IBM i
7.5

プログラミング
IBM Rational Development Studio for i
ILE RPG 解説書

SC09-2508-12

ノート

本書および本書で紹介する製品をご使用になる前に、971 ページの『特記事項』に記載されている情報をお読みください。

目次

ILE RPG 解説書

本書では、IBM i オペレーティング・システム で ILE RPG コンパイラーを使用してインプリメントされる RPG IV 言語について説明します。

本書には、以下の項目が含まれています。

- RPG IV の基本:
  - RPG IV 文字セット
  - RPG IV 予約語
  - コンパイラー指示
  - RPG IV プログラム・サイクル
  - ファイル
  - 標識
  - エラー処理
  - サブプロシージャー
- 定義:
  - データおよびプロトタイプの定義
  - データ・タイプおよびデータ形式
- RPG IV 仕様書:
  - 制御
  - ファイル記述
  - 定義
  - 入力
  - 演算
  - 出力
  - プロシー ジャー・
- データまたは装置を取り扱う方法:
  - 組み込み関数
  - 式
  - 命令コード

ILE RPG 解説書について

解説書については、このセクションをお読みください。

解説書については、このセクションをお読みください。

本書の対象読者

本書は、RPG IV プログラミング言語に精通しているプログラマーを対象にしています。

本書では、RPG IV 言語について詳しく説明します。 本書は、ILE RPG コンパイラーの使用法や RPG III プログラムを ILE RPG に変換する方法を説明するためのものではありません。 それらについては、「*IBM Rational Development Studio for i: ILE RPG プログラマーの手引き*」*(SD88-5042)* を参照してください。

本書を使用するためには、事前に下記についての知識が必要です。

- 該当する IBM i メニューと画面または制御言語 (CL) コマンドの使用法についての知識
- 「*ILE 概念*」*(SC41-5606)* で詳細に説明されている Integrated Language Environment® についての十分な理解

前提条件および関連情報

IBM i 技術情報を検索するための開始点として、IBM i Information Center を使用してください。 以下の Web サイトから Information Center にアクセスできます。

```
http://www.ibm.com/systems/i/infocenter/
```

IBM i Information Center には、ソフトウェアおよびハードウェア・インストール、Linux®、WebSphere®、Java™、高可用性、データベース、論理区画、CL コマンド、およびシステムのアプリケーション・プログラミング・インターフェース (API) などの新規および更新システム情報が含まれています。 また、システムのハードウェアおよびソフトウェアを計画、トラブルシューティング、および構成するのに役立つアドバイザーや検索機能も 提供されています。

関連資料のリストについては、969 ページの『参照文献』を参照してください。

ご意見送付方法 (英語のみ受付)

IBM にお客様のご意見をお寄せください。本書または他の IBM i の資料に関してご意見をお持ちの場合には、IBM 営業担当員までご連絡ください。

- 郵便の場合は下記あてにご送付ください。

  IBM Canada Ltd. LaboratoryInformation Development
  8200 Warden Avenue
  Markham, Ontario, Canada L6G 1C7

  「ご意見記入用紙」を米国以外の国からご郵送いただく場合は、 ご当地の IBM 事業所または IBM 担当員にお渡しいただいて構いませんが、所定の郵便料金はご負担いただきます。
- FAX の場合は、次の番号にお願いします。 1-845-491-7727
- 電子メールの場合は、以下の E メール・アドレスです。
  - 資料に関するご意見

    RCHCLERK@us.ibm.com
  - IBM i Information Center に関するご意見

    RCHINFOC@us.ibm.com

以下の項目の記入をお願いします。

- 資料名。
- 資料番号。
- ご意見の内容に該当するページ番号またはトピック。

新着情報

V3R1 以降の ILE RPG コンパイラーの各リリースで追加または変更された機能は、以下のとおりです。

RPG IV には、最初の V3R1 リリース以来いくつかのリリースが存在しています。 以下は、V3R1 から現行リリースに至るまでの各リリースにおいて行なわれた機能強化を列挙したものです。

このセクションを使用して、新しい RPG IV の機能にリンクして学習することができます。

注 : 本製品に対するこの情報は、RPG IV の 7.5 のリリースにおける最新のものです。 コンパイラーの前のリリースを使用している場合は、ユーザーのシステムでどの機能がサポートされるかを判断する必要があります。 例えば、7.4 システムを使用している場合は、7.5 リリースでの新機能はサポートされません。

7.5 での変更点

このセクションでは、7.5 で ILE RPG に対して行われた機能強化について説明します。

注 : これらの機能強化の多くは、7.3 および 7.4 の PTF でも使用可能です。 https://www.ibm.com/support/pages/rpg-cafe を参照して、プログラムをコンパイルして実行するシステムで必要となる 7.3 および 7.4 の PTF を判別してください。

新規命令コード DATA-GEN

DATA-GEN は、RPG 変数から JSON や CSV などの構造化文書を生成します。 文書の生成には、生成プログラムが必要です。 DATA-GEN 命令は、生成プログラムを呼び出して RPG 変数の名前と値を渡し、生成プログラムは、構造化された文書のテキストを DATA -GEN 命令に戻します。この命令は、情報を出力ファイルまたは出力 RPG 変数に入れます。

```
DCL-DS product QUALIFIED;
   name VARCHAR(25);
   id CHAR(10);
   price PACKED(9 : 2);
END-DS product;

DATA-GEN product %DATA('ProductInfo.JSON' : 'doc=file')
                %GEN('MYLIB/MYJSONGEN');
```

740 ページの『DATA-GEN (変数からの文書の生成)』、626 ページの『%DATA (文書 {:オプション})』、645 ページの『%GEN (生成プログラム {: オプション})』 を参照してください。

この機能強化は、2019 年後半のコンパイル時 PTF および実行時 PTF が適用された 7.3 および 7.4 でも使用可能です。

新規命令コード FOR-EACH

FOR-EACH 命令コードは、配列、%LIST、または %SUBARR の要素に対して反復処理を行う命令のグループを開始します。

FOR-EACH の第 1 オペランドは、配列または %LIST 内の現行要素の値を受け取る変数です。

ループは、ENDFOR 命令コードで終了します。

以下の例の *filenames* 配列の各要素は FOR-EACH グループで処理されます。 FOR-EACH 命令と ENDFOR 命令の間の命令コード・グループ内の *file* 変数には *filenames* 配列の現行要素の値が入ります。

```
DCL-S filenames CHAR(10) DIM(10);
DCL-S file CHAR(10);
DCL-S numFilenames INT(10);
...
FOR-EACH file in %SUBARR(filenames : 1 : numFilenames);
   process (file);
ENDFOR;
```

786 ページの『FOR-EACH (それぞれの場合)』を参照してください。

この機能強化は、2020 年後半のコンパイル時 PTF が適用された 7.3 および 7.4 でも使用可能です。

複数のサブフィールドによって配列データ構造をソートします。

配列データ構造をソートのためのサブフィールドをリストするには、%FIELDS を使用します。

以下の例では、データ構造配列 *info* がサブフィールド *dueDate* および *orderId* によってソートされています。 *info* の 2 つの要素の必要な順序を決定するために、サブフィールド *dueDate* が最初に比較されます。 2 つの要素の *dueDate* の値が等しい場合は、サブフィールド *orderId* を使用して *info* の 2 つの要素の順序が決定されます。

info(2).dueDate は *info(3).dueDate* より大きいため、要素 2 および要素 3 の相対順序を決定するために 2 番目のサブフィールド *orderId* を比較する必要はありません。

info(1).dueDate は *info(3).dueDate* と等しいため、要素 1 および要素 3 の相対順序を決定するために *info(1).orderId* が *info(3).orderId* と比較されます。

```
DCL-DS info QUALIFIED DIM(3);
   orderId packed(5);
   dueDate DATE(*ISO);
   quantity int(10);
END-DS;

info(1).orderId = 23456;
info(1).dueDate = D'2021-09-08';
info(1).quantity = 5;

info(2).orderId = 12345;
info(2).dueDate = D'2021-10-08';
info(2).quantity = 3;

info(3).orderId = 12345;
info(3).dueDate = D'2021-09-08';
info(3).quantity = 25;

SORTA info %FIELDS(dueDate : orderId);

// info(1).orderId = 12345
// info(1).dueDate = D'2021-09-08'
// info(1).quantity = 25
//
// info(2).orderId = 23456
// info(2).dueDate = D'2021-09-08'
// info(2).quantity = 5
//
// info(3).orderId = 12345
// info(3).dueDate = D'2021-10-08'
// info(3).quantity = 3
```

642 ページの『%FIELDS (ソートするためのサブフィールド)』を参照してください。

この機能強化は、2021 年後半のコンパイル時 PTF が適用された 7.3 および 7.4 でも使用可能です。

新規 IN 演算子

IN 演算子は、項目が配列または %LIST にあるか、あるいは範囲内にあるかを判別するために使用されます。

```
IF item IN %RANGE(lower_limit : upper_limit);
   ...
ENDIF;
IF item IN myArray;
   ...
ENDIF;

IF item IN %LIST(item1 : item2 : ...);
ENDIF;
```

601 ページの『IN 演算子』を参照してください。

この機能強化は、2020 年後半のコンパイル時 PTF が適用された 7.3 および 7.4 でも使用可能です。

新規組み込み関数 %RANGE

%RANGE 組み込み関数を使用することによって、値が他の 2 つの値の範囲にあるかどうかを検査することができます。

以下の例で、*num* が 1 以上 10 以下の場合、IF ステートメントは真になります。

```
DCL-S num PACKED(15:5);

IF num IN %RANGE(1 : 10);
   ...
ENDIF;
```

677 ページの『%RANGE (下限 : 上限)』を参照してください。

この機能強化は、2020 年後半のコンパイル時 PTF が適用された 7.3 および 7.4 でも使用可能です。

新規組み込み関数 %LIST

組み込み関数 %LIST を使用して一時配列を定義することができます。%LIST は、SORTA、%LOOKUPxx、および %SUBARR を除き、配列を使用できる場所であれば、どこででも使用できます。

以下の例では、*libraries* 配列の最初の 4 つの要素は %LIST 組み込み関数の値から割り当てられています。 *libraries* の最初の要素には値「QGPL」が割り当てられ、2 番目の要素には変数 *currentUserProfile* の値が割り当てられています。

```
DCL-S libraries CHAR(10) DIM(10);

libraries = %LIST ('QGPL'
                 : currentUserProfile
                 : 'QTEMP'
                 : getDevLib ());
```

656 ページの『%LIST (項目 { : 項目 { : 項目 ... } })』を参照してください。

この機能強化は、2020 年後半のコンパイル時 PTF が適用された 7.3 および 7.4 でも使用可能です。

新規組み込み関数 %LOWER および %UPPER

文字または UCS-2 ストリングの大/小文字の変換については、%lower 組み込み関数を使用して小文字に変換したり、%upper 組み込み関数を使用して大文字に変換したりできます。

変換の開始と長さを指定することにより、変換をストリングの一部のみに制限できます。

以下の例では、最初の文字の後に続く文字が小文字に変換されます。

```
DCL-S string VARCHAR(20);

string = %LOWER('GÖTEBORG' : 2);
// string = "Göteborg"
```

660 ページの『%LOWER および %UPPER (小文字または大文字に変換)』を参照してください。

この機能強化は、2021 年後半のコンパイル時 PTF および実行時 PTF が適用された 7.3 および 7.4 でも使用可能です。

新規組み込み関数 %SPLIT

%SPLIT を使用して、ストリングをサブストリングの一時配列に分割することができます。

デフォルトでは、ストリングはブランクで分割されます。 2 番目のパラメーターは、分割する文字を使用して指定できます。

以下の例では、%SPLIT の結果が配列に割り当てられています。

1. 最初の割り当てでは、ストリングがブランクで分割されます。
2. 2 番目の割り当てでは、ストリングがアスタリスクまたはブランクのいずれかで分割されます。

```
DCL-S array VARCHAR(20) DIM(10);

array = %SPLIT(' **One** **Two** **Three** '); // 1
// array(1) = '**One**'
// array(2) = '**Two**'
// array(3) = '**Three**'

array = %SPLIT(' **One** **Two** **Three** ' : ' *');  // 2
// array(1) = 'One'
// array(2) = 'Two'
// array(3) = 'Three'
```

689 ページの『%SPLIT (ストリングをサブストリングに分割)』を参照してください。

この機能強化は、2021 年後半のコンパイル時 PTF および実行時 PTF が適用された 7.3 および 7.4 でも使用可能です。

新規組み込み関数 %MAXARR および %MINARR

%MAXARR または %MINARR を使用して、配列内の最大値または最小値の指標を見つけることができます。

次に例を示します。

1. %LIST からの割り当ての後、配列内の最大値は「Yellow」(*array1(2)*) です。 最小値は「Blue」(*array1(3)*) です。
2. %MAXARR(array1) は 2 を返します。
3. %MINARR(array1) は 3 を返します。 %MINARR(array1) が *array1* の索引として使用される場合、*minVal* に割り当てられる値は「Blue」です。

```
DCL-S array1 VARCHAR(10) DIM(3);
DCL-S i INT(10);
DCL-S minVal LIKE(array1);

array1 = %LIST('Red' : 'Yellow' : 'Blue'); // 1

i = %MAXARR(array1); // 2
// i = 2

minVal = array1(%MINARR(array1)); // 3
// minVal = 'Blue'
```

663 ページの『%MAXARR および %MINARR (配列内の最大要素および最小要素)』を参照してください。

この機能強化は、7.4 および 2021 年後半のコンパイル時 PTF が適用された 7.4 でも使用可能です。

新規オプション *STRICTKEYS が EXPROPTS 制御キーワードに追加されます。

EXPROPTS(*STRICTKEYS) を指定すると、キーまたは %KDS のリストを使用してキー付きファイル操作の検索引数を指定するための規則がより厳密になります。

335 ページの『*STRICTKEYS』を参照してください。

この機能強化は、2021 年前半のコンパイル時 PTF が適用された 7.3 および 7.4 でも使用可能です。

文字を数値に変換する組み込み関数でのブランクを許可

新規オプション *ALWBLANKNUM が EXPROPTS 制御キーワードに追加されました。

EXPROPTS(*ALWBLANKNUM) を指定すると、数値変換組み込み関数 %DEC、%DECH、%FLOAT、%INT、%INTH、%UNS、および %UNSH でブランク文字の値を使用できます。 これらの組み込み関数は、文字オペランドがブランクであるか、長さがゼロである場合には、ゼロを返します。 DATA-INTO および XML-INTO で数値フィールドの値がブランクまたは空の場合、値にゼロが入ります。

以下の例では、EXPROPTS(*ALWBLANKNUM) が指定されているため、ブランク値による %DEC 命令の失敗は発生しません。

```
CTL-OPT EXPROPTS(*ALWBLANKNUM);

DCL-S num PACKED(15:5);
DCL-S string CHAR(10) INZ(*BLANKS);

num = %DEC(string : 63 : 15);
```

335 ページの『EXPROPTS(*MAXDIGITS | *RESDECPOS | *ALWBLANKNUM | *USEDECEDIT)』および 569 ページの『組み込み関数を使用して文字値を数値に変換するための規則』を参照してください。

この機能強化は、2020 年後半のコンパイル時 PTF および実行時 PTF が適用された 7.3 および 7.4 でも使用可能です。

文字を数値に変換する組み込み関数で桁区切り文字を許可

新規オプション *USEDECEDIT が EXPROPTS 制御キーワードに追加されました。

EXPROPTS(*USEDECEDIT) が指定されている場合、数値変換組み込み関数 %DEC、%DECH、%FLOAT、%INT、%INTH、%UNS、および %UNSH は、制御キーワード DECEDIT によって定義された設定に従ってピリオドとコンマ文字を解釈します。 このキーワードは、数値フィールドのデータを DATA-INTO および XML-INTO で処理する方法にも影響を与えます。

デフォルト、および DECEDIT('.') または DECEDIT('0.')、あるいはジョブ DECFMT の値が J でない場合の DECEDIT(*JOBRUN) では、ピリオドは小数点文字で、コンマは桁区切り文字 (桁区切り記号) です。

DECEDIT (',') または DECEDIT('0,')、あるいはジョブ DECFMT の値が J である場合の DECEDIT(*JOBRUN) では、コンマは小数点文字、ピリオドは数字区切り記号になります。

次に例を示します。

- キーワード EXPROPTS(*USEDECEDIT) が指定されています。
- キーワード DECEDIT が指定されていないため、小数点文字はデフォルトのピリオド、桁区切り文字はデフォルトのコンマになります。
- ストリング「1,234,567.89」は %DEC で許可されています。 コンマは桁区切り文字であるため、無視されます。
- %DEC 組み込み関数は 1234567.89 を返します。

```
CTL-OPT EXPROPTS(*USEDECEDIT);

DCL-S num PACKED(15:5);

num = %DEC('1,234,567.89' : 15 : 5);
```

335 ページの『EXPROPTS(*MAXDIGITS | *RESDECPOS | *ALWBLANKNUM | *USEDECEDIT)』および 569 ページの『組み込み関数を使用して文字値を数値に変換するための規則』を参照してください。

この機能強化は、2020 年後半のコンパイル時 PTF および実行時 PTF が適用された 7.3 および 7.4 でも使用可能です。

多重定義プロトタイプ

OVERLOAD キーワードは、プロトタイプの名前を OVERLOAD キーワードと一緒に使用することで呼び出すことができる他のプロトタイプのリストを定義します。 OVERLOAD キーワードが指定されたプロトタイプが呼び出し命令で使用される場合、コンパイラーは、呼び出しに指定されるパラメーターを使用して、OVERLOAD キーワードにリストされている候補プロトタイプのうちのどの候補プロトコルを呼び出すかを決定します。

/OVERLOAD 指示を使用すると、呼び出すプロシージャーをコンパイラーが決定する方法について、コンパイル・リストからより多くの情報を要求することができます。

以下の例で、*FORMAT* は OVERLOAD キーワードを使用して定義されています。

1. *FORMAT* の最初の呼び出しでは、パラメーターに日付タイプが指定されているため、*FORMAT_DATE* が呼び出されます。
2. *FORMAT* の 2 番目の呼び出しでは、パラメーターに時刻タイプが指定されているため、*FORMAT_TIME* が呼び出されます。
3. *FORMAT* の 2 番目の呼び出しでは、パラメーターに文字タイプが指定されているため、*FORMAT_MESSAGE* が呼び出されます。

```
DCL-PR format_date VARCHAR(100);
   dateParm DATE(*ISO) CONST;
END-PR;
DCL-PR format_time VARCHAR(100);
   timeParm TIME(*ISO) CONST;
END-PR;
DCL-PR format_message VARCHAR(100);
   msgid CHAR(7) CONST;
   replacement_text VARCHAR(100) CONST OPTIONS(*NOPASS);
END-PR;
DCL-PR format VARCHAR(100) OVERLOAD(format_time : format_date : format_message);
DCL-S result varchar(50);

result = format(%date());              // 1
result = format(%time());              // 2
result = format('MSG0100' : filename); // 3
```

470 ページの『OVERLOAD(プロトタイプ 1 { : プロトタイプ 2 ...})』および 83 ページの『/OVERLOAD DETAIL | NODETAIL』を参照してください。

この機能強化は、2019 年後半のコンパイル時 PTF が適用された 7.3 および 7.4 でも使用可能です。

プロトタイプ・パラメーターの OPTIONS(*EXACT)

プロトタイプ・パラメーターに OPTIONS(*EXACT) が指定されている場合、呼び出し先プロシージャーが渡されたパラメーターと同じ値を受け取るようにするために、追加の規則が適用されます。 例えば、OPTIONS(*EXACT) を指定しない場合、コンパイラーでは、渡されるパラメーターをプロトタイプ・パラメーターよりも長くすることができ、渡されるパラメーターとは異なるデータ構造に LIKEDS によって関連付けられているデータ構造を渡すことができます。 OPTIONS(*EXACT) を指定する場合、渡されたパラメーターをプロトタイプ・パラメーターよりも長くすることはできません。また、プロトタイプ・パラメーターが LIKEDS キーワードで定義されている場合、渡されるパラメーターは、LIKEDS によって同じデータ構造に関連付けられている必要があります。

以下の例のフィールド *fld10* をプロトタイプ p1 のパラメーター *parm5* に渡すことはできますが、呼び出されたプロシージャーは、*fld10* の完全な値「abcdefghij」ではなく、値「abcde」を受け取ります。

フィールド *fld10* をプロトタイプ p2 のパラメーター *parm5Exact* に渡すことはできません。*fld10* の長さは 10 であり、プロトタイプ・パラメーター *parm5Exact* の長さである 5 を超えているからです。

```
dcl-pr p1;
   parm5 char(5);
end-pr;
dcl-pr p2;
   parm5Exact char(5) OPTIONS(*EXACT);
end-pr;
dcl-s fld10 char(10) inz('abcdefghij');

p1 (fld10);
p2 (fld10);  // Error
```

459 ページの『OPTIONS(*EXACT)』を参照してください。

この機能強化は、2019 年後半のコンパイル時 PTF が適用された 7.3 および 7.4 でも使用可能です。

デバッガーで名前付き定数の値を表示
デバッガーで名前付き定数の値を表示できるようにするには、制御キーワード DEBUG(*CONSTANTS) を指定します。

```
CTL-OPT DEBUG(*CONSTANTS);
```

331 ページの『DEBUG{(*DUMP | *INPUT | *RETVAL | *XMLSAX | *NO | *YES)}』を参照してください。

この機能強化は、2021 年後半のコンパイル時 PTF が適用された 7.3 および 7.4 でも使用可能です。

デバッグ中にプロシージャーから戻り値を表示または変更
制御キーワード DEBUG(*RETVAL) が指定されている場合、特殊変数 *_QRNU_RETVAL* を評価して、プロシージャーが戻す値を表示または変更することができます。

この変数の値を表示または変更するには、プロシージャーの最後のステートメントにブレークポイントを設定します。

- 自由形式のコードでは、END-PROC ステートメントにブレークポイントを設定します。
- 固定形式のコードでは、Procedure-End 仕様にブレークポイントを設定します。

例えば、プログラマーが以下のプロシージャーをデバッグしているとします。

1. このプロシージャーは、値「abcde」を返します。
2. プログラマーは、END-PROC ステートメントにブレークポイントを設定します。
3. デバッグ・セッションで、プログラマーは戻り値を表示します。

```
CTL-OPT DEBUG(*RETVAL);
...
DCL-PROC myProc;
   DCL-PI *n CHAR(10) END-PI;

   RETURN 'abcde'; 1
END-PROC;          2
```

1. END-PROC ステートメントのデバッグ・セッションで、プログラマーは戻り値を表示します。
2. プログラマーは戻り値を「12345」に変更します。
3. 値「12345」がプロシージャーから戻されます。

```
> EVAL _qrnu_retval                    1
  _QRNU_RETVAL = 'abcde     '
> EVAL _qrnu_retval = '12345'          2
  _QRNU_RETVAL = '12345' = '12345     '
```

331 ページの『DEBUG{(*DUMP | *INPUT | *RETVAL | *XMLSAX | *NO | *YES)}』を参照してください。を参照してください。

この機能強化は、2020 年後半のコンパイル時 PTF が適用された 7.3 および 7.4 でも使用可能です。

オプションで、メイン・プロシージャーおよびエクスポートされたプロシージャーのプロトタイプを要求
パラメーター REQPREXP が CRTBNDRPG コマンドおよび CRTRPGMOD コマンドに追加され、エクスポートされるプロシージャーにプロトタイプが指定されていない場合のコンパイラーの応答を制御します。 さらに、キーワード REQPREXP を制御仕様書キーワードとして指定することもできます。

このパラメーターのデフォルトは *NO です。

*WARN レベルの場合、コンパイラーは警告診断メッセージを発行します。

*REQUIRE レベルの場合、コンパイラーは重大診断メッセージを発行します。これによりコンパイルが失敗します。

エクスポートされたプロシージャーまたはサイクル・メイン・プロシージャーのプロシージャー・インタフェースにキーワード REQPROTO(*NO) を指定して、プロトタイプがそのプロシージャーに必要でないことを指示できます。

エクスポートされたプロシージャーについては 344 ページの『REQPREXP(*NO | *WARN | *REQUIRE)』、538 ページの『REQPROTO(*NO)』、サイクル・メイン・プロシージャーのプロシージャー・インターフェースについては、476 ページの『REQPROTO(*NO)』 を参照してください。

この機能強化は、2020 年後半の 7.3 および 7.4 のコンパイル時 PTF でも提供されています。

マイクロ秒精度を戻す %TIMESTAMP

%TIMESTAMP() は、マイクロ秒精度のタイム・スタンプを返します。

%TIMESTAMP(*UNIQUE) を指定すると、返されるタイム・スタンプの 12 個の小数桁すべてにゼロ以外の値が入ります。 最初の 6 桁の小数秒はマイクロ秒精度を提供します。 最後の 6 桁は、固有のタイム・スタンプの作成に使用されます。

701 ページの『%TIMESTAMP (タイム・スタンプへの変換)』を参照してください。

この機能強化は、2020 年後半のコンパイル時 PTF および実行時 PTF が適用された 7.3 および 7.4 でも使用可能です。

%KDS の 2 番目のパラメーターには、変数または式を使用できます。

%KDS の 2 番目のパラメーターは、キー付き操作に使用するキーの数です。 2 番目のパラメーターに変数または式を指定できるようになりました。

```
DCL-F myfile KEYED;
DCL-DS keys LIKEREC(myRec : *KEY);
DCL-S numKeys INT(10);

numKeys = 3;
CHAIN %KDS(keys : numKeys) myRec;
READE %KDS(keys : numKeys - 1) myRec;
```

652 ページの『%KDS (データ構造の検索引数)』を参照してください。

この機能強化は、2020 年後半のコンパイル時 PTF および実行時 PTF が適用された 7.3 および 7.4 でも使用可能です。

LIKEDS キーワードの修飾名

LIKEDS キーワードに修飾名を指定できるようになりました。 次に例を示します。

1. データ構造サブフィールド *employees* は、データ構造 *employee_info* に直接定義されます。
2. プロシージャー check_employee のパラメーターの LIKEDS キーワードは、LIKEDS キーワードを使用して、修飾されたサブフィールド *employee_info.employeeds* と同じパラメーターを定義することによって定義されます。

```
DCL-DS employee_info QUALIFIED;
   num_employees INT(10);
   DCL-DS employees DIM(20);  // 1
      name VARCHAR(25);
      salary PACKED(7:2);
   END-DS;
END-DS;
DCL-PR check_employee IND;
   employee LIKEDS(employee_info.employees);  // 2
END-PI;
```

447 ページの『修飾データ構造名を使用した LIKEDS の指定』を参照してください。

この機能強化は、2019 年後半のコンパイル時 PTF が適用された 7.3 および 7.4 でも使用可能です。

***ON および *OFF を論理式にすることができる**

条件ステートメントに *ON および *OFF を直接使用できるようになりました。 例えば、無限 DOW ループが必要な場合は、「DOW *ON」とコーディングすることができます。

```
DOW *ON;
 ...
ENDDO;
```

この機能強化は、2019 年後半のコンパイル時 PTF が適用された 7.3 および 7.4 でも使用可能です。

LIKEDS キーワードの修飾名

LIKEDS キーワードに修飾名を指定できるようになりました。 次に例を示します。

1. データ構造サブフィールド *employees* は、データ構造 *employee_info* に直接定義されます。
2. プロシージャー check_employee のパラメーターの LIKEDS キーワードは、LIKEDS キーワードを使用して、修飾されたサブフィールド *employee_info.employeeds* と同じパラメーターを定義することによって定義されます。

```
DCL-DS employee_info QUALIFIED;
   num_employees INT(10);
   DCL-DS employees DIM(20);  // 1
      name VARCHAR(25);
      salary PACKED(7:2);
   END-DS;
END-DS;
DCL-PR check_employee IND;
   employee LIKEDS(employee_info.employees);  // 2
END-PI;
```

447 ページの『修飾データ構造名を使用した LIKEDS の指定』を参照してください。

この機能強化は、2019 年後半のコンパイル時 PTF が適用された 7.3 および 7.4 でも使用可能です。

ILE RPG では BOOLEAN データ・タイプがサポートされています。

データベース・ファイル内のフィールドがタイプ BOOLEAN で定義されている場合、RPG ではそのフィールドのタイプが標識タイプと見なされます。

表 *1. 7.4* 以降に変更された言語要素*:* 組み込み関数

要素	説明
%KDS	%KDS の 2 番目のパラメーターとして変数を指定できるようになりました。652 ページの『%KDS (データ構造の検索引数)』を参照してください。
%TIMESTAMP	%TIMESTAMP() は、マイクロ秒精度の値を戻します。%TIMESTAMP(*UNIQUE) は、固有のタイム・スタンプを返します。 701 ページの『%TIMESTAMP (タイム・スタンプへの変換)』を参照してください。

表 *2. 7.4* 以降に変更された言語要素*:* 制御仕様書キーワード

要素	説明
DEBUG キーワード	• プロシージャーから戻り値をデバッグする場合は、パラメーター *RETVAL を指定します。 • デバッガーで名前付き定数の値を表示する場合は、パラメーター *CONSTANTS を指定します。 331 ページの『DEBUG{(*DUMP | *INPUT | *RETVAL | *XMLSAX | *NO | *YES)}』を参照してください。 を参照してください。

表 2. 7.4 以降に変更された言語要素: 制御仕様書キーワード (続き)	
要素	**説明**
EXPROPTS キーワード	パラメーター *ALWBLANKNUM、*USEDECEDIT、および *STRICTKEYS を EXPROPTS キーワードに指定できるようになりました。 335 ページの『EXPROPTS(*MAXDIGITS | *RESDECPOS | *ALWBLANKNUM | *USEDECEDIT)』を参照してください。

表 3. 7.4 以降に変更された言語要素: 定義	
要素	**説明**
LIKEDS	修飾名を指定します。447 ページの『修飾データ構造名を使用した LIKEDS の指定』を参照してください。
OPTIONS(*EXACT)	プロトタイプ・パラメーターの OPTIONS キーワードには *EXACT 値を指定することができます。これは、渡されたパラメーターの妥当性を検査するときにコンパイラーが追加規則を適用する必要があることを示します。 459 ページの『OPTIONS(*EXACT)』を参照してください。

表 4. 7.4 以降に変更された言語要素: 命令コード	
要素	**説明**
SORTA 命令コード	%FIELDS は、ソートのためのサブフィールドをリストするために SORTA と一緒に指定することができます。 642 ページの『%FIELDS (ソートするためのサブフィールド)』を参照してください。

表 5. 7.4 以降の新規言語要素: 指示	
要素	**説明**
/OVERLOAD	指示の後のステートメントで多重定義プロトタイプの呼び出しに関する詳細情報を取得するには、/OVERLOAD DETAIL を指定します。 詳細情報の取得を停止するには、/OVERLOAD NODETAIL を指定します。 83 ページの『/OVERLOAD DETAIL | NODETAIL』を参照してください。

表 6. 7.4 以降の新規言語要素: 制御仕様書キーワード	
要素	**説明**
OVERLOAD	多重定義プロトタイプ (OVERLOAD キーワードを指定したプロトタイプ) の名前によって呼び出せる候補プロトタイプを定義します。470 ページの『OVERLOAD(プロトタイプ 1 { : プロトタイプ 2 ...})』を参照してください。
REQPROTO	REQPROTO(*NO) をサイクル・メイン・プロシージャーのプロシージャー・インターフェース上で指定して、プロトタイプが必要でないことを指示することができます。 476 ページの『REQPROTO(*NO)』を参照してください。

表 7. 7.4 以降の新規言語要素: 制御仕様キーワード	
要素	**説明**
REQPREXP	メイン・プロシージャーおよびエクスポートされたサブプロシージャーにプロトタイプが必要かどうかを制御します。 344 ページの『REQPREXP(*NO | *WARN | *REQUIRE)』を参照してください。

表 8. 7.4 以降の新規言語要素: 式演算子	
要素	**説明**
IN	第 1 オペランドが第 2 オペランドにあるかどうかをテストする 2 進条件ステートメントで使用されます。 601 ページの『IN 演算子』を参照してください。

表 9. 7.4 以降の新規言語要素: 組み込み関数	
要素	**説明**
%GEN	DATA-GEN 命令コードの生成プログラムを指定します。645 ページの『%GEN (生成プログラム {: オプション})』を参照してください。
%LIST	一時配列を返します。656 ページの『%LIST (項目 { : 項目 { : 項目 ... } })』を参照してください。
%LOWER	小文字に変換された入力ストリングを戻します。 660 ページの『%LOWER (小文字に変換)』を参照してください。
%MAXARR	配列内の最大値の索引を返します。663 ページの『%MAXARR および %MINARR (配列内の最大要素および最小要素)』を参照してください。
%MINARR	配列内の最小値の索引を返します。663 ページの『%MAXARR および %MINARR (配列内の最大要素および最小要素)』を参照してください。
%RANGE	IN 演算子を使用して検査する範囲を指定します。677 ページの『%RANGE (下限 : 上限)』を参照してください。
%SPLIT	サブストリングの配列を返します。689 ページの『%SPLIT (ストリングをサブストリングに分割)』を参照してください。
%UPPER	大文字に変換された入力ストリングを返します。 707 ページの『%UPPER (大文字に変換)』を参照してください。

表 10. 7.4 以降の新規言語要素: 命令コード	
要素	**説明**
DATA-GEN	RPG 変数から構造化文書を生成します。 740 ページの『DATA-GEN (変数からの文書の生成)』を参照してください。
FOR-EACH	配列、%LIST、またはサブ配列の項目を反復処理します。786 ページの『FOR-EACH (それぞれの場合)』を参照してください。

表 11. 7.4 以降の新規言語要素: プロシージャー仕様書キーワード	
要素	**説明**
REQPROTO	REQPROTO(*NO) は、エクスポートされたプロシージャーにプロトタイプが必要でないことを示します。 538 ページの『REQPROTO(*NO)』を参照してください。

7.4 での変更点

このセクションでは、ILE RPG の 7.4 における機能強化について説明します。

可変次元配列

可変次元配列は DIM(*AUTO:maximum_elements) または DIM(*VAR:maximum_elements) を使用して定義されます。 DIM キーワードの第 2 パラメーターは、配列エレメントの最大数を示します。

%ELEM 組み込み関数に値を代入することによって可変次元配列の次元を変更できます。

```
DCL-S array_auto CHAR(10) DIM(*AUTO:100);
DCL-S array_var CHAR(10) DIM(*VAR:100);
%ELEM(array_auto) = 5;
%ELEM(array_var) = 5;
```

DIM(*AUTO) で定義された可変次元配列の次元は、現在のエレメント数より大きいエレメントへの代入ステートメントがあると増やされます。 以下の例では、DIM(*AUTO:1000) で配列 *arr* が定義されています (1)。新規要素への代入があると、現在の配列要素数は自動的に増やされます。 配列の次元は、arr(5) への代入の前はゼロです。 代入の後の次元は 5 です (2)。要素 1 から 4 は、INZ キーワードで指定された値 ('?') で初期化されます。要素 5 は、代入ステートメントで指定された値 ('x') に設定されます。

```
DCL-s arr CHAR(10) DIM(*AUTO:1000) INZ('?'); // 1

arr(5) = 'x'; // 2
```

DIM(*AUTO) で定義された配列が代入ステートメントによって変更される際に、配列の索引として *NEXT を指定できます。 配列の次元は 1 だけ増やされます。 以下の例では、DIM(*AUTO:1000) で配列 *custArray* が定義されています (1)。custArray(*NEXT) に代入されるたびに、現在の配列要素数は自動的に 1 ずつ増えます (2).

```
DCL-F custfile;
DCL-DS custDs LIKEREC(custRec);
DCL-S numCustElems INT(10);
DCL-DS custArray LIKEDS(custDs) DIM(*AUTO:1000); // 1

READ file custDs;
DOW NOT %EOF;
   custArray(*next) = custDs; // 2
   READ file custDs;
ENDDO;
numCustElems = %ELEM(custArray);
```

422 ページの『可変次元配列』を参照してください

DIM(*CTDATA)

コンパイル時配列またはテーブルの次元がコンパイル時データのレコード数によって決まることを指定するには、DIM(*CTDATA) を指定します。

421 ページの『DIM({*AUTO:|*CTDATA|*VAR:}数値定数)』を参照してください

SAMEPOS キーワード

SAMEPOS キーワードは、サブフィールドを別のサブフィールドと同じ開始位置に置きます。

480 ページの『SAMEPOS(サブフィールド)』を参照してください

この機能強化は、PTF が適用された 7.3 でも使用可能です。

新規 PSDS サブフィールド

以下に示す新しいサブフィールドがプログラム状況データ構造に追加されました。

- 内部ジョブ ID (380 から 395 までの位置)。
- システム名 (396 から 403 までの位置)。

163 ページの『プログラム状況データ構造』を参照してください

この機能強化は、PTF が適用された 7.3 でも使用可能です。

ON-EXIT セクション

ON-EXIT セクションは、正常終了か異常終了かに関係なくプロシージャーが終了するたびに実行されます。

以下の例では、プロシージャーはヒープ・ストレージを割り振ります。 ON-EXIT 命令に続くコードは常に実行されるため、ヒープ・ストレージは必ず割り振り解除されます。 ON-EXIT セクションは、未処理の例外が原因でプロシージャーが突然終了した場合、ジョブが終了したためにプロシージャーが取り消された場合、あるいは、プロシージャーのメイン部分の終わりに達したため、または RETURN 命令に達したために正常にプロシージャーが戻った場合のいずれでも、実行されます。

```
dcl-proc myproc;
   p = %alloc(100);

   ...
on-exit;
   dealloc(n) p;
end-proc;
```

839 ページの『ON-EXIT (終了時)』を参照してください

この機能強化は、PTF が適用された 7.2 および 7.3 でも使用可能です。

ネストされたデータ構造サブフィールド

修飾されたデータ構造がフリー・フォーム構文を使用して定義される際、ネストされたデータ構造サブフィールドとしてサブフィールドを直接定義できます。 以下のデータ構造内の *street* サブフィールドは *product.manufacturer.address.street* として参照されます。

```
DCL-DS product QUALIFIED;
   name VARCHAR(25);
   id CHAR(10);
   DCL-DS manufacturer;
      name VARCHAR(10);
      DCL-DS address;
         street VARCHAR(50);
         city VARCHAR(25);
      END-DS address;
      active IND;
   END-DS manufacturer;
   price PACKED(9 : 2);
END-DS product;
```

400 ページの『ネストされたデータ構造サブフィールド』を参照してください。

この機能強化は、PTF が適用された 7.2 および 7.3 でも使用可能です。

新規命令コード DATA-INTO

DATA-INTO は、構造化文書 (JSON など) からデータを読み取って RPG 変数に入れます。 これには、文書を構文解析するためのパーサーが必要です。 DATA-INTO 命令はパーサーを呼び出し、パーサーが文書内の情報を DATA-INTO 命令に返すと、その情報を RPG 変数に入れます。

```
DCL-DS product QUALIFIED;
   name VARCHAR(25);
   id CHAR(10);
   price PACKED(9 : 2);
END-DS product;

DATA-INTO product %DATA('ProductInfo.JSON' : 'doc=file')
                 %PARSER('MYLIB/MYJSONPARS');
```

743 ページの『DATA-INTO (文書の変数への構文解析)』、626 ページの『%DATA (文書 {:オプション})』、および 675 ページの『%PARSER(パーサー {: オプション})』を参照してください。

この機能強化は、PTF が適用された 7.2 および 7.3 でも使用可能です。

組み込み関数 %PROC()

%PROC() は、現行プロシージャーの外部名を返します。 676 ページの『%PROC (現行プロシージャーの戻り値の名前)』を参照してください。

この機能強化は、PTF が適用された 7.2 および 7.3 でも使用可能です。

複合修飾名を使用できる場所の増加

以下の場所で複合修飾名を使用できるようになりました。

- 組み込み関数 %ELEM。 637 ページの『%ELEM (要素数の検索)』を参照してください。
- 組み込み関数 %SIZE。 687 ページの『%SIZE (サイズ (バイト数) の検索)』を参照してください。
- 命令コード DEALLOC。 752 ページの『DEALLOC (記憶域の解放)』を参照してください。
- 命令コード RESET。 862 ページの『RESET (リセット)』を参照してください。

この機能強化は、PTF が適用された 7.2 および 7.3 でも使用可能です。

新規組み込み関数 %MAX および %MIN

%MAX はオペランドの最大値を返し、%MIN はオペランドの最小値を返します。 計算式で %MAX または %MIN が使用される場合、少なくとも 2 つのオペランドが必要ですが、オペランド数の上限はありません。 宣言ステートメントで %MAX または %MIN が使用される場合は、正確に 2 つの数値オペランドがなければなりません。

以下の例では、代入命令の後、*maxval* の値は「Zorro」、*minval* の値は「Batman」です。

```
DCL-S maxval VARCHAR(10);
DCL-S minval VARCHAR(10);
DCL-S name1 VARCHAR(20) INZ('Robin');
DCL-S name2 VARCHAR(10) INZ('Batman');

maxval = %MAX(name1 : name2 : 'Zorro');
minval = %MIN(name1 : name2 : 'Zorro');
```

662 ページの『%MAX および %MIN (最大値または最小値)』を参照してください。

この機能強化は、PTF が適用された 7.2 および 7.3 でも使用可能です。

データ構造の長さを位置合わせの倍数で定義する ALIGN(*FULL)

ALIGN キーワードに *FULL が指定されている場合、データ構造の長さは位置合わせサイズの倍数になります。 *FULL が指定されていない場合、データ構造の長さはサブフィールドの最大終了位置によって決まります。 *FULL パラメーターは、関数または API にパラメーターとして渡される位置合わせされたデータ構造を定義する場合、または、位置合わせされたデータ構造の配列のエレメント間の距離を決定するために %SIZE を使用する場合に指定します。

例えば、以下の 2 つのデータ構造には、浮動小数点サブフィールドがあり、その後に長さ 1 の文字サブフィールドが続いています。 これらのデータ構造の両方とも、位置合わせサイズは 8 です。 データ構造 DS_ALIGN_FULL のサイズは 16 ですが、データ構造 DS_ALIGN のサイズは 9 です。

```
DCL-DS ds_align_full ALIGN(*FULL) QUALIFIED;
    sub1 FLOAT(8);
    sub2 CHAR(1);
END-DS;

DCL-DS ds_align ALIGN QUALIFIED;
    sub1 FLOAT(8);
    sub2 CHAR(1);
END-DS;
```

414 ページの『ALIGN{(*FULL)}』を参照してください。

この機能強化は、PTF が適用された 7.2 および 7.3 でも使用可能です。

CRTBNDRPG および CRTRPGMOD の新規パラメーター TGTCCSID によるユニコード・ソースからのコンパイルのサポート

ILE RPG コンパイラーは、通常は 1 次ソース・ファイルの CCSID でコンパイルのソース・ファイルを読み取ります。 コンパイラーは EBCDIC CCSID のみのソースの読み取りをサポートしているため、これは、1 次ソース・ファイルが UTF-8 や UTF-16 といったユニコード CCSID の場合はコンパイルが失敗することを意味します。

CRTBNDRPG および CRTRPGMOD の TGTCCSID パラメーターを使用すると、RPG プログラマーはコンパイルのためにコンパイラーがソース・ファイルの読み取りに使用する CCSID を指定できます。 TGTCCSID(*JOB) を指定するか、TGTCCSID(37) または TGTCCSID(5035) のように特定の EBCDIC CCSID を指定します。

デフォルトは TGTCCSID(*SRC) です。

この機能強化は、PTF が適用された 7.1、7.2、および 7.3 でも使用可能です。

表 *12. 7.3* 以降に変更された言語要素*:* 組み込み関数

要素	記述
%ELEM	配列が可変次元配列の場合、第 2 パラメーター *ALLOC、*KEEP、または *MAX を指定できます。 637 ページの『%ELEM (要素数の検索)』を参照してください。

表 *13. 7.3* 以降に変更された言語要素*:* 定義

要素	記述
DIM キーワード	• このパラメーターを *AUTO または *VAR にすることで、可変次元配列を定義できます。 422 ページの『可変次元配列』を参照してください。 • このパラメーターを *CTDATA にすることで、配列またはテーブルの次元がコンパイル時データのレコード数から決定されることを指示できます。 421 ページの『DIM({*AUTO:|*CTDATA|*VAR:}数値定数)』を参照してください。
DCL-DS 命令コード	修飾されたデータ構造内に DCL-DS をネストできます。 400 ページの『ネストされたデータ構造サブフィールド』を参照してください。
ALIGN(*FULL)	ALIGN キーワードにパラメーター *FULL を指定すると、データ構造の長さが位置合わせの倍数になります。 414 ページの『ALIGN{(*FULL)}』を参照してください。

表 *14. 7.3* 以降の新しい言語要素*:* 組み込み関数

要素	記述
%DATA	文書が DATA-INTO 命令コードによって構文解析されることを指定します。 626 ページの『%DATA (文書 {:オプション})』を参照してください。
%MAX	オペランドの最大値を返します。 662 ページの『%MAX (最大値)』を参照してください。
%MIN	オペランドの最小値を返します。 662 ページの『%MIN (最小値)』を参照してください。
%PARSER	DATA-INTO 命令コードのパーサーを指定します。 675 ページの『%PARSER(パーサー {: オプション})』を参照してください。
%PROC	現行プロシージャーの外部名を返します。 676 ページの『%PROC (現行プロシージャーの戻り値の名前)』を参照してください。

表 15. 7.3 以降の新しい言語要素: 定義キーワード	
要素	**記述**
SAMEPOS	サブフィールドを別のサブフィールドと同じ開始位置に置きます。 480 ページの『SAMEPOS(サブフィールド)』を参照してください。

表 16. 7.3 以降の新しい言語要素: 命令コード	
要素	**記述**
DATA-INTO	構造化文書から RPG 変数にデータをインポートします。 743 ページの『DATA-INTO (文書の変数への構文解析)』を参照してください。
ON-EXIT	正常終了か異常終了かに関わらずプロシージャーが終了した時点で実行されるコードのセクションを開始します。 839 ページの『ON-EXIT (終了時)』を参照してください。

7.3 での変更点

このセクションでは、7.3 において ILE RPG に対して行われた機能強化について説明します。

完全自由形式ソースのサポート

先頭行に特殊なディレクティブ **FREE が指定されている RPG ソースは、自由形式のコードのみを含みます。 コードは 1 桁目から始めて行の最後まで続けることができます。

完全自由形式ソースではソース行の長さに事実上制限はありません。

完全自由形式ソース内では固定形式のコードは許可されませんが、6 桁目から 80 桁目までのみを使用する桁限定ソースを /COPY ディレクティブまたは /INCLUDE ディレクティブを使用して組み込むことができます。

309 ページの『完全自由形式ステートメント』を参照してください。

新しい組み込み関数 %SCANR (スキャン・リバース)

%SCANR 組み込み関数は、%SCAN 組み込み関数に似ていますが、検索引数の最初の出現ではなく最後の出現を検出します。

以下の例では、ストリング中の「***」について、%SCAN を使用して最初の出現を検出し、%SCANR を使用して最後の出現を検出しています。

```
string = 'The title is *** Chapter 1 ***.';
p1 = %SCAN ('***' : string);
p2 = %SCANR ('***' : string);
// p1 = 14
// p2 = 28
```

683 ページの『%SCANR (文字の逆方向走査)』を参照してください。

組み込み関数 %SCAN の長さパラメーター

長さパラメーターを使用して、検索対象となるソース・ストリングの量を制限できます。

以下の例では、最初の %SCAN 組み込み関数では 26 が返されます。 2 番目の関数では、開始位置 1 と長さ 10 で示されるサブストリング中に値「abc」が見つからないため、0 が返されます。

```
string = 'The alphabet begins with abc.';
p1 = %SCAN ('abc' : string);
p2 = %SCAN ('abc' : string : 1 : 10);
// p1 = 26
// p2 = 0
```

681 ページの『%SCAN (文字の走査)』を参照してください。

ファイルに対する ALIAS サポートの拡張

ALIAS キーワードをすべての外部記述ファイルに指定できるようになりました。

修飾されていないグローバル・ファイルに対して ALIAS キーワードが指定されると、フィールドの代替名が RPG プログラム内で使用可能になります。

以下の例で、MYFILE ファイル内の *REQALC* フィールドの代替名は *REQUIRED_ALLOCATION* です。ALIAS キーワードは、このフィールドの名前が RPG プログラム内で REQUIRED_ALLOCATION になることを示します。 364 ページの『ALIAS』を参照してください。

```
dcl-f myfile ALIAS;

read myfile;
if required_allocation <> 0
and size > 0;
  ...
```

入出力操作用のデータ構造に関する規則の緩和

- タイプ *ALL を指定して定義された外部記述データ構造または LIKEREC データ構造を、任意の入出力操作の結果データ構造として使用できます。 576 ページの『ファイル操作』を参照してください。

```
dcl-f myfile usage(*input : *output : *update);
dcl-ds ds extname('MYFILE' : *ALL);

read myfile ds;
update myfmt ds;
write myfmt ds;
```

- DISK ファイルのレコード・フォーマットのデータ構造が、第 2 パラメーターなしで LIKEREC を使用して定義されていて、かつ、出力バッファーのレイアウトが入力バッファーのレイアウトと同一である場合、このデータ構造を任意の入出力操作の結果データ構造として使用できます。

```
dcl-f myfile usage(*input : *output : *update);
dcl-ds ds likerec(fmt);

read myfile ds;
update myfmt ds;
write myfmt ds;
```

ヌル可能フィールドに関する機能強化

- EXTNAME キーワードまたは LIKEREC キーワードを使用してデータ構造を定義するときに、*NULL を追加の抽出タイプとしてコーディングして、サブフィールドがすべて標識であることを指定できます。 外部ファイルがデータベース・ファイルである場合、定義されるデータ構造はファイルのヌル・バイト・マップと一致します。 431 ページの『EXTNAME(ファイル名{:形式名}{:*ALL| *INPUT| *OUTPUT|*KEY|*NULL})』および 449 ページの『LIKEREC(intrecname{:extract-types})』を参照してください。
- NULLIND キーワードを使用して以下を行います。
  - フィールドをヌル可能と定義する。
  - ユーザー独自の標識フィールドをフィールドのヌル標識であると定義する。
  - EXTNAME(*NULL) または LIKEREC(*NULL) を使用して定義されるユーザー独自の標識データ構造を、別のデータ構造のヌル標識であると定義する。 451 ページの『NULLIND{(ヌル標識)}』を参照してください。

PCML 機能強化

- プログラム・インターフェース情報内の名前が RPG ソース・ファイル内に定義される名前と同じ大/小文字で生成されるように、PGMINFO 制御仕様キーワードに *DCLCASE パラメーターを指定します。342 ページの『PGMINFO(*PCML | *NO | *DCLCASE { : *MODULE | *V6 | *V7 ... })』を参照してください。
- モジュールが作成されるときにプログラム・インターフェース情報に組み込まれる必要のあるプロシージャーに対してプロシージャー仕様キーワード内で PGMINFO(*YES) を指定するか、または、組み込まれる必要のないプロシージャーに対して PGMINFO(*NO) を指定します。536 ページの『PGMINFO(*YES | *NO)』を参照してください。

DCLOPT(*NOCHGDSLEN)

入力仕様、出力仕様、または計算仕様を使用するデータ構造の長さが変更されることを防止するには、DCLOPT(*NOCHGDSLEN) を指定します。 DCLOPT(*NOCHGDSLEN) を指定すると、%SIZE(*data-structure*) をより多くの自由形式宣言で使用できるようになります。330 ページの『DCLOPT(*NOCHGDSLEN)』を参照してください。

表 *17. 7.2* 以降に変更された言語要素*:* 制御仕様キーワード

要素	説明
PGMINFO キーワード	*DCLCASE パラメーターを指定すると、プログラム・インターフェース情報内の名前が RPG ソース･ファイルにコーディングされた名前と同じ大/小文字で生成されます。342 ページの『PGMINFO(*PCML | *NO | *DCLCASE { : *MODULE | *V6 | *V7 ... })』を参照してください。

表 *18. 7.2* 以降に変更された言語要素*:* ファイル仕様キーワード

要素	説明
ALIAS キーワード	すべての外部記述ファイルに対して指定できます。364 ページの『ALIAS』を参照してください。

表 *19. 7.2* 以降に変更された言語要素*:* 定義仕様キーワード

要素	説明
EXTNAME キーワード	抽出タイプ *NULL。431 ページの『EXTNAME(ファイル名{:形式名}{:*ALL| *INPUT| *OUTPUT|*KEY|*NULL})』を参照してください。
LIKEREC キーワード	抽出タイプ *NULL。449 ページの『LIKEREC(intrecname{:extract-types})』を参照してください。

表 *20. 7.2* 以降に変更された言語要素*:* 組み込み関数

要素	説明
%SCAN 組み込み関数	%SCAN 組み込み関数では、検索対象の長さを指定する第 4 パラメーターがサポートされるようになりました。681 ページの『%SCAN (文字の走査)』を参照してください。

表 *21. 7.2* 以降の新しい言語要素*:* ディレクティブ

要素	説明
特殊なディレクティブ **FREE	**FREE は、ソースが完全自由形式であり、ソース行の 1 桁目から最後までが RPG コードであることを示します。309 ページの『完全自由形式ステートメント』を参照してください。

表 22. 7.2 以降の新しい言語要素: 制御仕様キーワード	
要素	**説明**
DCLOPT (*NOCHGDSLEN) キーワード	入力仕様、出力仕様、または計算仕様を使用するデータ構造のサイズ変更を禁止します。 330 ページの『DCLOPT(*NOCHGDSLEN)』を参照してください。

表 23. 7.2 以降の新しい言語要素: 定義仕様キーワード	
要素	**説明**
NULLIND キーワード	1 つの項目を別の項目の 1 つまたは複数のヌル標識として関連付けます。 451 ページの『NULLIND{(ヌル標識)}』を参照してください。

表 24. 7.2 以降の新しい言語要素: プロシージャー仕様キーワード	
要素	**説明**
PGMINFO キーワード	これを使用すると、モジュールが作成されるときにプログラム・インターフェース情報内にインターフェースが記述されるプロシージャーを制御できます。 536 ページの『PGMINFO(*YES | *NO)』を参照してください。

表 25. 7.2 以降の新しい言語要素: 組み込み関数	
要素	**説明**
%SCANR (スキャン・リバース)	あるストリングが、別のストリング内で最後に出現する位置を検出します。 683 ページの『%SCANR (文字の逆方向走査)』を参照してください。

7.2 での変更点

このセクションでは、7.2 において ILE RPG に対して行われた機能強化について説明します。

自由形式の制御ステートメント、ファイル・ステートメント、定義ステートメント、およびプロシージャー・ステートメント

- 自由形式の制御ステートメントは、CTL-OPT で開始し、セミコロンで終了します。 319 ページの『自由形式の制御ステートメント』を参照してください。

```
CTL-OPT OPTION(*SRCSTMT : *NODEBUGIO)
        ALWNULL(*USRCTL);
```

- 自由形式のファイル定義ステートメントは、DCL-F で開始し、セミコロンで終了します。 350 ページの『自由形式のファイル定義ステートメント』を参照してください。

  以下のステートメントは、3 つのファイルを定義します。

  1. 入力と更新のためにオープンされる外部記述 DISK ファイル。
  2. 入力と出力のためにオープンされる外部記述 WORKSTN ファイル。
  3. レコード長が 132 のプログラム記述 PRINTER ファイル。

```
DCL-F custFile usage(*update) extfile(custFilename);
DCL-F screen workstn;
DCL-F qprint printer(132) oflind(qprintOflow);
```

- 自由形式のデータ定義ステートメントは DCL-C、DCL-DS、DCL-PI、DCL-PR、または DCL-S で開始し、セミコロンで終了します。 391 ページの『自由形式の定義ステートメント』を参照してください。

  以下のステートメントは、いくつかの項目を定義します。

  1. 名前付き定数 *MAX_ELEMS*。

2. 独立可変長文字フィールド *fullName*。
3. 整数サブフィールド *num* と UCS-2 サブフィールド *address* のある修飾データ構造。
4. プロシージャー「Qp0lRenameUnlink」のプロトタイプ。

```
DCL-C MAX_ELEMS 1000;
DCL-S fullName VARCHAR(50)
               INZ('Unknown name');
DCL-DS ds1 QUALIFIED;
  num INT(10);
  address UCS2(100);
END-DS;
DCL-PR Qp0lRenameUnlink INT(10) EXTPROC(*DCLCASE);
   oldName POINTER VALUE OPTIONS(*STRING);
   newName POINTER VALUE OPTIONS(*STRING);
END-PR;
```

- 自由形式のプロシージャー定義ステートメントは DCL-PROC で開始し、セミコロンで終了します。プロシージャーの終了には、END-PROC ステートメントが使用されます。 532 ページの『自由形式のプロシージャー・ステートメント』を参照してください。

  以下は、自由形式のサブプロシージャー定義の例です。

```
DCL-PROC getCurrentUserName EXPORT;
   DCL-PI *n CHAR(10) END-PI;
   DCL-S curUser CHAR(10) INZ(*USER);

   RETURN curUser;
END-PROC;
```

- /FREE ディレクティブと /END-FREE ディレクティブは不要になりました。 これらの指示は、コンパイラーによって無視されます。
- 自由形式のステートメントと固定形式のステートメントは、混用できます。

```
         IF endDate < beginDate;
C                    GOTO      internalError
         ENDIF;
         duration = %DIFF(endDate : beginDate : *days);
         . . .
C     internalError TAG
```

ヒント： 310 ページの『認識しておく必要のある固定形式と自由形式の相違点』を参照してください。

英数字データでの CCSID のサポート

- モジュールのデフォルトの英数字の CCSID には、UTF-8 や 16 進数などのより多くの CCSID を設定できるようになりました。
- 英数字データを CCSID を使用して定義できます。 サポートされる CCSID は、以下のとおりです。
  - 1 バイトと混合バイトの EBCDIC CCSID
  - 1 バイトと混合バイトの ASCII CCSID
  - UTF-8
  - 16 進数

  418 ページの『CCSID 定義キーワード』を参照してください。

外部英数字サブフィールドの CCSID
外部記述データ構造に CCSID(*EXACT) を使用して、英数字サブフィールドにファイル内のフィールドと同じ CCSID を指定するよう示します。

16 進データには CCSID 変換は実行されません。
16 進データの暗黙的または明示的な変換には、CCSID 変換は許可されません。

16 進データには、以下が含まれます。

- 16 進数リテラル
- CCSID(*HEX) を使用して定義された英数字データとグラフィック・データ

- ファイルに対して DATA キーワードが有効であり、ファイル内のフィールドの CCSID が 65535 の場合の、外部記述ファイルのバッファー内の英数字データとグラフィック・データ
- ファイル内のフィールドの CCSID が 65535 の場合の、CCSID(*EXACT) を使用して定義された外部記述データ構造内の英数字データとグラフィック・データ

連結のための暗黙的な変換
コンパイラーは、連結式に対して、英数字データ、グラフィック・データ、および UCS-2 間で暗黙的な変換を実行します。 261 ページの『変換』を参照してください。

ジョブ CCSID への変換なしのデータベース・ファイルのオープン
英数字データとグラフィック・データから入出力操作のジョブ CCSID に変換 (またはその逆に変換) されないようにデータベース・ファイルをオープンするように指定するには、制御キーワード OPENOPT(*NOCVTDATA) またはファイル・キーワード DATA(*NOCVT) を使用します。 340 ページの『OPENOPT (* {NO }INZOFL * {NO }CVTDATA)』を参照してください。

CCSID、日付形式、または時刻形式のデフォルトの一時的な変更
CCSID、日付形式、および時刻形式のデフォルト値を設定するには、/SET ディレクティブと /RESTORE ディレクティブを使用します。 81 ページの『/SET』を参照してください。

%SUBDT によって戻される長さの制御
%SUBDT のオプションの 3 つ目のパラメーターは、結果の桁数を指定することを可能にします。 例えば、年の値は、%SUBDT(MyDate:*YEARS:4) のように 4 桁の値として戻せます。

697 ページの『%SUBDT (日付、時刻、またはタイム・スタンプの一部の取り出し)』を参照してください。

タイム・スタンプ・データの精度の向上
タイム・スタンプ・データは、小数点以下 0 桁から 12 桁の秒数で指定できます。 277 ページの『タイム・スタンプ・データ・タイプ』を参照してください。

オープン・アクセス・ファイル
オープン・アクセス・ファイルは、オペレーティング・システムではなく、ユーザー作成のプログラムまたはプロシージャーによってすべての操作が処理されるファイルです。 このプログラムまたはプロシージャーは、「オープン・アクセス・ハンドラー」、または単純に「ハンドラー」と呼ばれます。 ハンドラーは、HANDLER キーワードによって指定します。 175 ページの『オープン・アクセス・ファイル』を参照してください。

新規の XML-INTO オプション

- XML 名前空間は、「ns」および「nsprefix」オプションでサポートされます。 ns オプションおよび nsprefix オプションを参照してください。
- サブフィールド名について RPG によってサポートされていない文字を含む XML 名は、「case=convert」オプションでサポートされます。 case オプションを参照してください。

ターゲット文字セットに、あるソース文字が存在しない場合に、データ損失の原因となる CCSID 変換のサポート
制御仕様書キーワード CCSIDCVT(*EXCP : *LIST)。 327 ページの『CCSIDCVT(*EXCP | *LIST)』を参照してください。

- ターゲット文字セットに、あるソース文字が存在しないために、CCSID 変換によってデータが失われる場合は、CCSIDCVT(*EXCP) を使用して例外を取得します。
- CCSIDCVT(*LIST) を使用して、モジュール内のすべての CCSID 変換のリストと、変換によってデータが失われる可能性があることを示す診断メッセージを取得します。

日付、時刻、およびタイム・スタンプのデータの処理中に、RPG コンパイラーによる検証ステップのスキップを可能にする VALIDATE(*NODATETIME)

RPG コンパイラーが、検証チェックを実行せずに、日付、時刻、およびタイム・スタンプのデータを文字データとして扱うようにするには、制御仕様書キーワード VALIDATE(*NODATETIME) を使用します。 348 ページの『VALIDATE(*NODATETIME)』を参照してください。

これにより、一部の日付、時刻、およびタイム・スタンプの操作のパフォーマンスが改善される場合があります。

警告 : 検証ステップをスキップすると、深刻なデータ破損の問題を引き起こす可能性があります。 この機能は、日付、時刻、およびタイム・スタンプのデータが、常に有効であることが確実な場合にのみ使用するようにしてください。

表 *26. 7.2* で変更された言語要素: 制御仕様書キーワード	
要素	**記述**
CCSID キーワード	• CCSID(*EXACT) は、モジュール内のすべての英数字データの CCSID を認識するようコンパイラーに指示します。 – 英数字リテラルとグラフィック・リテラルは、ソース・ファイルの CCSID を持っています。 – 英数字データは常に CCSID を持っていると見なされます。 CCSID(*EXACT) を指定しないと、RPG コンパイラーは、データベース・ファイルのリテラル、変数、または入出力バッファー内のデータの CCSID に関して、誤った想定をする可能性があります。 324 ページの『CCSID(*EXACT)』を参照してください。 • CCSID(*CHAR:ccsid) では、*HEX、*JOBRUNMIX、*UTF8、ASCII CCSID、および EBCDIC CCSID がサポートされます。 • CCSID(*GRAPH:ccsid) では、*HEX、*JOBRUN がサポートされます。 • CCSID(*UCS2:ccsid) では、*UTF16 がサポートされます。
DFTACTGRP キーワード	自由形式の制御仕様書が存在し、ACTGRP、BNDDIR、または STGMDL キーワードの少なくとも 1 つが使用されている場合は、DFTACTGRP(*NO) が想定されます。 333 ページの『DFTACTGRP(*YES | *NO)』を参照してください。
OPENOPT キーワード	OPENOPT(*{NO}CVT) は、データベース・ファイルの DATA キーワードのデフォルトを制御します。 • OPENOPT(*CVTDATA) は、ファイルに DATA キーワードが指定されていない場合、DISK ファイルと SEQ ファイルに対して DATA(*CVT) が想定されることを示します。 • OPENOPT(*NOCVTDATA) は、ファイルに DATA キーワードが指定されていない場合、DISK ファイルと SEQ ファイルに対して DATA(*NOCVT) が想定されることを示します。 340 ページの『OPENOPT (* {NO }INZOFL * {NO }CVTDATA)』を参照してください。

表 *27. 7.2* で変更された言語要素: ディレクティブ	
要素	**記述**
/FREE ディレクティブと /END-FREE ディレクティブ	これらのディレクティブは、自由形式のコードの開始と終了を示すのに必要ではなくなりました。 これらの指示は、コンパイラーによって無視されます。 80 ページの『/FREE... /END-FREE』を参照してください。

表 *28. 7.2* で変更された言語要素: 定義仕様書キーワード	
要素	**記述**
CCSID キーワード	• 英数字データでサポートされます。 • 英数字サブフィールドの CCSID をサポートするために、外部記述データ構造でサポートされます。 • グラフィック・データの場合は *HEX および *JOBRUN をパラメーターに指定できます。 • UCS-2 データの場合は *UTF16 をパラメーターに指定できます。
DTAARA キーワード	自由形式の定義では、以下のとおりです。 • *VAR は使用されません。 引用符なしで名前を指定した場合、その名前は、変数または名前付き定数の名前であると想定されます。 • データ構造の場合、データ域データ構造であることを示すには、*AUTO を使用します。IN、OUT、および UNLOCK 操作を使用してデータ域を操作できるよう指定するには、*USRCTL を使用します。 425 ページの『DTAARA キーワード』を参照してください。
EXTFLD キーワード	自由形式のサブフィールド定義では、以下のとおりです。 • パラメーターはオプションです。 • パラメーターを引用符なしで指定した場合、それは以前に定義された名前付き定数の名前であると想定されます。 429 ページの『EXTFLD{(フィールド名) }』を参照してください。
EXTNAME キーワード	自由形式のデータ構造定義では、以下のとおりです。 • ファイル名または形式名のパラメーターを引用符なしで指定した場合、それは以前に定義された名前付き定数の名前であると想定されます。 431 ページの『EXTNAME(ファイル名{:形式名}{:*ALL| *INPUT|*OUTPUT|*KEY| *NULL})』を参照してください。
EXPORT および IMPORT キーワード	自由形式の定義では、以下のとおりです。 • *DCLCASE を、外部名に指定できます。これは、外部名が、独立フィールドまたはデータ構造が指定されたのと同じ方法で指定され、同じ大/小文字混合の文字を使用していることを示します。 428 ページの『EXPORT{(外部名)}』および 441 ページの『IMPORT{(外部名)}』を参照してください。
EXTPROC キーワード	自由形式のプロトタイプ定義またはプロシージャー・インターフェース定義では、以下のとおりです。 • *DCLCASE を、外部プロシージャーまたはメソッドの名前に使用できます。これは、外部名が、プロトタイプまたはプロシージャー・インターフェースが指定されたのと同じ方法で指定され、同じ大/小文字混合の文字を使用していることを示します。 • プロシージャー・インターフェース名を *N と指定すると、外部名は DCL-PROC ステートメントから取得されます。 433 ページの『EXTPROC{({*CL|*CWIDEN|*CNOWIDEN|*JAVA:クラス名:}名前| *DCLCASE)}』を参照してください。
LIKE キーワード	自由形式の定義では、LIKE キーワードには、長さの調節を指定する、オプションの 2 つ目のパラメーターがあります。 444 ページの『LIKE(名前 {: 長さ調整 })』を参照してください。

表 *28. 7.2* で変更された言語要素: 定義仕様書キーワード *(続き)*	
要素	**記述**
LEN キーワード	自由形式の定義では、LEN キーワードは、データ構造の定義でのみ許可されます。 その他の自由形式の定義では、長さは、データ・タイプ・キーワードの一部として指定されます。 443 ページの『LEN(length)』を参照してください。
CLASS、DATFMT、PROCPTR、TIMFMT、および VARYING キーワード	これらのキーワードは、自由形式の定義では使用されません。 これらのキーワードによって指定される情報は、関連するデータ・タイプ・キーワードの一部として指定されます。
FROMFILE、PACKEVEN、および TOFILE キーワード	これらのキーワードは、自由形式の定義では許可されません。
OVERLAY キーワード	パラメーターは、自由形式のサブフィールド定義のデータ構造の名前であってはなりません。 代わりに、POS キーワードが使用されます。 474 ページの『POS(開始位置)』を参照してください。

表 *29. 7.2* で変更された言語要素: リテラル	
要素	**記述**
タイム・スタンプ・リテラル	タイム・スタンプ・リテラルは、小数点以下 0 桁から 12 桁の秒数で指定できます。 203 ページの『リテラル』を参照してください。

表 *30. 7.2* で変更された言語要素: ステートメントの順序	
要素	**記述**
ファイル・ステートメントと定義ステートメント	ファイル・ステートメントと定義ステートメントは混用できます。 305 ページの『RPG IV の仕様タイプ』を参照してください。

表 *31. 7.2* で変更された言語要素: 組み込み関数	
要素	**記述**
%CHAR	オペランドがタイム・スタンプの場合、戻り値の長さは、タイム・スタンプのバイト数によって異なります。 形式が *ISO0 の場合、バイト数には 14 から 26 を指定できます。 形式が *ISO の場合、バイト数では 19 か、21 から 32 を指定できます。 621 ページの『%CHAR(日付|時刻|タイム・スタンプ {: 形式})』を参照してください。
%DEC	オペランドがタイム・スタンプの場合、桁数は、タイム・スタンプの秒数の小数点以下の桁数に応じて、14 から 26 を指定できます。 629 ページの『日付、時刻、またはタイム・スタンプ式』を参照してください。
%DIFF	オペランドがタイム・スタンプの場合、オプションの 4 つ目のパラメーターは、戻される秒数の小数点以下の桁数を指定します。 632 ページの『%DIFF (2 つの日付、時刻、またはタイム・スタンプ値の差)』を参照してください。

表 *31. 7.2* で変更された言語要素*:* 組み込み関数 *(*続き*)*	
要素	**記述**
%SECONDS	タイム・スタンプの秒数を追加するために %SECONDS を使用する場合、パラメーターには、秒数の小数点以下の桁数を示す小数点位を指定できます。 686 ページの『%SECONDS (秒数)』を参照してください。
%SUBDT	• オプションの 3 つ目のパラメーターは、結果の桁数を指定します。 • 第 1 オペランドがタイム・スタンプで、第 2 オペランド が *SECONDS の場合、任意指定の第 4 オペランドは、結果に含まれる秒の小数部の桁数を示します。 697 ページの『%SUBDT (日付、時刻、またはタイム・スタンプの一部の取り出し)』を参照してください。
%TIMESTAMP	• 最初のパラメーターに、タイム・スタンプを指定できます。 • 最初のパラメーターに、*SYS を指定できます。 • 最初のパラメーターが日付、タイム・スタンプ、または *SYS の場合、オプションの 2 つ目のパラメーターには、秒数の少数点以下の桁数を示す 0 から 12 の値を指定できます。 • 最初のパラメーターが文字または数値の場合、オプションの 3 つ目のパラメーターには、秒数の小数点以下の桁数を示す 0 から 12 の値を指定できます。 701 ページの『%TIMESTAMP (タイム・スタンプへの変換)』を参照してください。

表 *32. 7.2* で変更された言語要素*:* 固定形式の定義仕様書	
要素	**記述**
長さ入力	タイム・スタンプの長さの項目には、19 か、21 から 32 の値を指定できます。 409 ページの『33 から 39 桁目 (終了位置/長さ)』を参照してください。
少数点位の項目	タイム・スタンプの小数点位の項目には、0 から 12 の値を指定できます。 411 ページの『41 から 42 桁目 (小数点以下の桁数)』を参照してください。

表 *33. 7.2* の新しい言語要素*:* ディレクティブ	
要素	**記述**
/SET ディレクティブ	以下の制御ステートメント・キーワードに対して、新規の値を一時的に設定します。 • CCSID(*CHAR:ccsid) • CCSID(*GRAPH:ccsid) • CCSID(*UCS2:ccsid) • DATFMT(形式) • TIMFMT(形式) これらの値は、定義ステートメントのデフォルト値が明示的に指定されていない場合に、デフォルト値を指定するために使用します。 81 ページの『/SET』を参照してください。

表 33. 7.2 の新しい言語要素: ディレクティブ (続き)	
要素	**記述**
/RESTORE ディレクティブ	前の設定を、最近の /SET ディレクティブが値を設定する前の値に復元します。 • CCSID(*CHAR) • CCSID(*GRAPH) • CCSID(*UCS2) • DATFMT • TIMFMT 83 ページの『/RESTORE』を参照してください。

表 34. 7.2 の新しい言語要素: 自由形式のステートメント	
要素	**記述**
CTL-OPT	自由形式の制御ステートメントを開始します。 319 ページの『自由形式の制御ステートメント』を参照してください。
DCL-F	自由形式のファイル定義を開始します。 350 ページの『自由形式のファイル定義ステートメント』を参照してください。
DCL-C	自由形式の名前付き定数定義を開始します。 395 ページの『自由形式の名前付き定数の定義』を参照してください。
DCL-DS	自由形式のデータ構造定義を開始します。 396 ページの『自由形式のデータ構造定義』を参照してください。
DCL-SUBF	自由形式のサブフィールド定義を開始します。 サブフィールド名が自由形式の演算で許可される命令コードと同じでない限り、「DCL-SUBF」の指定はオプションです。 399 ページの『自由形式のサブフィールド定義』を参照してください。
END-DS	自由形式のデータ構造定義を終了します。 サブフィールドがない場合は、これは DCL-DS ステートメントの最後のキーワードの後に指定できます。 396 ページの『自由形式のデータ構造定義』を参照してください。
DCL-PI	自由形式のプロシージャー・インターフェース定義を開始します。 403 ページの『自由形式プロシージャー・インターフェース定義』を参照してください。
DCL-PR	自由形式のプロトタイプ定義を開始します。 401 ページの『自由形式のプロトタイプ定義』を参照してください。
DCL-PARM	自由形式のパラメーター定義を開始します。 パラメーター名が自由形式の演算で許可される命令コードと同じでない限り、「DCL-PARM」の指定はオプションです。 405 ページの『自由形式のパラメーター定義』を参照してください。 .
END-PI	自由形式のプロシージャー・インターフェース定義を終了します。 パラメーターがない場合は、これは DCL-PI ステートメントの最後のキーワードの後に指定できます。 403 ページの『自由形式プロシージャー・インターフェース定義』を参照してください。

表 34. 7.2 の新しい言語要素: 自由形式のステートメント (続き)	
要素	**記述**
END-PR	自由形式のプロトタイプ定義を終了します。 パラメーターがない場合は、これは DCL-PR ステートメントの最後のキーワードの後に指定できます。 401 ページの『自由形式のプロトタイプ定義』を参照してください。
DCL-S	自由形式の独立フィールド定義を開始します。 395 ページの『自由形式の独立フィールド定義』を参照してください。
DCL-PROC	自由形式のプロシージャー定義を開始します。 532 ページの『自由形式のプロシージャー・ステートメント』を参照してください。
END-PROC	自由形式のプロシージャー定義を終了します。 532 ページの『自由形式のプロシージャー・ステートメント』を参照してください。

表 35. 7.2 の新しい言語要素: 制御仕様キーワード	
要素	**記述**
CCSIDCVT(*EXCP | *LIST)	異なる CCSID を持つデータ間でコンパイラーが変換を処理する方法を制御できます。 327 ページの『CCSIDCVT(*EXCP | *LIST)』を参照してください。
VALIDATE (*NODATETIME)	日付データ、時刻データ、およびタイム・スタンプ・データを、使用前に検証する必要があるかどうかを指定します。 348 ページの『VALIDATE(*NODATETIME)』を参照してください。

表 36. 7.2 の新しい言語要素: ファイル定義キーワード	
要素	**記述**
DATA(*{NO}CVT)	英数字フィールドとグラフィックス・フィールドからジョブ CCSID への (またはその逆の) CCSID 変換をデータベースが実行するように、ファイルをオープンするかどうかを制御します。 366 ページの『DATA(*CVT | *NOCVT)』を参照してください。
HANDLER(ハンドラー {:通信域})	ファイルがオープン・アクセス・ファイルであることを指定します。 372 ページの『HANDLER(プログラムまたはプロシージャー { : 通信域})』を参照してください。
DISK{(*EXT | レコード長)}	自由形式のファイル定義の装置タイプを指定する装置キーワード。 • デフォルトの装置タイプは DISK です。 • 各装置タイプ・キーワードのデフォルトのパラメーターは、外部記述ファイルであることを示す *EXT です。 174 ページの『ファイル装置』を参照してください。
PRINTER{(*EXT | レコード長)}	
SEQ{(*EXT | レコード長)}	
SPECIAL{(*EXT | レコード長)}	
WORKSTN{(*EXT | レコード長)}	

表 *36. 7.2* の新しい言語要素*:* ファイル定義キーワード *(*続き*)*	
要素	**記述**
USAGE(*INPUT *OUTPUT *UPDATE *DELETE)	自由形式のファイル定義でのファイルの使用方法を指定します。 389 ページの『USAGE(*INPUT *OUTPUT *UPDATE *DELETE)』を参照してください。
KEYED{(*CHAR : キーの長さ)}	自由形式のファイル定義にファイルがキー入力されることを示します。 375 ページの『KEYED{(*CHAR : キー長)}』を参照してください。

表 *37. 7.2* の新しい言語要素*:* 自由形式のデータ・タイプ・キーワード	
要素	**記述**
CHAR(長さ)	固定長の英数字データ・タイプ。 419 ページの『CHAR(長さ)』を参照してください。
VARCHAR(長さ {: 接頭部のサイズ})	可変長の英数字データ・タイプ。 486 ページの『VARCHAR(長さ {:2 | 4})』を参照してください。
GRAPH(長さ)	固定長のグラフィック・データ・タイプ。 440 ページの『GRAPH(長さ)』を参照してください。
VARGRAPH(長さ {:接頭部のサイズ})	可変長のグラフィック・データ・タイプ。 487 ページの『VARGRAPH(長さ {:2 | 4})』を参照してください。
UCS2(*length*)	固定長の UCS-2 データ・タイプ。 485 ページの『UCS2(length)』を参照してください。
VARUCS2(*length* {:*prefix-size*})	可変長の UCS-2 データ・タイプ。 487 ページの『VARUCS2(length {:2 | 4})』を参照してください。
IND	標識データ・タイプ。 442 ページの『IND』を参照してください。
INT(桁数)	整数データ・タイプ。 441 ページの『INT(桁数)』を参照してください。
UNS(桁数)	符号なし整数データ・タイプ。 486 ページの『UNS(桁数)』を参照してください。
PACKED(桁数 {:*10* 進数})	パック 10 進数データ・タイプ。 473 ページの『PACKED(桁数 {: 小数点以下の桁数})』を参照してください。
ZONED(桁数 {:*10* 進数})	ゾーン 10 進数データ・タイプ。 488 ページの『ZONED(桁数 {: 小数点以下の桁数})』を参照してください。
BINDEC(桁数 {:*10* 進数})	2 進化 10 進数データ・タイプ。 417 ページの『BINDEC(桁数 {: 小数点以下の桁数})』を参照してください。

表 37. 7.2 の新しい言語要素: 自由形式のデータ・タイプ・キーワード (続き)	
要素	**記述**
FLOAT(サイズ)	浮動データ・タイプ。 440 ページの『FLOAT(バイト数)』を参照してください。
DATE{(形式)}	日付データ・タイプ。 420 ページの『DATE{(形式 {区切り記号 }) }』を参照してください。
TIME{(形式)}	時刻データ・タイプ。 484 ページの『TIME{(形式 {区切り記号 }) }』を参照してください。
TIMESTAMP {(秒数の小数点以下の桁数)}	タイム・スタンプ・データ・タイプ。 484 ページの『TIMESTAMP { (秒の小数部の桁数) }』を参照してください。
POINTER{(*PROC)}	ポインター・データ・タイプ。 オプション・パラメーター *PROC は、これがプロシージャー・ポインターであることを示します。 473 ページの『POINTER{(*PROC)}』を参照してください。
OBJECT{(*JAVA : クラス名)}	オブジェクト・データ・タイプ。 これが Java コンストラクターの戻りタイプを定義している場合、パラメーターはオプションです。 453 ページの『OBJECT{(*JAVA:クラス名)}』を参照してください。

表 38. 7.2 の新しい言語要素: 自由形式のデータ定義キーワード	
要素	**記述**
EXT	データ構造が外部記述であることを示します。 データ構造定義の最初のキーワードに EXTNAME キーワードを指定した場合、このキーワードはオプションです。 429 ページの『EXT』を参照してください。
POS(サブフィールドの開始位置)	データ構造内のサブフィールドの開始位置を指定します。 474 ページの『POS(開始位置)』を参照してください。
PSDS	データ構造がプログラム状況データ構造であることを指定します。 475 ページの『PSDS』を参照してください。

7.1 での変更点

このセクションでは、7.1 において ILE RPG に対して行われた機能強化について説明します。

データ構造配列のソートおよび検索

サブフィールドのいずれかをキーとして使用して、データ構造配列をソートおよび検索することができます。

```
// Sort the custDs array by the amount_owing subfield
SORTA custDs(*).amount_owing;

// Search for an element in the custDs array where the
// account_status subfield is "K"
elem = %LOOKUP("K" : custDs(*).account_status);
```

昇順または降順での配列のソート

配列は、SORTA(A) を使用すると昇順にソートでき、SORTA(D) を使用すると降順にソートできます。配列は、順序付けされた配列であってはなりません (ASCEND または DESCEND キーワード)。

```
// Sort the salary array in descending order
SORTA(D) salary;
```

新規組み込み関数 %SCANRPL (スキャンおよび置換)

%SCANRPL 組み込み関数は、ストリング内をスキャンして、ある値の出現箇所をすべて検出し、それらを別の値に置き換えます。

```
// Replace NAME with 'Tom'
string1 = 'See NAME. See NAME run. Run NAME run.';
string2 = %ScanRpl('NAME' : 'Tom' : string1);
// string2 = 'See Tom. See Tom run. Run Tom run.'
```

%LEN(varying : *MAX)

%LEN 組み込み関数を使用して、可変長の文字フィールド、UCS-2 フィールド、または図形フィールドの最大文字数を取得できます。

外部記述データ構造内の ALIAS 名の使用

外部記述データ構造のサブフィールドに代替名を使用することを指示するには、定義仕様書で ALIAS キーワードを使用します。 ファイルのレコードから定義された LIKEREC データ構造に代替名を使用することを指示するには、ファイル仕様書で ALIAS キーワードを使用します。

```
A          R CUSTREC
A            CUSTNM        25A         ALIAS(CUSTOMER_NAME)
A            CUSTAD        25A         ALIAS(CUSTOMER_ADDRESS)
A            ID            10P 0

D custDs        e ds                   ALIAS
D                                      QUALIFIED EXTNAME(custFile)
 /free
   custDs.customer_name = 'John Smith';
   custDs.customer_address = '123 Mockingbird Lane';
   custDs.id = 12345;
```

戻り値を取得する時間の短縮

RTNPARM キーワードで定義されたプロシージャーは、戻り値を非表示パラメーターとして処理します。 プロシージャーが、非常に大きい値 (特に非常に大きな変化する値) を返すようにプロトタイプ化されている場合、RTNPARM キーワードを使用してプロシージャーを定義することによって、プロシージャー呼び出しのパフォーマンスを大幅に向上させることができます。

```
D getFileData     pr              a   varying len(1000000)
D                                     rtnparm
D   file                          a   const varying len(500)
D data            S               a   varying len(1000)
 /free
    data = getFileData ('/home/mydir/myfile.txt');
```

%PARMNUM 組み込み関数

%PARMNUM(parameter_name) 組み込み関数は、パラメーター・リスト内のパラメーターの序数を返します。 特に、プロシージャーが RTNPARM キーワードを使用してコーディングされている場合は、この組み込み関数を使用することが重要です。

```
D                 pi
D   name                       100a   const varying
D   id                          10i 0 value
D   errorInfo                        likeds(errs_t)
D                                     options(*nopass)
 /free
   // Check if the "errorInfo" parameter was passed
   if %parms >= %parmnum(errorInfo);
```

オプションのプロトタイプ

プログラムまたはプロシージャーが別の RPG モジュールから呼び出されない場合、プロトタイプの指定はオプションです。 プロトタイプは、以下のタイプのプログラムおよびプロシージャーでは省略される場合があります。

- 出口プログラムとして、またはコマンドのコマンド処理プログラムとしてのみ使用されることが意図されたプログラム
- 異なるプログラミング言語からのみ呼び出されることが意図されたプログラム
- モジュールからエクスポートされないプロシージャー
- モジュールからエクスポートされるが、異なるプログラミング言語からのみ呼び出されることが意図されたプロシージャー

任意のタイプのストリング・パラメーターの受け渡し

値によって、または読み取り専用参照によって渡されたストリング・パラメーターに対して、暗黙的な変換が行われます。 例えば、CONST UCS-2 パラメーターを含むようにプロシージャーをプロトタイプ化したり、プロシージャーへの呼び出しで文字式をパラメーターとして渡したりすることができます。 これにより、UCS-2 タイプでプロトタイプ化されたパラメーターおよび戻り値のある、単一のプロシージャーを作成できます。 そのプロシージャーを呼び出すために、任意のタイプのストリング・パラメーターを渡し、戻り値を任意のタイプのストリング変数に割り当てることができます。

```
// The makeTitle procedure upper-cases the value
// and centers it within the provided length
alphaTitle = makeTitle(alphaValue : 50);
ucs2Title = makeTitle(ucs2Value : 50);
dbcsTitle = makeTitle(dbcsValue : 50);
```

XML-INTO の 2 つの新しいオプション

- *datasubf* オプションを使用すると、属性も持つ XML エレメント用のテキスト・データを受け取るサブフィールドを指定できます。
- *countprefix* オプションを使用すると、*allowmissing=yes* オプションを指定する必要性が少なくなります。 XML-INTO 命令によって設定された RPG 配列エレメントまたは非配列サブフィールドの数を受け取る追加サブフィールドの名前の接頭部を指定します。

これらのオプションは、6.1 の PTF でも使用可能です。

テラスペース・ストレージ・モデル

テラスペース・ストレージ・モデルを使用するように、または呼び出し元のストレージ・モデルを継承するように、RPG モジュールおよびプログラムを作成できます。 テラスペース・ストレージ・モデルを使用した場合、自動ストレージに関するシステムしきい値は、単一レベル・ストレージ・モデルのシステムしきい値よりも大幅に高くなります。 単一のプロシージャーの自動ストレージの量の制限と、呼び出しスタックにあるすべてのプロシージャーの自動ストレージの合計の制限があります。

CRTRPGMOD または CRTBNDRPG コマンドでストレージ・モデル (STGMDL) パラメーターを使用するか、制御仕様書で STGMDL キーワードを使用します。

***TERASPACE**
プログラムまたはモジュールはテラスペース・ストレージ・モデルを使用します。

***SNGLVL**
プログラムまたはモジュールは単一レベル・ストレージ・モデルを使用します。

***INHERIT**
プログラムまたはモジュールは、呼び出し元のストレージ・モデルを継承します。

CRTBNDRPG コマンドの ACTGRP パラメーターおよび制御仕様書の ACTGRP キーワードへの変更

ACTGRP パラメーターおよびキーワードのデフォルト値が、QILE から *STGMDL に変更されています。

ACTGRP(*STGMDL) は、活動化グループがプログラムのストレージ・モデルに依存することを指定します。 ストレージ・モデルが *TERASPACE である場合、ACTGRP(*STGMDL) は ACTGRP(QILETS) と同じです。 それ以外の場合は、ACTGRP(*STGMDL) は ACTGRP(QILE) と同じです。

注 : ACTGRP パラメーターおよびキーワードへの変更は、活動化グループがプログラムに割り当てられるデフォルトの方法には影響しません。 STGMDL パラメーターおよびキーワードのデフォルト値は *SNGLVL であるため、ACTGRP パラメーターまたはキーワードが指定されない場合、プログラムの活動化グループは、以前のリリースと同様にデフォルトの QILE になります。

テラスペース・ストレージの割り振り

制御仕様書で ALLOC キーワードを使用して、モジュール内の RPG ストレージ管理命令でテラスペース・ストレージを使用するか単一レベル・ストレージを使用するかを指定します。 テラスペース・ストレージ割り振りの最大サイズは、単一レベル・ストレージ割り振りの最大サイズよりも大幅に大きくなっています。

リスト・デバッグ・ビューの暗号化

モジュールのリスト・デバッグ・ビューが暗号化されると、リスト・ビューは、デバッグを実行するユーザーが暗号鍵を知っている場合にデバッグ・セッション中のみ表示できます。 これにより、顧客がリスト・ビューを通じてソース・コードを見ることができない状態で、デバッグ可能なプログラムを顧客に送信することが可能です。 CRTRPGMOD、CRTBNDRPG、または CRTSQLRPGI コマンドで DBGENCKEY パラメーターを使用してください。

表 *39. 6.1* 以降に変更された言語要素

言語単位	要素	説明
制御仕様書キーワード	ACTGRP(*STGMDL)	*STGMDL は、ACTGRP キーワードおよびコマンド・パラメーターの新しいデフォルト値です。 プログラムがテラスペース・ストレージ・モジュールを使用する場合、活動化グループは QILETS です。 それ以外の場合、活動化グループは QILE です。
組み込み関数	%LEN(varying-field : *MAX)	現在は、可変長フィールドの最大文字数を取得するために使用できます。
命令コード	SORTA(A | D)	SORTA 命令コードでは、配列を昇順 (A) でソートするか降順 (D) でソートするかを示す A および D 命令拡張を指定できるようになりました。

表 *40. 6.1* 以降の新しい言語要素

言語単位	要素	説明
制御仕様書キーワード	STGMDL(*INHERIT | *TERASPACE | *SNGLVL)	モジュールまたはプログラムのストレージ・モデルを制御します。
	ALLOC(*STGMDL | *TERASPACE | *SNGLVL)	ストレージ管理命令 %ALLOC、%REALLOC、DEALLOC、ALLOC、REALLOC のストレージ・モデルを制御します。
ファイル仕様書のキーワード	ALIAS	LIKEREC キーワードによって定義されたデータ構造のサブフィールドに、代替フィールド名を使用します。

表 40. 6.1 以降の新しい言語要素 (続き)		
言語単位	**要素**	**説明**
定義仕様書キーワード	ALIAS	外部記述データ構造のサブフィールドに、代替フィールド名を使用します。
	RTNPARM	プロシージャーの戻り値が非表示パラメーターとして処理されることを指定します。
組み込み関数	%PARMNUM	パラメーター・リスト内のパラメーターの序数を返します。
	%SCANRPL	ストリング内をスキャンして、ある値の出現箇所をすべて検出し、それらを別の値に置き換えます。
XML-INTO オプション	datasubf	属性も持つ XML エレメント用のテキスト・データを受け取るサブフィールドを指定します。
	countprefix	XML-INTO 命令によって設定された RPG 配列エレメントまたは非配列サブフィールドの数を受け取る追加サブフィールドの名前の接頭部を指定します。

6.1 での変更点

このセクションでは、6.1 において ILE RPG に対して行われた機能強化について説明します。

THREAD(*CONCURRENT)

THREAD(*CONCURRENT) がモジュールの制御仕様書で指定されている場合、次のように複数のスレッドで同時に実行する機能が提供されます。

- 複数のスレッドが同時にモジュールで実行できる。
- デフォルトでは、各スレッドが静的変数の独自のコピーを持つように 静的変数が定義される。
- 個々の変数は、STATIC(*ALLTHREAD) を使用してすべてのスレッドで共用されるように定義できる。
- プロシージャー開始仕様書で SERIALIZE を指定することによって、 1 つのスレッドのみが個々のプロシージャーを一度に実行できるように、 それらを直列化することができる。

RPG サイクルを使用しないメイン・プロシージャーを定義する機能

サブプロシージャーは、制御仕様書で MAIN キーワードを使用して、 プログラム入力プロシージャーとして識別することができます。 これにより、RPG アプリケーションは、RPG サイクルを使用するモジュールがない場所に 開発できます。

サブプロシージャーで定義されるファイル

ファイルは、サブプロシージャーでローカルに定義できます。 ローカル・ファイルへの 入出力は、データ構造を使用してのみ実行できます。I 仕様書および O 仕様書はサブプロシージャーで 許可されず、コンパイラーは外部記述ファイルの I 仕様書および O 仕様書を生成しません。 デフォルトでは、ローカル・ファイルは自動記憶域に関連付けられます。 サブプロシージャーが戻るときに、ファイルがクローズされます。 STATIC キーワードを使用して、ファイルに関連付けられる記憶域が静的であることを 示すことができるため、サブプロシージャーのすべての呼び出しは同じファイルを使用します。 また、サブプロシージャーが戻ったときにファイルが開いている場合は、 そのサブプロシージャーへの次の呼び出しのためにファイルは開いたままになります。

修飾レコード様式

ファイルが QUALIFIED キーワードで定義される場合、 レコード様式は MYFILE.MYFMT というファイル名で修飾する必要があります。 修飾ファイルには コンパイラーによって生成される I 仕様書および O 仕様書はありません。入出力は データ構造を介してのみ行われます。

他のファイルのように定義されるファイル

ファイルは、LIKEFILE キーワードを使用して、他のファイル仕様書と同じ設定を使用するように 定義することができます。これは、ファイルをパラメーターとして渡すときに 重要です。 ファイルが外部記述される場合、QUALIFIED キーワードは暗黙となります。 新規ファイルへの入出力は、データ構造を介してのみ実行できます。

パラメーターとして渡されるファイル

プロトタイプ・パラメーターは、 LIKEFILE キーワードを使用して、ファイル・パラメーターとして定義できます。 同じ LIKEFILE 定義を介して関連付けられたファイルは、パラメーターとしてプロシージャーに渡される場合があります。 呼び出し先のプロシージャーまたはプログラム内では、サポートされているすべての操作をファイルに対して実行することができます。 ただし、入出力を実行するには、データ構造を使用する必要があります。

EXTDESC キーワードと EXTFILE(*EXTDESC)

EXTDESC キーワードは、コンパイル時にコンパイラーによって使用されるファイルを 識別して、ファイルの外部記述を取得します。 ファイル名は、'LIBNAME/FILENAME' または 'FILENAME' のいずれかの様式でリテラルとして指定されます。 これにより、ファイルのコンパイル時指定変更が必要 なくなります。

EXTFILE キーワードは、 特殊値 *EXTDESC を許可するように拡張され、EXTDESC によって指定されるファイルが ランタイム時にも使用されることを示します。

外部記述データ構造のライブラリーを指定するための EXTNAME

EXTNAME キーワードは、外部ファイルのライブラリーを指定するために、 リテラルを許可するように拡張されています。 EXTNAME('LIBNAME/FILENAME') または EXTNAME('FILENAME') がサポートされています。 これにより、ファイルのコンパイル時指定変更が必要 なくなります。

EXFMT は結果データ構造を許可する

データ構造が結果フィールドで指定されるように、EXFMT 命令が拡張されています。 データ構造を定義するには、使用タイプ *ALL を指定する必要があります。 このタイプは、レコード様式の外部記述データ構造として指定 (EXTNAME(file:fmt:*ALL) するか、またはレコード様式の LIKEREC を使用して指定 (LIKEREC(fmt:*ALL) します。

データ構造、文字、UCS-2 およびグラフィック変数の制限の拡大

- データ構造のサイズは最大 16,773,104 とすることができる。
- 文字定義の長さは最大 16,773,104 とすることができる。 (この制限は 可変長文字定義では 4 少ない。)
- UCS-2 定義の長さは最大 8,386,552 UCS-2 文字とすることができる。 (この制限は可変長 UCS-2 定義では 2 少ない。)
- グラフィック定義の長さは最大 8,386,552 DBCS 文字 とすることができる。 (この制限は可変長グラフィック定義では 2 文字 少ない。)
- VARYING キーワードは、長さの接頭部を保持するために使用されるバイト数を 示す 2 または 4 のいずれかのパラメーターを許可します。

%ADDR(varying : *DATA)

%ADDR 組み込み関数は、*DATA を 2 番目のパラメーターとして許可し、可変長フィールドのデータ部分のアドレスを取得できるように拡張されています。

DIM および OCCURS の制限の拡大

配列データ構造または複数オカレンス・データ構造には、最大 16,773,104 個までの要素を含めることができます。 ただし、合計サイズが 16,773,104 を超えないようにする必要があります。

リテラル (文字、UCS-2、および DBCS) の上限値の緩和

- 文字リテラルの最大長が 16380 文字になりました。
- UCS-2 リテラルの最大長が 8190 文字 (UCS-2) になりました。
- 図形リテラルの最大長が 16379 文字 (DBCS) になりました。

ファイルおよび定義の TEMPLATE キーワード

ファイルおよび変数定義に対して TEMPLATE キーワードをコーディングして、 その名前を LIKEFILE、LIKE、または LIKEDS キーワードのみと ともに使用して他のファイルまたは変数を定義することができます。 テンプレート定義は、プロトタイプ呼び出しのタイプを定義する 際に役立ちます。コンパイラーがコンパイル時にそれらのタイプのみを使用して、 他のファイルおよび変数の定義し、それらのタイプに関連するコードを生成しないためです。

テンプレート・データ構造では、データ構造とそのサブフィールドに対して INZ キーワードをコーディングすることができます。 これによって、簡単に INZ(*LIKEDS) を使用できるようになります。

一部の UCS-2 規則の緩和

コンパイラーは文字、 UCS-2 およびグラフィック値の間の暗黙的な変換を実行し、 多くの場合、%CHAR、%UCS2、または %GRAPH をコーディングする必要は無くなります。 この拡張は、 V5R3 および V5R4 の PTF でも使用可能です。 暗黙的な変換の現在のサポート対象は以下のとおりです。

- EVAL および EVALR による代入
- 式の中の比較演算。
- 固定形式演算 IFxx、DOUxx、DOWxx、WHxx、CASxx、 CABxx、COMP を使用した比較。
- 変換命令 MOVE および MOVEL に関しては、暗黙的な変換がすでにサポートされています。

UCS-2 変数は現在、%UCS2 組み込み機能を使用しないで、文字または図形リテラルを使用して初期化することができます。

コンパイル済みオブジェクトからの未使用の変数の除去

新規値 *UNREF および *NOUNREF が CRTBNDRPG コマンドと CRTRPGMOD コマンドの OPTION キーワード、および制御仕様書の OPTION キーワードに追加されています。 デフォルトは *UNREF です。*NOUNREF は、参照されない変数は RPG モジュールに生成してはならないことを示します。 これにより、プログラム・サイズは削減でき、 インポートされた変数が参照されない場合、モジュールをプログラムまたはサービス・プログラムに バインドするためにかかる時間を削減できます。

モジュールへの PCML の保存

PCML (Program Call Markup Language) は現在、モジュール内と ストリーム・ファイル内に保管できます。 PGMINFO コマンド・パラメーターまたは 制御仕様書の新規 PGMINFO キーワード、またはその両方の組み合わせを使用することにより、 RPG プログラマーは PCML 情報を置く場所を選択できます。PCML 情報がモジュール内に置かれる場合、 後で QBNRPII API を使用して取り出すことができます。この拡張は V5R4 の PTF でも使用可能ですが、 制御仕様書キーワードを介してのみ可能です。

表 *41. V5R4* 以降に変更された言語要素

言語単位	要素	説明
制御仕様書キーワード	OPTION(*UNREF | *NOUNREF)	未使用の変数はモジュールに生成してはならないことを指定します。
	THREAD(*CONCURRENT)	新規パラメーター *CONCURRENT では、複数のスレッドで 同時に実行可能です。
ファイル仕様書のキーワード	EXTFILE(*EXTDESC)	EXTDESC キーワードの値は EXTFILE キーワードにも使用されることを示します。

表 *41. V5R4* 以降に変更された言語要素 *(*続き*)*

言語単位	要素	説明
組み込み関数	%ADDR(varying-field : *DATA)	現在は、可変長変数のデータ部分のアドレスを取得するために使用できます。
定義仕様書キーワード	DIM(16773104)	配列は最大 16773104 要素を持つことができます。
	EXTNAME('LIB/FILE')	ファイル名のリテラルを許可します。 リテラルにはファイルのライブラリーを 含むことができます。
	OCCURS(16773104)	複数回繰り返しデータ構造は最大 16773104 要素を持つことができます。
	VARYING{(2|4)}	長さ接頭部のバイト数を示すパラメーターを 取ることが可能になりました。
定義仕様書	長さ入力	データ構造、およびタイプ A、C、または G の定義に対して最大 9999999 と することができます。 (より長い項目を定義するには、LEN キーワードを 使用する必要があります。)
入力仕様書	長さ入力	英数字フィールドの場合は最大 99999 で、UCS-2 および図形フィールドの場合は 99998 とすることができます。
演算仕様書	長さ入力	英数字フィールドの場合は最大 99999 とすることができます。
命令コード	EXFMT フォーマット { result-ds }	結果入力にデータ構造を持つことができます。

表 *42. V5R4* 以降の新しい言語要素

言語単位	要素	説明
制御仕様書キーワード	MAIN(subprocedure-name)	プログラムのプログラム・エントリー・プロシージャーを指定します。
	PGMINFO(*NO | *PCML { : *MODULE })	プログラム情報をモジュールに直接置くかどうかを示します。

表 42. V5R4 以降の新しい言語要素 (続き)		
言語単位	**要素**	**説明**
ファイル仕様書のキーワード	STATIC	サブプロシージャーへの呼び出し間で、ローカル・ファイルがプログラム状態を 保存することを示します。
	QUALIFIED	ファイルのレコード様式名はファイル名 FILE.FMT によって修飾されていることを示します。
	LIKEFILE(ファイル名)	ファイルは他のファイルと同じように定義されることを示します。
	TEMPLATE	ファイルは後の LIKEFILE 定義でのみ使用されることを示します。
	EXTDESC(constant-filename)	外部定義用にコンパイル時に使用される外部ファイルを 指定します。
定義仕様書キーワード	STATIC(*ALLTHREAD)	静的変数の同じインスタンスが、モジュールで実行中の すべてのスレッドによって使用されることを示します。
	LIKEFILE(ファイル名)	パラメーターがファイルであることを示します。
	TEMPLATE	定義は LIKE 定義または LIKEDS 定義に対してのみ使用されることを示します。
	LEN(length)	データ構造、または定義 A、C、または G の長さを指定します。
プロシージャー仕様書キーワード	SERIALIZE	プロシージャーが一度に 1 つのスレッドのみを実行できることを示します。

V5R4 での変更点

以下に、V5R4 の ILE RPG における機能強化について説明します。

新規の命令コード EVAL-CORR

```
EVAL-CORR{(EH)} ds1 = ds2
```

新規命令コード EVAL-CORR は、ソース・データ構造のサブフィールドからのデータおよび NULL 標識をターゲット・データ構造のサブフィールドに割り当てます。 代入されるサブフィールドは、両方のデータ構造に同じ名前および互換データ・タイプがあるサブフィールドです。

例えば、データ構造 DS1 に文字サブフィールド A、B、および C があり、データ構造 DS2 に文字サブフィールド B、C、および D がある場合、ステートメント EVAL-CORR DS1 = DS2; は、サブフィールド DS2.B および DS2.C から DS1.B および DS1.C にデータを代入します。 EVAL-CORR 命令が反映されたターゲットのデータ構造内のヌル対応サブフィールドでは、ソースのデータ構造のサブフィールドのヌル標識からヌル標識が設定されるか、またはソースのサブフィールドがヌルに非対応である場合は *OFF に設定されます。

```
    //  DS1 subfields         DS2 subfields
    //    s1  character         s1 packed
    //    s2  character         s2 character
```

```
//    s3  numeric
//    s4  date              s4 date
//                          s5 character
EVAL-CORR  ds1 = ds2;
// This EVAL-CORR operation is equivalent to the following EVAL operations
//    EVAL  ds1.s2 = ds2.s2
//    EVAL  ds1.s4 = ds2.s4
// Other subfields either appear in only one data structure (S3 and S5)
// or have incompatible types (S1).
```

EVAL-CORR は、外部記述ファイルおよびレコード・フォーマットへの入出力操作の結果のデータ構造を簡単に使用できるようにし、レコード・フォーマットのレイアウトが違うか、サブフィールドのタイプが少し違う場合に、異なるレコード・フォーマットのデータ構造間でのデータの自動転送を可能にします。

新規プロトタイプ・パラメーター・オプション OPTIONS(*NULLIND)

OPTIONS(*NULLIND) がパラメーターに指定された場合、そのパラメーターと共にヌル・バイト・マップが渡され、着呼側プロシージャーが発呼者のパラメーターのヌル・バイト・マップに直接アクセスできるようになります。

新規組み込み関数 %XML

```
%XML (xmldocument { : options } )
```

%XML 組み込み関数は、XML 文書を記述し、文書が構文解析される方法を制御するオプションを指定します。 ***xmldocument*** パラメーターは、文字または UCS-2 式であってもよく、その値は、XML 文書または XML 文書を含む IFS ファイルの名前でもかまいません。 ***xmldocument*** パラメーターの値がファイルの名前を持つ場合、"doc=file" オプションを指定する必要があります。

新規組み込み関数 %HANDLER

```
%HANDLER (handlingProcedure : communicationArea  )
```

%HANDLER は、1 つのイベントまたは一連のイベントを処理するプロシージャーを識別するために使用します。%HANDLER は値を戻さず、XML-SAX、XML-INTO および DATA-INTO の第 1 オペランドとしてのみ指定できます。

第 1 オペランド *handlingProcedure* は、処理プロシージャーのプロトタイプを指定します。 プロトタイプで指定される戻り値およびパラメーターは処理手順に必要なパラメーターに合致する必要があります。要件は、%HANDLER が指定された操作によって決定されます。

第 2 オペランド *communicationArea* は、処理プロシージャーのすべての呼び出しでパラメーターとして渡される変数を指定します。 オペランドは、参照によって渡されるプロトタイプ・パラメーターの検査で使用する規則と同じ規則に従って、 処理プロシージャーの最初のプロトタイプ・パラメーターに完全に一致してい る必要があります。 通信域パラメーターには、配列やデータ構造など、任意のタイプを使用できます。

新規命令コード XML-SAX

```
XML-SAX{ (e) } %HANDLER(eventHandler : commArea ) %XML(xmldoc { : options } );
```

XML-SAX は、%XML 組み込み関数で指定された XML 文書用の SAX 構文解析を開始します。 文書の構文解析を開始する XML パーサーを呼び出すことによって、XML-SAX 命令は開始します。 要素の開始の検出、属性名の検出、および要素の終了の検出などのイベントがパーサーにより発見された場合、パーサーはそのイベントを記述したパラメーターで ***eventHandler*** を呼び出します。 *commArea* オペランドは、パラメーターとして ***eventHandler*** に渡される変数であり、XML-SAX 命令コードが処理手順と通信する方法を提供します。 ***eventHandler*** が戻るときに、パーサーは、次のイベントを見つけて ***eventHandler*** を再度呼び出すまで構文解析を継続します。

新規命令コード XML-INTO

```
XML-INTO{ (EH) } variable   %XML(xmlDoc { : options });
XML-INTO{ (EH) } %HANDLER(handler : commArea ) %XML(xmlDoc { : options });
```

XML-INTO は、以下の 2 つの方法のいずれかで XML 文書からデータを読み込みます。

- 変数に直接。
- %HANDLER で指定されるプロシージャーに渡す配列パラメーターに徐々に。

命令を制御するためにさまざまなオプションを指定することができます。

第 1 オペランドには、構文解析されるデータのターゲットを指定します。 変数名または % HANDLER 組み込み関数を含むことができます。

第 2 オペランドは、XML 文書のソースおよび文書を構文解析する方法を制御する任意のオプションを指定する %XML 組み込み関数を含みます。 XML データを含むか、XML データのロケーションを含むことができます。 このオペランドが何を指定するかを示すために、doc オプションを使用します。

```
// Data structure "copyInfo" has two subfields, "from"
// and "to".  Each of these subfields has two subfields
// "name" and "lib".
// File cpyA.xml contains the following XML document
// <copyinfo>
//    <from><name>MASTFILE</name><lib>CUSTLIB</lib></from>
//    <to><name>MYFILE</name><lib>*LIBL</lib>
// <copyinfo>
xml-into copyInfo %XML('cpyA.xml' : 'doc=file');
// After the XML-INTO operation, the following
// copyInfo.from  .name = 'MASTFILE  ' .lib = 'CUSTLIB   '
// copyInfo.to    .name = 'MYFILE    ' .lib = '*LIBL     '
```

フィールド名の先頭から文字を除去するための PREFIX キーワードの使用

```
PREFIX('' : number_of_characters)
```

空の文字リテラル (中断文字なしで指定した 2 つの単一引用符) が「ファイルおよび定義仕様 (File and Definition specifications)」のための PREFIX キーワードの第 1 パラメーターとして指定された場合、指定した文字数がフィールド名から除去されます。 例えば、ファイルがフィールド XRNAME、XRIDNUM、 および XRAMOUNT を持っている場合、「ファイル仕様」に PREFIX('':2) を指定すると、内部フィールド名は、NAME、IDNUM、および AMOUNT になります。

ユーザーが、ファイル固有の接頭部以外同じ名前であるサブフィールドを持つ 2 つのファイルを所有している場合、この機能を使用して、そのファイルから定義される外部記述データ構造のサブフィールドの名前から接頭部を除去することができます。 これを使用すると、EVAL-CORR を使用して、1 つのデータ構造からの同じ名前のサブフィールドを他に割り当てることができます。 例えば、ファイル FILE1 がフィールド F1NAME を持ち、ファイル FILE2 がフィールド F2NAME を持ち、PREFIX('':2) が外部記述データ構造 DS1 を FILE1 に対して、外部記述データ構造 DS2 を FILE2 に対して指定した場合、サブフィールド F1NAME および F2NAME は両方 NAME になります。 データ構造 DS1 および DS2 間の EVAL-CORR 命令は NAME サブフィールドを割り当てます。

DEBUG キーワードの新規値

```
DEBUG { ( *INPUT  *DUMP  *XMLSAX *NO *YES ) }
```

DEBUG キーワードにより、モジュールに生成されるデバッグ補助が決定されます。*NO および *YES が既存の値です。*INPUT、*DUMP および *XMLSAX は *YES よりも詳細な設定を提供します。

***INPUT**
入力仕様書にはあるが、モジュールのその他の場所では使用されていないフィールドは、入力操作時にプログラム・フィールドに読み込まれます。

***DUMP**
(A) 拡張がない DUMP 操作が実行されます。

***XMLSAX**
SAX イベント名の配列が、SAX イベント・ハンドラーのデバッグ中に使用されるモジュールに生成されます。

***NO**
デバッグ援助機能がモジュールに生成されない予定であることを示します。 DEBUG(*NO) を指定しても、DEBUG キーワードを省略した場合と同じになります。

***YES**
この値は、互換性目的のために保持されています。 DEBUG(*YES) を指定することは、パラメーターなしで DEBUG を指定するか、または DEBUG(*INPUT : *DUMP) を指定することと同じです。

フリー・フォーム計算用の構文検査

SEU では、フリー・フォーム・ステートメントが正しい構文であるか検査されるようになりました。

修飾されたデータ構造のヌル可能サブフィールド用の改良されたデバッグ・サポート

ヌル可能サブフィールドを持つ修飾されたデータ構造をデバッグする際に、すべてのヌル可能サブフィールド用に標識サブフィールドを持つ同じデータ構造として、NULL 標識が編成されるようになりました。 データ構造の名前は、_QRNU_NULL_データ構造名 (例えば _QRNU_NULL_MYDS) です。 データ構造のサブフィールド自身がヌル可能サブフィールドを持つデータ構造である場合は、NULL 標識データ構造は、標識サブフィールドを持つデータ構造サブフィールドを同様に持ちます。 例えば、データ構造 DS1 がヌル可能サブフィールド DS1.FLD1、DS1.FLD2、および DS1.SUB.FLD3 を持つ場合、デバッグ命令を使用してデータ構造全体にすべての NULL 標識を表示することができます。

```
===> EVAL _QRNU_NULL_DS
> EVAL _QRNU_NULL_DS1
  _QRNU_NULL_DS1.FLD1 = '1'
  _QRNU_NULL_DS1.FLD2 = '0'
  _QRNU_NULL_DS1.SUB.FLD3 = '1'
===> EVAL _QRNU_NULL_DS.FLD2
  _QRNU_NULL_DS1.FLD2 = '0'
===> EVAL _QRNU_NULL_DS.FLD2 = '1'
===> EVAL DSARR(1).FLD2
  DSARR(1).FLD2 = 'abcde'

===> EVAL _QRNU_NULL_DSARR(1).FLD2

  _QRNU_NULL_DSARR(1).FLD2 = '0'
```

共用ファイルを持つファイルの終わりの振る舞いの変更

モジュールが共用ファイルにキー付き順次入力操作を実行し、EOF 条件になり、異なるモジュールが SETLL などの位置決め操作を使用してファイル・カーソルを設定する場合は、最初のモジュールによる後続の順次入力操作は正常に実行されます。 この変更の前に、最初の RPG モジュールは、他のモジュールが共用ファイルを位置変更したという事実を無視しました。

振る舞いでのこの変更は、リリース V5R2M0 (SI13932) および V5R3M0 (SI14185) の PTF で使用可能です。

表 *43. V5R3* 以降変更された言語要素

言語単位	要素	記述
制御仕様書キーワード	DEBUG(*INPUT|*DUMP *XMLSAX|*NO|*YES)	新規パラメーター *INPUT、*DUMP および *XMLSAX は、デバッグ補助機能のためのさらなるオプションを提供します。
ファイル仕様書のキーワード	PREFIX('':2)	PREFIX キーワードの第 1 パラメーターとして空のリテラルを指定し、名前の先頭から文字を除去することができます。
定義仕様書キーワード	OPTIONS(*NULLIND)	NULL 標識がパラメーターと共に渡されることを示します。
	PREFIX('':2)	PREFIX キーワードの第 1 パラメーターとして空のリテラルを指定し、名前の先頭から文字を除去することができます。

表 44. V5R3 以降の新しい言語要素		
言語単位	**要素**	**記述**
組み込み関数	%HANDLER(プロトタイプ: パラメーター)	イベント用の処理手順を指定します。
	%XML(文書{:オプション })	XML 文書およびそれを構文解析する方法を制御するオプションを指定します。
命令コード	EVAL-CORR	ソース・データ構造のサブフィールドからのデータおよび NULL 標識をターゲット・データ構造のサブフィールドに割り当てます。
	XML-INTO	XML 文書からのデータをプログラム変数に直接読み込みます。
	XML-SAX	XML 文書の SAX 構文解析を開始します。

V5R3 での変更点

以下に、V5R3 の ILE RPG における機能強化について説明します。

- **新規組み込み関数 %SUBARR:**

  新規組み込み関数 %SUBARR を使用すると、副配列への代入または値として副配列を返すことができます。

  既存の %LOOKUP 組み込み関数とともに、この拡張では要素数が変化する動的サイズ変更配列の実装が可能になりました。

  %SUBARR(array : start) は、array(start) から配列の末尾までの配列の要素を指定します。

  %SUBARR(array : start : num) は、array(start) から array(start + num - 1) までの配列の要素を指定します。

  例:

```
// Copy part of an array to another array:
  resultArr = %subarr(array1:start:num);
  // Copy part of an array to part of another array:
  %subarr(Array1:x:y) = %subarr(Array2:m:n);
  // Sort part of an array
  sorta %subarr(Array3:x:y);

  // Sum part of an array
  sum = %xfoot(%subarr(Array4:x:y));
```

- **SORTA 命令コードは、部分配列のソートを可能にするように拡張されています。**

  演算項目 2 として %SUBARR が指定されると、%SUBARR 組み込み関数で示された部分配列のみがソートされます。

- **%DEC による、date/time/timestamp から数値への直接変換:**

  %DEC は、最初のパラメーターに日付、時刻またはタイム・スタンプを指定し、 オプションの 2 番目のパラメーターで結果の数値のフォーマットを指定できるように拡張されました。

  例:

```
D numDdMmYy       s              6p 0
  D date            s                d    datfmt(*jul)
     date = D'2003-08-21';
     numDdMmYy = %dec(date : *dmy);     // now numDdMmYy = 210803
```

- **実行時に文字データを正しく変換するためには、制御仕様 CCSID (*CHAR : *JOBRUN) が使用されます:**

制御仕様書 CCSID キーワードは最初のパラメーターとして *CHAR を許容するように拡張されました。最初のパラメーターが *CHAR の場合、2 番目のパラメーターは *JOBRUN である必要があります。CCSID(*CHAR : *JOBRUN) は、実行時に文字データが UCS-2 に変換される方法を制御します。CCSID(*CHAR:*JOBRUN) が指定されると、文字デ ータはジョブ CCSID 内にあると見なされ、CCSID(*CHAR : *JOBRUN) が指定 されない場合は、文字データはジョブ CCSID に関連する混合バイト CCSID 内にあるものと見なされます。

- **%TRIM、%TRIMR および %TRIML の 2 番目のパラメーターは以下のように、どの文字をトリムするかを示します。**

  %TRIM には、トリムする文字のリストを指定する、オプションの 2 番目のパラメーターを取るように拡張されました。

  例:

```
trimchars = '*-.';
  data = '***a-b-c-.'
  result = %trim(data : trimchars);
  // now result = 'a-b-c'.  All * - and . were trimmed from the ends of the data
```

- **トリムされたパラメーターを渡すための新しいプロトタイプ・オプション OPTIONS(*TRIM) :**

  プロトタイプ・パラメーターに OPTIONS(*TRIM) が指定されると、渡さ れるデータの先頭および末尾のブランクはトリムされます。 OPTIONS(*TRIM) は文字、UCS-2 および CONST または VALUE で定義されたグラフィック・パラメーターに対して有効です。 また、OPTIONS(*STRING) で定義されたポインター・パラメーターに対しても有効です。 OPTIONS(*STRING : *TRIM) が指定されると、ポインターが呼び出しで渡される場合であっても、渡されるデータはトリムされます。

  例:

```
D proc            pr
D   parm1                        5a    const options(*trim)
D   parm2                        5a    const options(*trim : *rightadj)
D   parm3                        5a    const varying options(*trim)
D   parm4                         *    value options(*string : *trim)
D   parm5                         *    value options(*string : *trim)
D ptr             s               *
D data            s              10a
D fld1            s               5a

   /free
         data = ' rst ' + x'00';
         ptr = %addr(data);

         proc (' xyz ' : ' @#$ ' : ' 123 ' : ' abc ' : ptr);
         // the called procedure receives the following parameters
         //    parm1 = 'xyz  '
         //    parm2 = '  @#$'
         //    parm3 = '123'
         //    parm4 = a pointer to 'abc.' (where . is x'00')
         //    parm5 = a pointer to 'rst.' (where . is x'00')
```

- **63 桁のパック 10 進数値およびゾーン 10 進数 値のサポート**

  パック・データおよびゾーン・データを 63 桁および小数点以下 63 桁まで定義できます。 以前の上限は 31 桁でした。

- **I/O の結果データ構造として外部記述されたファイルおよびレコード様式を使用する規則の緩和**

  – レコード様式に対する I/O のための結果データ構造は、外部記述されたデータ構造にすることができます。

  – データ構造は、I/O の結果フィールド内で、 命令コード CHAIN、READ、 READE、READP および READPE の外部記述ファイル名に指定することができます。

  例:

  1. 以下のプログラムは外部記述されたデータ構造を使用してレコード様式への書き込みを行います。

```
Foutfile    o    e           k disk
D outrecDs      e ds                    extname(outfile) prefix(O_)
/free
      O_FLD1 = 'ABCDE';
```

```
      O_FLD2 = 7;
      write outrec outrecDs;
      *inlr = *on;
/end-free
```

2. 以下のプログラムはマルチフォーマット論理ファイルから、それぞ れのレコード様式のフィールドを保持する 2 つの重複したサブフ ィールドを含むデータ構造 INPUT への読み取りを行います。

```
Flog       if   e           k disk    infds(infds)
D infds           ds
D   recname             261    270
D input           ds                  qualified
D   rec1                              likerec(rec1) overlay(input)
D   rec2                              likerec(rec2) overlay(input)
   /free
      read log input;
      dow not %eof(log);
         dsply recname;
         if recname = 'REC1';
         // handle rec1
         elseif recname = 'REC2';
            // handle rec2
         endif;
         read log input;
      enddo;
      *inlr = *on;
   /end-free
```

- プログラム/モジュールが共用ファイルにキー付き順次入力操作を実行し、EOF 条件になる場合は、同じプログラム/モジュールによる後続の順次入力操作は試行できます。 入力要求がデータベースに送信され、入力用にレコードが使用可能であれば、データはプログラム/モジュールに移動され、EOF 条件はオフに設定されます。
- **Java メソッドを呼び出す RPG プログラムで使用するための新しい環境変数のサポート**
  - **QIBM_RPG_JAVA_PROPERTIES** は、RPG ユーザーが JVM の開始に使用する Java プロパティーを明示的に設定することを可能にします。

    この環境変数は、ジョブ内で RPG プログラムが Java メソッドを呼び出す前に設定する必要があります。

    この環境変数には、いずれのオプション・ストリングにも存在しない特定の文字で区切られて終了する、Java オプションが含まれています。 通常は、この文字としてセミコロンを使用することをお勧めします。

    例:

    1. **単一のオプションを指定:** システムのデフォルトの JDK が 1.3 であり、RPG プログラムで JDK 1.4 を使用する場合は、 環境変数 QIBM_RPG_JAVA_PROPERTIES に次の値を設定します。

    ```
    '-Djava.version=1.4;'
    ```

    なお、オプションが 1 つのみの場合でも、終了文字が必要であることに注意してください。 この例ではセミコロンを使用しています。

    2. **複数のオプションを指定する場合:** os400.stdout オプションもデフォルト値以外の値に設定したい場合には、 環境変数を次の値に設定することができます。

    ```
    '-Djava.version=1.4!-Dos400.stdout=file:mystdout.txt!'
    ```

    この例では、 区切り文字および終了文字として感嘆符を使用しています。 注: この機能は PTF 適用済みの V5R1 および V5R2 でもサポートされています。 V5R1: SI10069、V5R2: SI10101。
  - **QIBM_RPG_JAVA_EXCP_TRACE** は、Java メソッドに対する RPG 呼び出しが例外で終了した場合に、RPG ユーザーが例外トレースを取得することを可能にします。

    この環境変数はいつでも設定、変更、または除去できます。

    この環境変数に値「Y」が含まれている場合、RPG からの Java メソッドの呼び出し中に Java 例外が発生するか、呼び出された Java メソッドが呼び出し元に例外をスローすると、例外の Java トレースが出力されます。 デフォルトでは、画面に出力され、読み取れない場合もあります。 例外トレースをフ

ファイルに出力するには、Java オプションの os400.stderr を設定します。 (これは新規ジョブに対して行う必要があり、 QIBM_RPG_JAVA_PROPERTIES 環境変数を次の値に設定することで行います。

```
'-Dos400.stderr=file:stderr.txt;'
```

- **RPG プリプロセッサーにより、SQL プリプロセッサーにおける条件コンパイルおよびネストした /COPY の処理が可能**

  RPG コンパイラーは、パラメーター PPGENOPT に *NONE 以外の値を指定して呼び出された場合、RPG プリプロセッサーとして動作します。 この場合、プログラムが生成されるのではなく、新しいソース・ファイルが生成されます。 新しいソース・ファイルには、/DEFINE や /IF などの条件コンパイル・ディレクティブによって受け入れられたオリジナルのソース行が含まれます。 また、/COPY ステートメントで組み込まれたファイルのソース行、およびオプションで /INCLUDE ステートメントで組み込まれたソース行も含まれます。 PPGENOPT(*DFT) または PPGENOPT(*NORMVCOMMENT) が指定された場合、新しいソース・ファイルにはオリジナル・ソース・ファイルからのコメントも含まれます。

  SQL プリコンパイラーが新規パラメーター RPGPPOPT に *NONE 以外の値を指定して呼び出されると、プリコンパイラーは /COPY、条件コンパイル・ディレクティブ、そしておそらくは /INCLUDE ディレクティブを処理するために、この RPG プリプロセッサーを使用します。 これにより、SQLRPGLE ソースでネストされた /COPY ステートメントおよび条件付きで使用されるステートメントを使用できます。

表 *45. V5R2* 以降に変更された言語要素

言語単位	要素	説明
制御仕様書キーワード	CCSID(*GRAPH:パラメーター| *UCS2:数値| *CHAR:*JOBRUN)	実行時の文字データの扱いを制御するために、最初のパラメーターとして *CHAR、2 番目のパラメーターとして *JOBRUN を 指定できるようになりました。
組み込み関数	%DEC(式 {形式})	タイプ Date、Time または Timestamp のパラメーターを使用できるようになりました。
	%TRIM(式:式)	トリムする文字のセットを示す 2 番目のパラメ ーターを指定できるようになりました。
定義仕様書キーワード	OPTIONS(*TRIM)	渡されたパラメーターからブランクをトリム することを示します。
定義仕様書	長さおよび小数点以下の桁数の入力	パック・フィールドおよびゾーン・フィールドでは、長さおよび小数点以下の桁数として 63 桁まで指定できます。
入力仕様書	長さ入力	パック・フィールドの長さとして 32 まで、ゾーン ・フィールドの長さとして 63 まで指定できます。
	小数部の桁数入力	パック・フィールドおよびゾーン・フィールドでは、小数点以下の桁数として 63 桁まで指定できます。

表 45. V5R2 以降に変更された言語要素 (続き)		
言語単位	**要素**	**説明**
演算仕様書	長さおよび小数点以下の桁数の入力	パック・フィールドおよびゾーン・フィールドでは、長さおよび小数点以下の桁数として 63 桁まで指定できます。
	CHAIN、READ、READE、 READP、および READPE 命令	演算項目 2 が外部記述ファイルの名前である 場合に、結果フィールドにデータ構造を指定できます。
	CHAIN、READ、READC、 READE、READP、READPE、 WRITE、UPDATE 操作	演算項目 2 が外部記述レコード様式 の名前である場合に、結果フィールドに外部記述データ構造を指定できます。
	SORTA 操作	演算項目 2 が拡張され、%SUBARR を指定できる ようになりました。

表 46. V5R2 以降の新しい言語要素		
言語単位	**要素**	**説明**
組み込み関数	%SUBARR(配列:開始要素 {:要素の数})	配列のセクションを返すか、配列のセクションを変更できます。

V5R2 での変更点

以下に、V5R2 において ILE RPG に対して行われた機能強化について説明します。

- 文字から数値への変換

  組み込み関数 %DEC、%DECH、%INT、%INTH、%UNS、%UNSH、 および %FLOAT が、文字パラメーターを利用できるように拡張されました。 例えば、%DEC('-12345.67' : 7 : 2) と指定すると、数値 -12345.67 が戻されます。

- ビット単位の論理組み込み関数

  %BITAND、%BITOR、%BITXOR、および %BITNOT で、RPG 式内での直接ビット操作が認められるようになりました。

- 複雑なデータ構造

  データ構造定義が拡張され、 データ構造の配列、およびそれ自身がデータ構造である LIKEDS で定義されたデータ構造のサブフィールドが認められようになりました。 これにより、配列の配列、つまり構造の副配列が入っている構造の配列など、 複雑な構造のコーディングを行えるようになりました。

  ```
  Example:   family(f).child(i).hobbyInfo.pets(p).type = 'dog';
             family(f).child(i).hobbyInfo.pets(p).name = 'Spot';
  ```

  さらに、レコード様式と同じように、 新しい LIKEREC キーワードを使用してデータ構造を定義できるようになりました。

- 外部記述データ構造の拡張

  外部記述データ構造に、 プログラマーが選択した入力、出力、入出力、キー、またはすべてのフィールドを保持できるようになりました。 現在、外部記述データ構造に保持できるのは入力フィールドだけです。

- キーによる入出力の拡張

  キーによる入出力操作での検索引数を /FREE 計算で新たに 2 つの方法で指定できるようになりました。

  1. リストで検索引数 (式も可能) を指定する

2. 検索引数が入っているデータ構造を指定する

```
Examples: D custkeyDS    e ds        extname(custfile:*key)
           /free
              CHAIN  (keyA : keyB : key3) custrec;
              CHAIN  %KDS(custkeyDS) custrec;
```

- 外部記述ファイルのデータ構造結果

  外部記述ファイルに入出力操作を使うときに、 結果フィールドにデータ構造を指定できるようになりました。 この方法は、V5R2 より前ではプログラム記述ファイルでしか行えませんでした。 ファイル内のフィールド数が多い場合は、 データ構造を使用するとパフォーマンスを向上させることができます。

- 選択フィールドのみを更新する UPDATE 操作

  UPDATE 操作により、更新するフィールドのリストを指定できるようになりました。 これは、V5R2 より前では例外出力を使用しなければ実行できませんでした。

  例: 更新レコード %フィールド （サラリー ： ステータス)。

- 31 桁のサポート

  最大 31 桁のパックおよびゾーン数データをサポートします。 これは、DDS でサポートされる最大長です。 V5R2 より前では 30 桁しかサポートされませんでした。

- FEOD のパフォーマンス・オプション

  FEOD 操作は、ローカルでブロック・バッファーへの書き込みしか実行せず、コストのかかるディスク書き込みは実行しないことを示す、 拡張 N をサポートすることで拡張されました。

- データ域アクセスの拡張

  DTAARA キーワードが拡張され、 データ域の名前およびライブラリーを実行時に指定できるようになりました。

- 新しい代入演算子

  新しい代入演算子 +=、-=、*=、/=、**= により、 変数を古い値に基づいてより簡潔に変更できるようになりました。

  ```
  Example: totals(current_customer) += count;
  ```

  上記のステートメントは、 現在 "totals(current_customer)" に入っている値に "count" を追加します。 "totals(current_customer)" を 2 度コーディングする必要はありません。

- IFS ソース・ファイル

  ILE RPG コンパイラーは、 メイン・ソース・ファイルと /COPY ファイルの両方を IFS からコンパイルできるようになりました。 /COPY ディレクティブおよび /INCLUDE ディレクティブが IFS ファイル名をサポートするように拡張されました。

- プログラム呼び出しマークアップ言語 (PCML)

  ILE RPG コンパイラーは、PCML が入っている IFS ファイルを生成し、プログラム (CRTBNDRPG) またはエクスポートされたプロシージャー (CRTRPGMOD) に対するパラメーターを表します。

表 *47. V5R1* 以降に変更された言語要素

言語単位	要素	説明
組み込み関数	%DEC(式)	型を表す文字のパラメーターを取れるようになった。
	%DECH(式)	
	%FLOAT(式)	
	%INT(式)	
	%INTH(式)	
	%UNS(式)	
	%UNSH(式)	
定義仕様書キーワード	DTAARA({*VAR:}データ域名)	データ域名として、名前、'LIBRARY/NAME' を指定する文字リテラル、 または実行時に実際のデータ域を指定する文字変数を指定できる。
	DIM	データ構造の指定が可能。
	LIKEDS	サブフィールドの指定が可能。
	EXTNAME(ファイル名{:外部レコード 名} {:*ALL|*INPUT|*OUTPUT|*KEY})	オプションの "type" パラメーターを使って、 外部記述データ構造の場合にどのタイプのフィールドを取り出すかを制御する。
定義仕様書	長さおよび小数点以下の桁数の入力	パック・フィールドおよびゾーン・フィールドでは、長さおよび小数点以下の桁数として 31 まで指定できる。
命令コード	CHAIN、DELETE、READE、 READPE、SETGT、SETLL	自由形式演算では、演算項目 1 にキー値のリストを指定できる。
	CHAIN、 READ、READC、 READE、 READP、READPE、UPDATE、WRITE	外部記述ファイルまたはレコード様式で使用する場合は、 結果フィールドにデータ構造を指定できる。
	UPDATE	自由形式演算では、 最後の引数に、更新するフィールドのリストを指定できる。
	FEOD	命令拡張 N を指定できる。 これは、まだ書き込まれていないバッファーはデータベースには書き込む必要があるが、 必ずしもディスクには書き込む必要がないことを示す。
演算仕様書	長さおよび小数点以下の桁数の入力	パック・フィールドおよびゾーン・フィールドでは、長さおよび小数点以下の桁数として 31 まで指定できる。

表 *48. V5R1* 以降の新しい言語要素

言語単位	要素	説明
式	代入演算子 += -= *= /= **=	代入演算子を使用する場合、演算のターゲットは演算の第 1 オペランドでもある。
制御仕様書キーワード	DECPREC(30|31)	表示の場合 (例えば、%EDITC および %EDITW) の 10 進数中間値の精度を制御する。
定義仕様書キーワード	LIKEREC(内部レコード名{:*ALL| *INPUT|*OUTPUT|*KEY})	サブフィールドがレコード様式と同じであるデータ構造を定義する。

表 48. V5R1 以降の新しい言語要素 (続き)		
言語単位	**要素**	**説明**
組み込み関数	%BITAND(式 : 式)	オペランドの対応するビットが両方ともオンの場合にオンであるビットの結果を戻す。
	%BITNOT(式)	引数のビットとは逆のビットの結果を戻す。
	%BITOR(式 : 式)	オペランドの対応するビットのいずれかがオンの場合にオンであるビットの結果を戻す。
	%BITXOR(式 : 式)	オペランドの対応するビットの 1 つだけがオンである場合にオンであるビットの結果を戻す。
	%FIELDS(名前{:名前...})	自由形式の UPDATE で、更新するフィールドを指定する場合に使用。
	%KDS(データ構造)	自由形式のキー付き命令コード CHAIN、 SETLL、SETGT、READE、および READPE で、 操作のキーがデータ構造内にあることを示す場合に使用。

V5R1 での変更点

ILE RPG コンパイラーは、IBM Rational Development Studio for i 製品 (現在では C/C++ および COBOL コンパイラーが組み込まれています) とアプリケーション開発ツール・セット・ツールの一部です。

V4R4 以降の RPG IV の主な機能強化は、Java とのより容易なインターフェース接続、新しい組み込み関数、自由形式の演算仕様書、どのファイルをオープンするかの制御、修飾サブフィールド名、およびエラー処理の強化です。

以下に、これらの強化機能について説明します。

- Java Native Interface (JNI) の使用により、以下について、Java と ILE RPG の間の呼び出しのサポートが改善されました。
  - 新しいデータ・タイプ: オブジェクト
  - 新しい定義仕様書キーワード: CLASS
  - LIKE 定義仕様書キーワードが拡張され、オブジェクトをサポートするようになりました。
  - EXTPROC 定義仕様書キーワードが拡張され、Java プロシージャーをサポートするようになりました。
  - 新しい状況コード。
- 新しい組み込み関数。
  - 数字を期間に変換する次の関数。これらは算術式の中で使用することができます: %MSECONDS、%SECONDS、%MINUTES、%HOURS、%DAYS、 %MONTHS、および %YEARS。
  - %DIFF 関数。ある日付、時刻、またはタイム・スタンプの値を、もう 1 つの値から減算します。
  - 文字ストリング (または日付またはタイム・スタンプ) を日付、時刻、またはタイム・スタンプに変換する次の関数: %DATE、%TIME、および %TIMESTAMP。
  - %SUBDT 関数。日付、時刻、またはタイム・スタンプのサブセットを取り出します。
  - 記憶域の割り振りまたは再割り振りを行なう関数: %ALLOC および %REALLOC。
  - 配列の要素を検索する次の関数: %LOOKUP、%LOOKUPGT、 %LOOKUPGE、%LOOKUPLT、および %LOOKUPLE。
  - テーブルの要素を検索する次の関数: %TLOOKUP、%TLOOKUPGT、 %TLOOKUPGE、%TLOOKUPLT、および %TLOOKUPLE。
  - 指定された文字だけを含むストリングを検査する関数: %CHECK および %CHECKR。
  - %XLATE 関数。変換元文字と変換先文字のリストに基づいてストリングの変換を行います。

  - %OCCUR 関数。複数オカレンス・データ構造において現行のオカレンスを入手したり設定したりします。
  - %SHTDN 関数。オペレーターがシャットダウンを要求したかどうかを判断します。
  - %SQRT 関数。数値の平方根を計算します。
- 演算仕様書の新たな自由形式構文。 自由形式の演算仕様書のブロックは、コンパイラー・ディレクティブ /FREE および /END-FREE で区切られます。

  注 : これらのコンパイラー・ディレクティブは、現在では不要になりました。 80 ページの『/FREE... /END-FREE』を参照してください。
- ファイル仕様に EXTFILE キーワードおよび EXTMBR キーワードを指定して、ファイルがオープンされる時にどの外部ファイルが使用されるかを制御することができます。
- データ構造の中で次の通り修飾名がサポートされます。
  - 新しい定義仕様書キーワード: QUALIFIED。 このキーワードは、サブフィールド名がデータ構造名によって修飾されることを示します。
  - 新しい定義仕様書キーワード: LIKEDS。 このキーワードは、サブフィールドが別のデータ構造から複製されることを指定します。 サブフィールド名は新しいデータ構造名によって修飾されます。 LIKEDS はプロトタイプ化されたパラメーターに使用できます。 このキーワードにより、パラメーターのサブフィールドを直接に呼び出し先プロシージャーで使用することができます。
  - INZ 定義仕様書キーワードが拡張され、データ構造をその親データ構造を基にして初期化することができるようになりました。
- 次のような、エラー処理の機能拡張。
  - 新しい 3 つの演算コード (MONITOR、ON-ERROR、および ENDMON) によって、状況コードに合わせた条件付きエラー処理を伴う命令グループを定義することができるようになりました。

このリリースでは、これ以外の拡張も行われています。 これには，次のものがあります。

- パラメーターを持たないプロシージャー呼び出しにおいて、括弧を指定することができます。
- プロシージャーが ILE C 呼び出し規則または ILE CL 呼び出し規則を使用することを、EXTPROC 定義仕様書キーワードで指定することができます。
- 次の /DEFINE 名が事前定義されています: *VnRnMn、*ILERPG、 *CRTBNDRPG、および *CRTRPGMOD。
- %SCAN 命令における検索ストリングが、検索対象となるストリングよりも長くてもかまいません。 (そのストリングは検索されないことになりますが、これによってエラー条件が生成されることはなくなりました)。
- DIM キーワード、OCCURS キーワード、および PERRCD キーワードのパラメーターは、事前に定義されている必要がなくなりました。
- %PADDR 組み込み関数は、その引数に応じて、プロトタイプ名または入り口点名のいずれかをとることができるようになりました。
- 新しい命令コード ELSEIF は、ELSE 命令コードと IF 命令コードを結合したもので、追加の ENDIF は必要ありません。
- DUMP 命令コードは、DEBUG(*NO) が指定されていても必ずダンプが生成されるという意味の A 拡張を、新しくサポートするようになりました。
- 新しいディレクティブ /INCLUDE は、/INCLUDE が SQL プリプロセッサーによって展開されない点を除けば、/COPY と同等です。 組み込みファイルは、組み込み SQL またはホスト変数を含むことはできません。
- OFLIND ファイル仕様書キーワードは、名前付き標識を含むすべての標識を、引数として取ることが できるようになりました。
- LICOPT (ライセンス内部コード・オプション) キーワードが、CRTRPGMOD コマンドおよび CRTBNDRPG コマンドで使用可能になりました。
- PREFIX ファイル記述キーワードが、引数として上段シフト文字リテラルを取ることができるようになりました。 リテラルはピリオドで終わることができ、ファイルはこのピリオドがあれば修飾付きサブフィールドで使用できるようになります。

- PREFIX 定義仕様書キーワードも、引数として上段シフト文字リテラルを 取ることができるようになりました。 このリテラルはピリオドで終わることはできません。

以下の表に、影響を受ける言語の部分別に、変更のあった言語要素と新しい言語要素を 要約します。

表 *49. V4R4* 以降に変更された言語要素

言語単位	要素	説明
組み込み関数	%CHAR(式{:形式})	オプションである 2 番目のパラメーターには、使用したい日付、時刻、またはタイム・スタンプの形式を指定します。 結果は、指定された形式と形式のセパレーターを使用し、入力の形式とセパレーターは使用しません。
	%PADDR(プロトタイプ名)	この関数は、その引数に応じて、プロトタイプ名または入り口点名のいずれかをとることが できるようになりました。
定義仕様書キーワード	EXTPROC(*JAVA:クラス名:プロシージャー名)	Java メソッドが呼び出されることを指定します。
	EXTPROC(*CL:プロシージャー名)	プロシージャーが戻り値に ILE CL 規則を使用することを指定します。
	EXTPROC(*CWIDEN:プロシージャー名)	プロシージャーが、パラメーター拡大付きで ILE C 規則を使用することを指定します。
	EXTPROC(*CNOWIDEN:プロシージャー名)	プロシージャーが、パラメーター拡大なしで ILE C 規則を使用することを指定します。
	INZ(*LIKEDS)	LIKEDS キーワードを指定して定義されたデータ構造が、その親データ構造から初期設定を継承することを指定します。
	LIKE(オブジェクト名)	オブジェクトが、別のオブジェクトと同じクラスを持つことを指定します。
	PREFIX(文字リテラル{:数値})	指定された文字リテラルでサブフィールドに接頭語を付け、任意指定で指定された文字数を置換します。
ファイル仕様書のキーワード	OFLIND(名前)	このキーワードは、任意の名前付き標識をパラメーターとしてとることができるようになりました。
	PREFIX(文字リテラル{:数値})	指定された文字リテラルでサブフィールドに接頭語を付け、任意指定で指定された文字数を置換します。
命令コード	DUMP (A)	この命令コードは、DEBUG(*NO) が指定されていてもダンプが生成される、A 拡張を今後取ることができるようになりました。

表 *50. V4R4* 以降の新しい言語要素

言語単位	要素	説明
データ・タイプ	オブジェクト	Java オブジェクトに使用されます。
コンパイラー・ディレクティブ	/FREE ... /END-FREE	/FREE... /END-FREE コンパイラー・ディレクティブは、自由形式演算仕様書ブロックを示します。

表 *50. V4R4* 以降の新しい言語要素 *(*続き*)*		
言語単位	**要素**	**説明**
	/INCLUDE	SQL プリプロセッサーによって展開されない点を除き、/COPY と同等です。 コピーされたファイルの中にある、ネストされたファイルを組み込むために使用できます。 コピーされたファイルは、組み込み SQL またはホスト変数を持つことはできません。
定義仕様書キーワード	CLASS(*JAVA:クラス名)	オブジェクトのクラスを指定します。
	LIKEDS(データ構造名)	データ構造、プロトタイプ化されたパラメーター、あるいは戻り値が、 別のデータ構造のサブフィールドを継承することを指定します。
	QUALIFIED	データ構造内のサブフィールド名が、データ構造名によって修飾されることを示します。
ファイル仕様書のキーワード	EXTFILE(ファイル名)	どのファイルをオープンするか指定します。 値はリテラルまたは変数を指定できます。 デフォルトのファイル名は、ファイル仕様書の位置 **7** で指定されている名前になります。 デフォルトのライブラリーは *LIBL です。
	EXTMBR(メンバー名)	どのメンバーをオープンするか指定します。 値はリテラルまたは変数を指定できます。 デフォルト値は *FIRST です。

表 *50. V4R4* 以降の新しい言語要素 *(*続き*)*

言語単位	要素	説明
組み込み関数	%ALLOC(数値)	指定された容量の記憶域を割り振ります。
	%CHECK(コンパレーター:基本{:開始})	コンパレーターの中になく基本ストリングの中にある先頭文字を検索します。
	%CHECKR(コンパレーター:基本{:開始})	コンパレーターの中になく基本ストリングの中にある末尾文字を検索します。
	%DATE(式{:日付の形式})	式を日付に変換します。
	%DAYS(数値)	数値を日単位の期間に変換します。
	%DIFF(オプション 1:オプション 2:単位)	2 つの日付、時刻、またはタイム・スタンプの値の間の差 (期間) を、指定された単位で計算します。
	%HOURS(数値)	数値を時間単位の期間に変換します。
	%LOOKUPxx(引数:配列{:開始索引{:要素の数}})	指定された引数に近い引数、または指定されたタイプに近いタイプを、指定された配列内で検索します。
	%MINUTES(数値)	数値を分単位の期間に変換します。
	%MONTHS(数値)	数値を月単位の期間に変換します。
	%MSECONDS(数値)	数値をマイクロ秒単位の期間に変換します。
	%OCCUR(データ構造名)	複数オカレンス・データ構造の現在位置を設定するか、または入手します。
	%REALLOC(ポインター:数値)	指定された容量の記憶域を、指定されたポインター用に再割り振りします。
	%SECONDS(数値)	数値を秒単位の期間に変換します。
	%SHTDN	システム・オペレーターがシャットダウンを要求しているかどうかをチェックします。
	%SQRT(数値式)	指定された数値の平方根を計算します。
	%SUBDT(値:単位)	日付、時刻、またはタイム・スタンプの値から、指定された部分を抽出します。
	%THIS	代わりに固有メソッドが呼び出されるクラス・インスタンスへの参照を含むオブジェクト値を戻します。
	%TIME(式{:時刻形式})	式を時刻に変換します。
	%TIMESTAMP(式{:*ISO|*ISO0})	式をタイム・スタンプに変換します。
	%TLOOKUP(引数:検索テーブル{:代替テーブル})	指定された引数に近い引数、または指定されたタイプに近いタイプを、指定されたテーブル内で検索します。
	%XLATE(変換元:変換先:ストリング{:開始位置})	指定されたストリングを、変換元ストリングから変換先ストリングにもとづいて変換します。
	%YEARS(数値)	数値を年単位の期間に変換します。

表 50. V4R4 以降の新しい言語要素 (続き)		
言語単位	**要素**	**説明**
命令コード	MONITOR	条件付きエラー処理を持つ命令のグループを開始します。
	ON-ERROR	状況コードに基づいて、条件付きエラー処理を実行します。
	ENDMON	条件付きエラー処理を持つ命令のグループを終了します。
	ELSEIF	ELSE 命令コードに IF 命令コードを続けたものと同等です。
CRTBNDRPG キーワードおよび CRTRPGMOD キーワード	LICOPT(オプション)	ライセンス内部コード・オプションを指定します。

V4R4 での変更点

V4R2 以降 RPG IV に施された機能強化の主だったものとして、スレッド環境で ILE RPG モジュールを安全に実行するためのサポート、3 桁および 20 桁の符号付き および符号なしの新しい数字データ・タイプ、新しい汎用文字セット・バージョン 2 (UCS-2) データ・タイプおよび UCS-2 フィールドとグラフィックまたは単一バイト文字 フィールド間の変換についてのサポートが挙げられます。

以下に、これらの強化機能について説明します。

- Domino® あるいは Java のように、スレッド化されたアプリケーションから ILE RPG プロシージャーを呼び出すためのサポート。
  - 新しい制御仕様書キーワード THREAD(*SERIALIZE) は、マルチスレッド環境で実行 できるようになったモジュールを識別するものです。 モジュール内のプロシージャーへ のアクセスは、順番に行われます。
- 新しい 1 バイトおよび 8 バイト数字データ・タイプのサポート。3I および 20I 符号付き整数。3 および 20U 符号なし整数。
  - これらの新しい数字データ・タイプにより、これまでより広範囲な整数値が使用で きるようになるため、数字計算のパフォーマンスも向上し、64 ビット AS/400 RISC プロ セッサーをフルに活用できます。
  - 新しい 3U タイプにより、単一バイト文字 (CHAR) の戻りタイプおよびパ ラメーターが値別に渡される ILE C プロシージャーとの通信がさらに容易になります。
  - 新しい INTPREC 制御仕様書キーワードにより、式に入っている整数および符号なし 2 進算術演算の中間値について 20 桁の精度を指定することができます。
  - 整数の除算と剰余演算をサポートする組み込み関数 %DIV および %REM が追加さ れました。
- 新しい汎用文字セット・バージョン 2 (UCS-2) つまり Unicode データ・タイプの サポート
  - UCS-2 (Unicode) 文字セットは、多くの作成言語について文字をエンコードす ることができます。 フィールドは、2 バイトの長さの文字をもつ文字フィールドです。
  - Unicode についてのサポートが追加されたことにより、多国間企業を対象に開発する アプリケーションが 1 つで済むため、コード・ページ変換を行う必要がありません。 Unicode を使用することにより、保全性を失わずに、複数のスクリプトでの文字の処理 が可能になります。
  - MOVEL 演算と、新しい %UCS2 および %GRAPH 組み込み関数を使用した、UCS-2 フィ ールドとグラフィック・フィールドまたは単一バイト文字フィールド間の変換についての サポート。
  - EVAL、MOVE、MOVEL 演算と、新しい %UCS2 組み込み関数を使用した、別のコード化 文字セット識別コード (CCSID) をもつ UCS-2 フ ィールドまたはグラフィック・フィールド間の変換についてのサ ポート。

このリリースでは、これ以外の拡張も行われています。 これには，次のものがあります。

- OPTION 制御仕様書キーワードおよび作成コマンドの新規パラメーター
  - *SRCSTMT は、コンパイラー・リスト内のソース ID および SEU 順序番号からデバ ッグするためにステートメント番号を割り当てられるようにします。 (ステートメ ント番号は、デバッガーがコンパイラー・リスト内でエラーを識別し、実行時エラーが発生したステートメントを特定するために使用されます。)*NOSRCSTMT は、ステートメント番号がリストの行番号と関連付けられ、数字が順番に割り当てられるよう指定します。
  - *NODEBUGIO を使って、デバッグ・ビュー内の入出力仕様書について停止点 を生成しないよう選択できるようになりました。 このオプションを選択した場 合、デバッガー内の READ ステートメントの STEP は、入力仕様書のステップを実行する のではなく、次の計算ステップに進むことを指示します。
- 次のような、INZ 定義仕様書キーワードの新しい特殊語
  - INZ(*EXTDFT)。外部記述データ構造サブフィールドを初期化するために、DDS でデフォルト値を使用できるようにします。
  - INZ(*USER) によって初期化された文字変数は、現行ユーザー・プロファイルの名前 に初期設定されます。
- 新しい %XFOOT 組み込み関数。指定された配列式のすべての要素を合計しま す。
- 新しい EVALR 命令コード。式を評価して、その結果を固定長文字またはグラ フィック結果に割り当てます。 この割り当てにより、データは、結果内で右揃えされま す。
- 新しい FOR 命令コード。反復ループを実行して、初期値、増分値、および 限界値について自由形式を使用できるようにします。
- 新しい LEAVESR 命令コード。これを使用すると、サブルーチン内のどのポイントで も終了することができます。
- OVERLAY(名前:*NEXT) キーワードの新しい *NEXT パラメーター。これを使用すると、あるサ ブフィールドが、次の使用可能位置で別のサブフィールドをオーバーレイするよう指示 できます。
- SETLL 命令コードの新しい *START 値および *END 値は、ファイルの先頭または終端に位置付けます。
- 初期化演算および自由形式演算 (例えば、EVAL、IF など) で整数フィールドおよび符号 なし整数フィールドをもつ 16 進リテラルが使用可能。
- 新しい制御仕様書キーワード OPENOPT{(*NOINZOFL | *INZOFL)}。ファイルがオープンされたときにオーバーフロー標識を *OFF にリセットするかど うかを指示します。
- テラバイト・スペース内、つまり、1 つの割り振りで 16 メガバイトを超える連続する記憶域を許すメモリー・モデルでの ポインターの許容。

以下の表に、影響を受ける言語の部分別に、変更のあった言語要素と新しい言語要素を 要約します。

表 *51. V4R2* 以降変更された言語要素

言語単位	要素	記述
制御仕様書キーワード	OPTION(*{NO}SRCSTMT)	*SRCSTMT は、デバッグのためにステートメント番号を生成する際にコンパイラーが SEU 順序番号およびソース ID を使用するよう要求できるようにします。 *NOSRCSTMT の場合は、ステートメント番号はリストの行番号と関連付けられ、数字が順番に割り当てられます。
	OPTION(*{NO}DEBUGIO)	*{NO}DEBUGIO は、入出力仕様書について停止点を生成するかどうか を決定します。

表 *51. V4R2* 以降変更された言語要素 *(*続き*)*

言語単位	要素	記述
定義仕様書キーワード	INZ(*EXTDFT)	外部記述データ構造サブフィールドが、DDS 内で指定されているデフォルト値に初期設定できるようになりました。
	INZ(*USER)	いずれの文字フィールドまたはサブフィールドも、現行ユーザー・プロファイル の名前に初期設定できます。
	OVERLAY(名前:*NEXT)	特殊値 *NEXT は、サブフィールドをオーバーレイされたフィールド内の次に使用可能な 位置に配置することを指示します。
	OPTIONS(*NOPASS *OMIT *VARSIZE *STRING *RIGHTADJ)	関数プロトタイプ内の値または固定情報パラメーターに指定された新しい OPTIONS(*RIGHTADJ) は、パラメーターとして渡された文字、図形、または UCS-2 値は、プロシージャー呼び出しで渡される前に右寄せすることを示します。
定義仕様書 33 ～ 39 桁目 (終了位置 / 長さ)	データ・タイプ I および U について許されている 3 桁および 20 桁	1 バイトおよび 8 バイトの整数と符号なしデータをサポートするために、内部データ・タイプに使用できる値のリストに追加されました。
内部データ・タイプ	C (UCS-2 固定長形式または可変長形式)	定義仕様書で使用可能な内部データ・タイプのリストに追加されました。 UCS-2 (Unicode) 文字セットは、多くの作成言語について文字をエンコードす ることができます。フィールドは、2 バイトの長さの文字をもつ文字フィールドです。
データ形式	C (UCS-2 固定長形式または可変長形式)	UCS-2 形式が、プログラム記述ファイルの入力仕様書および出力仕様書で使用可能なデータ形式のリストに追加されました。
コマンド・パラメーター	OPTION	CRTBNDRPG および CRTRPGMOD コマンドの OPTION パラメーターに *NOSRCSTMT、*SRCSTMT、 *NODEBUGIO、および *DEBUGIO が追加されました。

表 *52. V4R2* 以降の新規言語要素

言語単位	要素	記述
制御仕様書キーワード	CCSID(*GRAPH: *IGNORE | *SRC | 番号)	モジュールについてデフォルトのグラフィック CCSID を設定します。 この設定値は、リ テラル、コンパイル時刻データ、およびプログラムで記述された入出力フィールドなら びに定義に使用されます。 デフォルトは *IGNORE です。
	CCSID(*UCS2: 番号)	モジュールにデフォルトの UCS-2 CCSID を設定します。 この設定値は、リ テラル、コンパイル時刻データ、およびプログラムで記述された入出力フィールドなら びに定義に使用されます。 デフォルトは 13488 です。
	INTPREC(10 | 20)	式の 2 進算術演算に整数の符号なし中間値の 10 進精度を指定します。 デフォルトであ る INTPREC(10) は、10 桁の精度が使用されることを示します。
	OPENOPT{(*NOINZOFL | *INZOFL)}	ファイルがオープンされたときにオーバーフロー標識を *OFF にリセットするかど うかを指示します。
	THREAD(*SERIALIZE)	モジュールがマルチスレッド化環境で実行できるようにすることを指示します。 モジュール内のプロシージャーへのアクセスは、順番に行われます。
定義仕様書キーワード	CCSID(番号 | *DFT)	定義に図形と UCS-2 の CCSID を設定します。
組み込み関数	%DIV(n:m)	2 つのオペランド n と m で整数の除算を実行します。その結果は、n/m の整数部分になります。 これらのオペランドは、小数点以下の桁数がない (ゼロの) 数値である必要があります。
	%GRAPH(文字式 | 図形式 | UCS2 式 {: ccsid})	単一バイト文字、グラフィック、または UCS-2 データからグラフィック・データに変換 します。
	%REM(n:m)	2 つのオペランド n と m で整数剰余の演算を実行します。その結果は n/m の剰余になります。 これらのオペランドは、小数点以下の桁数がない (ゼロの) 数値である必要があります。
	%UCS2(文字式 | 図形式 | UCS2 式 {:ccsid})	単一バイト文字、グラフィック、または UCS-2 データから UCS-2 データに変換します。
	%XFOOT(配列式)	指定された数値配列式内の全要素の合計を求めます。

表 52. V4R2 以降の新規言語要素 (続き)		
言語単位	**要素**	**記述**
命令コード	EVALR	形式 result=expression の割り当てステートメントを評価します。 結果は、右揃えされます。
	FOR	1 つの命令グループを開始し、そのグループが処理される回数を指示します。 初期値、増分値、および限界値は、自由形式の式でかまいません。
	ENDFOR	ENDFOR は、FOR 命令によって開始された命令のグループを終了します。
	LEAVESR	これを使用すると、サブルーチン内のどこからでも終了することができます。

V4R2 での変更点

V3R7 以降の RPG IV の主な拡張機能は、可変長フィールドのサポート、標識に関係するいくつかの拡張機能、および制御仕様書にコンパイル・オプションを指定する機能です。 これらの拡張機能により、オペレーティング・システムと ILE 言語間通信との統合において、RPG 製品がさらに改善されました。

以下に、これらの強化機能について説明します。

- 可変長フィールドのサポート

  この拡張機能によって、可変長の文字フィールドと図形フィールドが完全にサポートされます。 可変長フィールドを使用すれば、多くのストリング処理タスクを単純化することができます。
- INDARA 標識にユーザー固有のデータ構造を使用するための機能

  値をデータ管理機能に伝えるための *IN 配列を使用しなくても、 ユーザーは論理データ域にアクセスして、INDARA を 0 使用する各 WORKSTN および PRINTER ファイルに標識データ構造を関連付ける ことができるようになりました。
- 結果の標識の代わりに組み込み関数を使用する機能

  入出力操作の結果を照会するための組み込み関数 %EOF、%EQUAL、%FOUND、および %OPEN が追加されました。 エラー処理のための組み込み関数 %ERROR と %STATUS、および命令コード拡張 'E' が追加されました。
- 制御仕様書でのコンパイル・オプション

  コンパイル・オプションは、CRTBNDRPG および CRTRPGMOD コマンドで指定していましたが、制御仕様書キーワードで指定できるようになりました。 これらのコンパイル・オプションは、プログラムのコンパイルごとに使用されます。

さらに、以下の新しい機能が追加されました。

- 大文字と小文字が混合した名前を持つプロシージャーおよび変数のインポートおよびエクスポートのサポート
- DECEDIT 値を実行時に動的に設定する機能
- ストリング処理を容易にするための組み込み関数 %CHAR および %REPLACE
- 外部定義 *CMDY、*CDMY、および *LONGJUL 日付データ形式の新しいサポート
- 世紀日付形式の範囲の拡張
- 標識変数を定義する機能
- OVERLAY キーワードのパラメーターとして現行データ構造名を指定する機能
- 可変長のフィールド・エラーを示すための新しい状況コード 115
- アプリケーション・プロファイル作成のサポート

- FIXNBR(*INPUTPACKED) を使用して、ファイルからの検索時に 無効なパック 10 進数を処理する機能
- CRTRPGMOD コマンドに BNDDIR コマンド・パラメーターを指定する機能

以下の表に、影響を受ける言語の部分別に、変更のあった言語要素と新しい言語要素を 要約します。

表 *53. V3R7* 以降に変更された言語要素

言語単位	要素	説明
制御仕様書キーワード	DECEDIT(*JOBRUN | '値')	10 進数の編集値を、ジョブ値またはシステム値から、実行時に動的に決定できるようになりました。
定義仕様書キーワード	DTAARA {(データ域名)}	ユーザーが論理データ域にアクセスできるようになりました。
	EXPORT {(外部名)}	エクスポートする変数の外部名を、このキーワードのパラメーターとして指定できるようになりました。
	IMPORT {(外部名)}	インポートする変数の外部名を、このキーワードのパラメーターとして指定できるようになりました。
	OVERLAY(名前{:位置})	名前パラメーターとして現行データ構造の名前を指定できるようになりました。
世紀の拡張形式	*CYMD (cyy/mm/dd)	世紀の文字 'c' の有効値は次のようになりました。 'c' Years 0 1900-1999 1 2000-2099 . . . . . . 9 2800-2899
内部データ・タイプ	N (標識形式)	定義仕様書で使用可能な内部データ・タイプのリストに追加されました。 標識形式に文字データを定義します。
データ形式	N (標識形式)	標識形式が、プログラム記述ファイルの入力仕様書および出力仕様書で使用可能なデータ形式のリストに追加されました。
データ属性	*VAR	プログラム記述ファイルの入力仕様書および出力仕様書で使用可能なデータ属性のリストに追加されました。 この属性は、可変長フィールドを指定するために使用します。
コマンド・パラメーター	FIXNBR	*INPUTPACKED パラメーターが、無効なパック 10 進数データを処理するために追加されました。

表 *54. V3R7* 以降の新しい言語要素

言語単位	新規	説明
制御仕様書キーワード	ACTGRP(*NEW | *CALLER | '活動化グループ名')	ACTGRP キーワードによって、プログラムが呼び出された時に関連付ける活動化グループを指定することができます。

表 *54. V3R7* 以降の新しい言語要素 *(*続き*)*		
言語単位	**新規**	**説明**
	ALWNULL(*NO | *INPUTONLY | *USRCTL)	ALWNULL キーワードは、外部記述データベース・ファイルからの、ヌル可能フィールドを含んでいるレコードを使用する方法を指定します。
	AUT(*LIBRCRTAUT | *ALL | *CHANGE | *USE | *EXCLUDE | '権限リスト名')	AUT キーワードは、オブジェクトに対する特定の権限を持っていないユーザー、権限認可リストに載っていないユーザー、およびそのユーザー・グループがオブジェクトに対する特定の権限を持っていないユーザーに与える権限を指定します。
	BNDDIR('バインディング・ディレクトリー名'{:'バインディング・ディレクトリー名'...})	BNDDIR キーワードは、記号の解決で使用するバインディング・ディレクトリーのリストを指定します。
	CVTOPT(*{NO}DATETIME *{NO}GRAPHIC *{NO}VARCHAR *{NO}VARGRAPHIC)	CVTOPT キーワードは、ILE RPG コンパイラーが、外部記述データベース・ファイルから取り出した日付、時刻、タイム・スタンプ、図形データ・タイプ、および可変長データ・タイプを処理する方法を指定するために使用します。
	DFTACTGRP(*YES | *NO)	DFTACTGRP キーワードは、作成済みのプログラムが、呼び出されたときに実行する場所である活動化グループを指定します。
	ENBPFRCOL(*PEP | *ENTRYEXIT | *FULL)	ENBPFRCOL キーワードはパフォーマンス・コレクションを使用可能にするかどうかを指定します。
	FIXNBR(*{NO}ZONED *{NO}INPUTPACKED)	FIXNBR キーワードは、無効な 10 進数データをコンパイラーが修正するかどうかを指定します。
	GENLVL(番号)	GENLVL キーワードはオブジェクトの作成を制御します。
	INDENT(*NONE | '文字値')	INDENT キーワードは、構造化命令を読みやすくするために、ソース・リスト内で字下げするかどうかを指定します。
	LANGID(*JOBRUN | *JOB | '言語識別子')	LANGID キーワードは、分類順序が *LANGIDUNQ または *LANGIDSHR である場合に使用する言語識別子を指定します。
	OPTIMIZE(*NONE | *BASIC | *FULL)	OPTIMIZE キーワードは、もしあれば、オブジェクトの最適化のレベルを指定します。
	OPTION(*{NO}XREF *{NO}GEN *{NO}SECLVL *{NO}SHOWCPY *{NO}EXPDDS *{NO}EXT *{NO}SHOWSKP)	OPTION キーワードは、ソース・メンバーのコンパイル時に使用するオプションを指定します。
	PRFDTA(*NOCOL | *COL)	PRFDTA キーワードは、プロファイル作成データのコレクションを 使用可能にするかどうかを指定します。

表 *54. V3R7* 以降の新しい言語要素 (続き)

言語単位	新規	説明
	SRTSEQ(*HEX | *JOB | *JOBRUN | *LANGIDUNQ | *LANGIDSHR | 'ソート・テーブル名')	SRTSEQ キーワードは、ILE RPG ソース・プログラムで使用する 分類順序テーブルを指定します。
	TEXT(*SRCMBRTXT | *BLANK | '記述')	TEXT キーワードによって、オブジェクトおよびその機能について簡単に記述するテキストを入力することができます。
	TRUNCNBR(*YES | *NO)	TRUNCNBR キーワードは、オブジェクトの実行時に数値オーバーフローが発生した場合に、切り捨てた値を結果フィールドに転送するか、あるいは、エラーを生成するかどうかを指定します。
	USRPRF (*USER | *OWNER)	USRPRF キーワードは、作成済みのプログラム・オブジェクトを実行するユーザー・プロファイルを指定します。
ファイル仕様書のキーワード	INDDS(データ構造名)	INDDS キーワードによって、データ構造名をワークステーションまたはプリンター・ファイルの INDARA 標識に関連付けることができます。
定義仕様書キーワード	VARYING	文字データまたは図形データに指定された場合に、可変長フィールドを定義します。
組み込み関数	%CHAR(図形、日付、時刻またはタイム・スタンプの式)	文字データ・タイプの値を戻します。
	%EOF{ファイル名}	最後に実行されたファイル入力操作またはサブファイルへの書き出し (指定された場合は特定のファイルに対する) が、ファイルの終わり条件またはファイルの先頭条件で終了した場合に '1' を戻します。他の場合には '0' を戻します。
	%EQUAL{ファイル名}	最後に実行された SETLL (指定された場合は特定のファイルに対する) または LOOKUP 命令が等しい項目を見付けた場合に '1' を戻します。他の場合には '0' を戻します。
	%ERROR	最後に実行された、拡張 'E' が指定された命令コードの結果がエラーである場合に '1' を戻します。他の場合には '0' を戻します。
	%FOUND{ファイル名}	最後に実行された関係のある命令 (指定された場合は特定のファイルに対する) がレコード (CHAIN、DELETE、SETGT、SETLL)、要素 (LOOKUP)、または等しい項目 (CHECK、CHECKR、および SCAN) を見付けた場合に '1' を戻します。他の場合には '0' を戻します。
	%OPEN(ファイル名)	指定されたファイルがオープンされている場合に '1' を戻します。指定されたファイルがクローズされている場合には '0' を戻します。

表 54. V3R7 以降の新しい言語要素 (続き)		
言語単位	**新規**	**説明**
	%REPLACE(置換ストリング: ソース・ストリング {:開始位置 {:置換するソースの長さ}})	**開始位置**から開始し、指定された文字数を置換して、**置換ストリング**を**ソース・ストリング**に挿入することによって生成されるストリングを戻します。
	%STATUS{ファイル名}	最後に実行された、拡張「E」が指定された命令コード以降、プログラム・エラーまたはファイル・エラーが発生していない場合に「0」を戻します。 エラーが発生した場合には、プログラム状況またはファイル状況について最後に設定された値を戻します。 ファイルが指定されている場合、戻される値は、そのファイルに関する最新の状況になります。
命令コード拡張	E	CALLP 命令で %ERROR および %STATUS 組み込み関数を使用したエラー処理を可能にし、すべての命令でエラー標識を使用できるようにします。
新しい世紀形式	*CMDY (cmm/dd/yy)	MOVE、MOVEL、および TEST 命令で使用できるようになりました。
	*CDMY (cdd/mm/yy)	MOVE、MOVEL、および TEST 命令で使用できるようになりました。
新しい 4 桁の年形式	*LONGJUL (yyyy/ddd)	MOVE、MOVEL、および TEST 命令で使用できるようになりました。
コマンド・パラメーター	PRFDTA	PRFDTA パラメーターは、プロファイル作成データのコレクションを 使用可能にするかどうかを指定します。
	BNDDIR	BNDDIR パラメーターは CRTBNDRPG コマンドでしか使用できず、CRTRPGMOD コマンドでは使用できませんでしたが、両方のコマンドで使用できるようになりました。

V3R7 での変更点

V3R6 以降の RPG IV の主な拡張機能は、データベース・ヌル値フィールドに対する新規サポートと、式の中間結果の精度をさらに厳密に制御する機能です。 それ以外の拡張機能としては、浮動小数点データ・タイプの追加や、null 文字で終了するストリングのサポートなどが含まれます。 これらの拡張機能により、オペレーティング・システムと ILE 言語間通信との統合において、RPG 製品がさらに改善されました。 これは、アプリケーションにおける柔軟性が大きく向上するということです。

以下は、これらの拡張機能 (いくつかの新しい組み込み関数と使用可能度の強化も含む) をリストしたものです。

- データベース・ヌル値フィールドのサポート

  この拡張機能によって、ヌル可能フィールドにヌル値があるかどうかをテストし、そのフィールドをヌル値に設定できるようにすることにより、ユーザーが、ヌル可能フィールドを含むデータベース・ファイルを処理できるようになります。
- 式中間結果の精度

  フリー・フォームで表現される仕様書における新しい制御仕様書キーワードおよび新しい命令コード・拡張によって、ユーザーが、中間結果の精度をより厳密に制御することができるようになります。
- 新しい浮動小数点データ・タイプ

新しい浮動小数点データ・タイプには、他のデータ・タイプより広い範囲の値があります。 このデータ・タイプが追加されたことで、データベースの統合が進み、ILE 環境内における言語間通信、特に C および C++ 言語との通信が向上されます。

- NULL 文字で終了するストリングのサポート

  NULL 文字で終了するストリングを新しくサポートすることにより、言語間通信が向上しました。 このサポートにより、ユーザーは、null 文字で終了するストリングを定義、処理し、さらに null 文字で終了するストリングを待っているプロシージャーにパラメーターとして文字データを簡単に渡すことができるようになり、ひいては null 文字で終了するストリングを完全に制御できるようになりました。

- ポインターの加算および減算

  自由形式の式の機能が強化され、ポインターにオフセットを 加えたり、ポインターからオフセットを引いたり、また、2 つのポインター 間の差を求めることができるようになりました。

- 長名のサポート

  10 文字よりも長い名前が、RPG 言語に追加されました。 定義仕様書またはプロシージャー仕様書で定義されたものには長名を付けることができ、これらの名前は、記入項目の境界内に収まる位置であればどこでも使用することができます。 さらに、フリー・フォームの仕様書で参照される名前は、複数の行に連続する場合があります。

- 新しい組み込み関数

  この言語に新しい組み込み関数がいくつか追加され、それにより、以下の言語機能が向上しました。

  - 編集 (%EDITW、%EDITC、%EDITFLT)
  - 走査ストリング (%SCAN)
  - タイプ変換 (%INT、%FLOAT、%DEC、%UNS)
  - 四捨五入によるタイプ変換 (%INTH、%DECH、%UNSH)
  - 10 進数の式の場合の中間結果の精度 (%DEC)
  - 変数および式の 10 進数の長さ (%LEN, %DECPOS)
  - 絶対値 (%ABS)
  - ヌル可能フィールドの設定およびテスト (%NULLIND)
  - NULL 文字で終了するストリングの取り扱い (%STR)

- 条件付きコンパイル

  RPG IV は、条件付きコンパイルをサポートするように拡張されています。 このサポートには以下の内容が含まれます。

  - 条件の定義 (/DEFINE、/UNDEFINE)
  - 条件のテスト (/IF、/ELSEIF、/ELSE、/ENDIF)
  - 現行ソース・ファイルの読み取りの停止 (/EOF)
  - CRTBNDRPG および CRTRPGMOD コマンドで最高 32 までの条件を定義するための新しいコマンド・オプション (DEFINE)

- データ拡張

  データ処理命令を向上するために、いくつかの拡張が行われました。 TIME 命令コードは、結果フィールド内の日付、時刻またはタイム・スタンプの各フィールドをサポートするように拡張されています。 文字フィールドとの間で日付または時刻を転送する場合には、区切り記号が必要です。 UDATE および *DATE フィールドの転送には、形式コードを指定する必要はなくなりました。 日付フィールドは、定義仕様書のシステム (*SYS) またはジョブ (*JOB) の日付に初期設定することができます。

- 代替照合順序による文字比較

  特定の文字変数を定義して、比較において代替照合順序が使用されないようにすることができます。

- ネストされた /COPY メンバー

/COPY ディレクティブのネストが可能になりました。 すなわち、/COPY メンバーに 1 つ (または複数) の /COPY ディレクティブを含めることができます。そのディレクティブには、さらに別の /COPY ディレクティブなどを含めることができます。

- 記憶域管理

  新しい記憶管理命令コードを使用して、記憶域の割り振り、再割り振り、および割り振り解除が動的に行えるようになりました。

- 記憶管理および浮動アンダーフローのエラーの状況コード

  記憶管理エラーを示すための状況コードが 2 つ (425 および 426) 追加されました。 また、中間浮動結果が小さ過ぎることを示すための状況コード 104 も追加されています。

以下の表に、影響を受ける言語の部分別に、変更のあった言語要素と新しい言語要素を 要約します。

表 *55. V3R6* 以降に変更された言語要素

言語単位	要素	記述
定義仕様書キーワード	ALIGN	以前にサポートされている整数および符号なしの位置合わせに加え、浮動サブフィールドの位置合わせをするために、ALIGN が使用できるようになりました。
	OPTIONS(*NOPASS *OMIT *VARSIZE *STRING)	*STRING オプションによって、NULL 文字で終了するストリングとして文字値を渡すことができます。
レコード・アドレス・タイプ	F (浮動形式)	ファイル仕様書で使用可能なレコード・アドレス・タイプのリストに追加されました。 プログラム記述ファイルについての浮動処理をシグナルします。
内部データ・タイプ	F (浮動形式)	定義仕様書で使用可能な内部データ・タイプのリストに追加されました。 浮動小数点独立フィールド、パラメーター、またはデータ構造サブフィールドを定義します。
データ形式	F (浮動形式)	プログラム記述ファイルの入力仕様書および出力仕様書で使用可能なデータ形式のリストに追加されました。

表 *56. V3R6* 以降の新しい言語要素

言語単位	新規	記述
制御仕様書キーワード	COPYNEST('1 ～ 2048')	/COPY ディレクティブのネストのための最大の深さを指定します。
	EXPROPTS(*MAXDIGITS | *RESDECPOS)	精度のタイプの式オプション (デフォルト値または「結果の小数点以下の桁数」精度規則)
	FLTDIV {(*NO | *YES)}	式の中のすべての除算命令が浮動小数点で計算されることを示します。
定義仕様書キーワード	ALTSEQ(*NONE)	代替照合順序が指定されている場合でも、文字比較に通常の照合順序を使用することを強制します。
組み込み関数	%ABS	パラメーターとして指定されている数値式の絶対値を戻します。

表 *56. V3R6* 以降の新しい言語要素 (続き)

言語単位	新規	記述
	%DEC および %DECH	数値式の値を、パラメーターとして指定されている桁数および小数点以下の桁数を持つ 10 進 (パック) 形式に変換します。%DECH は %DEC と同じですが、四捨五入が適用されます。
	%DECPOS	数値変数または数値式の小数点以下の桁数を戻します。 戻される値は定数で、定数が予期されているところに使用されます。
	%EDITC	この関数は、編集コードに従って編集された数値を表す文字結果を戻します。
	%EDITFLT	数値式の値を、浮動の文字外部表示表現に変換します。
	%EDITW	この関数は、編集ワードに従って編集された数値を表す文字結果を戻します。
	%FLOAT	数値式の値を、浮動形式に変換します。
	%INT および %INTH	数値式の値を、整数に変換します。 10 進数はすべて %INT によって切り捨てられ、%INTH によって丸められます。
	%LEN	変数式の数字または文字の数を戻します。
	%NULLIND	ヌル可能フィールドのヌル標識を照会または設定するために使用されます。
	%SCAN	ソース・ストリングの中の検索引数の 1 桁目、またはそれが見付からない場合には 0 を戻します。
	%STR	NULL 文字で終了するストリングを作成または使用するために使用します。このストリングは、C および C++ アプリケーションで非常に一般的に使用されています。
	%UNS および %UNSH	数値式の値を符号なし形式に変換します。 10 進数はすべて %UNS によって 切り捨てられ、%UNSH によって丸められます。
命令コード拡張	N	DEALLOC が正常に行われた後、 ポイントを *NULL に設定します。
	M	デフォルトの精度規則
	R	小数点以下の桁数が、結果の小数点以下の桁数より少ない中間値はなくなります ("結果の小数点以下の桁数" 精度の規則)。
命令コード	ALLOC	記憶域を動的に割り振るために使用します。
	DEALLOC	記憶域を動的に割り振り解除するために使用します。
	REALLOC	記憶域を動的に再割り振りするために使用します。

V3R6/V3R2 での変更点

V3R1 以降の RPG IV の主な拡張機能は、1 つのモジュールを複数のプロシージャーを使用してコーディングする機能です。 これは何を意味するでしょう? 簡単に言えば、1 つのモジュールを 1 つ以上のプロトタイプ・プロシージャーを使用してコーディングできることを意味します。この場合、プロシージャーは RPG サイクルを使用しないで戻り値を持ち、実行することができます。

1 つのモジュールを複数のプロシージャーを使用して作成すると、 作成するアプリケーションの機能を高めることができます。 すべてのアプリケーションは、 特定のタスクを実行すると考えられる一連の論理単位から構成されます。 柔軟性を保持して アプリケーションを開発するには、各論理単位をできるだけ独立させることが重要です。 各単位を独立させることによって、 以下のことが可能になります。

- 特定のタスクを実行する視点から各単位を作成しやすくなります。
- 変更できるように設計したデータ・オブジェクト以外のデータ・オブジェクトを変更する可能性が少なくなります。
- 論理およびデータ項目をより個別化できるので、デバッグが容易になります。
- 変更が必要なアプリケーションの部分を容易に分離できるので、保守が容易になります。

複数のプロシージャーを使用して 1 つのモジュールをコーディングする主な利点は、 モジュラー・アプリケーションのコーディングの制御が容易になり、効率が上がることです。 この利点はいくつかの方法で実現することができます。 以下を行うことができます。

- 同じ呼び出し命令および構文を使用してプロシージャーおよびプログラムを呼び出す。
- 呼び出しインターフェースのコンパイル時に検査を行うプロトタイプを定義する。
- 値によって、または参照によってパラメーターを渡す。
- 値を戻すプロシージャーを定義し、式の中でそのプロシージャーを呼び出す。
- 変数のローカル定義を実行することによって、データ項目へのアクセスを制限する。
- サイクルを使用しないモジュールをコーディングする。
- プロシージャーを回帰的に呼び出す。

モジュール内のメイン・プロシージャーの実行時の動作は、V3R1 プロシージャーの場合と同じです。 それ以降のプロシージャーの実行時の動作は、V3R1 プログラムとはいくらか異なります。プロシージャーの最後および例外処理の領域で特に異なります。 これらの相違点は、これらのプロシージャーについてはサイクル・コードが生成されないために生じます。

他の拡張は、このリリースについても行われています。 これには，次のものがあります。

- 2 つの新しい整数データ・タイプ、符号付き整数 (I) と符号なし整数 (U) のサポート

  整数データ・タイプによって、2 進データ・タイプより広い範囲の値を使用することができます。 整数データ・タイプは、整数計算のパフォーマンスを向上することもできます。
- MOVE、MOVEL、および TEST 命令の *CYMD サポート

  既にこのデータ形式であるシステム値を処理するために、特定の命令の中で *CYMD 日付形式を使用できるようになりました。
- 制御仕様書の COPYRIGHT キーワードを使用して、プログラムおよびモジュールの版権を示す機能

  このキーワードを使用して指定した版権情報は、DSPMOD、DSPPGM、または DSPSRVPGM 情報の一部になります。
- キーワード BLOCK を使用するレコードのブロック化のユーザー制御

  DISK または SEQ ファイルに SETLL、SETGT、または CHAIN 命令が使用されている場合でも、ファイルのレコードのブロック化を要求することができます。 ブロック化を実行しないように要求することもできます。 このような場合でのブロック化の使用によって、実行時のパフォーマンスがかなり向上する可能性があります。
- PREFIX 機能の改善

  ファイル記述および定義仕様書に関する PREFIX キーワードへの変更によって、既存のフィールド名内の文字を接頭部ストリングで置換できるようになりました。

- トリガー・プログラム・エラーに関する状況コード

  トリガー・プログラム・エラーを示すために 2 つの状況コード 1223 および 1224 が追加されました。

以下の表に、影響を受ける言語の部分別に、変更のあった言語要素と新しい言語要素を 要約します。

表 *57. V3R1* 以降に変更された言語要素

言語単位	要素	説明
ファイル仕様書のキーワード	PREFIX(接頭部ストリング {:置換する文字数})	フィールド名に対するストリングの接頭部を付けること、またはフィールド名の部分的な変更を可能にします。
定義仕様書キーワード	CONST{(定数)}	名前付き定数の値を指定するか、または参照によって渡されるプロトタイプ・パラメーターが定数値を持っていることを示します。
	PREFIX(接頭部ストリング {:置換する文字数})	フィールド名に対するストリングの接頭部を付けること、またはフィールド名の部分的な変更を可能にします。
命令コード	RETURN	呼び出し元に制御を戻し、指定されている場合は、値を戻します。

表 *58. V3R1* 以降の新しい言語要素

言語単位	新規	説明
制御仕様書キーワード	COPYRIGHT('版権ストリング')	版権情報をモジュールおよびプログラムに関連付けることができるようにします。
	EXTBININT{(*NO | *YES)}	プログラム処理中に、外部記述ファイル内の 2 進数フィールドに整数形式を割り当てていることを指定します。
	NOMAIN	モジュールにサブプロシージャーのみがあることを示します。
ファイル仕様書のキーワード	BLOCK(*YES |*NO)	レコードのブロック化の発生を制御できるようにします (他の条件が満たされた場合)
定義仕様書キーワード	ALIGN	整数フィールドまたは符号なしフィールドの位置合わせを行うかどうかを指定します。
	EXTPGM(名前)	プロトタイプ・プログラムの外部名を示します。
	EXTPROC(名前)	プロトタイプ・プロシージャーの外部名を示します。
	OPDESC	プロトタイプ・バインド呼び出しで、操作記述子を渡すかどうかを指定します。
	OPTIONS(*NOPASS *OMIT *VARSIZE)	プロトタイプ・パラメーターに関する各種のオプションを指定します。
	STATIC	ローカル変数が静的記憶域を使用するように指定します。
	値	プロトタイプ・パラメーターを値によって渡すように指定します。
組み込み関数	%PARMS	呼び出しで渡すパラメーターの個数を戻します。

表 *58. V3R1* 以降の新しい言語要素 (続き)		
言語単位	**新規**	**説明**
命令コード	CALLP	プロトタイプ・プログラムまたはプロシージャーを呼び出します。
仕様書タイプ	プロシージャー仕様書	サブプロシージャー定義の先頭と終端を示します。
定義タイプ	PR	プロトタイプ定義の先頭を示します。
	PI	プロシージャー・インターフェース定義の先頭を示します。
	24～25 桁目のブランク	プロトタイプ・パラメーターを定義します。

RPG IV の概念

RPG IV の一般概念

この部では、次の RPG IV の基本の一部について説明しています。

- 記号名
- コンパイラー・ディレクティブ
- RPG IV プログラム・サイクル
- 標識
- エラー処理
- サブプロシージャー
- ファイルに関する一般的な考慮事項

記号名および予約語

RPG IV 言語で有効な文字セットは、次の文字から構成されています。

- 英字 A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
- RPG IV では、小文字の記号名が受け入れられますが、コンパイル時にそれらは大文字に変換されます。
- 数字 0 1 2 3 4 5 6 7 8 9
- 文字 + - * , . ' &/$ # : @ _ > < = ( ) %
- ブランク文字

注 : $、#、および @ は、コード・ページによっては 異なる記号として現れる場合があります。 詳しくは、IBM i Information Center のグローバリゼーションに関するトピックを参照してください。

記号名

記号名は、プログラムまたはプロシージャー中の特定のエンティティー を固有に識別する名前です。 RPG IV 言語では、記号名は次の目的で使用されます。

- 配列 (74 ページの『配列名』を参照)
- 条件付きコンパイル名 (74 ページの『条件付きコンパイル名』を参照)
- データ構造 (74 ページの『データ構造名』を参照)
- 例外出力レコード (74 ページの『EXCEPT 名』を参照)
- フィールド (74 ページの『フィールド名』を参照)
- キー・フィールド・リスト (74 ページの『KLIST 名』を参照)
- ラベル (74 ページの『ラベル』を参照)
- 名前付き定数 (207 ページの『名前付き定数』を参照)
- パラメーター・リスト (75 ページの『PLIST 名』を参照)
- プロトタイプ名 (75 ページの『プロトタイプ名』を参照)
- レコード名 (75 ページの『レコード名』を参照)
- サブルーチン (75 ページの『サブルーチン名』を参照)
- テーブル (75 ページの『テーブル名』を参照)

以下の規則は、各記号名の説明中に特に注意がある場合を除き、すべての記号名に適用されます。

- 名前の最初の文字は英字でなければなりません。 これには、$、#、および @ が含まれます。
- 残りの文字は英字または数値でなければなりません。 これには下線 (_) が含まれます。

- 名前は、名前を浮動させることができる フィールド (定義仕様書、キーワード・フィールド、および 拡張演算項目 2 フィールド) の場合を除き、仕様書の 記入項目で左寄せしなければなりません。
- 記号名に RPG IV 予約語を指定することはできません。
- 記号名は 1 から 4096 文字とすることができます。 実際上の限界値は、その名前を定義するために使用する記入項目のサイズ によって決まります。 15 文字までの名前は、定義仕様書またはプロシージャー仕様書の 名前記入項目に指定することができます。 15 文字を超える名前の場合は、継続の指定を行います。 詳しくは、「305 ページの『仕様書』」を参照してください。
- 記号名は、それが定義されたプロシージャーの中で固有のものでなければ なりません。

配列名

配列名には、次の補足規則が適用されます。

- 独立フィールドの配列名を TAB という文字で始めることはできません。 配列名がプロトタイプ・パラメーターまたは DIM キーワードで定義されたデータ構造である場合には、TAB という文字で始めることができます。

条件付きコンパイル名

条件付きコンパイルに使用する記号名は、他の記号名と関連していません。 例えば、MYFILE というファイルを定義する場合、後で /DEFINE を 使用して条件名 MYFILE を定義し、さらに /UNDEFINE を使用して条件名 MYFILE を除去します。 これは、MYFILE という名前のファイルには 影響しません。

条件付きコンパイル名の長さは最大 50 文字までとする ことができます。

データ構造名

データ構造は記憶域中の区域であって、文字フィールドと見なされます。

EXCEPT 名

EXCEPT 名は、例外出力レコードに割り当てられる記号名です。 EXCEPT 名には、次の補足規則が適用されます。

- 2 つ以上の出力レコードに同じ EXCEPT 名を割り当てることができます。

フィールド名

フィールド名には、次の補足規則が適用されます。

- フィールド名は、その名前を使用する各定義が同じデータ・タイプ、同じ 長さ、および小数点以下の桁数が同じである場合には、複数回定義することが できます。 同じ名前を使用するすべての定義は、単一のフィールド (すなわち、記憶域中 の同一区域) を参照することになります。 しかし、定義仕様書では 1 回しか定義することができません。
- 1 つのフィールドは、修飾されているデータ構造 (QUALIFIED または LIKEDS で定義されている) の場合を除き、データ構造サブフィールドとしては 1 回しか定義することができません。 この場合にサブフィールドを使用するには、そのサブフィールドを修飾する必要があります (*dsname.subfieldname* の形式で指定します)。
- サブフィールド名は、*ENTRY PLIST パラメーターで結果のフィールドと して指定することはできません。

KLIST 名

KLIST 名は、キー・フィールドのリストに割り当てられる記号名です。

ラベル

ラベルは、プログラム内の特定の場所 (例えば、TAG または ENDSR 命令 に割り当てられる名前) を識別する記号名です。

名前付き定数

名前付き定数は、定数に割り当てられる記号名です。

PLIST 名

PLIST 名は、パラメーターのリストに割り当てられる記号名です。

プロトタイプ名

プロトタイプ名は、プロトタイプ定義に割り当てられる記号名です。 この名前は、プロトタイプ・プロシージャーまたはプログラムを呼び出す時に 使用しなければなりません。 プロトタイプは、明示的に指定することも、あるいは、呼び出しと同じモジュールにプロシージャーが定義されている場合には、 プロシージャー・インターフェースからコンパイラーが暗黙的に生成することもできます。

レコード名

レコード名は､外部記述ファイルのレコード様式に割り当てられる記号名で す。 RPG IV プログラム中のレコード名には、次の補足規則が適用されます。

- ファイル仕様書でキーワード QUALIFIED または LIKEFILE が指定されているためにファイルが修飾されている場合、レコード名は FILENAME.FMTNAME という形式の修飾名で指定されます。 このレコード名は、そのファイルの他のレコード名とは異なる、固有のものでなければなりません。
- ファイルが修飾されていない場合、レコード名は FMTNAME という修飾なしの形式で指定されます。 ファイルがグローバル・ファイルである場合、レコード名は他のグローバル名とは異なる、固有のものでなければなりません。 ファイルがサブプロシージャーのローカル・ファイルである場合、レコード名は他のローカル名とは異なる、固有のものでなければなりません。

  注 : RPG プログラム内で使用されている名前とレコード名が競合する状況の処理方法について詳しくは、386 ページの『RENAME(外部形式:内部形式)』 を参照してください。

サブルーチン名

この名前は、BEGSR (サブルーチン開始) 命令の演算項目 1 で定義されます。

テーブル名

テーブル名には、次の補足規則が適用されます。

- テーブル名には 3 から 10 文字を使用することができます。
- テーブル名は TAB という文字で始めなければなりません。
- サブプロシージャーでテーブル名を定義することはできません。

特別な機能を持つ RPG IV の用語/予約語

次の RPG IV 予約語は、プログラムの中で特別な機能を持ちます。

- 次の予約語によって、プログラムで使用するジョブ日付またはその部分を アクセスすることができます。

  UDATE
  *DATE
  UMONTH
  *MONTH
  UYEAR
  *YEAR
  UDAY
  *DAY
- 次の予約語は、 報告書のページの番号付け、レコードの順序番号付け、ま たは出力フィールドの順序番号付けに使用することができます。

ページ
PAGE1-PAGE7

- 形象定数は、長さを指定しないで指定できるリテラルを 暗黙に指定します。

  *BLANK/*BLANKS
  *ZERO/*ZEROS
  *HIVAL
  *LOVAL
  *NULL
  *ON
  *OFF
  *ALLX'x1..'
  *ALLG'oK1K2i'
  *ALL'X..'

- 次の予約語は、データベース・ファイルの位置指定に使用されます。*START はファイルの始めに位置指定し、*END はファイルの終わりに位置指定します。

  *END
  *START

- 次の予約語によって、RPG IV 標識をデータとして参照することができます。

  *IN
  *INxx

- 次の特殊語は日付および時刻に使用されます。

  *CDMY
  *CMDY
  *CYMD
  *DMY
  *EUR
  *HMS
  *ISO
  *JIS
  *JOB
  *JOBRUN
  *JUL
  *LONGJUL
  *MDY
  *SYS
  *USA
  *YMD

- 次の特殊語は変換に使用されます。

  *ALTSEQ
  *EQUATE
  *FILE
  *FTRANS

- *PLACE によって、出力レコードにフィールドを繰り返して 入れることができます。 (詳しくは、525 ページの『*PLACE』を参照してください。)
- *ALL によって、外部記述ファイル用に定義されたすべてのフィールドを 出力時に書き出すことができます。 (*ALL について詳しくは、209 ページの『表意定数に関する規則』を参照してください)
- 次の特殊語は式の中で使用されます。
  - AND

  - NOT
  - または

    注 : NOT は式の中でしか使用することができません。 ソース内のどこでも名前として使用することはできません。
- 次の特殊語はパラメーターの受け渡しで使用されます。

  *NOPASS
  *OMIT
  *RIGHTADJ
  *STRING
  *TRIM
  *VARSIZE
- 以下の特殊語は、XML-SAX 命令コードのイベント処理プロシージャーにおけるイベント・パラメーターの解釈に役立ちます。

  XML_ATTR_UCS2_REF
  XML_ATTR_NAME
  XML_ATTR_PREDEF_REF
  XML_ATTR_CHARS
  XML_CHARS
  XML_COMMENT
  XML_UCS2_REF
  XML_PREDEF_REF
  XML_DOCTYPE_DECL
  XML_ENCODING_DECL
  XML_END_CDATA
  XML_END_DOCUMENT
  XML_END_ELEMENT
  XML_END_PREFIX_MAPPING
  XML_EXCEPTION
  XML_PI_TARGET
  XML_PI_DATA
  XML_STANDALONE_DECL
  XML_START_CDATA
  XML_START_DOCUMENT
  XML_START_ELEMENT
  XML_START_PREFIX_MAPPING
  XML_UNKNOWN_ATTR_REF
  XML_UNKNOWN_REF
  XML_VERSION_INFO
  XML_END_ATTR

ユーザー日付の特殊語

ユーザー日付の特殊語 (UDATE、*DATE、UMONTH、*MONTH、UDAY、*DAY、 UYEAR、*YEAR) では、 プログラマーが実行時にプログラムに日付を与えることができます。 ユーザー日付の特殊語では、ジョブ記述に指定されたジョブ日付がアクセスされます。 ユーザー日付は出力時に書き出すことができます。 UDATE および *DATE は、 Y 編集コードを制御仕様書に指定された形式で使用して書き出すことができます。

(ジョブ日付の説明については、「実行管理」マニュアルを参照してください。)

ユーザー日付に関する規則

ユーザー日付の使用にあたっては、以下の規則を忘れないでください。

- UDATE が出力仕様の 30 から 43 桁目に指定された場合には、6 文字の数値日付フィールドが印刷されます。同様に *DATE が指定された場合には、8 文字 (年部分が 4 桁) の数値日付フィー ルドが印刷されます。 これらの特殊語は、次の 3 つの異なる日付の形式で使用することができます。

  月/日/年
  年/月/日
  日/月/年

  制御仕様書の DATEDIT キーワードを使用して、UDATE および *DATE の形式を指定してください。

DATEDIT	UDATE の形式	*DATE の形式
*MDY	*MDY	*USA (mmddyyyy)
*DMY	*DMY	*EUR (ddmmyyyy)
*YMD	*YMD	*ISO (yyyymmdd)

  DATEDIT キーワードも Y 編集コードの形式を制御することに注意してください。

  このキーワードが指定されていない場合には、 デフォルトの値は *MDY になります。
- 対話式ジョブまたはバッチ・プログラムの場合に、ユーザー日付の特殊語 は、システムでプログラムの実行が開始された時のジョブ日付の値に設定されます。 プログラムが深夜を過ぎて実行されている場合、あるいはジョブ日付が変更 された場合であっても、プログラムの処理中にユーザー日付の特殊値は更新され ません。 プログラムの実行中に時刻および日付を入手するためには、TIME 命令コード を使用してください。
- UMONTH、*MONTH、UDAY、*DAY、および UYEAR が出力仕様の 30 から 43 桁 目に指定された場合には、2 桁の数値日付フィールドが印刷されます。4 桁の数値日付フィールドを印刷するには、*YEAR を使用することができます。 月だけを印刷するには UMONTH または *MONTH、日だけを印刷するには UDAY また は *DAY、および年だけを印刷するには UYEAR または *YEAR を使用します。
- 出力仕様の 44 桁目に Y 編集コードを指定した場合には、UDATE および *DATE を書き出す時に編集することができます。 制御仕様書に関する 330 ページの『DATEDIT(形式 {区切り記号 })』 キーワードによって、例えば、12/31/88、31.12.88.、12/31/1988 のように形式および挿入される区切り文字が決まります。
- UMONTH、*MONTH、UDAY、*DAY、UYEAR、および *YEAR は、 出力仕様の 44 桁目の Y 編集コードでは編集できません。
- ユーザー日付フィールドを変更することはできません。 これはユーザー日付フィールドについて次の使用ができないことを意味します。
  - 演算の結果のフィールドの中で
  - PARM 命令の演算項目 1 として
  - LOOKUP 命令の演算項目 2 の指標として
  - 出力仕様の「後で消去」と一緒に
  - 入力フィールドとして
- ユーザー日付の特殊語は、数値フィールドを使用する命令コードの 演算仕様書の演算項目 1 または演算項目 2 に使用できます。
- ユーザー日付フィールドは日付データ・タイプ・フィールドではなく、 数値フィールドです。

PAGE、PAGE1 から PAGE7

PAGE は報告書のページに番号を付けるため、ファイル中の出力レコードに 通し番号を付けるため、あるいは出力フィールドに順序番号を付けるために 使用されます。 改ページは行われません。

異なるタイプの出力ページに番号を付けるため、あるいは異なる印刷装置ファイ ルのページに番号を付けるために、使用可能な 8 つの PAGE フィールド (PAGE、PAGE1、PAGE2、PAGE3、PAGE4、PAGE5、PAGE6、および PAGE7) が必要になること があります。

出力仕様の 30 から 43 桁目あるいは入力または演算仕様書に PAGE フィールドを 指定することができます。

PAGE、PAGE1 から PAGE7 に関する規則

PAGE フィールドの使用にあたっては、以下の規則を忘れないでください。

- PAGE フィールドを他の個所で定義しないで出力仕様に指定した場合には、小数点以下の桁数のない 4 桁の数値フィールドと見なされます。
- 他で指定されない限り、ページ番号付けは 0001 から始まり、新しい ページごとに 1 が自動的に加算されます。
- 1 以外のページ番号から始めるためには、PAGE フィールドの値を 開始ページ番号より 1 だけ小さい値に設定します。 例えば、番号付けを 24 から始める場合には、PAGE フィールドに 23 を 記入してください。PAGE フィールドの長さに制限はありませんが、小数点以下の桁数があってはなりません (79 ページの図 1 を参照)。
- ページ番号付けは、ジョブのどこからでも改めて開始することができます。 以下の方式を使用して、PAGE フィールドをリセットすることができます。
  - 後で消去 (出力仕様の 45 桁目) を指定します。
  - 演算仕様書における命令の結果フィールドとして PAGE フィールドを 指定します。
  - 出力フィールド仕様書において出力標識を指定します (79 ページの図 2 を参照)。 出力標識がオンになっている場合、PAGE フィールドは 1 にリセットされます。 PAGE フィールドは常に書き出されるので、出力標識を使用して PAGE フィールドの印刷を制御することはできません。
  - 79 ページの図 1 に示されているとおりに PAGE フィールドを入力フィールドとして指定します。
- PAGE フィールドの先行ゼロは、編集コード、編集語、 またはデータ形式 (52 桁目の P/B/L/R) を指定した場合を除いて、PAGE フィールドを 印刷する時に自動的に消去されます (Z 編集コードと見なされます)。 編集およびデータ形 式によって、先行ゼロの消去は一時変更されます。 PAGE フィールドを入力仕様および演算仕様書で定義した場合には、 それが出力仕様ではフィールド名として取り扱われ、自動的なゼロ抑制とはなりません。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC..............................
I.....................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
IINPUT     PG  50   1  CP
I                                 2    5 0PAGE
```

図 *1.* ページ・レコード記述

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+..........................
O.............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat
O*  When indicator 15 is on, the PAGE field is set to zero and 1 is
O*  added before the field is printed.  When indicator 15 is off, 1
O*  is added to the contents of the PAGE field before it is printed.
OPRINT     H    L1                     01
O               15      PAGE          1     75
```

図 *2. PAGE* フィールドのゼロへのリセット

コンパイラー指示

コンパイラー指示ステートメント /TITLE、 /EJECT、/SPACE、/COPY、および /INCLUDE によって、コンパイル・リストの見出し情報を指定し、 コンパイル・リストのスペースを制御し、 コンパイル中に他のファイル・メンバーからレコードを挿入することができます。

条件付きコンパイル指示ステートメント /DEFINE、/UNDEFINE、/IF、/ELSEIF、/ELSE、/ENDIF、および /EOF によって、ソース・レコードを選択または省略することができます。

/SET 指示および /RESTORE 指示を使用すると、制御ステートメントに指定されているキーワードの一部を一時的に変更することができます。

/OVERLOAD 指示を使用すると、多重定義プロトタイプの呼び出しに使用する候補プロトタイプをコンパイラーが判別する方法についての追加情報を入手することができます。

コンパイラー指示ステートメントは、コンパイル時配列レコード、コンパイル時テーブル・レコード、変換レコード、および代替照合順序レコードよりも前になければなりません。

注 : コンパイラー指示ステートメント /FREE および /END-FREE は 使用されなくなりました。 これらを指定しても無視されます。80 ページの『/FREE... /END-FREE』を参照してください。

指示は、桁制限付きソースの場合は 7 桁目以降で、また完全自由形式ソースの場合は 1 桁目以降で書き始めることができます。

すべての指示は、単一の固定形式ステートメント内と、 任意のステートメント間に指定することができます。

単一の自由形式の演算ステートメント内に指示を指定することはできません。

/IF、/ELSEIF、/ELSE、および /ENDIF 指示は、単一の自由形式の制御ステートメント、 ファイル・ステートメント、定義ステートメント、またはプロシージャー・ステートメント内に指定できます。 他の指示をこれらのステートメント内に指定することはできません。

自由形式ステートメント内で、そのステートメント内では許可されない指示のように見えるものが行の先頭にある場合、 それは 1 個のスラッシュの後に名前が続いているものであると解釈されます。 例えば、次のステートメントの「/TITLE」は、「TITLE」という名前の変数での除算であると解釈されます。

```
x = y
  /title + 5;
```

特殊指示 **FREE を記述できるのはソースの先頭行の 1 桁目のみです。 **FREE が指定されるときは、ソース・メンバー全体が自由形式でなければなりません。309 ページの『完全自由形式ステートメント』を参照してください。

/FREE... /END-FREE

前のリリースでは、/FREE コンパイラー指示によって、自由形式の演算仕様ブロックの始まりが指定されました。/END-FREE によってブロックの終了が指定されていました。

/FREE または /END-FREE 指示を コーディングした場合、指示は無視されますが、指示の構文はチェックされます。 指示の後の桁はブランクでなければなりません。 自由形式ステートメントについては、307 ページの『自由形式ステートメント』を参照してください。

/TITLE

コンパイラー指示 /TITLE は、コンパイル・リストの各ページ の最上部に表示される見出し情報 (機密保護分類またはタイトルなど) を 指定するために使用します。

指示に使用できる桁については、80 ページの『コンパイラー指示』を参照してください。

/TITLE の後には 1 個のブランクがなければなりません。 このブランクの後にあるデータが、リストの後続の各ページの最上部に 印刷されます。

/TITLE を自由形式ステートメントの内部にコーディングすることはできません。

1 つのプログラムに複数の /TITLE ステートメントを含めることができます。 各 /TITLE ステートメントは、別の /TITLE ステートメントが見付かる まで、コンパイル・リストの見出し情報を提供します。 /TITLE ステートメントは、コンパイル・リストの最初のページに情報を 印刷するために見付けられる最初の RPG 指定でなければなりません。 /TITLE ステートメントによって指定された情報は、コンパイラー見出し 情報に加えて印刷されます。

/TITLE ステートメントがあると、タイトルが印刷される前に次のページへの スキップが起こります。 /TITLE ステートメントはコンパイル・リストには印刷されません。

/EJECT

指示に使用できる桁については、80 ページの『コンパイラー指示』を参照してください。

/EJECT の後には少なくとも 2 個のブランクがなければなりません。 残りの桁には注記を入れることができます。

/EJECT を自由形式ステートメントの内部にコーディングすることはできません。

/EJECT は、後続の仕様がコンパイル・リストの新規ページで始まるように するために使用します。 スプール・ファイルが既に新しいページの最上部に位置している場合には、/EJECT による改ページは行われません。/EJECT はコンパイル・リストには印刷されません。

/SPACE

コンパイラー指示 /SPACE は、コンパイル・リストのソース・セク ション内の行送りを制御するために使用します。

指示に使用できる桁については、80 ページの『コンパイラー指示』を参照してください。

/SPACE の後には、ちょうど 1 個のブランク、コンパイラー・リストで何行間隔を空けるかを定義する 1 から 112 までの正整数値、および少なくとも 2 個のブランクが、この順序で含まれていなければなりません。残りの桁には注記を入れることができます。

/SPACE を自由形式ステートメントの内部にコーディングすることはできません。

行数が 112 よりも大きい場合、/SPACE の値として 112 が 使用されます。 行数が現行ページの残りの行数より大きい場合、 後続の仕様は次ページの先頭から始まります。

/SPACE はコンパイル・リストには印刷されませんが、指定された 行送りによって置き換えられます。 /SPACE によって行われる行送りは、仕様の タイプが変わる場合にスキップされる 2 行に対する追加分です。

/SET

コンパイラー指示 /SET は、定義に対して新しいデフォルト値を 一時的に設定するために使用します。

1 つ以上のキーワードに対する /SET 指示の影響を 取り消すには、/RESTORE 指示を使用します。

/SET 指示と共に以下のキーワードを指定できます。

CCSID(*CHAR : ccsid)
CCSID キーワードも LIKE キーワードも指定せずに定義された英数字項目に対して、 および、CCSID(*EXACT) キーワードなしで定義された データ構造の外部記述英数字サブフィールドに対して、デフォルト CCSID を指定します。 325 ページの『CCSID(*CHAR : *JOBRUN | *JOBRUNMIX | *UTF8 | *HEX | 番号)』を参照してください。

CCSID(*GRAPH : ccsid)
CCSID キーワードなしで指定されたグラフィック項目のデフォルト CCSID を 指定します。 326 ページの『CCSID(*GRAPH : *JOBRUN | *SRC | *HEX | *IGNORE | 番号)』を参照してください。

CCSID(*UCS2 : ccsid)
CCSID キーワードなしで指定された UCS-2 項目のデフォルト CCSID を 指定します。 326 ページの『CCSID(*UCS2 : *UTF16 | 番号)』を参照してください。

DATFMT(形式)
日付形式なしで指定された (自由形式定義でパラメーターなしで DATE キーワードが指定されているか、 または、固定形式定義で DATFMT キーワードが指定されていない) 日付項目 のデフォルトの形式および区切り記号を指定します。 330 ページの『DATFMT(形式 {区切り記号 })』を参照してください。

TIMFMT(形式)
時刻形式なしで指定された (自由形式定義でパラメーターなしで TIME キーワードが指定されているか、 または、固定形式定義で TIMFMT キーワードが指定されていない) 時刻項目 のデフォルトの形式および区切り記号を指定します。 347 ページの『TIMFMT(形式 {区切り記号 })』を参照してください。

SET 指示をコピー・ファイル内に指定することによって、そのコピー・ファイルを含んでいるすべてのモジュールが、時刻形式と日付形式、および CCSID に確実に同じ値を使用するようにします。 コピー・ファイル内の /SET 指示で設定された値は、 /COPY または /INCLUDE 指示の前の値に暗黙的に戻されます。

コピー・ファイルを変更できない場合、/COPY または /INCLUDE 指示の前に /SET 指示を コーディングし、/COPY または /INCLUDE 指示の後に /RESTORE 指示を コーディングして /SET 指示の前に有効だった値にデフォルトを戻すことができます。

/SET および /RESTORE に関する規則

- /SET 指示をネストできます。
- /RESTORE 指示に指定されるキーワードは、前の /SET 指示に指定された キーワードと完全に一致する必要はありません。 1 つの /RESTORE 指示で、それより前にあるいずれかの /SET 指示で設定された値の一部またはすべてを 元に戻すことができます。

/SET および /RESTORE の例

1. 英数字項目およびグラフィック項目のデフォルト CCSID と日付項目のデフォルト日付形式が、制御仕様キーワードを使用して指定されています。 CCSID(*UCS2) は 13488 にデフォルト設定され、TIMFMT は *ISO にデフォルト設定されます。
2. フィールド *char1* は英数字です。 CCSID キーワードは 指定されていないため、CCSID は *UTF8 にデフォルト設定されます。
3. フィールド *graph1* はグラフィックです。 CCSID キーワード は指定されていないため、CCSID は 835 にデフォルト設定されます。
4. /SET 指示はデフォルト英数字 CCSID を 37 に設定し、 デフォルト UCS-2 CCSID を 1200 に設定しています。
5. フィールド *char2* は英数字です。 CCSID キーワード は指定されていないため、CCSID は 37 にデフォルト設定されます。
6. フィールド *char3* は LIKE キーワードを使用して定義されています。 CCSID(*DFT) が指定されていて、デフォルト CCSID を使用することを示しています。 英数字フィールドと同種であると定義されているため、 現行のデフォルト英数字 CCSID である 37 が使用されます。
7. /RESTORE 指示は、CCSID(*CHAR) を、前の値である *UTF8 に 戻します。 この /RESTORE 指示には CCSID(*UCS2) は指定されていないため、 前の /SET 指示で設定された値 1200 は有効なままです。
8. フィールド *ucs1* は UCS-2 です。 CCSID キーワード は指定されていないため、CCSID は 1200 にデフォルト設定されます。これは、現行のデフォルト UCS-2 CCSID です。
9. ソース・メンバー *copyfile* を組み込むために /COPY が使用されています。 この時点で、デフォルトは、CCSID(*CHAR:*UTF8)、CCSID(*GRAPH:835)、 CCSID(*UCS2:1200)、DATFMT(*YMD)、TIMFMT(*ISO) です。
10. コピー・ファイルは /SET 指示で始まっていて、 このコピー・ファイル内で使用されるデータ・タイプに対してデフォルトが設定されています。 この時点から、 デフォルトは、CCSID(*CHAR:*UTF8)、CCSID(*GRAPH:835)、CCSID(*UCS2:13488)、DATFMT(*ISO)、 TIMFMT(*HMS) になります。
11. フィールド *time1* は時刻フィールドです。 TIMFMT キーワード は指定されていないため、時刻形式は、コピー・ファイル内の /SET 指示で設定されたデフォルトである *HMS に デフォルト設定されます。
12. フィールド *char4* は英数字です。 CCSID キーワードは 指定されていないため、CCSID は *UTF8 にデフォルト設定されます。

13. コピー・ファイルの最後で、このコピー・ファイル内の /SET 指示で設定された すべての値は暗黙的に元に戻されます。
14. この時点で、デフォルトは、/COPY 指示の前と同じ 値、つまり、CCSID(*CHAR:*UTF8)、CCSID(*GRAPH:835)、CCSID(*UCS2:1200)、 DATFMT(*YMD)、TIMFMT(*ISO) になります。

```
CTL-OPT CCSID(*CHAR : *UTF8) CCSID(*GRAPH : 835) 1
        DATFMT(*YMD); 1
DCL-S char1 char(10); 2
DCL-S graph1 graph(10); 3
/SET CCSID(*CHAR : 37) CCSID(*UCS2:1200) 4
DCL-S char2 char(10); 5
DCL-S char3 LIKE(char1) CCSID(*DFT); 6
/RESTORE CCSID(*CHAR) 7
DCL-S ucs1 UCS2(10); 8
/COPY copyfile 9
14
```

図 *3.* メイン・ソース・ファイル

```
/SET CCSID(*UCS2 : 13488) DATFMT(*ISO) TIMFMT(*HMS) 10
DCL-S time1 time; 11
DCL-S char4 char(10); 12
13
```

図 *4. /COPY* ファイル *copyfile*

/RESTORE

コンパイラー指示 /RESTORE は、同じソース・メンバー内で 前に /SET によって設定された値を元に戻すために使用されます。

CCSID(*CHAR)
CCSID(*CHAR) キーワードを含む、前の /SET 指示の影響をなくします。

CCSID(*GRAPH)
CCSID(*GRAPH) キーワードを含む、前の /SET 指示の影響をなくします。

CCSID(*UCS2)
CCSID(*UCS2) キーワードを含む、前の /SET 指示の影響をなくします。

DATFMT
DATFMT キーワードを含む、前の /SET 指示の影響をなくします。

TIMFMT
TIMFMT キーワードを含む、前の /SET 指示の影響をなくします。

注 : コピー・ファイル内の /SET 指示で設定された値は、/COPY または /INCLUDE 指示の前の値に暗黙的に戻されます。

/SET および /RESTORE の規則および例については、81 ページの『/SET』を参照してください。

/OVERLOAD DETAIL | NODETAIL

/OVERLOAD 指示のパラメーターには、DETAIL または NODETAIL を指定できます。

指示の後のステートメントに、多重定義プロトタイプの呼び出しに関する詳細情報を表示する場合は、DETAIL を指定します。

指示の後のステートメント内のリストに詳細情報を表示しない場合は、NODETAIL を指定します。

/OVERLOAD 指示の前に指定された呼び出しについては、リストに詳細情報は表示されません。

注 : /OVERLOAD 指示は、同じパラメーターまたは別のパラメーターを使用して繰り返し指定することができます。 特定の演算ステートメントの場合は、最新の /OVERLOAD 指示が有効になります。

次の例では、ステートメント 13、15、16、および 23 の候補プロトタイプへの呼び出しについて、「多重定義プロトタイプ」セクションに詳細情報が示されます。

```
   ...
 7 DCL-PR ov1 OVERLOAD(p1 : p2 : p3);
 8
 9 ov1 ('a');
10 /OVERLOAD NODETAIL
11 ov1 ('b');
12 /OVERLOAD DETAIL
13 ov1 ('c');
14 /OVERLOAD DETAIL
15 ov1 ('d');
16 ov1 ('e');
17 /OVERLOAD NODETAIL
18 ov1 ('f');
19
20 dcl-proc p1;
21    ov1 ('g');
22    /OVERLOAD DETAIL
23    ov1 ('h');
   ...
```

多重定義プロトタイプの情報、および /OVERLOAD DETAIL を指定する場合のリストの「多重定義プロトタイプ」セクションで提供される詳細の例については、472 ページの『コンパイラー・リスト内の「多重定義プロトタイプ」セクション』を参照してください。

/COPY または /INCLUDE

/COPY 指示および /INCLUDE 指示は同じ目的と同じ構文を持ちますが、SQL プリコンパイラーによる処理が異なってきます。 ユーザーのプログラムに組み込み SQL がない場合は、ユーザーはどちらの指示を使用するか自由に選択できます。 ユーザーのプログラムに組み込み SQL がある場合に使用すべき指示については、86 ページの『組み込み SQL のあるソース・ファイルへの /COPY、/INCLUDE の使用』を参照してください。

/COPY コンパイラー指示および /INCLUDE コンパイラー指示を使用すると、 これらのコンパイラー指示が出された時点で、 他のファイルからのレコードがコンパイル中のファイルに挿入されます。 挿入されるファイルには、/COPY および /INCLUDE を含む有効な仕様であれば、COPYNEST キーワードによって指定されたネストの 最大の深さ (指定されていない場合は 32) まで、いかなる仕様が含まれていても差し支えありません。

/COPY ファイルおよび /INCLUDE ファイルは物理ファイルでも IFS ファイルでもかまいません。 物理ファイルを指定するには、 /COPY ステートメントおよび /INCLUDE ステートメントを次の方法でコーディングします。

- /COPY または /INCLUDE の後には、正確に 1 つのスペースを入れ、その後にファイル名またはパスが必要です。
- 物理ファイルを指定する場合には、ライブラリー、ファイル、およびメンバー名に次の形式のいずれかを使用できます。

```
libraryname/filename,membername
filename,membername
membername
```

  - メンバー名の指定が必要です。
  - ファイル名が指定されていない場合には、QRPGLESRC と見なされます。

  - ライブラリー名が指定されていない場合には、ライブラリー・リスト からファイルが検索されます。 メンバーが見付かるか検索が完了するまで、ライブラリー・リスト内の 指定されたソース・ファイル のすべてが検索されます。
  - ライブラリー名を指定する場合には、ファイル名も指定しなければ なりません。
- IFS (Integrated File System) ファイルを指定する場合には、パスは絶対パス (/ で開始) でも相対パスでも かまいません。
  - パスは、単一引用符または二重引用符で囲むことができます。 パスにブランクが含まれている場合は、引用符で囲む必要があります。
  - パスが接尾部 (".txt" など) で終了しない場合、コンパイラーはファイルが名前付きであるものとして検索します。 接尾部が ".rpgle" または ".rpgleinc" であるファイルについても同様です。
  - IFS /COPY ファイルの使用については、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」を参照してください。
- オプションで、少なくとも 1 つのスペースとコメント。

ヒント:

アプリケーションの維持を容易にするために、エクスポート・プロシージャーのプロトタイプを 別のソース・メンバーに入れることができます。 これを行う場合は必ず、そのメンバーのための COPY 指示または INCLUDE 指示を、エクスポート・プロシージャーが入っているモジュールとエクスポート・プロシージャーに対する呼び出しが入っているすべてのモジュールの両方に入れてください。

85 ページの図 5 は、/COPY 指示ステートメントおよび /INCLUDE 指示ステートメントの例をいくつか示しています。

```
C/COPY MBR1 1
I/INCLUDE SRCFIL,MBR2 2
O/COPY SRCLIB/SRCFIL,MBR3 3
O/INCLUDE "SRCLIB!"/"SRC>3","MBR¬3" 4
O/COPY /dir1/dir2/file.rpg 5
O/COPY /dir1/dir2/file 6
O/COPY dir1/dir2/file.rpg 7
O/COPY "ifs file containing blanks"  8
O/COPY 'ifs file containing blanks'  8
```

図 *5. /COPY* コンパイラー指示ステートメントおよび */INCLUDE* コンパイラー指示ステートメントの例

1

ソース・ファイル QRPGLESRC のメンバー MBR1 からコピーします。 ファイル QRPGLESRC を検索するために現行ライブラリー・リストが使用されます。 ファイルがライブラリー・リストで見付からない場合、 検索は IFS に進み、ファイル MBR1、MBR1.rpgle、または MBR1.rpgleinc をインクルード・サーチ・パスから検索します。 IFS ソース・ファイルの使用について詳しくは、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」を参照してください。

2

ファイル SRCFIL のメンバー MBR2 からコピーします。 ファイル SRCFIL を検索するために現行ライブラリー・リストが使用 されます。 ファイル名とメンバー名を区切るためにコンマが使用されることに注意 してください。 ファイルがライブラリー・リストで見付からない場合、検索は IFS に進み、ファイル *SRCFIL*, *MBR* 1 (.rpgle または .rpgleinc 接尾部が付いている可能性があります) をインクルード・サーチ・ パスから検索します。

3
ライブラリー SRCLIB にある SRCFIL ファイルのメンバー MBR3 またはディレクトリー SRCLIB にある IFS ファイル *SRCFIL, MBR3* からコピーしま す。

4
ライブラリー「SRCLIB!」にあるファイル「SRC>3」のメンバー「MBR¬3」からコピーします。

5
ディレクトリー /dir1/dir2 にある IFS ファイル file.rpg からコピーします。

6
ディレクトリー /dir1/dir2 にあるファイル、file.rpgleinc、または file.rpgle からコピーします。

7
IFS サーチ・パスを使用して、ディレクトリー dir1/dir2 を検索し、ディレクトリー dir1/dir2 にある IFS ファイル file.rpg からコピーします。

8
名前にブランクがあるファイルからコピーします。

コンパイル時の /COPY または /INCLUDE の結果

コンパイル時には、指定されたファイル・メンバーが /COPY ステートメントまたは /INCLUDE ステートメントが出てきた場所でプログラムに組み合わされます。 すべてのメンバーは、COPY メンバー・テーブルに組み込まれます。

ネストされた /COPY または /INCLUDE

/COPY 指示および /INCLUDE 指示はネストすることができます。 /COPY メンバーまたは /INCLUDE メンバー に 1 つ (または複数) の /COPY 指示または /INCLUDE 指示を含めることができます (その指示には、さらに別の /COPY 指示または /INCLUDE 指示などを含めることができます)。 ネストが可能な最大の深さは、COPYNEST 制御仕様書キーワードを 使用して設定することができます。 デフォルトの最大の深さは 32 です。

ヒント:

ネストされた /COPY ファイルまたは /INCLUDE ファイル同士が無限に組み込み合うことがないようにしてください。 また、その /COPY ファイルまたは /INCLUDE ファイルの最初にある条件付きコンパイル指示を使用して、ソース行が 2 度以上使用されることを防止してください。

複数回の組み込みを防止する方法の例について は、91 ページの図 6 を参照してください。

組み込み SQL のあるソース・ファイルへの /COPY、/INCLUDE の使用

/COPY 指示および /INCLUDE 指示は、SQL プリコンパイラーによる処理が異なる点以外は、同一です。

SQL プリコンパイラーが /COPY 指示および /INCLUDE 指示を処理する方法は、CRTSQLRPGI コマンド で指定された RPG プリプロセッサー・オプション・パラメーター (RPGPPOPT) によって異なります。 詳細については、トピック『組み込み SQL プログラミング (Embedded SQL Programming)』の『ILE RPG アプリケーションでの SQL ステートメントのコーディング』、またはトピック『CL』の CRTSQLRPGI コマンドの説明を参照してください。

条件付きコンパイル指示

条件付きコンパイル指示ステートメントによって、 ソース・コードのセクションを、コンパイルに組み込んだり、 コンパイルから除外したりすることができます。

- 定義条件指示 /DEFINE および /UNDEFINE を 使用して、現在定義されている条件のリストに条件名を追加したり、 そのリストから条件名を除去することができます。
- 条件式 DEFINED(条件名) および NOT DEFINED(条件名) は、 テスト条件 /IF グループ内で使用されます。
- テスト条件指示、/IF、/ELSEIF、/ELSE および /ENDIF は、 どのソース行がコンパイラーによって読み取られるかを制御します。

- /EOF 指示は、コンパイラーに対して、 現行ソース・メンバー内のソース行の残りを無視するように指示します。

条件の定義

定義条件指示 /DEFINE および /UNDEFINE を使用して、現在定義されている条件の リストに条件名を追加したり、そのリストから条件名を除去することができます。

/DEFINE

/DEFINE コンパイラー指示は、条件付きコンパイルの条件 を定義します。 条件名域内の記入項目は自由形式 (左寄せされない) です。

/DEFINE の後には 1 つ以上のスペースを置き、その後に条件名を同じ行に指定しなければなりません。 行の残りの部分はブランクでなければなりません。

指示に使用できる桁については、80 ページの『コンパイラー指示』を参照してください。

/DEFINE を自由形式ステートメントの内部に指定することはできません。

/DEFINE 指示は、現在定義されている条件のリストに条件名を追加します。 後続の /IF DEFINED(条件名) は真となります。 後続の /IF NOT DEFINED(条件名) は偽となります。

注 : コマンド・パラメーターの DEFINE は、CRTBNDRPG および CRTRPGMOD コマンドに最高 32 までの条件を事前定義する ために使用することができます。

/UNDEFINE

/UNDEFINE 指示は、条件がもやは定義されていないことを示す ために使用します。

/UNDEFINE の後には 1 つ以上のスペースを置き、 その後に条件名を同じ行に指定しなければなりません。 行の残りの部分はブランクでなければなりません。

指示に使用できる桁については、80 ページの『コンパイラー指示』を参照してください。

/UNDEFINE を自由形式ステートメントの内部に指定することはできません。

/UNDEFINE 指示は、現在定義されている条件のリストから 条件名を除去します。 後続の /IF DEFINED(条件名) は偽となります。 後続の /IF NOT DEFINED(条件名) は真となります。

注 : DEFINE パラメーター上に指定された条件は すべて、/IF および /ELSEIF 指示を処理するときに定義されるものと 見なされます。 これらの条件は、 /UNDEFINE 指示を使用して除去することができます。

事前定義条件

いくつかの条件は、ユーザーの代わりに RPG コンパイラーが定義します。 これらの条件は、/DEFINE または /UNDEFINE とともに使用することはできません。 これらは、/IF および /ELSEIF とともにのみ、使用できます。

環境に関連する条件

***ILERPG**

この条件は、ユーザーのプログラムが ILE RPG IV コンパイラー (本書で説明しているコンパイラー) によってコンパイル中である場合に定義されます。

```
     * This module is to be defined on different platforms.  With
     * the ILE RPG compiler, the BNDDIR keyword is used to
     * indicate where procedures can be found.  With a different
     * compiler, the BNDDIR keyword might not be valid.
     /IF DEFINED(*ILERPG)
    H BNDDIR('QC2LE')
     /ENDIF
```

RPG IV コンパイラーの他のバージョンでどの条件が使用可能であるかについては、 そのコンパイラーの解説書を参照してください。

使用されているコマンドに関連する条件

***CRTBNDRPG**
この条件は、ユーザーのプログラムが、プログラムを作成する CRTBNDRPG コマンドによってコンパイル中である場合に定義されます。

```
      /IF DEFINED(*CRTBNDRPG)
     H DFTACTGRP(*NO)
      /ENDIF
```

***CRTRPGMOD**
この条件は、ユーザーのプログラムが、モジュールを作成する CRTRPGMOD コマンドによってコンパイル中である場合に定義されます。

```
      * This code might appear in a generic Control specification
      * contained in a /COPY file.  The module that contains the
      * main procedure would define condition THIS_IS_MAIN before
      * coding the /COPY directive.

      * If the CRTRPGMOD command is not being used, or if
      * THIS_IS_MAIN is defined, the NOMAIN keyword will not
      * be used in this Control specification.

      /IF DEFINED(*CRTRPGMOD)
      /IF NOT DEFINED(THIS_IS_MAIN)
     H NOMAIN
      /ENDIF
      /ENDIF
```

ターゲット・リリースに関連する条件

***VxRxMx**
この条件は、ユーザーのプログラムが、この条件で指定されるリリース以降でコンパイルされようとしている場合に定義されます。 *V4R4M0 (バージョン 4 リリース 4 モディフィケーション 0) から始まります。

この条件は、ユーザーが同一のプログラムを異なるターゲット・リリースで実行し、かつすべてのリリースにおいて使用可能ではない機能の利点を利用したい場合に使用します。 この条件に対するサポートは、適切な PTF がインストールされている *V4R4M0 システムから始まっています。

```
      /IF DEFINED(*V5R1M0)

      * Specify code that is valid in V5R1M0 and subsequent releases

     I/INCLUDE SRCFIL,MBR2

      /ELSE
      * Specify code that is available in V4R4M0

     I/COPY SRCFIL,MBR2

      /ENDIF
```

制御仕様書キーワードに関連する条件

***THREAD_CONCURRENT**
この条件は、制御ステートメントに THREAD(*CONCURRENT) キーワードが指定された場合に定義されます。

次の例では、制御ステートメントに THREAD キーワードが指定されていないモジュールから変数 GLOBAL_DATA がエクスポートされます。 この変数の記憶域タイプは all-thread 静的です。 制御ステートメントに THREAD(*CONCURRENT) が指定されているソースに、このファイルがコピーされると、この変数は、エクスポートされた変数とは互換性のない記憶域タイプ (スレッド・ローカル静的) となるようにデフォルト設定されます。 この変数を STATIC(*ALLTHREAD) キーワードで定義すると、この変数はエクスポートされた変数との互換性を持つようになります。

```
   DCL-S GLOBAL_DATA CHAR(100)
            IMPORT
```

```
            /IF DEFINED(*THREAD_CONCURRENT)
                STATIC(*ALLTHREAD)
            /ENDIF
          ;
```

***THREAD_SERIALIZE**
この条件は、制御ステートメントに THREAD(*SERIALIZE) キーワードが指定された場合に定義されます。

条件式

条件式の形式は次のいずれかになります。

- DEFINED(条件名)
- NOT DEFINED(条件名)

条件式は自由形式ですが、複数行にすることはできません。

条件のテスト

条件のテストは /IF グループを使用して行います。この グループの構成は、前から順に /IF 指示、0 または 1 つ以上 の /ELSEIF 指示、オプションの /ELSE 指示となり、最後に /ENDIF 指示が 続きます。

/IF グループの指示と指示の間では、コンパイル時データを 除くソース行はすべて有効です。 これには、ネストされた /IF グループも含まれます。

注 : /IF グループのネスト・レベルには、実際上の限界は ありません。

/IF 条件式

/IF コンパイラー指示は、条件付きコンパイルについて条件式を テストするために使用します。

/IF の後には 1 つ以上のスペースを置き、その後に条件式 を同じ行に指定しなければなりません。 条件式の後の行の残りの部分はブランクでなければなりません。

指示に使用できる桁については、80 ページの『コンパイラー指示』を参照してください。

/IF は、自由形式演算ステートメント以外の任意の自由形式ステートメント 内に指定できます。309 ページの『自由形式ステートメント内の条件付き指示』を参照してください。

その条件式が真である場合、/IF 指示の後に続くソース行は、 コンパイラーによって選択され、読み取られます。 それ以外の場合、同じ /IF グループ中で次の /ELSEIF、/ELSE または /ENDIF が見付かるまで、行は除外されます。

/ELSEIF 条件式

/ELSEIF コンパイラー指示は、/IF または /ELSEIF グループ内の 条件式をテストするために使用します。

/ELSEIF の後には 1 つ以上のスペースを置き、その後に条件式 を同じ行に指定しなければなりません。 条件式の後の行の残りの部分はブランクでなければなりません。

指示に使用できる桁については、80 ページの『コンパイラー指示』を参照してください。

/ELSEIF は、自由形式演算ステートメント以外の任意の自由形式ステートメント 内に指定できます。309 ページの『自由形式ステートメント内の条件付き指示』を参照してください。

直前の /IF または /ELSEIF が該当しなかった場合、その条件式は 真となり、/ELSEIF 指示の後のソース行は選択され、読み取られます。 それ以外の 場合、同じ /IF グループ内で次に /ELSEIF、/ELSE または /ENDIF が見付かるまで、行は除外されます。

/ELSE

/ELSE コンパイラー指示は、/IF または /ELSEIF テストが 失敗した後に読み取られるソース行を無条件で選択するために 使用します。

指示に使用できる桁については、80 ページの『コンパイラー指示』を参照してください。

/ELSE は、自由形式演算ステートメント以外の任意の自由形式ステートメント の内部に指定できます。 309 ページの『自由形式ステートメント内の条件付き指示』を参照してください。

/ELSE が含まれている行の残りの部分はブランクでなければなりません。

直前の /IF または /ELSEIF が該当しなかった場合、次 の /ENDIF が見付かるまでソース行は選択されます。

直前の /IF または /ELSEIF が該当した場合、次 の /ENDIF が見付かるまでソース行は除外されます。

/ENDIF

/ENDIF コンパイラー指示は、最新の /IF、/ELSEIF また は /ELSE グループを終了するために使用します。

/ENDIF は、自由形式演算ステートメント以外の任意の自由形式ステートメント の内部に指定できます。 309 ページの『自由形式ステートメント内の条件付き指示』を参照してください。

/ENDIF が含まれている行の残りの部分はブランクでなければなりません。

/ENDIF 指示の後は、突き合わせ /IF 指示が選択された行であった 場合、行は無条件に選択されます。 それ以外の場合、/IF グループは選択されていないので、 行は引き続き選択されません。

条件をテストする場合の規則

- /ELSEIF および /ELSE は /IF グループの外部では無効です。
- /IF グループに含められる /ELSE 指示は 1 つです。 /ELSEIF 指示は、/ELSE 指示の 後に続けることはできません。
- /ENDIF は、/IF、/ELSEIF または /ELSE グループの外部では 無効です。
- /IF はすべて、後続の /ENDIF によって突き合わせをする必要があります。
- 1 つの /IF グループに関連したすべての指示は、同じソース・ファイル の中にある必要があります。 1 つのファイルの中に /IF があり、別のファイルの中に 突き合わせ /ENDIF が あるということは、その 2 つ目のファイルがネストされた /COPY の 中にある場合でも無効です。 ただし、/IF グループ全体はネストされた /COPY の中に入れる ことができます。

/EOF 指示

/EOF は、コンパイラーに対して、 現行ソース・メンバー内のソース行の残りを無視するように指示します。

/EOF

/EOF コンパイラー指示は、現行ソース・ファイル の「ファイルの終わり」に達していることをコンパイラーが考慮する 必要があることを示すために使用します。

指示に使用できる桁については、80 ページの『コンパイラー指示』を参照してください。

/EOF が含まれている行の残りの部分はブランクでなければなりません。

/EOF は、現行ソース・メンバーの読み取り中に活動状態になった 活動 /IF グループをすべて終了します。 /EOF が /COPY ファイル内に ある場合、/COPY 指示が読み取られた ときに活動状態だった条件は、そのまま活動状態となります。

注 : 除外された行がリスト上に印刷されている場合、/EOF の後でも ソース行の読み取りおよびリストは引き続き行われますが、それらの行の内容は、コンパイラーによって完全に無視されます。 /EOF の 後には、診断メッセージが出されることはありません。

ヒント :

/EOF 指示を使用すると、/COPY メンバー全体が 1 度しか使用 されないで、複数回コピーされる可能性がある場合、コンパイル時パフォーマンスが向上します。 (これは、除外された行が印刷されている場合には該当しません。)

以下は、/EOF 指示の例です。

```
*----------------------------------------------------------------
* Main source file
*----------------------------------------------------------------
....
/IF DEFINED(READ_XYZ)                          1
/COPY XYZ
/ENDIF                                         2
....
*----------------------------------------------------------------
* /COPY file XYZ
*----------------------------------------------------------------
/IF DEFINED(XYZ_COPIED)                        3
/EOF
/ELSE
/DEFINE XYZ_COPIED
 D .....
/ENDIF
```

図 *6. /EOF* 指示

最初にこの /COPY メンバーが読み取られるとき は、XYZ_COPIED は定義されないので、/EOF は考慮されません。

2 回目にこのメンバーが読み取られるときには、XYZ_COPIED は 定義され、/EOF は処理されます。 /IF DEFINED(XYZ_COPIED) (3) は終了したと見なされ、そのファイルは閉じられます。 ただし、メイン・ソース・メンバーの /IF DEFINED(READ_XYZ) (1) は、それ自身の /ENDIF (2) に達するまでは、活動状態のままです。

RPG プリプロセッサーによる指示の処理

RPG プリプロセッサーによるコンパイラー指示の処理は、 コンパイル・コマンドの PPGENOPT パラメーターで指定されたオプションによって異なります。 このプリプロセッサーは、特定の指示に対応して、次のいずれかのアクションを行う可能性があります。

- 生成されたソース・ファイルで指示が保持される (以下の表では「保持」で表されています)
- 生成されたソース・ファイルから指示が除去される (以下の表では「除去」で表されています)
- 生成されたソース・ファイルで、指示がコメントとして保持される (以下の表では「コメント」で表されています)

一般に、オプション *RMVCOMMENT を指定した場合には、 正常にコンパイルを行うために必要な指示のみが、生成されるソース・ファイルに出力されます。 オプション NORMVCOMMENT を指定した場合には、生成されるソース・ファイルを正常にコンパイル するのに必要ではない指示はコメントに変換されます。

次の表は、さまざまな PPGENOPT パラメーター値を指定した場合に、それぞれの指示がプリプロセッサーによってどのように処理されるのかを要約したものです。

指示	*RMVCOMMENT		*NORMVCOMMENT	
	*EXPINCLUDE	*NOEXPINCLUDE	*EXPINCLUDE	*NOEXPINCLUDE
/COPY	除去	除去	comment	comment
/DEFINE	除去	保持	comment	保持
/EJECT	除去	除去	保持	保持
/ELSE	除去	除去	comment	comment
/ELSEIF	除去	除去	comment	comment
/END-EXEC	保持	保持	保持	保持
/END-FREE	保持	保持	保持	保持
/ENDIF	除去	除去	comment	comment

指示	*RMVCOMMENT		*NORMVCOMMENT	
	*EXPINCLUDE	*NOEXPINCLUDE	*EXPINCLUDE	*NOEXPINCLUDE
/EOF	除去	除去	comment	comment
/EXEC	保持	保持	保持	保持
/FREE	保持	保持	保持	保持
/IF	除去	除去	comment	comment
/INCLUDE	除去	保持	comment	保持
/RESTORE	保持	保持	保持	保持
/SET	保持	保持	保持	保持
/SPACE	除去	除去	保持	保持
/TITLE	除去	除去	保持	保持
/UNDEFINE	除去	保持	comment	保持

プロシージャーおよびプログラム論理サイクル

プロシージャーは、呼び出して実行することのできるステートメントの集合です。

RPG のプロシージャーには、通常のサブプロシージャー、リニア・メイン・サブプロシージャー、サイクル・メイン・サブプロシージャーの 3 種類があります。 存在するプロシージャーのタイプに応じて、RPG ソース・プログラムを 3 種類のモジュール (サイクル、Nomain、リニア・メイン) のいずれかにコンパイルできます。 これらは、制御仕様書で NOMAIN または MAIN のキーワードで示されます。

「サブプロシージャー」という用語は、通常のサブプロシージャーおよびリニア・メイン・プロシージャーを指すために使用されます。

RPG ソース・プログラムは、次のセクションに分類できます (それぞれプロシージャーを含んでいます)。

- **メイン・ソース・セクション:** ソース・プログラム内の最初の行から、最初のプロシージャー仕様書までのソース行。 サイクル・モジュールでは、このセクションにサイクル・メイン・プロシージャーを構成する演算仕様書 (標準形式または自由形式) が含まれている場合があります。 このセクションに演算仕様書がない場合でも、サイクル・メイン・プロシージャーが暗黙的に指定されます。 この種のプロシージャーには、それを特定するプロシージャーの始めの指定と「プロシージャーの終わり」の指定がありません。

  サイクル・モジュールは、サブプロシージャーなしでも設計可能であるため、個別のプロシージャー・セクションがありません。
- **プロシージャー・セクション:** ソース・プログラム内で定義される、リニア・メイン・プロシージャー (ゼロまたは 1 つ) と通常のサブプロシージャー (1 つ以上)。 各プロシージャーは、プロシージャーの始めの指定で始まり、「プロシージャーの終わり」の指定で終わります。

  リニア・メイン・プロシージャーは、制御仕様書の MAIN キーワードによって指定され、特殊なサブプロシージャーになります。

サブプロシージャーの定義

サブプロシージャーとは、メイン・ソース・セクションの後に定義されるプロシージャーです。

サブプロシージャーは、サイクル・メイン・プロシージャーと異なる点がいくつかあります。 その中でも大きな違いは、サブプロシージャーでは実行中に RPG サイクルを使用しない (使用できない) という点です。

メイン・ソース・セクションの定義仕様書の中に、サブプロシージャーの対応プロトタイプを指定することができます。 このプロトタイプが指定された場合、これは、プログラムまたはプロシージャーを正しく

呼び出し、呼び出 し元が確実に正しいパラメーターを渡せるように、コンパイラーによって使用されます。 指定されない場合、プロシージャーはプロシージャー・インターフェースから暗黙的に生成されます。

ヒント：

プロシージャーを定義したモジュール内では、プロトタイプの指定はオプションですが、 プロシージャーがモジュールからエクスポートされて、他の RPG モジュールから呼び出される場合には、オプションにはなりません。 この場合には、プロトタイプをコピー・ファイルに指定して、 サブプロシージャーを定義したモジュール、およびサブプロシージャーを呼び出すすべてのモジュールにコピーする必要があります。

以下に示すのは、各構成部分を分かりやすく表示したサブプロシージャーの例です。 最初に自由形式定義を使用する例を示し、2 番目に固定形式を使用する例を 示します。

このプロシージャーは、値パラメーターとして渡される 3 つの数値に対して関数を 実行します。 この例は、プロシージャーに対してプロシージャー・インターフェースがどのように 指定されるのか、および、値がプロシージャーからどのように戻されるのかを示しています。

サブプロシージャーには、プロシージャーが終了するたびに稼働するコードを含む ON-EXIT セクションをオプションで設定できます。 このコードは、プロシージャーが正常終了しても異常終了しても実行されます。 詳しくは、839 ページの『ON-EXIT (終了時)』を参照してください。

```
// Prototype for procedure FUNCTION
 DCL-PR Function INT(10);          1
   TERM1 INT(5) VALUE;
   TERM2 INT(5) VALUE;
   TERM3 INT(5) VALUE;
 END-PR;

 DCL-PROC Function;                2
    DCL-PI *N INT(10);             3
      TERM1 INT(5) VALUE;
      TERM2 INT(5) VALUE;
      TERM3 INT(5) VALUE;
    END-PI;
    DCL-S Result INT(10);          4

    Result = Term1 ** 2 * 17
                + Term2 * 7        5
                + Term3;
    return Result * 45 + 23;
 END-PROC;                         6
```

図 7. 自由形式サブプロシージャーの例

```
 * Prototype for procedure FUNCTION
D FUNCTION        PR            10I 0                    1
D    TERM1                       5I 0 VALUE
D    TERM2                       5I 0 VALUE
D    TERM3                       5I 0 VALUE

P Function        B                                      2
D Function        PI            10I 0                    3
D    Term1                       5I 0 VALUE
D    Term2                       5I 0 VALUE
D    Term3                       5I 0 VALUE
D Result          S             10I 0                    4

    Result = Term1 ** 2 * 17
                + Term2 * 7                               5
                + Term3;
    return Result * 45 + 23;

P                 E                                      6
```

図 8. 固定形式サブプロシージャーの例

1
名前、戻り値 (ある場合)、パラメーター (ある場合) を指定 するプロトタイプ。 プロシージャーがこのモジュールからエクスポートされないため、プロトタイプの指定はオプションです。

2
プロシージャー仕様の開始。

3
プロシージャー・インターフェース定義。これは、 戻り値およびパラメーターがあった場合に、それらを 指定します。 プロシージャー・インターフェースは、対応するプロトタイプと一致していなけ ればなりません。 サブプロシージャーから値が戻されず、また、そのサブプロシージャ ーに渡されるパラメーターがない場合には、プロシージャー・インターフェースの定義はオプションです。 プロトタイプが指定されていないと、 コンパイラーがプロシージャー・インターフェース定義を使用して、プロトタイプを暗黙的に定義します。

4
他のローカル定義。

5
プロシージャーのタスクを実行するためには、標準または自由形式のなんらかの演算仕様書が必要です。 演算では、ローカルとグローバルの両方の定義を 参照することができます。 サブプロシージャーの中に組み込まれたサブルーチンは、 いずれもローカルです。 それらをサブプロシージャーの外で使用することはできません。 サブプロシージャーから値が戻される場合には、 そのサブプロシージャーに RETURN 命令が含まれていなければなりません。

6
プロシージャー仕様の終了。

定義仕様書のどこにでも入れることができるプロシージャー・インターフェース定 義を除き、サブプロシージャーは上記の順序でコーディングしなければなりま せん。

サブプロシージャーの場合には、サイクル・コードは生成されません。 したがって、次 のものをコーディングすることはできません。

- 実行時前およびコンパイル時配列およびテーブル
- *DTAARA 定義
- 合計演算

演算仕様書は一度だけ処理され、演算仕様書の終わりでプロシージャーに戻ります。 詳しくは、116 ページの『サブプロシージャー演算』を参照してください。

サブプロシージャーはエクスポートすることができますが、これは、プログラム 内の他のモジュールにあるプロシージャーでそれを呼び出すことができることを 意味します。 サブプロシージャーがエクスポートされることを指示するためには、プロシージャーの始めの指定でキーワード EXPORT を指定してください。 これを指定しなかった場合には、サブプロシージャーはモジュールの中からしか 呼び出すことができません。

プロシージャー・インターフェース定義

プロトタイプ・プロシージャーに呼び出しパラメーターまたは戻り値がある場合には、プロシージャー・イン ターフェース定義が含まれていなければなりません。

プロシージャー・インターフェース定義について詳しくは、228 ページの『プロシージャー・インターフェース』を参照してください。

戻り値

値を戻すプロシージャーは、本質的には、組み込み関数に似たユーザー定義関数 です。 サブプロシージャーの戻り値を定義するためには、以下のことが必要です。

1. 戻り値をサブプロシージャーのプロトタイプとプロシージャー・インターフ ェース定義の両方に定義します。
2. 戻り値が入れられる式を指定した RETURN 命令をコーディングします。

プロシージャー・インターフェース仕様 (DCL-PI ステートメント、 または、24 から 25 桁目に PI がある定義仕様書) に、戻り値の長さとタイプを定義します。 次のキーワードも使用することができます。

DATFMT(形式)
戻り値は、キーワードによって指定された日付の形式を持ちます。

DIM(N)
戻り値は、要素数が N の配列です。

LIKE(名前)
戻り値は、キーワードによって指定された項目と同じに定義されます。

LIKEDS(名前)
戻り値は、キーワードによって指定されたデータ構造と同様に定義されているデータ構造です。

LIKEREC(名前 {,タイプ })
戻り値は、キーワードによって指定されたレコード名と同様に定義されているデータ構造です。

PROCPTR
戻り値は、プロシージャー・ポインターです。

TIMFMT(形式)
戻り値は、キーワードによって指定された時刻の形式を持ちます。

値を呼び出し元に戻すためには、戻り値が入れられる式を指定した RETURN 命令を コーディングしなければなりません。 RETURN 命令の オペランドには、EVAL の式と同じ規則が適用 されます。 実際の戻り値が EVAL 式の左側と同じ 働きをし、RETURN 命令のオペランドが右側と同じ役割を果たします。 サブプロシージャーに戻り値が定義されている場合には、RETURN 命令が実行され ることを確認しなければなりません。これを確認しておかないと、サブプロシー ジャーの呼び出し元には例外が出されることになります。

定義の有効範囲

サブプロシージャーの中で定義される項目は、すべてサブプロシージャーに対してローカルになります。ローカル項目がグローバル・データ項目と同じ名前で定義されていた場合には、 サブプロシージャー内部でのその名前に対するすべての参照にはローカル定義が 使用されます。

ただし、次の点を承知しておいてください。

- サブルーチン名およびタグ名は、サイクル・メイン・プロシージャーで定義された場合でも、それらが定義されたプロシージャーでのみ認識されます。
- 入力および出力仕様に指定されたすべてのフィールドは、グローバル・フ ィールドになります。 サブプロシージャーに入力または出力仕様が使用される場合 (例えば、読み取 り操作の処理時) には、同じ名前のローカル変数があっても、グローバル名が使 用されます。

サブプロシージャーでグローバルな KLIST または PLIST を使用している場合に は、一部のフィールドにローカル・フィールドと同じ名前が入っていることがあ ります。 これが起こった場合には、グローバル・フィールドが使用されます。 これは、その使用前の KLIST または PLIST のセットアップ時に問題の原因となることがあります。

例えば、次のようなソースを考えてみます。

```
 * Main procedure definitions
D Fld1            S              1A
D Fld2            S              1A

 * Define a global key field list with 2 fields, Fld1 and Fld2
C     global_kl     KLIST
C                   KFLD                    Fld1
C                   KFLD                    Fld2

 * Subprocedure Section
P Subproc         B
D Fld2            S              1A

 * local_kl has one global kfld (fld1) and one local (fld2)
C     local_kl      KLIST
C                   KFLD                    Fld1
C                   KFLD                    Fld2

 * Even though Fld2 is defined locally in the subprocedure,
 * the global Fld2 is used by the global_kl, since global KLISTs
 * always use global fields.  As a result, the assignment to the
 * local Fld2 will NOT affect the CHAIN operation.

C                   EVAL      Fld1 = 'A'
C                   EVAL      Fld2 = 'B'
C     global_kl     SETLL     file

 * Local KLISTs use global fields only when there is no local
 * field of that name.  local_kl uses the local Fld2 and so the
 * assignment to the local Fld2 WILL affect the CHAIN operation.
C                   EVAL      Fld1 = 'A'
C                   EVAL      Fld2 = 'B'
C     local_kl      SETLL     file
 ...
P                 E
```

図 9. モジュール内のキー・フィールドの有効範囲

有効範囲について詳しくは、200 ページの『定義の有効範囲』を参照してください。

サブプロシージャーおよびサブルーチン

サブプロシージャーはサブルーチンと類似していますが、サブプロシージャーで は以下のように強化されている点が異なります。

- パラメーターを値によってもサブプロシージャーに渡すことができます。

  このことは、サブプロシージャーとの連絡に使用されるパラメーターは変更可能 である必要はないことを意味します。 プログラムの場合のように参照によって渡されるパラメーターは、変更可能でな ければなりません。したがって、その信頼性が低下することもあります。
- サブプロシージャーとの間で受け渡しされるパラメーターの整合性は、コンパ イル時に検査されます。このことは、より負担が大きくなる可能性がある実行時エラーを少なくするのに 役立ちます。
- サブプロシージャーは、式の中で組み込み関数と同じように使用することがで きます。

  このようにして使用した場合には、呼び出し元に値が戻されます。 これによって、基本的 には、式の中に必要な任意の演算子をユーザーが定義する ことができます。
- サブプロシージャーの中に定義された名前は、そのサブプロシージャーの外 部では見ることができません。

  このことは、プロシージャーが他のプロシージャーと共用されている項目を 不用意に変更してしまう機会が少なくなることを意味します。 さらに、プロシージャーの呼び出し元では、サブプロシージャーの中で使用されて いる各項目について多くを知っている必要はありません。
- サブプロシージャーがエクスポートされた場合には、それをモジュールの外 部から呼び出すことができます。
- サブプロシージャーは反復して呼び出すことができます。

- プロシージャーは、異なる仕様タイプ、すなわち、プロシージャー仕様書に 定義されます。 このタイプが異なることで、別の単位を取り扱っていることをた だちに認識するのに役立ちます。

サブプロシージャーによって提供される改良点が必要でない場合は、サブプロシージャーを呼び出すよりも EXSR 命令のほうが高速であるため、サブルーチンを使用したほうがよい場合もあります。

RPG モジュールでのプログラム・フロー: サイクル対リニア

RPG モジュールの論理の一部は、ILE RPG コンパイラーによって提供されます。 選択したモジュールのタイプに応じて、モジュールの制御フローの大部分または一部分が、この提供された論理によって制御されます。 RPG モジュールには、完全な RPG サイクルがデフォルトで含まれています。 このサイクルは、*INIT フェーズで始まり、*TERM フェーズで終わります。 その他 2 つのタイプの RPG モジュールには、完全な RPG サイクルは含まれません。RPG サイクルで唯一残されるものは、*INIT フェーズに類似しているモジュールの初期化です。 ILE RPG コンパイラーは、追加の暗黙論理 (例: サブプロシージャーでのローカル・ファイルの暗黙的なオープンとクローズ) を提供します。 これは、RPG サイクルとは区別されます。

すべての ILE RPG モジュールには、プロシージャーを 1 つ以上含めることができます。

3 つのタイプの RPG モジュールは、モジュール内のメイン・プロシージャーの性質によって区別されます。

プログラムまたはサービス・プログラムは、複数のモジュールで構成することができ、各モジュールに RPG メイン・プロシージャーを含めることができます。 RPG モジュールがプログラムのプログラム入り口モジュールとなるように選択されている場合は、プログラム呼び出しを使用してメイン・プロシージャーを呼び出します。 RPG モジュールがプログラムのプログラム入り口モジュールでない場合、またはそれがサービス・プログラムのモジュールである場合には、結合呼び出しを使用してメイン・プロシージャーを呼び出します。 結合呼び出しを使用したメイン・プロシージャーの呼び出しは、サイクル・メイン・プロシージャーに対してしか使用できません。 モジュールにリニア・メイン・プロシージャーが含まれており、そのモジュールがプログラム入り口モジュールになるよう選択されていない場合、そのプロシージャーを呼び出すことはできません。

サイクル・メイン・プロシージャーを含むモジュール
このモジュールには、サイクル・メイン・プロシージャーが 1 つと、サブプロシージャーが 1 つ以上 (ゼロの場合もあり) 含まれます。 サイクル・メイン・プロシージャーには、完全な RPG サイクルの論理が含まれています。 サイクル・メイン・プロシージャーは、結合呼び出しまたはプログラム呼び出しを通じて呼び出すことができます。詳しくは、98 ページの『サイクル・モジュール』 および 102 ページの『プログラム・サイクル』 を参照してください。

リニア・メイン・プロシージャーを含むモジュール
このモジュールには、リニア・メイン・プロシージャーが 1 つと、通常のサブプロシージャーが 1 つ以上 (ゼロの場合もあり) 含まれます。 リニア・メイン・プロシージャーは、制御仕様書では MAIN キーワードで識別されます。 メイン・プロシージャー自体は、サブプロシージャーとして (プロシージャー仕様書に従って) コーディングされます。 リニア・メイン・プロシージャーは、プログラム呼び出しを通じてのみ呼び出すことができます。 結合呼び出しで呼び出すことはできません。

注 : 呼び出し方法を除いて、リニア・メイン・プロシージャーはサブプロシージャーとみなされます。

このモジュールには、RPG サイクルの論理は含まれていません。 詳しくは、101 ページの『リニア・メイン・モジュール』を参照してください。

メイン・プロシージャーを含まないモジュール
制御仕様書の NOMAIN キーワードは、モジュールにメイン・プロシージャーがないことを示しています。 このモジュールに含まれているのは、サブプロシージャーのみです。 このモジュールには、RPG サイクルの論理は含まれていません。

このタイプのモジュールは、メイン・プロシージャーが含まれていないため、プログラムのプログラム入り口モジュールにすることはできません。

詳しくは、101 ページの『NOMAIN モジュール』を参照してください。

表 *59. RPG* モジュール・タイプの要約

モジュール・タイプ	キーワード	許可されるサイクル機能	メイン・プロシージャー	グローバル変数の初期化、グローバル・ファイルのオープン、および UDS データ域のロック	グローバル・ファイルの暗黙的なクローズおよびデータ域のアンロック
サイクル・メイン		はい	メイン・ソース・セクションで暗黙的に定義される	• 活動化グループが作成された後に、モジュールの最初のプロシージャーが呼び出されたとき。 • メイン・プロシージャーの呼び出し時に、メイン・プロシージャーが LR オンの状態で終了してしまっているとき、または異常終了してしまっているとき。	メイン・プロシージャーが LR オンの状態で終了したとき、または異常終了したとき。
リニア・メイン	MAIN	いいえ	MAIN キーワードおよびプロシージャー仕様書で明示的に定義される	活動化グループが作成された後にメイン・プロシージャーが初めて呼び出されたとき、または何らかの形でサブプロシージャーが初めて呼び出されたとき。	なし
メインなし	NOMAIN	いいえ	なし。NOMAIN キーワードの存在によって示される。	活動化グループが作成された後に、モジュールの最初のプロシージャーが呼び出されたとき	なし

サイクル・モジュール

サイクル・モジュールには、RPG プログラム・サイクルを使用するサイクル・メイン・プロシージャーが 1 つ含まれています。 このプロシージャーは、メイン・ソース・セクションで暗黙的に指定されます。(102 ページの『プログラム・サイクル』を参照してください。) メイン・プロシージャーを定義するには特別なコーディングは何も必要ありません。 メイン・プロシージャーは、最初のプロシージャー仕様書より前にあるすべてによって成り立っています。 サイクル・メイン・プロシージャー用のパラメーターは、 グローバル定義仕様書のプロシージャー・インターフェースとオプションのプロトタイプを使用するか、 またはサイクル・メイン・プロシージャーの演算で *ENTRY PLIST を使用することで、コーディングすることができます。

サイクル・メイン・プロシージャーの名前は、作成するモジュールの名前と一致させる必要があります。 プロトタイプおよびプロシージャー・インターフェースにこの名前を使用するか、プロトタイプの EXTPROC キーワード、あるいはプロトタイプが指定されない場合にはプロシージャー・インターフェースの EXTPROC キーワードにこの名前を指定することができます。

グローバル定義で見つかったプロシージャー・インターフェースはすべて、サイクル・メイン・プロシージャー用のプロシージャー・インターフェースであるとみなされます。 プロトタイプが指定される場合、サイクル・メイン・プロシージャー用のプロシージャー・インターフェースには、名前が必要になります。 その名前と一致する名前を持つプロトタイプを、ソース内のプロシージャー・インターフェースよりも前に置く必要があります。

以下の例では、モジュール CheckFile が作成されます。 このサイクル・メイン・プロシージャーには、次の 3 つのパラメーターがあります。

1. ファイル名 (入力)
2. ライブラリー名 (入力)

3. ファイルが見付かったかどうかを示す標識 (出力)

この例では、プロシージャーが別のモジュールから呼び出されるため、 /COPY ファイルにプロトタイプが指定されていなければなりません。

/COPY ファイルの CHECKFILEC (サイクル・メイン・プロシージャー用のプロトタイプ付き):

```
D CheckFile       PR
D   file                        10a   const
D   library                     10a   const
D   found                        1N
```

モジュール CheckFile:

```
 /COPY CHECKFILEC
D CheckFile       PI
D   file                        10a   const
D   library                     10a   const
D   found                        1N
C   ... code using parameters file, library and found
```

*ENTRY PLIST を使用してこのようにパラメーターを定義します

```
D file            S             10a   const
D library         S             10a   const
D found           S              1N
C     *ENTRY        PLIST
C                   PARM                    file
C                   PARM                    library
C                   PARM                    found
C   ... code using parameters file, library and found
```

プロトタイプとプロシージャー・インターフェースを使って、サイクル・メイン・プロシージャーをプログラムとして定義することもできます。 この場合、プロトタイプに対して EXTPGM キーワードを指定します。 この例では、プログラムが別の RPG プログラムによって呼び出されるため、 /COPY ファイルにプロトタイプが指定されていなければなりません。

/COPY ファイルの CHECKFILEC (プログラム用のプロトタイプ付き):

```
 D CheckFile       PR                  extpgm('CHECKFILE')
 D   file                        10a   const
 D   library                     10a   const
 D   found                        1N
```

モジュール・ソースの中で、プロシージャー・インターフェースを同じように定義しておきます。

次の例では、プログラムが別の RPG プログラムによって呼び出されないため、 プロトタイプは不要です。この場合、EXTPGM キーワードがプロシージャー・インターフェースに指定されます。 プロトタイプは指定されないため、プロシージャー・インターフェースの名前は不要です。

EXTPGM キーワードを指定したプロシージャー・インターフェース:

```
 F ... file specifications
 D                 PI                  extpgm('CUSTREPORT')
 D   custfile                    10a   const
 D   custlib                     10a   const
  ... code using the custfile and custlib parameters
```

サイクル・モジュールにおけるサブプロシージャーのエクスポート時の注意

サイクル・メイン・プロシージャーとエクスポートされたサブプロシージャーの両方が、単一のモジュール内にある場合、サブプロシージャーが使用するグローバルのデータ、ファイル、およびデータ域に、サイクル・メイン・プロシージャーの RPG サイクルが悪影響を及ぼすことがないよう、十分に注意してください。

ファイルがいつ暗黙的にオープンおよびクローズされるか、データ域はいつ暗黙的にロックされ、 またアンロックされるか、そしてグローバル・データがいつ初期化または再初期化されるか、 ユーザーは知っておく必要があります。

起こりうる問題

エクスポートされたサブプロシージャーがサイクル・モジュールに含まれている場合、予期しない時にサイクル・メイン・プロシージャーの初期化が実行されるシナリオが生じる可能性があります。 これによって、ファイル、データ域のロック、およびグローバル・データに影響を及ぼし、エラーが発生します。 サイクル・メイン・プロシージャーを呼び出す前に、モジュール外のプロシージャーから、エクスポートされたサブプロシージャーを最初に呼び出すことができます。 サイクル・メイン・プロシージャーは、呼び出された時点で初期化されます。

- 呼び出される最初のプロシージャーがサブプロシージャーであるという理由でモジュールの初期化が行なわれ、その後にサイクル・メイン・プロシージャーの初期化が行なわれた場合、ファイルがすでにオープンしていたりあるいはデータ域がすでにロックされていたりすると、エラーが発生する可能性があります。
- サブプロシージャーがサイクル・メイン・プロシージャーを呼び出したときに、前回の呼び出し時におけるメイン・プロシージャーの終了状態によって、今回の呼び出し時にグローバル・データの再初期化が行われる場合と行われない場合とがあります。 サブプロシージャーが何らかのグローバル・データを使用していると、予期しない結果を招くことがあります。
- サイクル・メイン・プロシージャーが最後に呼び出され、終了された後、エクスポートされたサブルーチンがモジュール外から呼び出されたときに、本来ならファイルのオープンとデータ域のロックが求められているにもかかわらず、ファイルが暗黙的にクローズされてデータ域がアンロックされていた場合、エラーが発生する可能性があります。

推奨事項

サイクル・メイン・プロシージャーの論理をサブプロシージャーに移動して、モジュールを NOMAIN モジュールにするか、サイクル・メイン・プロシージャーをリニア・メイン・プロシージャーに変更するか、いずれかを検討してください。

エクスポートされたサブプロシージャーをサイクル・メイン・プロシージャーと混用する場合は、いかなるサブプロシージャーよりも先に、サイクル・メイン・プロシージャーを必ず最初に呼び出すようにしてください。

サイクル・メイン・プロシージャーの初期化を複数回行なわないようにしてください。 初期化が再度行なわれると、ユーザーのグローバル・データも再初期化されてしまいます。 再初期化を防ぐ最良の方法は、LR 標識を使用しないことです。

サイクル・メイン・プロシージャーとサブプロシージャーを混合して呼び出したい場合は、すべてのユーザーのファイルを USROPN として宣言し、UDS データ域は使用しないでください。 ファイルのオープンとデータ域のロックはユーザーが必要な時点で行ない、必要でなくなったらファイルをクローズしデータ域をアンロックしてください。 オープンしているファイルをすべてクローズし、ロックされているデータ域をすべてアンロックするサブプロシージャーを、モジュールに含めることを考えた方がよい場合もあります。

リニア・モジュール

キーワード **MAIN** または **NOMAIN** を制御仕様書で指定しているモジュールは、プログラム・サイクルを組み込まずにコンパイルされます。

プログラム・サイクルがモジュールに含まれていないと、メイン・ソース・セクションにコーディングできる対象が制限されます。 具体的には、次のものに関する仕様はコーディングすることができません。

- 1 次ファイルおよび 2 次ファイル
- 見出し、明細出力および合計出力
- 実行可能演算 (*INZSR 初期化サブルーチンを含む)
- *ENTRY PLIST

代わりに、メイン・ソース・セクションには次のものがコーディングされます。

- 全手順ファイル
- 入力仕様書
- 定義仕様書

- DEFINE、KFLD、KLIST、PARM、および PLIST などの宣言演算 (ただし、*ENTRY PLIST は除く)
- 例外出力

注意: リニア・モジュールでは、グローバル・ファイルの暗黙的なクローズまたはデータ域のアンロックは行いません。 これらのオブジェクトは、クローズやアンロックが明示的に行われるまで、オープンされたままあるいはロックされたままになります。

リニア・メイン・モジュール

プログラム入り口プロシージャーはあるが、RPG プログラム・サイクルは使用しないモジュールを、制御仕様書で MAIN キーワードを指定することにより生成できます。

このタイプのモジュールにはプロシージャーが 1 つ以上含まれており、その中のいずれかがメイン・プロシージャーとして特定されます。 RPG プログラム・サイクルに関連する仕様は許可されません。

詳しくは、338 ページの『MAIN(main_procedure_name)』を参照してください。

NOMAIN モジュール

メイン・プロシージャーをコーディングせずに、モジュールの中に 1 つまたは複数のサ ブプロシージャーをコーディングすることができます。 このようなモジュールは、制御仕様書に NOMAIN キーワードの指定を必要とするこ とから、**NOMAIN モジュール**と呼ばれます。 NOMAIN モジュールの場合には、サイクル・コードは生成されません。

ヒント :

プログラム用のプログラム入り口プロシージャーが実際に含まれているものを除いて、すべてのサイクル・モジュールを NOMAIN モジュールに変換することを検討してみてください。 これにより、各モジュールから不要なサイクル・コードが除去されるため、個々のモジュール・サイズを縮小できます。

注 : NOMAIN が指定されたモジュールには、プログラム・エントリー・プロシージャーは含まれません。 したがって、ソース仕 様をコンパイルするための CRTBNDRPG コマンドを使用することはできません。

詳しくは、340 ページの『NOMAIN』を参照してください。

モジュールの初期化

モジュール初期化は、最初のプロシージャー (メイン・プロシージャーでもサブプロシージャーでもどちらでも) が呼び出された時に行なわれます。

サイクル・モジュールには、繰り返し行われる可能性のある初期化形式が他にもあります。 サイクル・メイン・プロシージャーが初めて呼び出されたときには、サイクル・メイン・プロシージャーの初期化が行われます。 以降の呼び出しにおいても、サイクル・メイン・プロシージャーが異常終了した場合、あるいは LR オンで終了した場合に、この初期化が行なわれます。

グローバル・データの初期化

モジュールの初期化時およびサイクル・メイン・プロシージャーの初期化時に、モジュール内のグローバル・データが初期化されます。

サイクル・メイン・プロシージャーの初期化に特有の問題については、99 ページの『サイクル・モジュールにおけるサブプロシージャーのエクスポート時の注意』を参照してください。

RPG サイクルおよびその他の暗黙論理

RPG プログラムの論理の一部は、ILE RPG コンパイラーによって提供されます。

- サイクル・メイン・プロシージャーの場合は、コンパイラーによってプログラム・サイクルが提供されます。 このプログラム・サイクルは、論理サイクル または *RPG* サイクル とも呼ばれます。
- サブプロシージャーまたはリニア・メイン・プロシージャーの場合、コンパイラーによってサブプロシージャーの初期化および終了が提供されます。

プログラム・サイクル

RPG プログラムの論理の一部は、ILE RPG コンパイラーによって提供されます。 サイクル・メイン・プロシージャーの場合、プログラム・サイクルまたは論理サイクルと呼ばれる論理が、コンパイラーによって提供されます。 プログラム・サイクルとは、 読み取った各レコードをメイン・プロシージャーが処理していく、順序付けられた一連の ステップのことです。

ソース・プログラムの中で RPG IV 仕様書にコーディングする情報では、レコードの読み取りまたは書き出しの時期を明示的に指定する必要はありません。 ソース・プログラムのコンパイル時に、ILE RPG コンパイラーによってこれらの操作の論理順序を得ることができます。 コーディングした仕様書によって、プログ ラムでサイクル内の各ステップが使用されることもあれば、使用されないことも あります。

1 次ファイル (ファイル仕様書の 18 桁目の P で識別される) お よび 2 次ファイル (ファイル仕様書の 18 桁目の S で 識別される) では、プログラム・サイクルによって入力が制御されることを指示します。 全手順ファイル (自由形式 SCL-F ステートメントを使用して定義されるか、 または、ファイル仕様書の 18 桁目の F で識別される) は、 プログラム指定の演算命令 (たとえば、READ および CHAIN) によって入力が 制御されることを指示します。

サイクルを制御するために、以下のものを含めることができます。

- 1 つの 1 次ファイル、および、オプションの 1 つまたは複数の 2 次ファイル
- 全手順ファイルのみ
- 1 つの 1 次ファイル、任意指定の 2 次ファイル、およ び 1 つまたは複数の全手順ファイルの組み合わせ。 ファイル中にはサイクルによって 制御される入力がありますが、プログラムによって制御される入力もあります。
- ファイルなし (たとえば、パラメーター・リストまたはデータ域データ構造 からの入力が可能な場合)

注 : 制御仕様書に MAIN または NOMAIN が指定されている場合には、モジュール用のサイクル・コードは生成されません。 詳しくは、100 ページの『リニア・モジュール』 を参照してください。

一般的な *RPG IV* プログラム・サイクル

103 ページの図 10 は、RPG IV プログラム・サイクルの一般的なフローにおける特定のステップを示しています。 プログラム・サイクルは、ステップ 1 から始まってステップ 7 まで続き、次にま たステップ 1 から始まります。

プログラムが RPG IV サイクルを通過する場合､最初と最後は通常のサイクルと多少異なります。 サイクルを通じて最初のレコードを初めて読み取る前に、プログラム では、そこに渡されたすべてのパラメーターが分析解決され、1P (1 ページ目) 標識 によって条件付けされたレコードが書き出され、ファイルとデータの初期化が実行さ れて、条件付け標識がないかまたはすべてが否定の条件付け標識である見出しま たは明細出力操作が処理されます。 たとえば、最初のレコードを読み取る 前に印刷される見出し行は、定数、ページ見出し情報、または PAGE およ び *DATE などの予約語のフィールドから構成されることがあります。 さらに、最初のサイ クルでは、合計演算および合計出力のステップはプログラムによって回避されます。

プログラムがサイクルを最後に通過する場合に使用できるレコードがなければ、LR (最終レコード) 標識および L1 から L9 (制御レベル) 標識がオンに設定され、ファイルおよびデータ域の終結処置が実行されます。

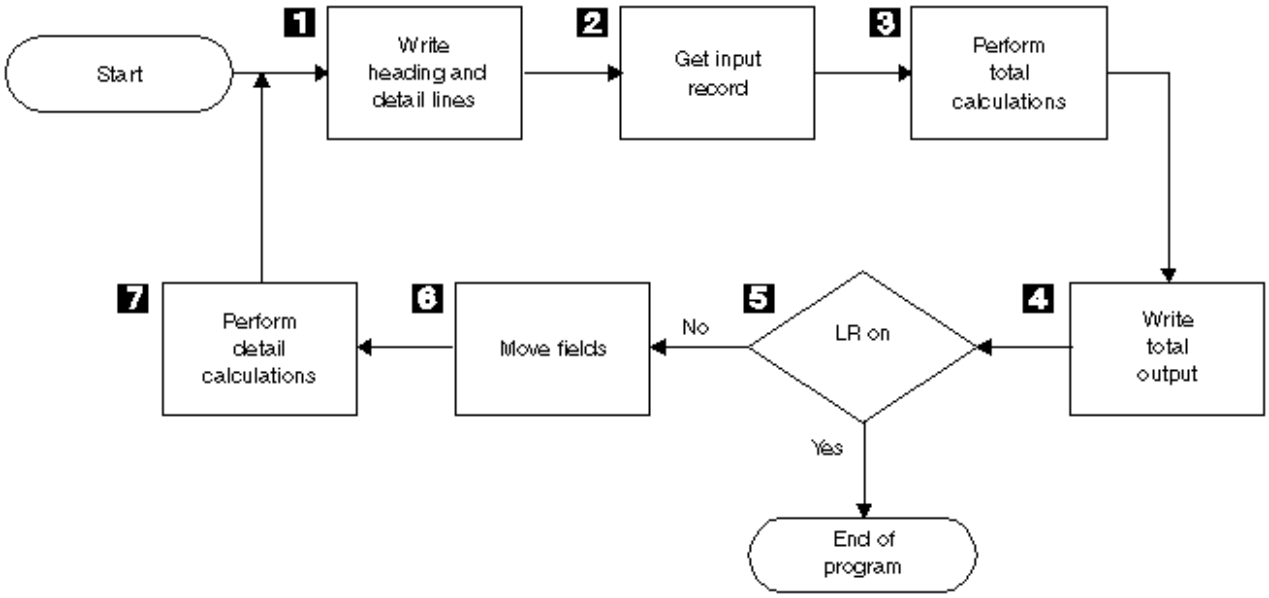


図 *10. RPG IV* プログラムの論理サイクル

1
すべての見出し行および明細行 (出力仕様の 17 桁目が H または D) が処 理されます。

2
次の入力レコードが読み取られ、レコード識別標識および制御レベル標識が オンに設定されます。

3
合計演算が処理されます。 これらの合計演算は L1 から L9 または LR 標識、あるいは L0 記入項目によって条 件付けされます。

4
すべての合計出力行が処理されます。(出力仕様の 17 桁目の T によって識別されます。)

5
LR 標識がオンかどうかが判別されます。 オンの場合には、プログラムは終了されます。

6
選択された入力レコードの各フィールドが、レコードから処理区域 に転送されます。 フィールド標識がオンに設定されます。

7
サイクルの始めで読み取ったレコードのデータに対して、明細演算 (演算仕様書の 7 から 8 桁目の制御レベル標識によって条件付けされていない もの) がすべて処理されます。

詳細な *RPG IV* プログラム・サイクル

102 ページの『一般的な RPG IV プログラム・サイクル』で、基本的な RPG IV 論理サイクルについて 紹介しました。 次の図には、RPG IV 論理サイクルの詳細な説明が示されています。

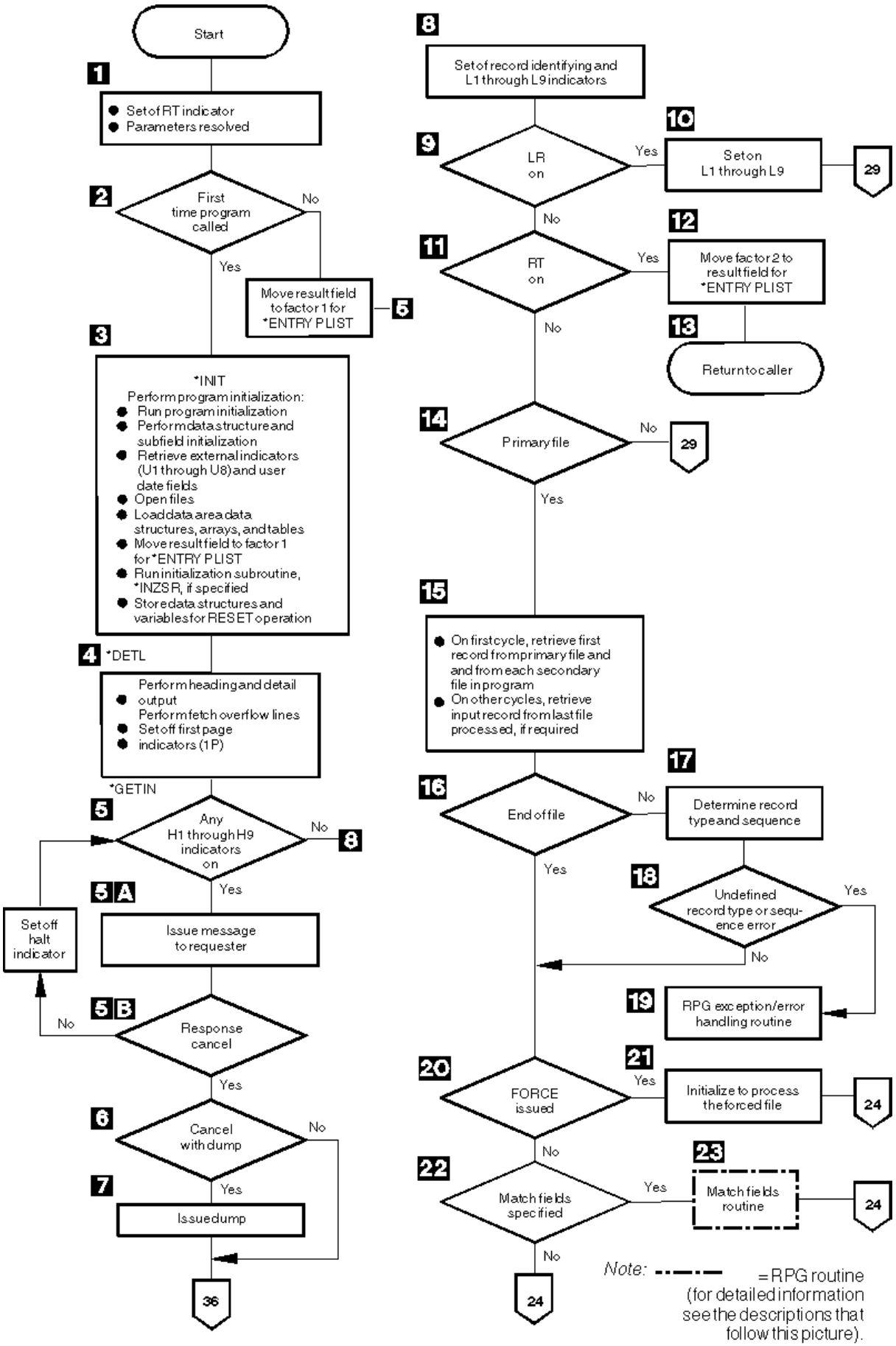


図 *11.* 詳細な *RPG IV* オブジェクト・プログラム・サイクル

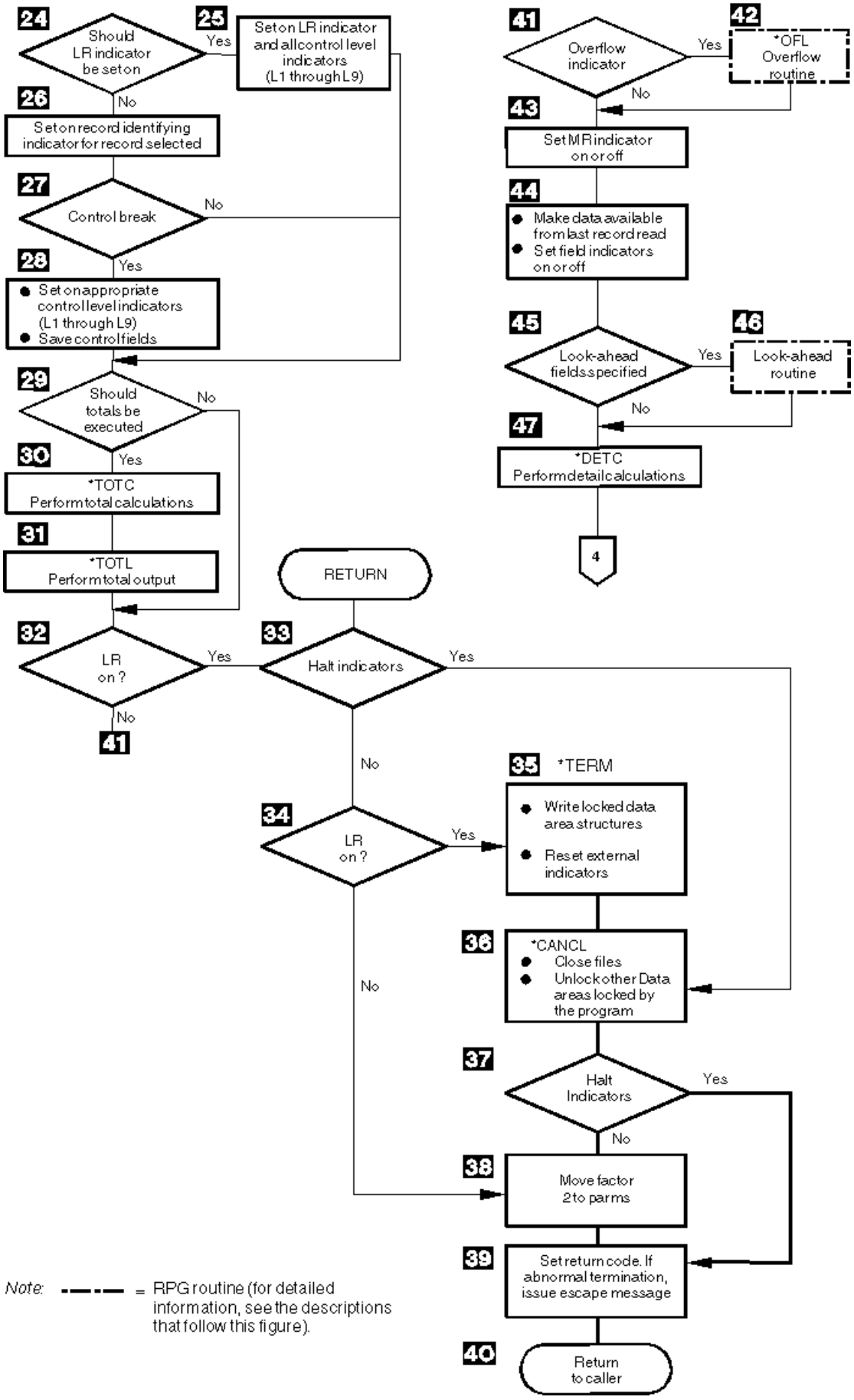


図 *12.* 詳細な *RPG IV* オブジェクト・プログラム・サイクルの続き

詳細な *RPG IV* オブジェクト・プログラム・サイクル

104 ページの図 11 は、RPG IV プログラム・サイクルの詳細なフローの特定のステップを示しています。以下の説明の項目番号は、図の中の 番号を示しています。 114 ページの図 15 および 110 ページの図 13 には、ルーチンがフローチャートで示されています。

1

RT 標識がオフに設定されます。 *ENTRY PLIST の指定がある場合には、パラメーターが 分析解決されます。

2

RPG IV は、プログラムの最初の呼び出しを検査します。 これが 最初の呼び出しであれば、プログラムの初期化が続行されます。 そうでない場合 には、結果のフィールドを *ENTRY PLIST の PARM ステートメントの演算項目 1 に転送し、ステップ 5 に分岐します。

3

プログラムがサイクルの *INIT で初期化されます。 ここで行われる処理は、以下のとおりです: データ構造およびサブフィールドの初期化の実行、ユーザー日付フィールドの設定、グローバル・ファイルのオープン、データ域のすべてのデータ構造、配列、およびテーブルのロード、*ENTRY PLIST の PARM ステートメントの演算項目 1 への結果フィールドの転送、初期化サブルーチン *INZSR の実行、RESET 命令のための構造および変数の保管。 グローバル・ファイルは、ファイル仕様書での指定とは逆の順序でオープンされます。

4

見出しおよび明細行 (出力仕様の 17 桁目の H または D によって識別され る) を、最初のレコードが読み取られる前に書き出します。 見出し行と 明細行は常に同時に処理されます。 条件付け標識の指定がある場合には、適切な標識設定値が満たされて いなければなりません。 フェッチ・オーバーフロー・ルーチン論理が指定され、 オーバーフロー標識が オンの場合には、該当するオーバーフロー行が書き出されます。 ファイル変換の指定がある場合には、見出し行、明細行、およびオーバーフロー 出力について変換が実行されます。 ENDSR 命令の演算項目 2 に値 *DETL が入って いる場合には、このステップがプログラム内の戻り点になります。

5

停止標識 (H1 から H9) がテストされます。 すべての停止標識がオフであれば、プログラムはステップ 8 に分岐します。 停止標識は、プログラムのどの時点でもオンに設定することができます。 ENDSR 命令の演算項目 2 に値 *GETIN が入っている場合には、このステップがプ ログラム内の戻り点になります。

a.
停止標識が 1 つでもオンであれば、メッセージがユーザーに出されます。

b.
続行するという応答の場合には、停止標識はオフに設定され、プログラムは ステップ 5 に戻ります。 応答が取り消しの場合には、プログラムはステップ 6 に進みます。

6

ダンプをとって取り消すという応答の場合には、プログラムはステップ 7 に 進みます。そうでない場合には、プログラムはステップ 36 に分岐します。

7

プログラムがダンプを出し、ステップ 36 (異常終了) に分岐します。

8

レコード識別、1P (1 ページ目)、および制御レベル (L1 から L9) 標識が、す べてオフに設定されます。オーバーフロー標識 (OA から OG、OV) は、前の明細演算または 明細出力でオンに設定されたのでなければ、すべてオフに設定されます。 その他 のオンの標識はオンのまま残ります。

9

LR (最終レコード) 標識がオンの場合には、プログラムはステップ 10 から 続行されます。 オンでなければ、プログラムはステップ 11 に分岐します。

10

該当する制御レベル (L1 から L9) 標識がオンに設定され、プログラムはステ ップ 29 に分岐します。

11
RT 標識がオンの場合には、プログラムはステップ 12 から続行され、そうで ない場合には、プログラムはステップ 14 に分岐します。

12
*ENTRY PLIST のパラメーターの演算項目 2 が結果のフィールドへ 転送されます。

13
RT 標識がオンである (戻りコードが 0 に設定されている) 場合には、プログラムは 呼び出し元に戻ります。

14
プログラムに 1 次ファイルが存在する場合には、プログラムはス テップ 15 から続行され、そうでない場合には、プログラムは ステップ 29 に分岐します。

15
最初のプログラム・サイクルでは、プログラム中の 1 次ファイル および各 2 次ファイルから最初のレコードが読み取られます。 入力レコードの ファイル変換が実行されます。 その他のプログラム・サイクルでは、最後に処理された ファイルからレコードが 読み取られます。 このファイルがレコード・アドレス・ファイルによって処理さ れる場合には、レコード・アドレス・ファイル中のデータによって検索するレコ ードが定義されます。 最後に処理されたレコードに先読みフィールドが 指定されていた場合には、レコードはすでに記憶域の中にある可能性があるため、この時 点では読み取りは行われません。

16
読み取ったばかりのファイルでファイルの終わりが起こった場合には、プログ ラムはステップ 20 に分岐します。 そうでなければ、プログラムはステップ 17 から続行されます。

17
ファイルからレコードが読み取られた場合に、レコード・タイプおよびレコ ード順序 (入力仕様の 17 から 20 桁目) が判別されます。

18
レコード・タイプがプログラム内で定義されているかどうか、およびレコー ド順序は正しいかどうかが判別されます。 レコード・タイプが未定義であるか、あるいはレコード順序が正しくない場合には、プログラムはステップ 19 から続 行され、そうでない場合には、プログラムはステップ 20 に分岐します。

19
RPG IV の例外/エラー処理ルーチンが制御を受け取ります。

20
前のサイクルで FORCE 命令が処理されているかどうかが判別されます。 FORCE 命令が処理されていた場合には、プログラムによってそのファイルが処理 のために選択され (ステップ 21)、突き合わせフィールドの処理の近辺 (ステップ 22 および 23) に分岐します。 FORCE 命令によって処理されるすべてのレコード は、突き合わせレコード (MR) 標識をオフにして処理されるので、この分岐が処理されます。

21
前のサイクルで FORCE が出されていた場合には、プログラムは読み取ったば かりのファイルからすべての突き合わせフィールドを保管した後に、処理用の強制フ ァイルを選択します。 強制ファイルがファイルの終わりになっている場には、通 常の 1 次/2 次の複数ファイルの論理によって次のレコードが処理のために選択され、プログラムはステップ 24 に分岐します。

22
突き合わせフィールドが指定された場合には、プログラムはステップ 23 から続 行され、そうでない場合には、プログラムはステップ 24 に分岐します。

23
突き合わせフィールド・ルーチンが制御を受け取ります。(突き合わせフィールド・ ルーチンについて詳しくは、110 ページの『突き合わせフィールド・ルーチン』を参照してください。)

24
ファイル仕様書の 19 桁目に E の指定があるファイルからのすべてのレ コードが処理され、すべての突き合わせ 2 次レコードが処理された時に、 LR (最終レコード) 標識がオンに設定されます。 LR 標識がオンに設定されて いない場合には、処理はステップ 26 から続行されます。

25
LR (最終レコード) 標識およびすべての制御レベル (L1 から L9) 標識がオンに設定され、処理はステップ 29 から続行されます。

26
処理のために選択されたレコードのレコード識別標識がオンに設定されま す。

27
処理のために選択されたレコードで制御の切れ目が起こったかどうかが判別 されます。 処理中のレコードの制御フィールドの値が最後に処理されたレコード の制御フィールドの値と異なる場合に、制御の切れ目が起こります。 制御の切れ目が 起こっていなければ、プログラムはステップ 29 に分岐します。

28
制御の切れ目が起こった場合には、該当する制御レベル標識 (L1 から L9) がオ ンに設定されます。 これより低いレベルの制御標識もすべてオンに設定されます。 プログラムは、次の比較のために制御フィールドの内容を保管します。

29
合計時演算および合計時出力を実行する必要があるかどうかが 判別されます。 LR 標識がオンである場合には、合計が常に処理されます。 入力仕様に 制御レベルが指定されていない場合には、合計は最初のサイクルで は回避され、最初のサイクルの後は、すべてのサイクルで合計が処理されます。 入力仕様 に制御レベルが指定されている場合には、制御フィールドが含まれる 最初のレコードが処理されるまで、合計は回避されます。

30
制御レベル項目 (演算仕様書の 7 から 8 桁目) によって条件付けられたすべての合計演算が処理されます。 ENDSR 命令の演算項目 2 に値 *TOTC が入っている場合には、 このステップがプログラム内の戻り点になります。

31
すべての合計出力が処理されます。 フェッチ・オーバーフロー論理が指定され、 ファイルと関連したオーバーフロー標識 (OA から OG、OV) がオンの場合には、オーバーフロー行が書き出されます。 ファイル変換の指定がある場合には、 すべての合計出力およびオーバーフロー行について変換が実行されます。 ENDSR 命令の演算項目 2 に値 *TOTL が入っている場合には、 このステップがプログラム内の戻り点になります。

32
LR がオンの場合には、プログラムはステップ 33 から続行され、そうでない 場合には、プログラムはステップ 41 に分岐します。

33
停止標識 (H1 から H9) がテストされます。 停止標識が 1 つでもオンであれば、プログラム はステップ 36 (異常終了) に分岐 します。 停止標識がオフの場合には、プログラムはステップ 34 から続行されます。 RETURN 命令コードを演算で使用した場合には、プログラムはその命令を実行 した後でステップ 33 に分岐します。

34
LR がオンの場合にはプログラムはステップ 35 から続行されます。 オンで なければ、プログラムはステップ 38 に分岐します。

35
RPG IV プログラムは、定義指定において TOFILE キーワードが指定されているすべての配列またはテーブルを書き出し、すべてのロックされたデータ域データ構造を書き出します。 出力配列およびテーブルは必要に応じて変換されます。

36
オープンされているグローバル・ファイルが、すべてクローズされます。 RPG IV プログラムは、プログラムでロックされたにもかかわらずアンロックされなかったすべてのデータ域をアンロックします。 ENDSR 命令の演算項目 2 に値 *CANCL が入っている場合には、このステップが戻り 点になります。

37
停止標識 (H1 から H9) がテストされます。 停止標識が 1 つでもオンであれば、プログラム はステップ 39 (異常終了) に分岐 します。 停止標識がオフの場合には、プログラムはステップ 38 から続行されます。

38
演算項目 2 フィールドは *ENTRY PLIST の PARM の結果のフィールドに転送 されます。

39
戻りコードが設定されます。1 = LR オン、2 = エラー、3 = 停止。

40
呼び出し元へ制御が戻されます。

注 : ステップ 32 から 40 によって通常の終了ルーチンが構成されます。 異常終了の場合には、ステップ 34 から 35 は回避されます。

41
オーバーフロー標識 (OA から OG、OV) でオンのものがあるかどうかが判別され ます。 オーバーフロー標識がオンの場合には、プログラムはステップ 42 から続 行され、そうでない場合には、プログラムはステップ 43 に分岐します。

42
オーバーフロー・ルーチンが制御を受け取ります。(オーバーフロー・ルー チンについて詳しくは、111 ページの『オーバーフロー・ルーチン』を参照してください。) ENDSR 命令の 演算項目 2 に値 *OFL が入っている場合には、このステップがプロ グラム内の戻り点になります。

43
MR 標識がオンに設定され、これが複数ファイル・プログラムの場合、およ び処理するレコードが突き合わせレコードの場合には、突き合わせレコードを処理するサ イクルが完了するまでオンになったままです。 そうでない場合には、MR 標識はオ フに設定されます。

44
読み取った最後のレコードのデータを処理に使用できるようになります。 フィールド 標識の指定があれば、オンに設定されます。

45
先読みフィールドが指定された場合には、プログラムはステップ 46 から続 行され、そうでない場合には、プログラムはステップ 47 に分岐します。

46
先読みルーチンが制御を受け取ります。先読みルーチンについて詳しくは、111 ページの『先読みルーチン』を参照してください。

47
明細演算が処理されます。 ENDSR 命令の演算項目 2 に値 *DETC が入っている 場合には、このステップがプログラム内の戻り点になります。 プログラムはステップ 4 へ分岐します。

初期化サブルーチン

RPG IV 初期化サブルーチンについて詳しくは、104 ページの図 11 を参照してください。

初期化サブルーチンによって、1P 出力の前に演算仕様書を処理することができます。 プログラムの初期化時に実行される特定のサブルーチンは、そのサブルーチン の BEGSR 命令の演算項目 1 に *INZSR を指定することによって定義することがで きます。 初期化サブルーチンとして定義できるのは、1 つのサブルーチンだけ です。 それが呼び出されるのは、プログラム・サイクルのプログラム初期化ステップの終了時 (つまり、データ構造およびサブフィールドの初期化、外部標識およびユーザー・データ・フィールドの検索、グローバル・ファイルのオープン、データ域のデータ構造、配列、およびテーブルのロード、*ENTRY PLIST の演算項目 1 への PARM の結果フィールドの転送のすべてが実行された後) です。*INZSR をファイル/プログラム・エラー/例外サブルーチンとして指定することはできません。

プログラムが LR をオフにして終了した場合には、サブルーチンはプログラムの 初期化ステップの一部なので、初期化サブルーチンはそのプログラムの次の 呼び出し時には自動的に実行されません。 しかし、LR をオフにしたプログラムから出口が 作成される前に初期化サブルー チンが完了していない場合には、そのプログラムの次の呼び出し時に初期化サブ ルーチンが再実行されます。

初期化サブルーチンは、プログラムの初期化時に呼び出される点を除き、プ ログラム内の他のサブルーチンと類似のものです。 これは、EXSR または CASxx 命令によって呼び出したり、このサブルーチンから他のサブルーチンまたは他の プログラムを呼び出すことができます。 サブルーチン内で有効な命令は、RESET 命令を 除いて、初期化サブルーチン内 でもすべて有効です。 RESET 命令が例外であるのは、変数のリセットに使用され る値が、初期化サブルーチンが実行された後でなければ定義されないためです。

初期化サブルーチンの実行中に変数を変更すると、後続の RESET 命令でその変 数が設定される値に影響します。 レコード様式内のフィールドのデフォルトの値 については、たとえば、それを初期化サブルーチンに設定しておき、そのデフォルトの 値を使用するたびにレコード様式に対して RESET を使用することによって、定義 することができます。 初期化サブルーチンでは、また、1P 出力の現在の時刻な どの情報を検索することができます。

サブプロシージャーと関連した *INZSR はありません。 サブプロシージャーがモジュール 内で最初に呼び出されるプロシージャーであっ た場合には、他のグローバル・データの初期化は実行されても、メインプロシージ ャーの *INZSR は実行されません。 メイン・プロシージャーの *INZSR は、そのメイン・プロシージャーが呼び出され た時に実行されます。

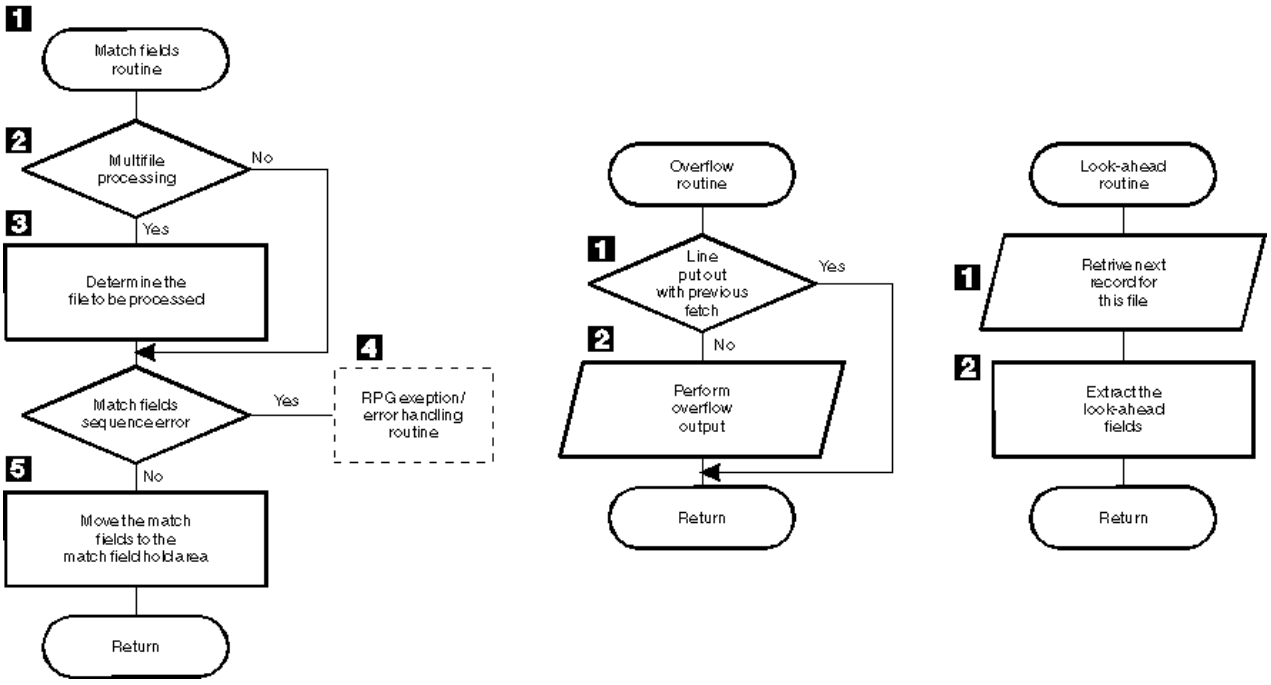


図 *13. RPG IV* 突き合わせフィールド、オーバーフロー、および先読みルーチンの詳細なフロー

突き合わせフィールド・ルーチン

110 ページの図 13 は、RPG IV 突き合わせフィールド・ルーチンの特定のステップを示しています。 以下の説明の項目番号は、図の中の番号を示しています。

1

複数ファイル処理を使用している場合には、処理はステップ 2 から続行され、 そうでない場合には、プログラムはステップ 3 に分岐します。

2

保留域の突き合わせフィールドの値がテストされ、次に処理するファイルが判別 されます。

3

RPG IV プログラムによって突き合わせファイルから突き合わせフィールドが抜き出され、順序検査が実行されます。 突き合わせフィールドの順序が正しければ、プログ ラムはステップ 5 に分岐します。

4

突き合わせフィールドの順序が違っている場合には、RPG IV 例外/エラー処理ルーチンが制御を受け取ります。

5

突き合わせフィールドがそのファイルの保留域に転送されます。 保留域は突き合わせ フィールドのあるファイルごとに、それぞれ用意されます。 突き合わせフィールドの値に基づいて、処理する次のレコードが選択されます。

オーバーフロー・ルーチン

110 ページの図 13 は、RPG IV オーバーフロー・ルーチンの特定のステップを示しています。 以下の説明の項目番号は、図の中の番号を示しています。

1
RPG IV プログラムは、フェッチ・オーバーフロー・ルーチン論理 (104 ページの図 11 のステップ 30) を使用して、前にオーバーフロー行が書き出されたかどうかを判別します。 オーバーフロー行が前に書き出されていた 場合には、プログラムは指定された戻り点に分岐し、そうでない場合には、処理 はステップ 2 から続行されます。

2
オーバーフロー標識によって条件付けされたすべての出力行がテストされ、 条件付けされたオーバーフロー行が書き出されます。

フェッチ・オーバーフロー・ルーチンによって、ミシン線上への印刷を防止し、ページをできるだけ有効に使用できるように、基本 RPG IV オーバーフローの論理を変更することができます。 通常のプログラム・サイクル中は、オーバーフロー標識がオンになっているかどうかを確認するために、合計出力の直後に RPG IV プログラムによって一度のみ検査が行われます。 フェッチ・オーバーフロー機能を指定した場合には、フェッチ・オーバーフローが指定されている行ごとに RPG IV プログラムによるオーバーフローの検査が行われます。

フェッチ・オーバーフローは、出力仕様で PRINTER ファイルに関する任意 の明細行、合計行、または例外行の 18 桁目に F を記入して指定し ます。 フェッチ・オーバーフロー・ルーチンによって、次のページへ用紙が進められる ことは自動的になくなります。

出力時には、出力行の条件付け標識がテストされ、その行が書き出されるかど うかが判別されます。 それが書き出される行で、18 桁目に F が指定されている場合には、オーバーフロー標識がオンになっているかどうかが RPG IV プログラムによってテストされ判別されます。 オーバーフロー標識がオンのときは、オーバーフロー・ルーチンが 取り出され、以下の処理がなされます。

- 取り出しが指定されたファイルのオーバーフロー行だけが出力について検査さ れます。
- オーバーフロー標識によって条件付けされた合計行はすべて書き出されます。
- 現在印刷中の行より小さい番号の行へのスキップが、オーバーフロー標識に よって条件付けされている行に指定されている場合に、用紙が次のページに進め られます。
- オーバーフロー標識によって条件付けされている見出し行、明細行、および 例外行は書き出されます。
- オーバーフロー・ルーチンを取り出した行は書き出されます。
- そのプログラム・サイクルで書き出されることになっている明細行および 合計行はすべて書き出されます。

各 OR 行の 18 桁目は、オーバーフロー・ルーチンが OR 関係で各レコードに使われるものである 場合は、F でなければなりません。 フェッチ・オーバーフローは、オーバーフロー標識が 同じ仕様書行の 21 から 29 桁目で指定されていると使われることはありま せん。 これを使用した場合には、オーバーフロー・ルーチ ンは取り出されません。

オーバーフロー標識によって条件付けされた残りの明細行、合計行、例外行、 および見出し行を印刷するのに十分なスペースがページ上に残されていない場合 には、フェッチ・オーバーフロー・ルーチンを使用してください。 オーバーフロ ー・ルーチンの取り出し時期を決定するためには、考えられるオーバーフローのす べての状況を検討してください。 行およびスペースをカウントすることによって、各明細行、合計行、および例外 行でオーバーフローが起こるとどうなるかを推測することができます。

先読みルーチン

110 ページの図 13 は、RPG IV 先読みルーチンの特定のステップを示しています。 以下の説明の項目番号は、図の中の番号を示しています。

1
処理中のファイルの次のレコードが読み取られます。 しかし、入出力共用ファ イルまたは更新ファイル (ファイル仕様書の 17 桁目の C または U によっ てそれぞれ識別される) の場合には、処理中の現行レコードから先読みフィール ドが抜き出されます。

2
先読みフィールドが抜き出されます。

1 次ファイルのないプログラムの終了

プログラムに 1 次ファイルが含まれていない場合には、プログラム の終了を次の方法によって指定しなければなりません。

- LR 標識をオンに設定する
- RT 標識をオンに設定する
- H1 から H9 標識をオンに設定する
- RETURN 命令コードを指定する

LR、RT、H1 から H9 標識、および RETURN 命令コードは、相互に組み合わせて使 用することができます。

ファイル処理のプログラム制御

プログラムの入力のすべてまたは一部を制御するためには、全手順ファイル (固定形式ファイル記述仕様書の 18 桁目に F があるか、 または、自由形式ファイル定義で定義されたすべてのファイル) を指定します。全手順ファイルでは、 入力 がプログラム指定の演算命令 (たとえば、READ、CHAIN) によっ て制御されることを指示します。 全手順ファイルと 1 次ファイル (フ ァイル仕様書の 18 桁目に P) の両方がプログラムに指定されている場合に は、入力の一部はプログラムによって制御され、その他の入力はサイクルによって制御されます。 モジュールにプログラム・サイクルが存在する場合でも、全手順ファイルの処理は演算ですべて行われます。

ファイル命令コードを入力のプログラム制御に使用することができます。これらのファイル命令コードは、576 ページの『ファイル操作』で説明されています。

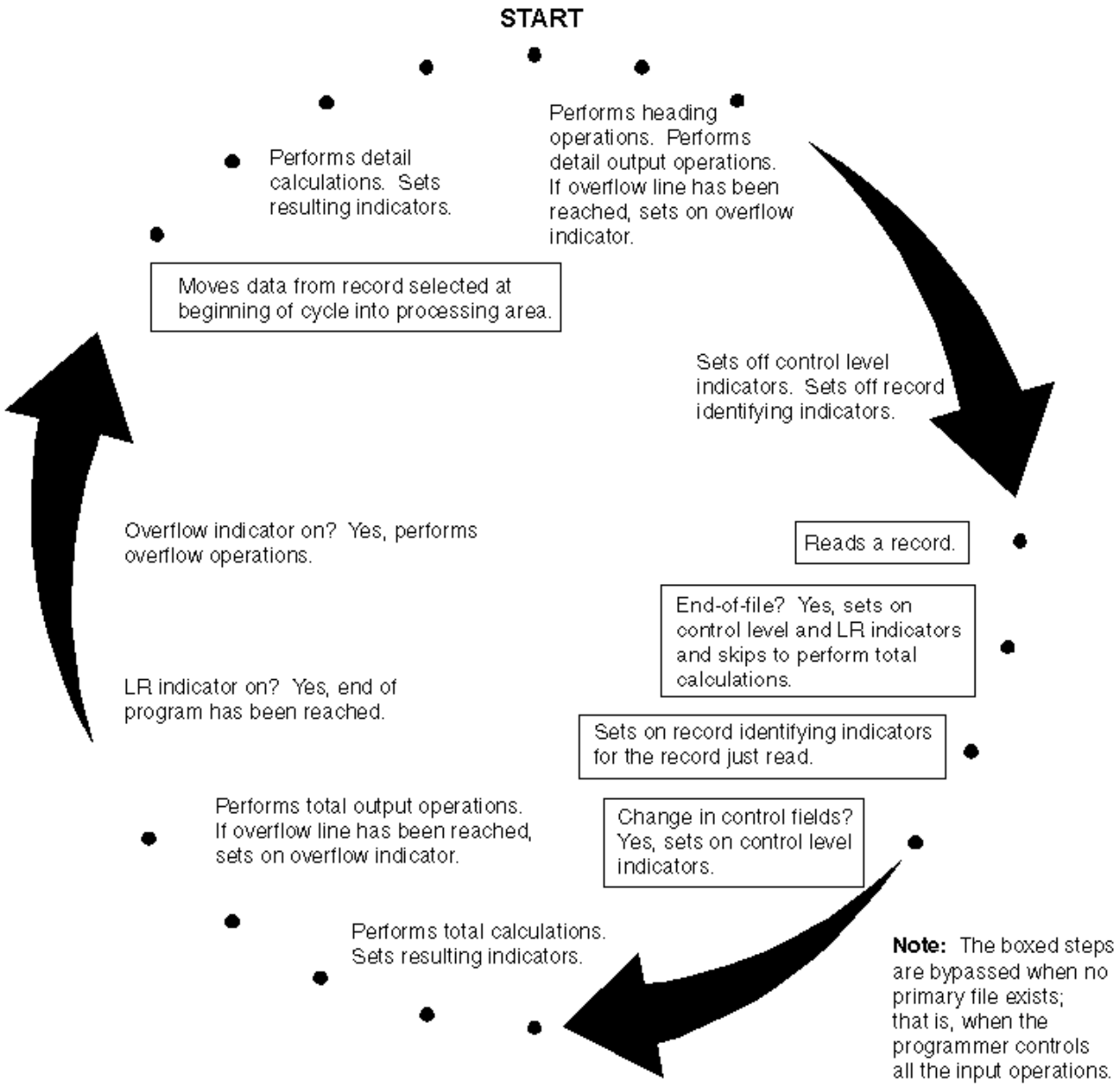


図 *14.* プログラム・サイクル内の入力操作のプログラマー制御

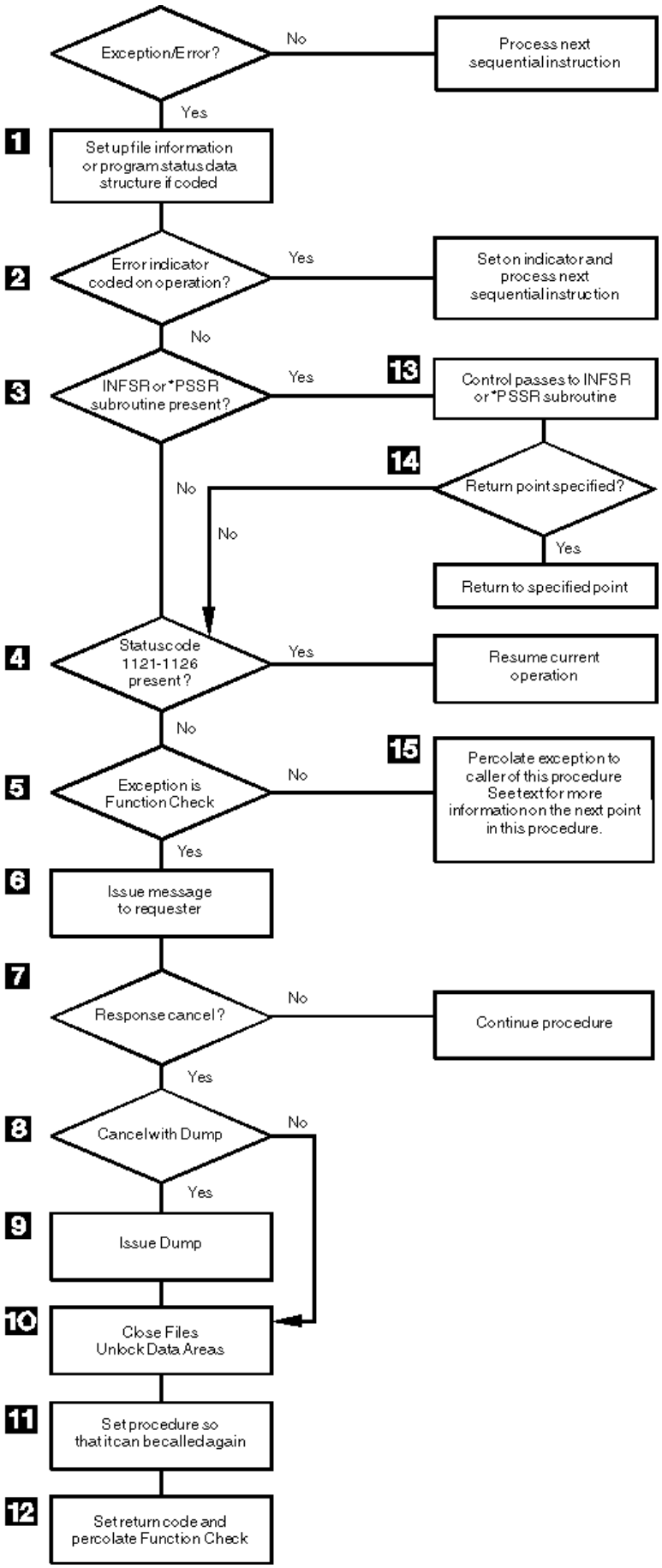


図 *15. RPG IV* 例外/エラー処理ルーチンの詳細フロー

RPG IV 例外/エラー処理ルーチン

114 ページの図 15 は、RPG IV 例外/エラー処理ルーチン内の特定のステップを示しています。 以下の説明の項目番号は、図の中の 番号を示しています。

1
ファイル情報データ構造またはプロシージャー状況データ構造が指定された 場合には、それが状況情報によってセットアップされます。

2
例外/エラーが 73 から 74 桁目に標識の指定がある命令コードで起こった場合 には、標識がオンに設定され、演算の次の順序の命令に制御が戻されます。

3
該当する例外/エラー処理サブルーチン (INFSR または *PSSR) がプロシージ ャーに存在する場合には、プロシージャーはステップ 13 に分岐し、そうでない 場合には、プロシージャーはステップ 4 から続行されます。

4
状況コードが 1121 から 1126 の場合には (158 ページの『ファイル状況コード』を参照)、演算の現在の命令に制御が戻されます。 そうでない場合には、プロシージャーはステップ 5 から続行されます。

5
例外が機能チェックである場合には、プロシージャーはステップ 6 から続行されます。 そうでない場合には、ステップ 15 に分岐します。

6
照会メッセージが要求元端末に出されます。 対話式ジョブの場合には、メッ セージは要求元に送られます。 バッチ・ジョブの場合には、メッセージは QSYSOPR に送られます。 QSYSOPR が中断モードでなければ、デフォルトの応答が出さ れます。

7
ユーザーの応答がプロシージャーを取り消すためのものである場合には、プロシージャーはステップ 8 から続行されます。 そうでない場合には、プロシージャーは 続行されます。

8
ユーザーの応答がダンプをとってプロシージャーを取り消すためのものであ る場合には、プロシージャーはステップ 9 から続行されます。 そうでない場合 には、プロシージャーはステップ 10 に分岐します。

9
ダンプが出力されます。

10
すべてのグローバル・ファイルがクローズされ、データ域がアンロックされます。

11
プロシージャーは再び呼び出せるように設定されます。

12
戻りコードが設定され、機能チェックがパーコレートされます。

13
制御が例外/エラー処理サブルーチン (INFSR または *PSSR) に渡されます。

14
例外/エラー処理サブルーチンの ENDSR 命令の演算項目 2 に戻り点が指定され ている場合には、プロシージャーは指定された戻り点に進みます。 戻り点が指定 されていない場合には、プロシージャーはステップ 4 に進みます。 ENDSR 命令の演算項目 2 にフィールド名が指定されており、その内容が RPG IV 定義の戻り点 (*GETIN または *DETC など) の 1 つではない場合には、プロシージャーはステップ 6 に進みます。 エラーは示されず、元のエラーは演算項目 2 の指定がブラ ンクであるかのように処理されます。

15
呼び出しで例外が処理されない場合は、機能チェックにプロモートされ、プロシージャーはステップ 5 に分岐します。 それ以外の場合は、処理プログラムによって取られた処置に応じて、ステップ 10 か、または例外が発生したポイントの後の次の機械命令のいずれかでこのプロシージャーにおける制御が再開されます。

サブプロシージャー演算

サブプロシージャーの場合、サイクル・コードは生成されないため、サイクル・メイン・プロシージャーとは異なる方法でコーディングする必要があります。 サブプロシージャーは、次の 1 つが起こった時に終了します。

- RETURN 命令が処理された。
- サブプロシージャーの本体の最後の演算が処理された。

116 ページの図 16 は、サブプロシージャーの通常の処理ステップを示しています。117 ページの図 17 は、例外/エラーの処理順序を示しています。

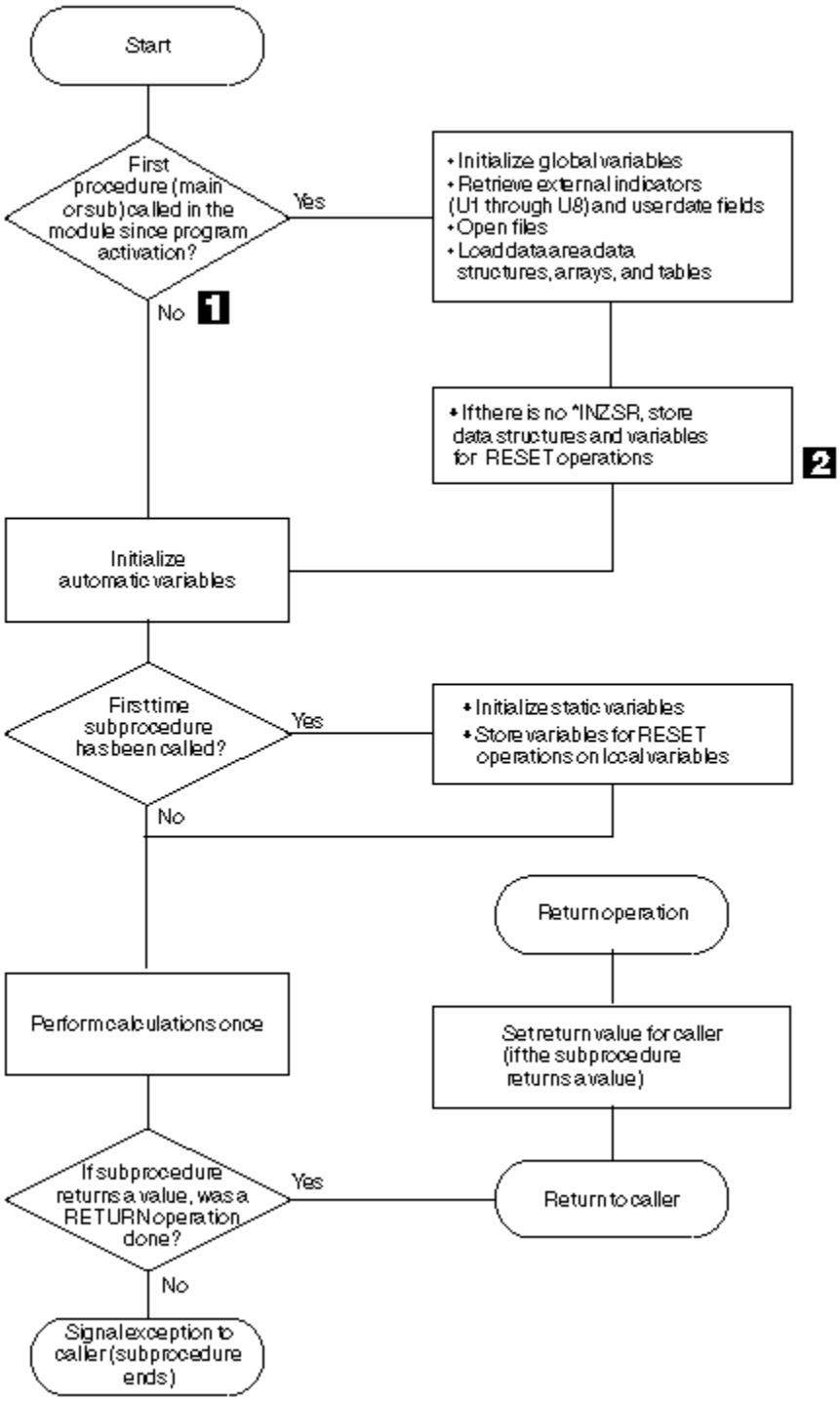


図 *16.* サブプロシージャーの通常の処理順序

1

「いいえ」の分岐をとることは、プログラムが活動化されていたために、別 のプロシージャーがすでに呼び出されていることを意味します。 別のプロシージャーではファイルがクローズされていたり、データ域がアンロッ クされていたりすることがあるため、ファイル、データ域などの状態について間 違った前提をとっていないことを確認しなければなりません。

2

モジュールのどこかでメイン・プロシージャーへの入り口パラメーターが RESET にな っている場合には、これによって例外が起こります。 サブプロシージャーをメイン・プロシージャーの前に呼び出すことが可能な場合、サイクル・メイン・プロシージャーの入り口パラメーターを RESET しないことをお勧めします。

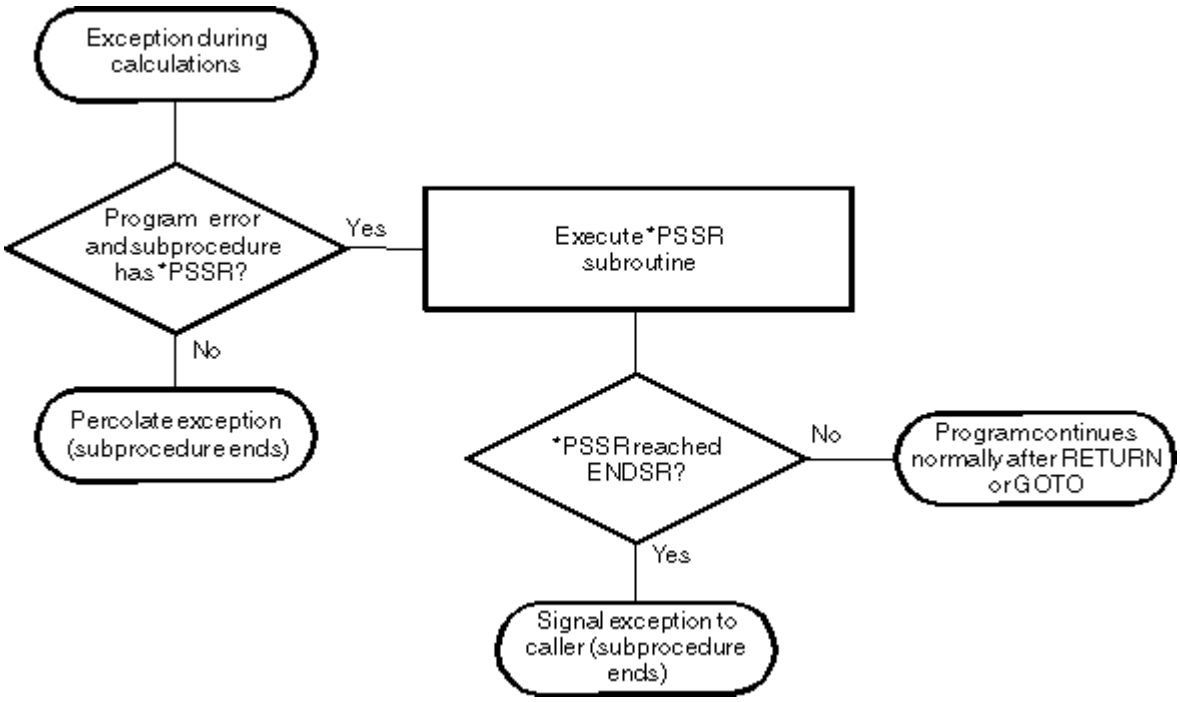


図 *17.* サブプロシージャーの例外/エラーの処理順序

以下は、サブプロシージャーのコーディング時に考慮すべきいくつかの点です。

- サブプロシージャーと関連した *INZSR はありません。 データは、サブプロシージャーが初めて呼び出された時に、演算を開始する前に (INZ 値かまたはデフォルトの値によって) 初期化されます。

  モジュール内で呼び出される最初の プロシージャーがサブプロシージャーである場合、他のグローバル・データの初期化は実行されますが、サイクル・メイン・プロシージャーの *INZSR (存在する場合) は実行されないので、注意してください。 サイクル・メイン・プロシージャーの *INZSR は、そのサイクル・メイン・プロシージャーが呼び出された時に実行されます。
- サブプロシージャーが通常戻り値を戻してきた時に、呼び出されるプログ ラムまたはプロシージャーのプロトタイプに指定されていれば、その戻り値が呼び出し元に渡されます。 自動的には何も行われません。 ファイルおよびデータ域はすべて手操作でクローズしなければなりません。 ファイルへは手操作で書き出さなければなりません。 LR 標識または RT 標識を設定することはできますが、プログラム終了処理に対する即効性はありません。 サブプロシージャーがサイクル・メイン・プロシージャーによって呼び出された場合、設定された LR 標識または RT 標識は、RPG が各標識をチェックするポイントに RPG サイクルが達したときに、有効になります。
- サブプロシージャー内での例外処理は、サイクル・メインプロシージャーとは基本的に異なるものになります。 サブプロシージャー用のデフォルトの例外処理プログラムは存在しないため、このような状況ではサブプロシージャーの異常終了と対応して、サイクル・メイン・プロシージャーのデフォルトの処理プログラムが呼び出されるからです。 例えば、サブプロシージャーの中の *PSSR サブルーチンのための ENDSR 命令の 演算項目 2 はブランクでなければなりません。 サイクル・メイン・プロシージャー内の *PSSR サブルーチンの ENDSR でブランクの演算項目 2 が発生した場合、デフォルトの処理プログラムに

制御が渡されます。 サブプロシージャーにおいて *PSSR サブルーチンの ENDSR に達した場合、そのサブプロシージャーは異常終了し、サブプロシージャーの呼び出し元に RNX9001 信号が送られます。

異常終了は、*PSSR に RETURN 命令をコーディングするか、あるいはサブプロシ ージャーに処理を続行するための GOTO およびラベルをコーディングすることに よって、避けることができます。

- *PSSR エラー処理サブルーチンは、サブプロシージャーにとってローカルなものです。
- INFSR をサブプロシージャーにコーディングすることはできません。 また、INFSR がコーディングされたファイルを使用することもできません。
- サイクルを制御する標識は、リニア・モジュールで使用される (制御仕様書における MAIN または NOMAIN) 場合、あるいは活動中のサブプロシージャーで使用される (ただしモジュールのサイクル・メイン・プロシージャーが存在しない) 場合には、条件付け標識としてのみ機能します。 サイクルを制御する標識には、LR、RT、H1 から H9、および制御レベル標識があります。

ファイルの暗黙的なオープンおよびデータ域のロック

USROPN キーワードおよび UDS データ域を持たないグローバル・ファイルは、モジュールの初期化およびサイクル・メイン・プロシージャーの初期化の際に、暗黙的にオープンまたはロックされます。 USROPN キーワードを持たないサブプロシージャー内の静的ファイルは、サブプロシージャーの最初の呼び出し時に暗黙的にオープンされます。 USROPN キーワードを持たないサブプロシージャー内の自動ファイルは、プロシージャーが呼び出されるたびにオープンされます。

暗黙的なファイルのクローズおよびデータ域のアンロック

サイクル・メイン・プロシージャーが異常終了したり、LR オンの状態で終了したりした場合は、サイクル・メイン・プロシージャーの終了時に、オープンしているグローバル・ファイルは暗黙的にクローズされ、ロックされているデータ域は暗黙的にアンロックされます。 サブプロシージャー内の自動ファイルは、サブプロシージャーが正常終了しても異常終了しても、暗黙的にクローズされます。

注意: サブプロシージャー内の静的ファイルの場合、暗黙的にクローズされることはありません。 リニア・モジュール内では、グローバル・ファイルのクローズおよびデータ域の暗黙的なアンロックは起こりません。 これらのオブジェクトは、クローズやアンロックが明示的に行われるまで、オープンされたままあるいはロックされたままになります。

RPG IV 標識

標識は、'1' (オン) または '0' (オフ) のいずれかが入れられる 1 バイトの文字 です。 これは、一般に、命令の結果を示すか、あるいは命令の処理を 条件付け (制御) するために使用されます。

標識形式は、標識変数を定義するために定義仕様書で指定することができます。 標識形式で文字データを定義する方法については、250 ページの『文字形式』および 410 ページの『40 桁目 (内部データ・タイプ)』を参照してください。 この章では、定義済みの RPG IV 標識 (*INxx) の特殊な設定について説明します。

RPG IV 標識は、仕様書の記入項目か RPG IV プログラム自体のいずれかによって定義されます。 標識が定義された仕様書上の桁によって、 標識の使用法が決まります。 定義された標識は、その後、演算および出力命令の条件付けに使用することがで きます。

RPG IV プログラムは、プログラム・サイクル中の特定の時点に、ある種の標識を設定およびリセットします。 さらに、多くの標識の状態を、演算命令 によって変更することができます。 MR、1P、KA から KN、および KP から KY を除くすべての 標識は、SETON 命令コードによってオンに設定することができ、MR および 1P を除くすべての 標識は、SETOFF 命令コードによってオフに設定することができます。

この章は、以下のトピックに分けられています。

- RPG IV 仕様書で定義される標識
- RPG IV 仕様書で定義されない標識
- 標識の使用
- データとして参照される標識

RPG IV 仕様書で定義される標識

RPG IV 仕様書には、以下の標識を指定することができます。

- オーバーフロー標識 (ファイル仕様書 での OFLIND キーワード)
- レコード識別標識 (入力仕様 の 21 から 22 桁目)
- 制御レベル標識 (入力仕様 の 63 から 64 桁目)
- フィールド標識 (入力仕様 の 69 から 74 桁目)
- 結果標識 (演算仕様書 の 71 から 76 桁目)
- *IN 配列、*IN(xx) 配列要素、または *INxx フィールド (これらの予約語の 1 つを使用して標識を定義する方法の説明については、141 ページの『データとして参照される標識』を参照してください。)

定義された標識は、次に、プログラム中の命令を条件付けするために使用する ことができます。

オーバーフロー標識

オーバーフロー標識は、ファイル仕様書の OFLIND キーワードによって定 義されます。 これは、ページの最後の行を印刷したりまたは通過した時に設定されます。 有 効な標識は *INOA から *INOG、*INOV、および *IN01 から *IN99 です。 定義されたオー バーフロー標識は、次に、演算および出力命令を条件付けするために使用するこ とができます。オーバーフロー標識およびフェッチ・オーバーフロー・ルーチンの論理の説明は、111 ページの『オーバーフロー・ルーチン』にあります。

レコード識別標識

レコード識別標識は、入力仕様の 21 から 22 桁目の記入項目によって定義され 、対応するレコード・タイプが処理のために選択された時にオンに設定されます。 その標識は、次に、特定の演算および出力命令を条件付けするために使用するこ とができます。 レコード識別標識は、特定の順序で割り当てる必要はありません。

有効なレコード識別標識は次のとおりです。

- 01 から 99
- H1-H9
- L1-L9
- LR
- U1-U8
- RT

外部記述ファイルの場合には、レコード識別標識は任意指定ですが、指定する場 合には、プログラム記述ファイルの場合と同じ規則に従ってください。

一般に、標識 01 から 99 がレコード識別標識として使用されます。 しかし、制御レベル標識 (L1 から L9) および最終レコード標識 (LR) を使用するこ ともできます。 レコード識別情報として L1 から L9 を指定した場合には、より低いレベルの標識は オンに設定されません。

レコード・タイプを処理のために選択した時に、対応するレコード識別標識が オンに設定されます。 その他すべてのレコード識別標識は、明細および合計の 演算時にファイルからレコードを検索するためにファイル命令コードが使用され ている場合 (以下を参照) を除き、オフに設定されます。 レコード識別標識は、レコードが 選択された後、入力フィールドが入力域へ転送 される前にオンに設定されます。 新しいレコードのレコード識別標識は、古い レコードの合計時にオンに設定されます。古いレコードのフィールドを使用し て合計時に処理される演算を、その古いレコードのレコード識別標識を使用して 条件付けすることはできません。 プログラム・サイクルの任意の時点に標識をオフに設定することができます。 それらは、次の 1 次または 2 次レコードが選択される前にオ フに設定されます。

演算仕様書でファイル命令コードを使用して レコードを検索する場合には、フ ァイルからレコードが検索されると同時にレコード識別標識がオンに設定され ます。 このレコード識別標識は、RPG IV サイクル中の適切な点になるまで、オフに設定されません(113 ページの図 14 を参照してください。) したがって、同じ RPG IV プログラム・サイクル内で同じファイルに対して複数の命令を出している場合には、同じファイル

に対する複数のレコード識別標識だけでなく、レコード不在標識も同時にオンに設定される可能性があります。

レコード識別標識の割り当てに関する規則

プログラム記述ファイル中のレコードにレコード識別標識を割り当てる場合には、 以下の点に留意してください。

- すべてのレコード・タイプに対して同じ命令を処理する場合には、2 つ以上 の異なるレコード・タイプに同じ標識を割り当てることができます。 そのためには、21 から 22 桁目にレコード識別標識を指定し、各種のレコード・タイ プのレコード識別コードを OR 関係で指定します。
- レコード識別標識を AND 関係で関連付けることができますが、そのレコード 識別標識はグループの最初の行に指定しなければなりません。 AND 行にレコード識別標識を 指定することはできません。
- 不定形式レコード (プログラム記述ファイル中のレコードで、23 から 46 桁目に レコード識別コードで記述されていないもの) は、プログラムの停止の原因とな ります。
- 1 つのレコード識別標識を、別のレコード・タイプのレコード識別標識とし て、フィールド標識として、または結果の標識として指定することができます。 診断メッセージは出されませんが、こうした標識の使用法は、間違った結果の原 因となることがあります。

外部記述ファイル中のレコードにレコード識別標識を割り当ている場合には、以 下の点に留意してください。

- AND/OR 関係をレコード様式名と一緒に使用することはできませんが、同 じレコード識別標識を複数のレコードに割り当てることはできます。
- 7 から 16 桁目には、ファイル名でなく、レコード様式名を指定しなければなり ません。

レコード識別標識の例については、120 ページの図 18 を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC..............................
I.....................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
 *
I*Record identifying indicator 01 is set on if the record read
I*contains an S in position 1 or an A in position 1.
IINPUT1    NS  01    1 CS
I         OR         1 CA
I                                1   25  FLD1
 * Record identifying indicator 02 is set on if the record read
 * contains XYZA in positions 1 through 4.
I          NS  02    1 CX    2 CY    3 CZ
I         AND        4 CA
I                                1   15  FLDA
I                               16   20  FLDB
 * Record identifying indicator 95 is set on if any record read
 * does not meet the requirements for record identifying indicators
 * 01 or 02.
I          NS  95
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IRcdname+++....Ri.....................................................
 *
 * For an externally described file, record identifying indicator 10
 * is set on if the ITMREC record is read and record identifying
 * indicator 20 is set on if the SLSREC or COMREC records are read.
IITMREC         10
ISLSREC         20
ICOMREC         20
```

図 *18.* レコード識別標識の例

制御レベル標識 (L1 から L9)

制御レベル標識は、入力仕様の 63 から 64 桁目の記入項目によって定義され、 ある入力フィールドを制御フィールドとして指定するものです。 これは、次に、演算および出力命令を条件付けするために使用することができま す。 有効な制御レベル標識は L1 から L9 です。

制御レベル標識は、ある入力フィールドを制御フィールドとして指定するもの です。 制御フィールドが読み取られると、制御フィールドのデータは、前のレコードか らの同じ制御フィールドのデータと比較されます。 データが異なっている場合に は、制御の切れ目が起こり、この制御フィールドに割り当てられた制御レベル標 識がオンに設定されます。 次に、制御レベル標識を使用して、制御フィールドに同じ 情報が入っているすべ てのレコードが読み取られた時にだけ処理されるように、命令を条件付けするこ とができます。 標識は、合計と最初の明細の両方の時点でオンになったままであ るため、合計印刷 (制御グループの最後のレコード) および明細印刷 (制御グル ープの最初のレコード) の条件付けにも使用することができます。 制御レベル標 識は、次のレコードが読み取られる前にオフに設定されます。

制御フィールドが入っている最初のレコードが 読み取られた後で、制御の切れ 目が起こる可能性があります。 このレコードの制御フィールドは、16 進数のゼロが入って いる記憶域内の区域と 比較されます。 2 つの異なるレコードからのフィールドが比較されているわけで はないので、このサイクルの合計演算および合計出力命令は回避されます。

制御レベル標識は、その重要度の順に、最低の L1 から最高の L9 までランク づけされます。 制御の切れ目の結果としてより高いレベルの標識がオンに設定 されると、それより低いレベルのすべての標識がオンに設定されます。 ただし、より 低いレベルの標識は、定義されている場合にしかプログラム中で使 用することができません。 例えば、制御の切れ目によって L8 がオンに設定された 場合には、L1 から L7 もオンに設定されます。 LR (最終レコード) 標識は、入力ファイルがファイル の終わりになった時にオンに設定されます。 LR は最高レベルの標識と見なされ、 L1 から L9 も強制的にオンに設定されます。

制御レベル標識は、レコード識別標識または結果の標識として定義することも できます。 この方法で制御レベル標識を使用した場合には、高いレベルの標識が オンに設定されても、それより低いレベルの標識の状況は変更されません。 例えば、L3 が結果の標識として使用されている場合には、L3 がオンに設定さ れても、L2 および L1 の状況は変わりません。

他のフィールドとの関係での制御フィールドの重要度に よって、制御レベル標 識を割り当ている方法が決まります。 例えば、小計を必要とするデータには、最 終合計を必要とするデータより低い制御レベル標識が必要です。各部門の従業員をグループ分けする場合に、部門番号が入っている制御フィールドには、従業員番号が入っている制御フィールドより高い制御レベル標識が必要です (122 ページの図 19 を参照)。

制御レベル標識に関する規則

制御レベル標識を割り当ている場合には、以下の点に留意してください。

- 制御フィールドは、1 次または 2 次ファイルに対してだ け指定することができます。
- 制御フィールドを、全手順ファイル、 タイプが 2 進数、整数、符号なしまたは浮動の数値入力フィールド あるいは先読みフィールドに対して指定することはできません。
- 入力仕様の 49 から 62 桁目に配列名が指定された時には、制御レベル標識を 使用することはできません。しかし、配列要素と一緒に制御レベル標識を使 用することはできます。 制御レベル標識は、ヌル値可能フィールドに使用することはできません。
- 制御レベル比較操作の処理は、レコードが入っていたファイルにかかわりな く、レコードが見付かった順序で行われます。
- 異なるレコード・タイプまたは異なるファイルに同じ制御レベル標識を使用する場合には、その制御レベル標識と関連した制御フィールドは、同じ長さである必要があります (122 ページの図 19 を参照)。ただし、日付、時刻、およびタイム・スタンプ・フィールドについてはタイプが一致しているのみで (つまり、形式が異なっていても) かまいません。
- 制御レベル標識フィールドの長さは、制御レベル標識のレコード内における 長さです。 例えば、L1 がレコード内で 10 バイトのフィールド長を持っていれ ば、L1 の制御レベル標識フィールドの長さは 10 桁です。

  分割制御フィールドの場合の制御レベル標識フィールドの長さは、ある制御レ ベル標識と関連した、レコード内のすべてのフィールドの長さの合計です。 L2 が 長さ 12 バイト、2 バイト、および 4 バイトの 3 つのフィールドから構成される 分割制御フィールドを持っている場合には、L2 の制御レベル標識フィールドの長 さは 18 桁になります。

  複数のレコードが同じ制御レベル標識を使用する場合、制御レベル標識フィー ルドの長さは、1 つのレコードだけの長さです。それらのレコードのすべての長 さの合計ではありません。

プログラム内では、すべての制御レベル標識の制御レベル標識フィールドの長 さの合計は 256 桁以内でなければなりません。

- レコード・タイプが同じ場合には、異なる制御レベル標識を割り当てた制御フィールド内のレコードの位置がオーバーラップしていても差し支えありません (123 ページの図 20 を参照)。 制御フィールドまたは突き合わせフィールドを必要とするレコード・タイプの場合に は、制御フィールドまたは突き合わせフィールドの合計の長さは 256 桁以内でなけれ ばなりません。 例えば、123 ページの図 20 では、制御レベルに 15 桁が割り当てられています。
- 制御レベルの命令では、フィールド名は無視されます。 したがって、同じ制 御レベル標識を割り当てた異なるレコード・タイプのフィールドが、同じ名前であ っても差し支えありません。
- 制御レベルは、決まった順序で記入する必要はありません。 L2 を L1 より前 に表すことができます。 より低いレベルの標識をすべて割り当ている必要もあり ません。
- 1 つのファイル中の異なるレコード・タイプの制御フィールドが同数でない 場合には、不要な制御の切れ目が生ずることがあります。

124 ページの図 21 は、このような不要な制御の切れ目を防ぐ方法の例を示しています。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
A* EMPLOYEE MASTER FILE -- EMPMSTL
A          R EMPREC                    PFILE(EMPMSTL)
A            EMPLNO         6
A            DEPT           3
A            DIVSON         1
A*
A*                 (ADDITIONAL FIELDS)
A*
A          R EMPTIM                    PFILE(EMPMSTP)
A            EMPLNO         6
A            DEPT           3
A            DIVSON         1
A*
A*                 (ADDITIONAL FIELDS)
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC.............................
I.....................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
 *
 *  In this example, control level indicators are defined for three
 *  fields.  The names of the control fields (DIVSON, DEPT, EMPLNO)
 *  give an indication of their relative importance.
 *  The division (DIVSON) is the most important group.
 *  It is given the highest control level indicator used (L3).
 *  The department (DEPT) ranks below the division;
 *  L2 is assigned to it.  The employee field (EMPLNO) has
 *  the lowest control level indicator (L1) assigned to it.
 *
IEMPREC       10
I                                       EMPLNO        L1
I                                       DIVSON        L3
I                                       DEPT          L2
 *
 *  The same control level indicators can be used for different record
 *  types.  However, the control fields having the same indicators must
 *  be the same length.  For records in an externally described file,
 *  the field attributes are defined in the external description.
 *
IEMPTIM       20
I                                       EMPLNO        L1
I                                       DEPT          L2
I                                       DIVSON        L3
```

図 *19.* 制御レベル標識 *(2* つのレコード・タイプ*)*

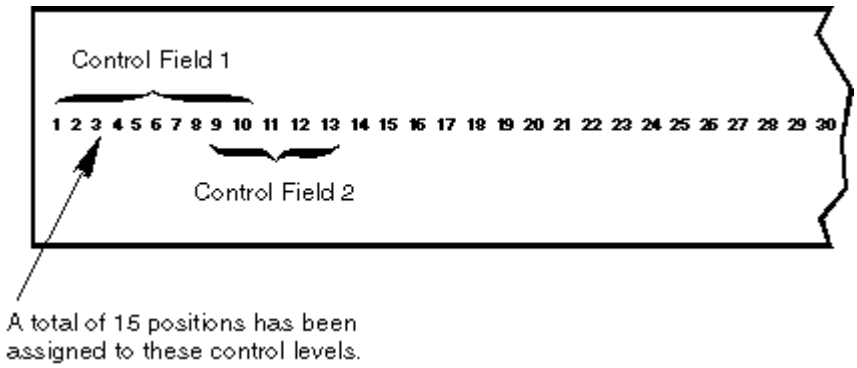


図 *20.* オーバーラップする制御フィールド

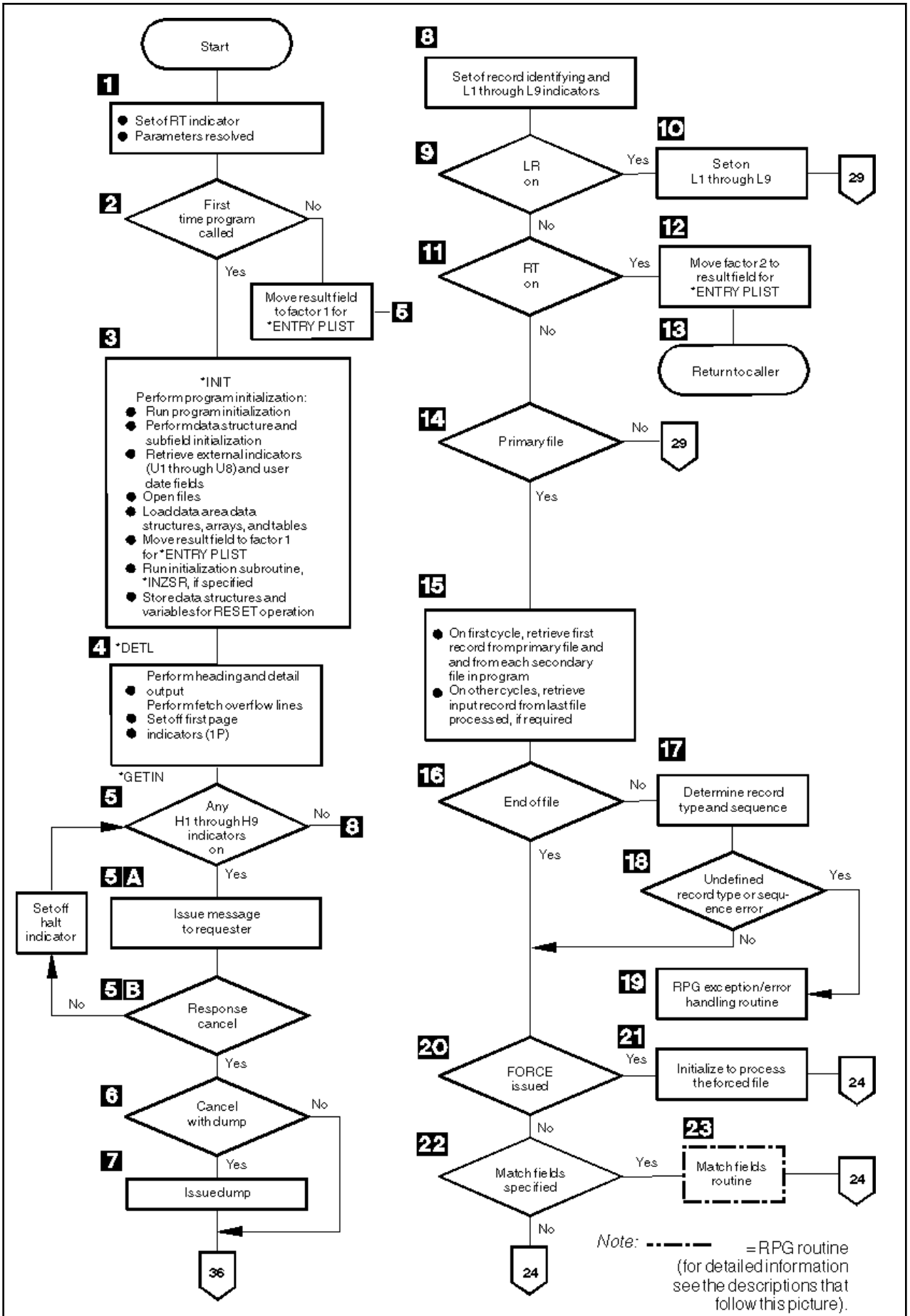
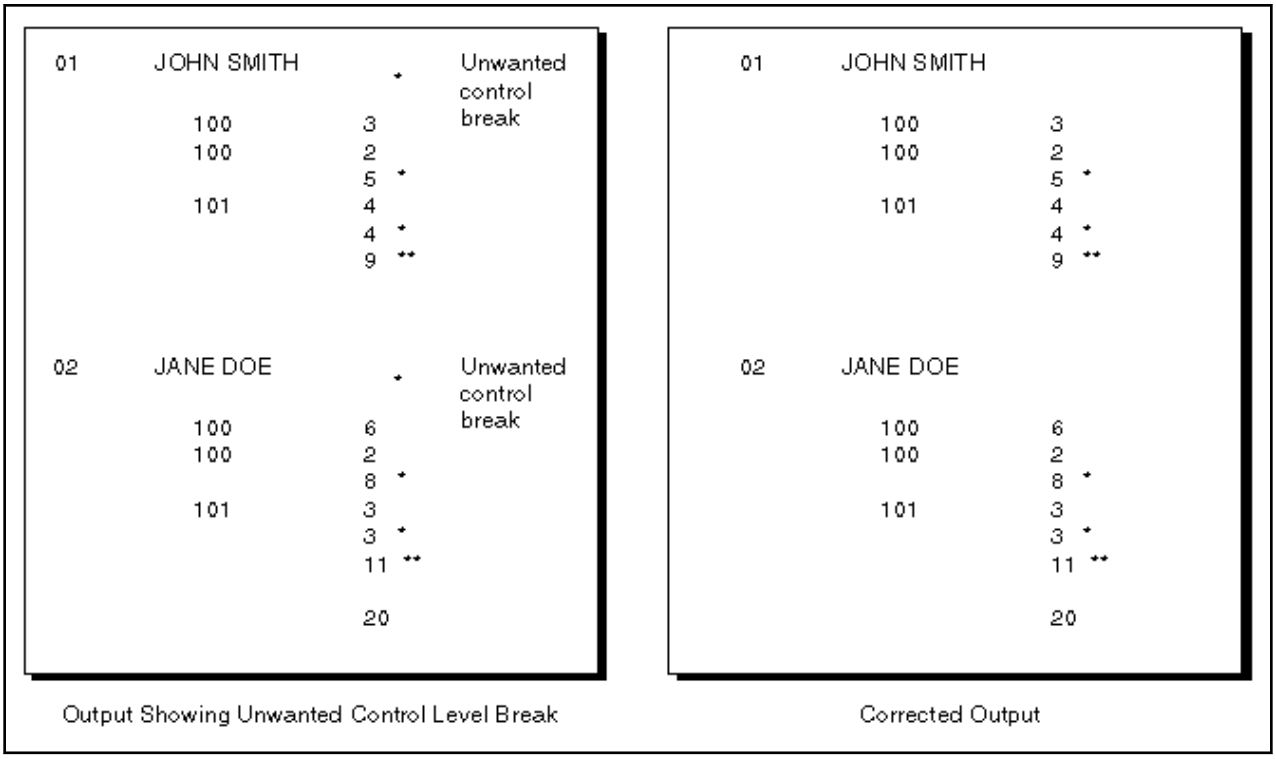


図 *21.* 不要な制御の切れ目を防ぐ方法

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC..............................
I.....................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
ISALES         01
I                                  1    2  L2FLD         L2
I                                  3   15  NAME
IITEM          02
I                                  1    2  L2FLD         L2
I                                  3    5  L1FLD         L1
I                                  6    8  AMT
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
 *  Indicator 11 is set on when the salesman record is read.
 *
C   01              SETON                                      11
 *
 *  Indicator 11 is set off when the item record is read.
 *  This allows the normal L1 control break to occur.
 *
C   02              SETOFF                                     11
C   02AMT           ADD       L1TOT         L1TOT            5 0
CL1   L1TOT         ADD       L2TOT         L2TOT            5 0
CL2   L2TOT         ADD       LRTOT         LRTOT            5 0
 *
```

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+..........................
O.............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat
OPRINTER   D    01                 1  1
O                       L2FLD                5
O                       NAME                25
O          D    02                    1
O                       L1FLD               15
O                       AMT           Z     15
 *
 *  When the next item record causes an L1 control break, no total
 *  output is printed if indicator 11 is on.  Detail calculations
 *  are then processed for the item record.
 *
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+..........................
O.............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat
O          T    L1N11              1
O                     L1TOT          ZB    25
O                                          27 '*'
O          T    L2                 1
O                     L2TOT          ZB    25
O                                          28 '**'
O          T    LR                 1
O                     LRTOT          ZB    25
```



レコード・タイプが異なっていても、通常は同数の制御フィールドがあります。 しかし、アプリケーションによっては、一部のレコードに異なる数の制御フィールドが 必要な場合があります。

販売担当者レコードには、L2 の制御フィールドしかありません。 品目レコード には、L1 と L2 の両方の制御フィールドがあります。 通常の RPG IV のコーディングでは、販売担当者レコードに続く最初の品目レコードによって不要な制御の切れ目が生じます。 これは、販売担当者レコードの直後の L1 の制御の切れ目によって認識され、その結果、販売担当者レコードの下の行に アスタリスクが印刷されます。

- 数値制御フィールドは、ゾーン 10 進数形式で比較されます。 パック形式の数値入力フィールドの長さは次の式により決まります。

  ```
  d = 2n - 1
  ```

  ここで、d = フィールドの桁数、および n = 入力フィールドの長さです。 パック形式の数値フィールドの桁数は常に奇数です。したがって、パック形式の 数値フィールドをゾーン 10 進数値フィールドと比較する時には、ゾーン形式の フィールドの長さは奇数でなければなりません。
- 制御の切れ目が起こっているかどうかを判別するために、小数点以下の桁数 を持つ数値制御フィールドが比較される場合には、常に小数点以下の桁数はない ものとして取り扱われます。 例えば、3.46 は 346 と等しいと見なされます。
- フィールドを数値として指定する場合には、正の数値だけが制御の切れ目が 起こったかどうかを判別します。すなわち、フィールドは常に正であると見なされます。 例えば、-5 は +5 と等しいと見なされます。
- 日付および時刻フィールドは、比較する前に *ISO 形式に変換されます。
- 図形データは 16 進数値によって比較されます。

分割制御フィールド

入力レコード中の複数のフィールドに同じ制御レベル標識を割り当てていると、分 割制御フィールドが形成されます。プログラム記述ファイルの場合には、同じ制御レベル標識を持つフィールドは入力仕様に指定された順序でプログラムによって結合され、単一の制御フィールドとして取り扱われます (127 ページの図 22 を参照)。 最初に定義されたフィールドは、制御フィールドの高位 (左端) 位置に入れら れ、最後に定義されたフィールドは、制御フィールドの低位 (右端) 位置に入 れられます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC.............................
I.....................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
IMASTER        01
I                                 28   31  CUSNO         L4
I                                 15   20  ACCTNO        L4
I                                 50   52  REGNO         L4
```

図 *22.* 分割制御フィールド

外部記述ファイルの場合には、同じ制御レベル標識を持つフィールドは、その フィールドが入力仕様に指定された順序ではなく、データ記述仕様 (DDS) に記 述されている順序で結合されます。 例えば、DDS に次のフィールドが次の順序で指定されているとします。

- EMPNO
- DPTNO
- REGNO

また、これらのフィールドが、入力仕様で次の順序により同じ制御レベル標 識を持つように指定されているとします。

- REGNO L3
- DPTNO L3
- EMPNO L3

フィールドは EMPNO DPTNO REGNO の順序で結合されて分割制御フィールドが構 成されます。

分割制御フィールドについては、次のような特別な規則があります。

- 1 つの制御レベル標識について、フィールド名が異なっていれば、一部のレ コード・タイプではフィールドを分割し、別のレコード・タイプでは分割しない ことができます。 ただし、分割にかかわらず、フィールドの長さはすべてのレコード・タイプで同 じでなければなりません。
- フィールド名が異なっていれば、異なるレコード・タイプごとに分割制御フ ィールドの各部分の長さを変えることができます。 ただし、各部分の合計の長さ は常に同じでなければなりません。
- フィールドの長さ (桁数または文字数) が同じである限り、パック 10 進数 フィールドとゾーン 10 進数フィールドを組み合わせて 1 つの分割制御フィール ドを形成することができます。
- 1 つのレコード・タイプの分割制御フィールドのすべての部分に同じフィールドとレコードの関連標識を割り当て、連続した仕様行に定義する必要があります。
- 分割制御フィールドに日付、時刻、またはタイム・スタンプ・フィールドが含ま れている時には、分割制御フィールド中のすべてのフィールドが同じタイプでな ければなりません。

128 ページの図 23 は、これらの規則の例を示しています。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC.............................
I.....................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
IDISK      BC  91   95 C1
I         OR   92   95 C2
I         OR   93   95 C3
I
 * All portions of the split control field must be assigned the same
 * control level indicator and all must have the same field record
 * relation entry.
I                                  1    5  FLD1A         L1
I                                 46   50  FLD1B         L1
I                                 11   13  FLDA          L2
I                                 51   60  FLD2A         L3
I                                 31   40  FLD2B         L3
I                                 71   75  FLD3A         L4  92
I                                 26   27  FLD3B         L4  92
I                                 41   45  FLD3C         L4  92
I                                 61   70  FLDB              92
I                                 21   25  FLDC              92
I                                  6   10  FLD3D         L4  93
I                                 14   20  FLD3E         L4  93
```

図 *23.* 分割制御フィールド ― 特別な規則

95 桁目の 1 で識別されるレコードには、次の 2 つの分割制御フィールドがあ ります。

1. FLD1A および FLD1B
2. FLD2A および FLD2B

95 桁目の '2' で識別されるレコードには、次の 3 つの分割制御フィールドがあ ります。

1. FLD1A および FLD1B
2. FLD2A および FLD2B
3. FLD3A、FLD3B、および FLD3C

95 桁目の '3' で識別される 3 番目のレコード・タイプにも、次の 3 つの分割制 御フィールドがあります。

1. FLD1A および FLD1B
2. FLD2A および FLD2B
3. FLD3D および FLD3E

フィールド標識

フィールド標識は、入力仕様の 69 から 70 桁目、71 から 72 桁目、または 73 から 74 桁目の記入項目によって定義されます。 有効なフィールド標識は次のとおりです。

- 01 から 99
- H1-H9
- U1-U8
- RT

フィールド標識を使用して、指定のフィールドまたは配列要素が正、負、ゼロ 、またはブランクであるかどうかを判別することができます。 69 から 72 桁目が有効 なのは数値フィールドの場合だけです。73 から 74 桁目が有効なのは、数値フィール ドまたは文字フィールドの場合です。 69 から 70 桁目に指定された標識は、数値入力フィールドがゼロより大きい時にオン に設定されます。71 から 72 桁目に指定された標識は、数値入力フィールドがゼロ より小さい時にオンに設定されます。また、73 から 74 桁目に指定された標識は、数値フィールドがゼロの時または文字フィールドがブランクの時にオンに設定 されます。 フィールド標識を使用して、演算または出力命令の条件付けをすることができます。

フィールドまたは配列要素のデータがレコードから抜き出され、その入力レコ ードにフィールド標識が表す条件があった場合には、そのフィールド標識がオ ンに設定されます。 このフィールド標識は、同じタイ

プの別のレコードが読み 取られ、その入力レコードにこの標識が表す条件が存在しなくなるまで、あるいは演算の結果としてその標識がオフに設定されるまで、オンのままです。

停止標識 (H1 から H9) をフィールド標識として使用し、フィールドまたは配列要 素がプログラムに読み込まれた時にそれらのエラー条件を検査することができます。

フィールド標識の割り当てに関する規則

フィールド標識を割り当ている場合には、以下の点に留意してください。

- プラス、マイナス、ゼロ、またはブランクの標識は、プログラムの始めでは オフに設定されています。これらは、読み取られたばかりのレコード上のテス トされているフィールドが条件 (プラス、マイナス、ゼロ、またはブランク) を 満たすまでオンに設定されません。
- 配列全体または先読みフィールドでフィールド標識を使用することはできま せん。 ただし、項目は配列要素として作成されます。 フィールド標識は、ALWNULL(*USRCTL) が使用される 場合にのみ、ヌル値可能フィールドに使用することができます。
- 数値入力フィールドには、2 つまたは 3 つのフィールド標識を割り当てるこ とができます。 ただし、そのフィールドのテストの結果について信号を送ってきた標識だけがオ ンに設定され、その他の標識はオフに設定されます。
- 異なるレコード・タイプのフィールドに同じフィールド標識を割り当てた場 合に、その標識の状態 (オンまたはオフ) は、常に最後に選択されたレコード・ タイプのフィールドの状態に基づいています。
- 異なるレコード・タイプのフィールドに異なるフィールド標識を割り当てた 場合には、そのタイプの別のレコードが読み取られるまで、フィールド標識はオ ンのままです。 同様に、1 つのレコード・タイプの中で複数のフィールドに割り 当てられたフィールド標識は、常に最後に定義されたフィールドの状況を反映し ます。
- 同じフィールド標識を、別の入力仕様のフィールド標識、結果の標識、レコ ード識別標識、またはフィールドとレコードの関連標識として指定することがで きます。 診断メッセージは出されませんが、とくに突き合わせフィールドまたはレベル制御が 関連している場合には、このような標識の使用は間違った結果の原因となる可能 性があります。
- 3 つの位置すべてに同じ標識を指定すると、該当するフィールドを含むレコ ードが選択された時に、この標識は常にオンに設定されます。

結果標識

結果標識は、従来の形式の演算仕様書 (C 仕様書) で使用されます。 これらは、自由形式演算仕様書では使用されません。 ほとんどの命令コードの場合、従来の形式でも自由形式でも、結果の標識の代わりに組み込み関数が使用できます。 詳しくは、「551 ページの『組み込み関数』」を参照してください。

結果の標識は、演算仕様書の 71 から 76 桁目の記入項目によって定義されます。 結果の標識の目的は、26 から 35 桁目に指定された命令コードによって異なります。 (結果の標識の目的については、710 ページの『命令コード』にある個々の命令コード の説明を参照してください。) 例えば、結果の標識を使用して、演算操作の後で結果のフィールドをテストし たり、レコード不在条件を識別したり、ファイル操作の例外/エラー条件を指示 したり、あるいはファイルの終わり条件を指示したりすることができます。

有効な結果の標識は次のとおりです。

- 01 から 99
- H1-H9
- OA から OG、OV
- L1-L9
- LR
- U1-U8
- KA から KN、KP から KY (SETOFF の場合にだけ有効)
- RT

結果の標識は、演算仕様書の 3 つの位置 (71 から 72 桁目、73 から 74 桁目、および 75 から 76 桁目) に指定することができます。 結果の標識が定義された位置によって、テストされる条件が決まります。

多くの場合、演算が処理されている時には結果の標識がオフに設定され、結 果の標識によって指定された条件が満たされた場合に、その標識がオンに設定 されます。 しかし、この規則には、特に 798 ページの『LOOKUP (テーブルまたは配列要素の検索)』、879 ページの『SETOFF (標識をオフに設定)』、および 880 ページの『SETON (標識をオンに設定)』 に示されているようないくつかの例外があります。 結果の標識は、同じ演算行で、あるいは他の演算また出力命令で条件付け標識と して使用することができます。 同じ行で使用する場合には、この標識の前の設定状況によって演算が処理され るかどうかが決まります。それが処理される場合には、結果のフィールドがテストされ、この標識の現在の設定状況が決められます (130 ページの図 24 を参照)。

結果標識の割り当てに関する規則

結果の標識を割り当てている場合には、以下の点に留意してください。

- 結果のフィールドが配列全体を参照している場合には、結果の標識を使用す ることはできません。
- 複数の命令の結果をテストするために同じ標識を使用した場合には、処理さ れた最後の命令によって、標識の設定状況が決まります。
- L1 から L9 標識が結果の標識として使用され、オンに設定されても、それより 低いレベルの標識はオンに設定されません。 例えば、L8 がオンに設定されても、L1 から L7 はオンに設定されません。
- H1 から H9 標識が結果の標識として使用された時にオンに設定されると、プログラム・サイクル中の検査の前に停止標識をオフに設定しない限り、プログラムは停止します。 (101 ページの『RPG サイクルおよびその他の暗黙論理』 を参照してください)。
- 指定した命令によっては、複数の条件のテストに同じ標識を使用することが できます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
 *
 *  Two resulting indicators are used to test for the different
 *  conditions in a subtraction operation.  These indicators are
 *  used to condition the calculations that must be processed for
 *  a payroll job.  Indicator 10 is set on if the hours worked (HRSWKD)
 *  are greater than 40 and is then used to condition all operations
 *  necessary to find overtime pay.  If Indicator 20 is not on
 *  (the employee worked 40 or more hours), regular pay based on a
 *  40-hour week is calculated.
 *
C     HRSWKD        SUB       40            OVERTM            3 01020
 *
C  N20PAYRAT       MULT (H)  40            PAY               6 2
C   10OVERTM       MULT (H)  OVRRAT        OVRPAY            6 2
C   10OVRPAY       ADD       PAY           PAY
 *
 *  If indicator 20 is on (employee worked less than 40 hours), pay
 *  based on less than a 40-hour week is calculated.
C   20PAYRAT       MULT (H)  HRSWKD        PAY
 *
```

図 *24.* 演算の条件付けに使用される結果の標識

RPG IV 仕様書で定義されない標識

RPG IV プログラムの中で条件付け標識として使用できるすべての標識が、仕様書で定義されるわけではありません。 外部標識 (U1 から U8) は、CL コマンドまたは前の RPG IV プログラムによって定義されます。内部標識 (1P、LR、MR、および RT) は、RPG IV プログラム・サイクル自体によって定義されます。

外部標識

外部標識は U1 から U8 です。 これらの標識は、CL プログラムまたは RPG IV プログラムによって設定することができます。 CL プログラムでは、CL コマンドの CHGJOB (ジョブ変更) または CRTJOBD (ジョ ブ記述作成) の SWS (スイッチ設定) パラメーターによって設定することがで きます。 RPG IV プログラムでは、結果の標識またはフィールド標識として設定することができます。

外部標識の状況は、演算仕様書で結果の標識として指定したり、入力仕様でフ ィールド標識として指定することにより、プログラム中で変更することができます。 ただし、RPG IV プログラムの処理中に CL プログラムによって IBM i ジョブ・スイッチの状況を変更しても、RPG IV プログラムが使用する外部標識のコピーには影響がありません。 プログラムの中で外部標識をオンまたはオフに設定しても、ファイル操作には 影響がありません。 ファイル操作は、プログラムの初期化時の標識 U1 から U8 の 状況に応じて機能します。 しかし、LR がオンでプログラムが正常に終了した場合には、外部標識は記憶域にコピーし戻され、その状況は RPG IV プログラムの最後の状況を反映します。 次に、この外部標識の現在の状況を他のプログラムで 使用することができます。

注 : LR 標識をオフにして 867 ページの『RETURN (呼び出し元への戻し)』 を使用する場合は、終了せずに戻り値を指定することになり、結果として外部標識は更新されません。

内部標識

内部標識には次のものがあります。

- 1 ページ目標識
- 最終レコード標識
- 突き合わせレコード標識
- 戻り標識

1 ページ目標識 *(1P)*

1 ページ目 (1P) 標識は、プログラムの実行開始時に RPG IV プログラムによってオンに設定され、明細時出力の後で RPG IV プログラムによってオフに設定されます。 明細時出力の後で、最初のレコードが処理されます。 1P 標識は、1P 時に書き出される見出しまたは明細レコードの条件付けに使用す ることができます。 1P 標識は、次のいずれかの方法では使用しないようにしてください。

- 入力レコードからのデータが必要な出力フィールドを条件付けするため。こ れは、入力データが使用可能にならないからです。
- 合計または例外出力行を条件付けするため
- 制御レベル標識との AND 関係の中で
- 結果標識として
- 制御仕様書に MAIN または NOMAIN が指定されている場合

最終レコード標識 *(LR)*

最終レコード標識 (LR) は、1 次ファイルを含んでいるプログラムで、1 次/2 次ファイルからの最後のレコードが処理された後にオンに設定されるか、またはプログラマーによって設定されます。

LR 標識は、プログラムの終わりで実行される演算または出力命令を条件付けす るために使用することができます。 LR 標識がオンに設定された場合には、他の すべての制御レベル標識 (L1 から L9) もオンに設定されます。 標識 L1 から L9 の中で制御レベル標識として、レコード識別標識として、結果の標 識として、あるいは *INxx によって定義されていない標識は、LR がオンに設定された時にはオンに設定されますが、他の仕様では使用することができません。

1 次ファイルが含まれていないプログラムでは、プログラムを終了 する方式の 1 つとして LR 標識をオンに設定することができます。(1 次ファイルなしでプログラムを終了する方法について詳しくは、101 ページの『RPG サイクルおよびその他の暗黙論理』を参照してください。) LR 標識をオンに設定するために、LR 標識をレコード識別標識または結果の標 識として指定することができます。 明細演算時に LR がオンに設定されると、 次のサイクルの始めに、他のすべての制御レベル標識がオンに設定されます。 残りの明細サイクルの間、LR 標識とレコード識別標識はどちらもオンのままですが、レコード識別標識は、LR の合計時の前にオフに設定されます。

突き合わせレコード標識 *(MR)*

突き合わせレコード標識 (MR) は、突き合わせフィールドの M1 から M9 の指定と対応して います。 これは、突き合わせフィールドが 1 次ファイルと少なくとも 1 つの 2 次ファイルに定義されている場合にだけ、プログラム中で使用することが できます。

MR 標識は、2 次ファイルのレコード中のすべての突き合わせフィール ドが 1 次ファイル中のレコードのすべての突き合わせフィールドと一致し た時にオンに設定されます。 これは、1 次と 2 次のレコードが完全に処理されるまでオンの ままです。 MR 標識は、これらのレコードに対する合計演算、合計出力、およびオーバーフロ ーがすべて処理された時点でオフに設定されます。

明細時の MR は、常に処理のために選択されたばかりのレコードの突き合わせ状況 を示します。合計時には、前のレコードの突き合わせ状況を反映します。 すべての 1 次ファイル・レコードがすべての 2 次ファイル・レコード と一致している場合には、MR 標識は常にオンになっています。

MR 標識は、フィールドとレコードの関連標識として、あるいはレコードが一 致する場合にだけ処理したい命令を指示するための演算仕様書または出力仕様 の条件付け標識として使用してください。 MR 標識を結果の標識として使用することはできません。

突き合わせフィールドおよび複数ファイル処理について詳しくは、173 ページの『ファイルに関する一般的な考慮事項』を参照してください。

戻り標識 (RT)

戻り標識 (RT) を使用して、制御を呼び出し側プログラムに戻す必要があることを内部 RPG IV 論理に指示することができます。 RT がオンかどうかを判別するためのテストは、LR の状況のテストの後、次のレ コードが読み取られる前に行われます。 RT がオンであれば、制御は呼び出し側プログラムに戻ります。 RT は、そのプログラムが再び呼び出された時にオフに設定されます。

RT 標識の状況の検査は停止標識 (H1 から H9) および LR 標識のテストの後で行われ るため、停止標識または LR 標識の状況が RT 標識の状況に優先します。 停止標識と RT 標識が両方ともオンの場合には、停止標識が優先します。 LR 標識と RT 標識が両方ともオンの場合には、プログラムは正常に終了します。

RT は、レコード識別標識、結果の標識、またはフィールド標識としてオンに設定することができます。 これは、次に、演算または出力命令の条件付け標識として使用することができます。

RT 使用して呼び出し側プログラムに制御を戻す方法の説明については、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」の呼び出し側プログラムに関する章を参照してください。

標識の使用

オーバーフロー標識、制御レベル標識、レコード識別標識、フィールド標識、結果標識、*IN、*IN(xx)、*INxx、あるいは、RPG IV 言語によって定義された標識は、演算 命令、または出力命令をファイルに条件付けるために使用できます。 標識を条件付け標識として使用するためには、その前に定義しておくことが必要 です。 標識が条件付け標識として使用されても、その標識の状況 (オンまたはオ フ) は影響を受けません。 この状況は、特定の条件を表すように標識を定義す ることによってのみ変更することができます。

注 : サイクルを制御する標識は、MAIN モジュールまたは NOMAIN モジュールで使用される場合、あるいは活動中のサブプロシージャーで使用される (ただしモジュールのサイクル・メイン・プロシージャーは存在しない) 場合には、条件付け標識としてのみ機能します。 サイクルを制御する標識には、LR、RT、H1 から H9、および制御レベル標識があります。

ファイルの条件付け

ファイル条件付け標識は、ファイル仕様書の EXTIND キーワードによって指 定されます。 外部標識 U1 から U8 が有効なのは、ファイルの条件付けの場合だけです。 (USROPN キーワードを指定して、暗黙の OPEN を実行する必要はないことを指定 することができます。)

プログラムが呼び出された時に指定された外部標識がオフになっていた場合には、 そのプログラムの実行時に、 ファイルはオープンされず、ファイルとの間のデータ転送は 行われません。 1 次および 2 次入力ファイルは、ファイルの終わりになったかの ように処理されます。 ファイルの終わり標識は、そのファイルに対するすべての READ 命令についてオン に設定されます。 このファイルの入力、演算、および出力仕様は、外部標識によって条件付けする 必要はありません。

ファイルの条件付けに関する規則

ファイルを条件付けする場合には、以下の点に留意してください。

- ファイルの条件付けの記入項目は、入力、出力、更新、または入出力共用フ ァイルに対して指定することができます。
- テーブルまたは配列の入力に対しては、ファイルの条件付けの記入項目を指 定することはできません。
- テーブルの出力ファイルは、U1 から U8 で条件付けすることができます。 この標識がオフの場合には、テーブルは書き出されません。
- レコード・アドレス・ファイルは U1 から U8 で条件付けすることができますが 、レコード・アドレス・ファイルによって処理されるファイルを U1 から U8 で条件付 けすることはできません。
- 突き合わせレコードを持つ 1 次ファイルを条件付けする標識がオフの 場合には、MR 標識はオンに設定されません。
- ファイルを条件付けする標識がオフの場合には、入力、更新、または入出力 共用ファイルについて入力は行われません。 関連した入力仕様の 63 から 74 桁目に定義されたすべての標識は、入力フィール ドの既存の値を使用して通常どおりに処理されます。
- ファイルを条件付けする標識がオフの場合には、出力、更新、または 入出力共用ファイルについてファイルへのデータ転送は行われません。 これらのファイルについて出力仕様に定義された条件付け標識、数値編集、ま たは後で消去は、すべて通常どおりに処理されます。
- 入力、更新、または入出力共用ファイルを条件付けしている標識がオフの場 合には、このファイルはファイルの終わりになっているものと見なされます。 定義されたすべての結果の標識は、指定された各入出力操作の始めでオフに設定されます。 ファイルの終わり標識は、READ、READC、READE、READPE、および READP 命令に対し てオンに設定されます。 CHAIN、EXFMT、SETGT、SETLL、および UNLOCK 命令は無視され、定義されたすべ ての結果の標識はオフに設定されたままです。

フィールドとレコードの関連標識

フィールドとレコードの関連標識は、入力仕様の 67 から 68 桁目に指定されます。 有効なフィールドとレコードの関連標識は次のとおりです。

- 01 から 99
- H1-H9
- MR
- RT
- L1-L9
- U1-U8

外部記述ファイルに対しては、フィールドとレコードの関連標識を指定すること はできません。

フィールドとレコードの関連標識は、特定のレコード・タイプが OR 関係にある いくつかのレコード・タイプの 1 つである場合に、フィールドをそのレコード・ タイプと関連付けるために使用します。 仕様行に記述されたフィールドは、フィ ールドとレコードの関連記入項目に指定された標識がオンであるか、または記入 項目がブランクの場合にだけ入力に使用することができます。 記入項目がブラン クの場合には、フィールドは OR 関係によって定義されたすべてのレコード・タ イプに共用です。

フィールドとレコードの関連標識の割り当て

67 から 68 桁目にレコード識別標識 (01 から 99) を使用して、フィールドを特定の レコード・タイプに関連付けることができます。 複数のレコード・タイプが OR 関係で 指定されている場合には、67 から 68 桁目にフィールドとレコードの関連標識のない すべてのフィールドが、OR 関係にあるすべてのレコード・タイプと 関連付けられます。フィールドを 1 つのレコード・タイ プにのみ関連付けるには、そのレコード・タイプに割り当てられているレコ ード識別標識を 67 から 68 桁目に記入します (135 ページの図 25 を参照)。

入力域から入力フィールドへのフィールドの転送を条件付けするために、レコー ド識別標識でない標識 (01 から 99) を 67 から 68 桁目に使用することもできます。

フィールドとレコードの関連標識が指定されている場合には、入力仕様の 63 から 64 桁目の L1 から L9 標識で定義した制御フィールド、および 入力仕様の 65 から 66 桁目の突き合わせ値 (M1 から M9) で指定した突き合わせフィールド も、OR 関係にある特定の レコード・タイプに関連付けることができます。 OR 関

係にあって、フィールドと レコードの関連標識のない制御フィールドまたは 突き合わせフィールドは、OR 関係にあるすべてのレコード・タイプで使用されます。

2 つの制御フィールドが同じ制御レベル標識を持っているか、2 つの突き合わせフ ィールドが同じ突き合わせレベルの値を持っている場合には、それらの突き合わせフィー ルドの 1 つにだけフィールドとレコードの関連標識を割り当てていることができます 。 この場合には、その標識がオンである時に、フィールドとレコードの関連標識 を持つフィールドだけが使用されます。 その制御フィールドまたは突き合わせフィールドについてオンになっているフィール ドとレコードの関連標識がない場合には、フィールドとレコードの関連標識のな いフィールドが使用されます。 制御フィールドおよび突き合わせフィールドに可能な記入項目は、67 から 68 桁目の 01 から 99 または H1 から H9 だけです。

67 から 68 桁目を使用して、一定の条件が起こった (例えば、レコードが一致 する、制御の切れ目が起こる、または外部標識がオンである) 時にだけ、プログ ラムが特定のフィールドからデータを受け入れて使用することを指定することが できます。 67 から 68 桁目に標識 L1 から L9、MR、または U1 から U8 を指定することによ って、プログラムがフィールドからデータを受け入れる条件を指示することがで きます。 標識を指定することによって、プログラムがフィールドからデータを受け入れ る条件を指示することができます。49 から 62 桁目で指定したフィールドのデー タが受け入れられるのはフィールドとレコードの関連標識がオンの場合だけで す。

外部標識は、ファイル記述仕様書の 371 ページの『EXTIND(*INUx)』 キーワードによってファイルの条件付けが指定される場合に主に使用されます。 しかし、ファイルの条件付けを指定しない場合であっても、外部標識を使用する ことはできます。

67 から 68 桁目の停止標識 (H1 から H9) は、OR 関係にあり、かつ 21 から 22 桁目に停 止標識が指定されたレコードにフィールドを関連付けます。

フィールドとレコードの関連標識を使用する場合には、以下の点に留意してく ださい。

- 同じフィールドとレコードの関連標識を持つ制御レベル (63 から 64 桁目) と突き合わせフィールド (65 から 66 桁目) は、1 つにグループ化しなければなりません。
- フィールドとレコードの関連標識のない制御レベル (63 から 64 桁目) および突き合わせフィールド (65 から 66 桁目) の記入項目として使用されるフィールドは、フィ ールドとレコードの関連標識で使用されるフィールドの前になければなりません。
- フィールドとレコードの関連標識 (67 から 68 桁目) のある制御レベル (63 から 64 桁目) と突き合わせフィールド (65 から 66 桁目) は、この標識がオンの時に、同じ レベルで標識のない制御レベルおよび突き合わせフィールドに優先します。
- 突き合わせフィールドおよび制御レベル・フィールド (63 から 66 桁目) のフィール ドとレコードの関連 (67 から 68 桁目) は、その突き合わせフィールドが参照する主要仕 様行または OR 関係行からレコード識別標識 (01 から 99 または H1 から H9) で指定し なければなりません。 複数のレコード・タイプを OR 関係で指定する場合には、フィールドの 関係を指定する標識を使用して、突き合わせおよび制御レベル・フィールドを該当する レコード・タイプに関連付けることができます。
- 制御レベル (63 から 64 桁目) と突き合わせフィールド (65 から 66 桁目) のない仕様 は、フィールドとレコードの関連の指定 (67 から 68 桁目) があるグループとグルー プの間に入れることができます。
- 入力レコードの特定のフィールドが対応する条件がある時だけ必要になる場 合には、MR 標識をフィールドとレコードの関連標識として使用し、処理時間を短 縮することができます。
- OR 関係にある異なるレコード・タイプには、異なる数の制御レベル (L1 から L9) を指定することができます。 あるレコード・タイプには制御レベルがなく、他のレコード・タイプには制御レ ベルがいくつもあることがあります。
- すべての突き合わせフィールド (65 から 66 桁目) にフィールドとレコードの関連標 識 (67 から 68 桁目) を指定する場合には、フィールドとレコードの関連標識の 1 つ ごとに、突き合わせフィールドの完全セットを関連付けなければなりません。
- 1 つの突き合わせフィールドをフィールドとレコードの関連標識なしで指定する 場合には、フィールドとレコードの関連標識のないフィールドについて突き合わせフ ィールドの完全セットを指定しなければなりません。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC..............................
I.....................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
IREPORT    AA  14    1 C5
I        OR    16    1 C6
I                                  20   30  FLDB
I                                   2   10  FLDA             07
 *
 *  Indicator 07 was specified elsewhere in the program.
 *
I                                  40   50  FLDC             14
I                                  60   70  FLDD             16
```

図 *25.* フィールド・レコード 関係

このファイルには 2 つの異なるレコード・タイプが入っており、その 1 つは 1 桁目 の 5 によって識別され、もう 1 つは 1 桁目の 6 によって識別されます。 FLDC フィールドは、レコード識別標識 14 によって、1 桁目の 5 によって識別されるレコード・タイプと関連付けられています。 FLDD フィールドは、レコード識別 標識 16 によって、1 桁目に 6 を持つレコード・タイプと関連付けられて います。 これは、FLDC が 1 つのタイプのレコード (1 桁目の 5 によって識別される) にだ けあり、FLDD はもう 1 つのタイプのレコードにだけあることを意味します。 FLDA は、プログラムの別の個所ですでに定義されている標識 07 によって条件付 けされています。 FLDB は、レコード識別標識によってどちらのタイプにも関連付 けられていないので、両方のレコード・タイプに入っています。

機能キー標識

関連した機能キーがデータ記述仕様 (DDS) に指定されている場合 には、WORKSTN 装置を含むプログラムで機能キー標識を使用することが できます。 機能キーは、CFxx または CAxx キーワードによって DDS に指定されます。 ワークステーション・ファイルと一緒に使用する機能キーの使用例については、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」の WORKSTN の章を参照してください。

機能キー標識	対応する機能キー	機能キー標識	対応する機能キー
KA	1	KM	13
KB	2	KN	14
KC	3	KP	15
KD	4	KQ	16
KE	5	KR	17
KF	6	KS	18
KG	7	KT	19
KH	8	KU	20
KI	9	KV	21
KJ	10	KW	22
KK	11	KX	23
KL	12	KY	24

機能キー標識は、機能キー 1 から 24 と対応します。 機能キー標識 KA は機能キー 1 と対応し、KB は機能キー 2 と対応し、以下同様に、 KY は機能キー 24 と対応します。

機能キー標識がオンに設定された後に、演算または出力命令の条件付けに使 用することができます。 機能キー標識は、SETOFF 命令によってオフに設定する ことができます。

停止標識 (H1 から H9)

停止標識 (H1 から H9) を使用して、プログラムの実行中に起こるエラーを指示す ることができます。 停止標識は、レコード識別標識、フィールド標識、または結果の 標識としてオン に設定することができます。

停止標識は、RPG IV サイクルの *GETIN ステップでテストされます (101 ページの『RPG サイクルおよびその他の暗黙論理』を参照)。 停止標識がオンになっている場合には、メッセージ がユーザーに出されます。 次の応答が有効です。

- 停止標識をオフに設定して、プログラムを続行する。
- ダンプを指示して、プログラムを終了する。
- ダンプなしで、プログラムを終了する。

停止標識がオンになっている場合に、サイクル・メイン・プロシージャー内部の RETURN 命令が処理された時、または LR 標識がオンになっている時には、呼び出されたプログラムは異常終了します。 呼び出し側プログラムには、呼び出されたプログラムが停止標識 のオンによって終了されたことが通知されます。

注 : キーワード MAIN または NOMAIN が制御仕様書に指定されている場合、条件付け標識としての停止標識を除き、すべての停止標識が無視されます。

停止標識がオンのときに取られるステップについて詳しくは、101 ページの『RPG サイクルおよびその他の暗黙論理』にある RPG IV サイクルの詳細なフローチャートを参照してください。

演算の条件付け標識

従来の形式の演算仕様書 (C 仕様書) では、7 桁目および 8 桁目、そして 9 から 11 桁目に条件付け標識を含むことができます。 条件付け標識は、自由形式演算仕様書では使用されません。

演算を実行する条件を指定するために使用される標識は、プログラムのどこかで定義されていなければなりません。

7 から *8* 桁目

演算仕様書の 7 から 8 桁目に制御レベル標識 (L1 から L9 および LR) を指定する ことができます。

7 から 8 桁目がブランクであれば、演算は明細時に実行されるか、サブルーチン 内のステートメントであるか、または宣言ステートメントです。 標識 L1 から L9 が 指定された場合に、合計時に演算が処理されるのは指定の標識がオンになっている 時だけです。 LR 標識が指定された場合には、演算は最終合計時に処理されます。

注 : 演算がすべてのプログラム・サイクルで処理したい合計演算であることを 指示するために、L0 の記入項目を使用することができます。

9 から *11* 桁目

演算仕様書の 9 から 11 桁目を使用して、演算が処理される条件を制御する標識 を指定することができます。 9 桁目に N を指定して標識の値がオフ ('0') であるかどうかを テストする必要があることを指示することができます。10 から 11 桁目に有効な記入項目は以下のとおり です。

- 01 から 99
- H1-H9
- MR
- OA から OG、OV
- L1-L9
- LR
- U1-U8
- KA から KN、KP から KY
- RT

9 から 11 桁目に使用する標識は、すべて次のタイプの標識の 1 つとして事前に定義 しておかなければなりません。

- オーバーフロー標識 (ファイル記述仕様書 380 ページの『OFLIND(標識)』)
- レコード識別標識 (入力仕様の 21 から 22 桁目)
- 制御レベル標識 (入力仕様の 63 から 64 桁目)
- フィールド標識 (入力仕様の 69 から 74 桁目)
- 結果標識 (演算仕様書の 71 から 76 桁目)
- 外部標識
- LR および MR のようにオンに設定される標識
- *IN 配列、*IN(xx) 配列要素、または *INxx フィールド (これらの予約語の 1 つを使用して標識を定義する方法については、141 ページの『データとして参照される標識』を参照してください。)

演算を条件付けするために標識がオフである必要がある場合には、9 桁目に N を入れてください。 グループ化された AND/OR 行の標識の他に制御レベル標識 (7 から 8 桁目に指定される場合) も、137 ページの図 26 に従って演算を実行する前に、すべてがこの図に示されるとおりに正確に指定されている必要があります。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
 *
C   25
CAN L1             SUB       TOTAL         TOTAL       A
CL2 10
CANNL3TOTAL        MULT      05            SLSTAX      B
 *
```

図 *26.* 演算の条件付け *(*制御レベル標識*)*

標識 25 がレコード・タイプを表し、レコード・タイプ 25 が読み取られた 時に制御レベル 2 の切れ目が起こったとします。 L1 と L2 は両方ともオンになっ ています。 7 から 8 桁目の制御レベル標識によって条件付けされたすべての演算 は、9 から 11 桁目の制御レベル標識によって条件付けされた演算より前に実行されます。 そのため、 B の演算が A の演算の前に行われます。 A の演算は、25 によって示される新しい制御グループの最初のレコードについて実行されますが、 B の演算は、前の制御グループのすべてのレコードに対して実行される合計演算です。

The operation in B の演算を、他の条件が満たされて L2 がオンのときに実行することができます。標識 10 はオンである必要があり、L3 標識がオンであってはなりません。

L2 と NL3 の両方によって条件付けされた演算は、制御レベル 2 の切れ目が起 こった時にだけ実行されます。 制御レベル 3 が起こった場合には、L2 もオンにな りますが、その時点ではこの演算は実行したくないため、これら 2 つの標識が同 時に使用されています。

9 から 11 桁目に条件付け標識を使用する場合に知っておく必要があるいくつかの特 別な考慮事項は、次のとおりです。

- 外部記述ワークステーション・ファイルを使用する場合に、演算仕様書の条件 付け標識は RPG プログラムの中で定義されているか、またはワークステーション ・ファイルの DDS ソース仕様の中で定義されていなければなりません。
- プログラム記述ワークステーション・ファイルを使用する場合に、そのワー クステーション・ファイルに使用される標識は、RPG プログラムのコンパイル時 には不明です。 したがって、標識 01 から 99 が宣言されたものと見なされ、それら の標識を定義せずに演算仕様書の条件付けに使用することができます。
- 停止標識を使用して、入力データまたは別の演算で指定のエラー条件が見付 かった場合に、プログラムを終了したり、命令が処理されないようにすることが できます。 プログラムは、停止の原因となったレコードが完全に処理されてから停止するの で、停止標識の使用が必要になります。 したがって、命令がエラー状態のまま処 理された場合には、結果は誤りとなります。 また、停止標識を使用して、エラー が起こった時にだけ実行したい命令を条件付けすることができます。
- 9 から 11 桁目に LR を指定した場合には、演算は最後のレコードが処理された 後、あるいは LR がオンに設定された後で実行されます。

- 9 から 11 桁目に制御レベル標識を使用し、7 から 8 桁目を使用しない (明細時の) 場合には、この標識によって条件付けされた演算は、制御の切れ目またはより高 いレベルの制御の切れ目の原因となったレコードに対してだけ実行されます。
- 7 から 8 桁目に制御レベル標識を指定し (合計時)、9 から 11 桁目に MR を指定し た場合に、MR は、制御の切れ目の原因となった読み取られたばかりのレコードで はなく、前のレコードの突き合わせ条件を指示することになります。 7 から 8 桁目の 制御レベル標識によって条件付けされたすべての演算が実行された後 に、MR は、読み取られたばかりのレコードの突き合わせ条件を指示します。
- 7 から 8 桁目および 9 から 11 桁目がブランクの場合には、その行に指定された演 算が明細演算時に実行されます。

138 ページの図 27 と 138 ページの図 28 は、条件付け標識の例を示しています。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilenameSqNORiPos1NCCPos2NCCPos3NCC.PFromTo++DField+L1M1FrPlMnZr...*
I.....................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
 *
 *  Field indicators can be used to condition operations.  Assume the
 *  program is to find weekly earnings including overtime.  The over-
 *  time field is checked to determine if overtime was entered.
 *  If the employee has worked overtime, the field is positive and -
 *  indicator 10 is set on.  In all cases the weekly regular wage
 *  is calculated.  However, overtime pay is added only if
 *  indicator 10 is on.
 *
ITIME      AB  01
I                                  1    7  EMPLNO
I                                  8   10 0OVERTM             10
I                                 15   20 2RATE
I                                 21   25 2RATEOT
CL0N01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++
 *
 *  Field indicator 10 was assigned on the input specifications.
 *  It is used here to condition calculation operations.
 *
C                   EVAL (H)  PAY = RATE * 40
C   10              EVAL (H)  PAY = PAY + (OVERTM * RATEOT)
```

図 *27.* 演算の条件付け *(*フィールド標識*)*

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC..............................
I.....................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
 *
 *  A record identifying indicator is used to condition an operation.
 *  When a record is read with a T in position 1, the 01 indicator is
 *  set on.  If this indicator is on, the field named SAVE is added
 *  to SUM.  When a record without T in position 1 is read, the 02
 *  indicator is set on.  The subtract operation, conditioned by 02,
 *  then performed instead of the add operation.
 *
IFILE       AA  01    1 CT
I          OR   02    1NCT
I                                 10   15 2SAVE
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
 *
 *  Record identifying indicators 01 and 02 are assigned on the input
 *  specifications.  They are used here to condition calculation
 *  operations.
 *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C   01              ADD       SAVE          SUM               8 2
C   02              SUB       SAVE          SUM               8 2
```

図 *28.* 演算の条件付け *(*レコード識別標識*)*

式で使用されている標識

標識は演算仕様書の拡張演算項目 2 フィールドの式の中でブールとして使用す ることができます。 これらの標識はデータとして (すなわち、*IN または *INxx を使用 して) 参照しなければなりません。 以下の例はこれを例示しています。

```
CL0N01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++
 * In these examples, the IF structure is performed only if 01 is on.
 * *IN01 is treated as a boolean with a value of on or off.

 * In the first example, the value of the indicator ('0' or '1') is
 * checked.
C                   IF        *IN01

 * In the second example, the logical expression B < A is evaluated.
 * If true, 01 is set on.  If false 01 is set off.  This is analogous
 * to using COMP with A and B and placing 01 in the appropriate
 * resulting indicator position.
C                   EVAL      *IN01 = B < A
```

図 *29.* 式で使用されている標識

さらに多くの例および詳細については、本書の式の章および命令コードの章を 参照してください。

出力の条件付け標識

出力レコードまたは出力フィールドが書き出される条件を指定するために使用さ れる標識は、プログラム内で事前に定義されていなければなりません。 出力を条件付けする標識は、21 から 29 桁目に指定されます。 出力の条件付けには、すべての標識が有効です。

出力の条件付けに使用する標識は、次のタイプの標識の 1 つとして、事前に定義 されていなければなりません。

- オーバーフロー標識 (ファイル記述仕様書、380 ページの『OFLIND(標識)』)
- レコード識別標識 (入力仕様の 21 から 22 桁目)
- 制御レベル標識 (入力仕様の 63 から 64 桁目)
- フィールド標識 (入力仕様の 69 から 74 桁目)
- 結果標識 (演算仕様書の 71 から 76 桁目)
- 1P や LR のように RPG IV プログラムによって設定される標識
- プログラムの処理前または処理中に設定される外部標識
- *IN 配列、*IN(xx) 配列要素、または *INxx フィールド (これらの予約語の 1 つを使用して標識を定義する方法については、141 ページの『データとして参照される標識』 を参照してください。)

標識によってレコード全体を条件付けする場合には、レコード・タイプを指定する行に標識を記入します (140 ページの図 30 を参照)。標識によってフィールドの書き出し時点を条件付けする場合には、フィールド名と同じ行に記入します (140 ページの図 30 を参照)。

出力行には、条件付け標識は必須ではありません。 条件付け標識が指定されていない場合には、レコードのタイプが出力用に検査されるたびにその行が出力されます。 条件付け標識を指定する場合には、3 つの別個の出力標識フィールド (22 から 23、25 から 26、および 28 から 29 桁目) のそれぞれに標識を 1 つずつ 記入することができます。 これらの標識がオンの場合には、出力操作が実行されます。 各標識の前の桁 (21、24、または 27 桁目) の N は、標識がオンでない場合にだけ出力操作が 実行されることを意味します (否定標識)。出力行を否定標識だけで条件付けする必要はありません が、少なくとも 1 つの標 識は肯定とする必要があります。 見出しまたは明細操作を否定標識だけで条件付 けした場合には、その操作は、プログラム・サイクルの始めか、あるいは 1 ペー ジ目 (1P) の行が書き出される時に実行されます。

16 から 18 桁目に AND/OR を指定することによって、AND/OR 関係にある出力標識を指定できます。 使用することができる AND/OR 行の数に制限はあり ません。 AND/OR 行は、出力レコードの条件付けに使用することはできますが、フィールド の条件付けに使用することはできません。 しかし、演算仕様書で EVAL

命令を使 用することによって、4 つ以上の標識でフィールドを条件付けすることができます。 以下の例はこれを例示しています。

```
CL0N01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++
 * Indicator 20 is set on only if indicators 10, 12, 14,16, and 18
 * are set on.
C                   EVAL      *IN20 = *IN10 AND *IN12 AND *IN14
C                              AND *IN16 AND *IN18
OFilename++DAddN01N02N03Excnam++++.....................................
O.............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat
 * OUTFIELD is conditioned by indicator 20, which effectively
 * means it is conditioned by all the indicators in the EVAL
 * operation.
OPRINTER   E
O               20      OUTFIELD
```

出力標識について知っておくべき、その他の特別な考慮事項は、次のとおりです。

- 1 ページ目標識 (1P) によって、1 次ファイルを読み取る前に、 最初のサイクルで 1 ページ目の印刷などの出力を行うことができます。 1P 標識によって 条件付けされる行には、見出しまたは PAGE および UDATE など の予約語用のフィールドとして使用される定数が入っていなければなりませ ん。 この定数は、出力仕様の 53 から 80 桁目に指定されます。1P がオーバーフロー標識との OR 関係で使用される場合には、この情報がすべてのページに印刷されます (141 ページの図 31 を参照)。 1P 標識は、見出しまたは明細出力行に だけ使用してください。 合計出力行または 例外出力行の条件付けに使用することはできません。また、 制御レベル標識との AND 関係にもこの標識は使用しないようにしてください。
- 特定のエラー条件が起こり、出力操作の処理の取り止めが必要になることが あります。エラーの原因となったデータを使用しないようにするには、停止標識を使用してください (141 ページの図 32 を参照)。
- ある種の出力レコードを外部条件によって条件付けするには、外部標識を使 用して、それらのレコードを条件付けしてください。

出力仕様でのオーバーフロー標識の割り当てについては、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」の印刷装置ファイルのセクションを参照して ください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+.........................
O.............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat
 *
 *  One indicator is used to condition an entire line of printing.
 *  When 44 is on, the fields named INVOIC, AMOUNT, CUSTR, and SALSMN
 *  are all printed.
 *
OPRINT     D    44                    1
O                       INVOIC              10
O                       AMOUNT              18
O                       CUSTR               65
O                       SALSMN              85
 *
 *  A control level indicator is used to condition when a field should
 *  be printed.  When indicator 44 is on, fields INVOIC, AMOUNT, and
 *  CUSTR are always printed.  However, SALSMN is printed for the
 *  first record of a new control group only if 44 and L1 are on.
 *
OPRINT     D    44                  1
O                       INVOIC              10
O                       AMOUNT              18
O                       CUSTR               65
O                  L1   SALSMN              85
```

図 *30.* 出力標識

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+.........................
O.............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat
 *
 *  The 1P indicator is used when headings are to be printed
 *  on the first page only.
 *
OPRINT     H      1P                   3
O                                          8 'ACCOUNT'
 *
 *  The 1P indicator and an overflow indicator can be used to print
 *  headings on every page.
 *
OPRINT     H    1P                     3  1
O         OR    OF
O                                          8 'ACCOUNT'
```

図 *31. 1P* 標識

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC.............................
I.....................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
 *
 *  When an error condition (zero in FIELDB) is found, the halt
 *  indicator is set on.
 *
IDISK      AA  01
I                                  1   3  FIELDA        L1
I                                  4   8 0FIELDB                  H1
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
 *
 *  When H1 is on, all calculations are bypassed.
 *
C   H1              GOTO      END
C                     :
C                     :               Calculations
C                     :
C     END             TAG
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+.........................
O.............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat
 *
 *  FIELDA and FIELDB are printed only if H1 is not on.
 *  Use this general format when you do not want information that
 *  is in error to be printed.
 *
OPRINT     H    L1                 0  2 01
O                                            50 'HEADING'
O          D    01NH1                 1  0
O                       FIELDA              5
O                       FIELDB        Z     15
```

図 *32.* フィールドの印刷防止

データとして参照される標識

RPG IV の標識を参照し、処理する代替方式として、RPG IV の予約語 *IN および *INxx が用意されています。

*IN

配列 *IN は、1 桁の文字要素が 99 個ある定義済み配列で、 その要素は標識 01 から 99 を表しています。この配列の要素には、文字値 '0' (ゼロ) または '1' だけを入れてください。

入力レコードのフィールドとして、 結果のフィールドとして、あるいは PARM 命令の演算項目 1 として、*IN 配列または *IN(xx) 変数指標配列要素を指定すると、 標識 01 から 99 がプログラムでの使用に備えて定義されます。

1 桁の文字要素の配列に対して有効な命令または参照は、配列 *IN で有効で す。ただし、配列 *IN は、データ構造のサブフィールドとして、または PARM 命 令の結果のフィールドとして指定することはできません。

*INxx

フィールド *INxx は事前定義された 1 桁の文字フィールドです。ここで、xx は RPG IV 標識の 1 つを表します。

入力レコードのフィールドとして、結果のフィールドとして、あるいは PARM 命 令の演算項目 1 として、*INxx フィールドまたは *IN(n) 固定指標配列 要素 (n = 1 から 99) を指定すると、対応する標識がプログラムでの使用に備えて定義されます。

1 桁の文字フィールドが使用できるところであればどこにでも、 フィールド *INxx を指定することができますが、次の例外があります。すなわち、 フィールド *INxx は、データ構造のサブフィールドとして指定したり、PARM 命令の結果 のフィールドとして指定したり、または SORTA 命令で指定したりすることはできません。

追加規則

配列 *IN、配列要素 *IN(xx)、またはフィールド *INxx を処理する場合には 、次の規則に留意してください。

- これらのフィールドのいずれかに文字 '0' (ゼロ) または *OFF を 転送した場合には、対応する標識がオフに設定されます。
- これらのフィールドのいずれかに文字 '1' または *ON を転送した場 合には、対応する標識がオンに設定されます。
- *INxx に転送する値は '0' (ゼロ) または '1' だけにして ください。 それ以外の値を転送した場合、その後の通常の RPG IV 標識テストで予期できない結果が出される可能性があります。
- *IN、*IN01 から *IN99、または *IN(索引) のアドレスを使用 する場合は、標識 *IN01 から *IN99 が定義されることになります。 それ以外の、*INLR あるいは *INL1 などの標識のアドレスを使用する場合は、その標識だけが定義されることになります。

データとして参照される標識の例については、143 ページの図 33 を参照し てください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
 *
 *  When this program is called, a single parameter is passed to
 *  control some logic in the program.  The parameter sets the value
 *  of indicator 50.  The parameter must be passed with a character
 *  value of 1 or 0.
 *
C     *ENTRY        PLIST
C     *IN50         PARM                    SWITCH            1
 *
 *
 *  Subroutine SUB1 uses indicators 61 through 68.  Before the
 *  subroutine is processed, the status of these indicators used in
 *  the mainline program is saved.  (Assume that the indicators are
 *  set off in the beginning of the subroutine.)  After the subroutine
 *  is processed, the indicators are returned to their original state.
 *
 *
C                   MOVEA     *IN(61)        SAV8              8
C                   EXSR      SUB1
C                   MOVEA     SAV8           *IN(61)
 *
 *  A code field (CODE) contains a numeric value of 1 to 5 and is
 *  used to set indicators 71 through 75.  The five indicators are set
 *  off.  Field X is calculated as 70 plus the CODE field.  Field X is
 *  then used as the index into the array *IN.  Different subroutines
 *  are then used based on the status of indicators 71 through 75.
 *
C                   MOVEA     '00000'        *IN(71)
C   70              ADD       CODE           X                 3 0
C                   MOVE      *ON            *IN(X)
C   71              EXSR      CODE1
C   72              EXSR      CODE2
C   73              EXSR      CODE3
C   74              EXSR      CODE4
C   75              EXSR      CODE5
```

図 *33.* データとして参照される標識の例

標識の要約

144 ページの表 60 および 145 ページの表 61 は、RPG IV 標識 が定義される個所、有効な記入項目、標識が使用される個所、および標識がオンおよびオフに設定される時点を要約して示しています。145 ページの表 61 は、各タイプの標識が RPG IV プログラムによってオンおよびオフに設定される基本的な条件を示しています。135 ページの『機能キー標識』には、機能キー標識とそれに対応する機能キーがリストされています。

表 *60.* 標識の記入項目および使用法

	定義個所/使用個所	01 から 99	1P	H1-H9	L1-L9	LR	MR	OA から OG OV	U1-U8	KA から KN KP から KY	RT
ユーザー定義	オーバーフロー標識、ファイル仕様書、OFLIND キーワード	X						X			
	レコード識別標識、入力仕様の 21 から 22 桁目	X		X	X	X			X		X
	制御レベル、入力仕様の 63 から 64 桁目				X						
	フィールド・レベル、入力仕様の 69 から 74 桁目	X		X					X		X
	結果標識、演算仕様書の 71 から 76 桁目	X		X	X	X		X[1]	X	X[2]	X
RPG 定義	内部標識		X			X	X				X
	外部標識								X		
使用個所	ファイルの条件付け、ファイル仕様書								X		
	ファイルとレコードの関連、入力仕様の 67 から 68 桁目 [3]	X		X	X		X		X		X
	制御レベル、演算仕様書の 7 から 8 桁目				X	X					
	条件付け標識、演算仕様書の 9 から 11 桁目	X		X	X	X	X	X	X	X	X
	出力標識、出力仕様の 21 から 29 桁目	X	X[4]	X	X	X	X	X	X	X	X

注 :

1. オーバーフロー標識は、まずファイル仕様書で定義しな ければなりません。
2. KA から KN および KP から KY は、SETOFF 命令の結果の標識として使用できます。
3. 制御フィールドまたは突き合わせフィールドの条件付けには、主要レコード または OR レコードのレコード識別標識しか使用できません。 L1 または L9 を使用 して、制御フィールドまたは突き合わせフィールドを条件付けす ることはできません。
4. 1P 標識を使用できるのは、見出し行および明細行だけです。

表 *61. RPG IV* 論理サイクルによって標識がオンおよびオフに設定される時点

標識のタイプ	オンに設定	オフに設定
オーバーフロー	オーバーフロー行の印刷時、あるいはスペースまたはスキップにより通過した時。	OA-OG、OV: 次の見出しおよび明細行が完了した後、またはファイルが開いた後(H 仕様書のキーワードの OPENOPT(*NOINZOFL) が使用されていない場合)。01 から 99: ユーザーによる。
レコード識別	指定された 1 次/2 次レコードが読み取られた時点および 合計計算が処理される前。レコードが全手順ファイルから読み取られた直後。	次の処理サイクルで次の 1 次/2 次レコードが読み取られ る前。
制御レベル	制御フィールドの値が変わった時。 これより低いレベルの標識もすべてオンに 設定される。	次の明細サイクルの終わり。
フィールド標識	指定したフィールドのブランクまたはゼロ、指定したフィールドのプラス、ま たは指定したフィールドのマイナスによる。	このフィールドの状況の次回のテストの前。
結果	演算が処理され、標識が表す条件が満たされた時。	同じ標識を結果の標識として指定した演算の次回の実行時に、指定した条件が 満たされない時。
機能キー	ワークステーション・ファイルについて対応する機能キーが押された時点および関連サブフ ァイルに対する後続の読み取り時点。	SETOFF によるか、あるいは ワークステーション・ファイルのフィールド転送論理による。
外部 U1 から U8	プログラムの開始前に CL コマンドによるか、あるいは結果の標識またはフィ ールド標識として使用された時。 **注 :** 外部標識の値は、初期化時にジョブ・スイッチから設定されます。 サイクル・モジュールの場合は、サイクルの *INIT フェーズ時に設定されます。 その他のモジュールの場合は、モジュール内の最初のプロシージャーが呼び出されたときに 1 度だけ設定されます。	プログラムの開始前に CL コマンドによるか、または結果の標識として使用さ れるか、あるいは結果の標識またはフィールド標識として使用された時。
H1-H9	プログラマーの指定による。	メッセージに対する応答として続行のオプションが選択された時、またはプロ グラマーによる。
RT	プログラマーの指定による。	このプログラムが再び呼び出された時。
内部標識 1P	処理の始めで、入力レコードが読み取られる前。	最初のレコードが読み取られる前。
LR	最後のファイルの最終 1 次/2 次レコードの処理後、また はプログラマーによる。	処理の始め、またはプログラマーによる。
MR	2 次ファイルのレコードの突き合わせフィールドの内容が、1 次ファイルのレコードの突き合わせフィールドの内容と一致した時。	突き合わせグループの最終レコードに対する合計演算および出力がすべて完了した時。

ファイルおよびプログラムの例外/エラー

RPG は、例外/エラーをプログラムとファイルと いう 2 つのクラスに分類します。 ファイルおよびプログラムの例外/エラーに関する 情報は、それぞれ、ファイル情報データ構造およびプログラム状況データ構造を使用 することによって、RPG IV プログラムで使用することができます。 これらのタイプの例外/エラーを処理するために、ファイルおよびプログラムの 例外/エラー処理サブルーチンを指定することができます。

ファイル例外/エラー

ファイル例外/エラーの例としては、未定義のレコード・タイプ、 トリガー・プログラムのエラー、クローズされているファイルに対する入出力操作、 装置エラー、および配列/テーブルのロード順序エラーなどがあります。 これらは、以下 のいずれかの方法によって処理することができます。

- 命令コード拡張 'E' を指定することができます。 この拡張を、命令の開始前に指定すれば、%ERROR および %STATUS 組み込み関数 がゼロを戻すように設定することができます。 例外/エラーが操作中に発生した 場合、%ERROR 命令が '1' を戻した後、%STATUS がファイル状況を戻します。 任意指定の ファイル情報データ構造は、例外/エラー情報によって更新されます。 %ERROR および %STATUS をテストすることによって、 とるべき処置を決定することができます。
- 命令コードの演算仕様書の 73 から 74 桁目に標識を指定することができます。 この標識は、指定された命令の処理中に例外/エラーが起こった場合にオンに設定されます。 任意指定のファイル情報データ構造は、 例外/エラー情報によって更新されます。 標識をテストすることによって、とるべき処置 を判別するこ とができます。
- ON-ERROR グループを使用して、MONITOR ブロックの中で処理されるステートメントのエラーを処理することができます。 ステートメントの処理の際にエラーが発生すると、適切な ON-ERROR グループに制御が渡ります。
- 例外が起こった時に制御を受け取ることになるユーザー定義の ILE 例外処理 プログラムを作成することができます。 詳しくは、*Rational Development Studio for i: ILE RPG* プログラマーの手引きを参照してください。
- サイクル・モジュールのグローバル・ファイルに対して、ファイル例外/エラー処理サブルーチンを指定することができます。 このサブルーチンは、ファイル仕様書上で、制御を渡す先のサブルーチンの名前を 指定して、INFSR キーワードによって定義します。 ファイル例外/エラーに関する情報は、ファイル仕様書で INFDS キーワードによって指定されるファイル情報データ構造を介して、使用可能にすることができます。 また、プログラムまたは ファイル状況に関して設定された最新の値を戻す %STATUS 組み込み関数も 使用することができます。 ファイルが指定されている場合、%STATUS は、指定されたファイルに 関する INFDS *STATUS フィールドに入っている値を戻します。
- 標識、'E' 拡張、MONITOR ブロック、またはファイル例外/エラー処理サブルーチンがない場合には、ファイル例外/エラーは RPG IV のデフォルトのエラー処理プログラムによって処理されます。

ファイル情報データ構造

ファイル情報データ構造 (INFDS) を各ファイルに定義することによって、ファイル例外/エラーおよびファイル・フィードバックに関する情報を、プログラムまたはプロシージャーで使用可能にすることができます。

ファイル情報データ構造は、各ファイルごとに固有のものであって、ファイルと同じ有効範囲内で定義されていなければなりません。 グローバル・ファイルの場合は、メイン・ソース・セクションで INFDS を定義する必要があります。 サブプロシージャーのローカル・ファイルの場合は、そのサブプロシージャーの定義仕様書で INFDS を定義する必要があります。 さらに、そのファイルと同じ記憶域タイプ (自動または静的) を指定して、INFDS を定義する必要があります。

あるファイルについて定義された INFDS は、そのファイルを使用するすべてのプロシージャーで共用されます。 そのファイルをパラメーターとして渡せば、呼び出し先のプログラムまたはプロシージャーで同じ INFDS が使用されるようになります。

INFDS には、次のフィードバック情報が入れられます。

- ファイル・フィードバック (長さは 80)

- オープン・フィードバック (長さは 160)
- 入出力フィードバック (長さは 126)
- 装置固有のフィードバック (長さは可変長)
- 属性入手フィードバック (長さは可変長)

注 : 属性入手フィードバックは、入出力フィードバックおよび装置固有のフィー ドバックと同じ INFDS の桁を使用します。 これは、属性入手フィードバックが ある場合には、入出力フィードバックまたは 装置固有のフィードバックは持てないこと、あるいはその逆の場合を意味します。

INFDS の長さは、INFDS の中で宣言したフィールドによって異なります。 INFDS の最小の長さは 80 です。

ファイル・フィードバック情報

ファイル・フィードバック情報は、ファイル情報データ構造の 1 桁目から始まり 、80 桁目で終わります。 ファイル・フィードバック情報には、RPG に固有のファイルに関するデータが入 れられます。 これには、次のものを識別するエラー/例外に関する情報が含まれます。

- 例外/エラーが起こったファイルの名前
- 例外/エラーが起こった時に処理されていたレコードまたは例外/エラーの 原因となったレコード
- 例外/エラーが起こった時に処理されていた最後の操作
- 状況コード
- 例外/エラーが発生した RPG IV のルーチン。

INFDS のファイル・フィードバック・セクションの 1 から 66 桁目のフィールドは、プログラム中で INFDS が指定されなくとも常に提供および更新されます。 INFDS の ファイル・フィードバック・セクションの 67 から 80 桁目のフィールドは 、特定の装置に対する POST 命令の後にのみ更新されます。

INFDS が指定されていない場合には、DUMP 命令を使用して、INFDS のファイル・ フィードバック・セクションの情報を出力することができます。詳しくは、768 ページの『DUMP (プログラム・ダンプ)』を参照してください。

INFDS のファイル・フィードバック・セクションの重ね書きは、後続のエラー処 理で予期しない結果になることがあり、お勧めできません。

INFDS のファイル・フィードバック・セクションで一般によく使用される サブフィールドの一部の位置は、特殊なキーワードによって定義されます。 INFDS のファイル・フィードバック・セクションの内容および特殊なキーワード とその説明は、次の表にあります。

表 *62.* ファイル情報データ構造 *(INFDS)* 内で使用できるファイル・フィードバック情報の内容

開始 (Pos.26 から 32)	宛先 (Pos.33 から 39)	様式	length	キーワード	情報
1	8	文字	8	*FILE	ファイル名の最初の 8 文字。
9	9	文字	1		オープン指示 (1 = オープン)。
10	10	文字	1		ファイルの終わり (1 = ファイルの終わり)。
11	15	ゾーン 10 進数	5、0	*STATUS	状況コード。これらのコードの説明については、158 ページの『ファイル状況コード』を参照してください。

表 62. ファイル情報データ構造 *(INFDS)* 内で使用できるファイル・フィードバック情報の内容 *(*続き*)*

開始 (Pos.26 から 32)	宛先 (Pos.33 から 39)	様式	length	キーワード	情報
16	21	文字	6	*OPCODE	命令コード。最初の 5 桁 (左寄せ) は、演算命令コードの文字表現を使用する ことによって命令のタイプを指定します。 たとえば、READE が処理されていた場 合には、READE が左端の 5 桁に入れられます。 命令が暗黙の操作 (たとえば、1 次ファイルの読み取りまたは出力仕様 に対する更新) であった場合には、同等の命令コード (READ または UPDAT など) が生成され、 位置 *OPCODE に入れられます。 6 文字の英字名をもつ命令コードは 5 文字の英字に短縮されます。 **DELETE** DELET **EXCEPT** EXCPT **READPE** REDPE **UNLOCK** UNLCK **UPDATE** UPDAT 残りの桁には、次のいずれかが入ります。 **F** 最後の命令はファイル名に対して指定されました。 **R** 最後の命令はレコードに対して指定されました。 **I** 最後の命令は暗黙のファイル命令でした。
22	29	文字	8	*ROUTINE	ファイル操作が実行されたルーチン (サブルーチンを含む) の名前の最初の 8 文字。
30	37	文字	8		OPTION(*NOSRCSTMT) が指定されている場合、これは、ファイル操作 のソース・リスト行番号です。 OPTION(*SRCSTMT) が指定されている 場合、これは、ファイル操作のソース・リスト・ステートメント番号 です。 そのステートメント番号がルート・ソース・メンバーに適用される 場合、ステートメント番号全体が組み込まれます。 ステートメント番号が 6 桁を超える場合、すなわち、ゼロ以外 のソース ID を組み込んでいる場合、8 バイト・フィードバック域の 最初の 2 桁は、ステートメント番号の残りの部分が 53 から 54 桁目に保管される ことを示す "+" になります。
38	42	ゾーン 10 進数	5、0		SPECIAL ファイルでのエラーに対するユーザー指定の理由。

表 62. ファイル情報データ構造 *(INFDS)* 内で使用できるファイル・フィードバック情報の内容 *(続き)*

開始 (Pos.26 から 32)	宛先 (Pos.33 から 39)	様式	length	キーワード	情報
38	45	文字	8	*RECORD	プログラム記述ファイルの場合には、レコード識別標識は左寄せしてフィールド に入れられ、残りの 6 桁には、ブランクが埋め込まれます。 外部記述ファイルの場合には、例外/エラーが起こった時に処理されていたレコ ードの名前の最初の 8 文字。
46	52	文字	7		マシンまたはシステム・メッセージ番号。
53	66	文字	14		未使用。
77	78	Binary	2		ソース ID は、30 から 37 桁目からのステートメント番号と突き合わせ られます。

表 63. *POST* の後にファイル情報データ構造 *(INFDS)* で使用可能なファイル・フィードバック情報の内容

開始 (Pos.26 から 32)	宛先 (Pos.33 から 39)	様式	length	キーワード	情報
67	70	ゾーン 10 進数	4,0	*SIZE	画面サイズ (装置の画面の行数と桁数の積)。
71	72	ゾーン 10 進数	2,0	*INP	表示装置のキーボード・タイプ。 表示装置が英数字またはカタカナの場合は 00 に設定されます。 キーボードが漢字の場合は 10 に設定されます。
73	74	ゾーン 10 進数	2,0	*OUT	表示装置タイプ。 表示装置が英数字またはカタカナの場合は 00 に設定されます。 表示装置が漢字の場合は 10 に設定されます。 表示装置が DBCS の場合は 20 に設定されます。
75	76	ゾーン 10 進数	2,0	*MODE	常に 00 に設定されます。

INFDS ファイル・フィードバックの例

ファイル・フィードバック・セクションのフィールドが入る INFDS を指定す るために、次の指定を行うことができます。

- ファイル仕様書でファイル情報データ構造の名前と一緒に INFDS キ ーワードを指定します。
- 定義仕様書で使用したいファイル情報データ構造およびサブフィールドを指定 します。
- 自由形式サブフィールド定義の最初のキーワードとして特殊キーワードを指定するか、あるいは、 定義仕様書の FROM フィールド (26 から 32 桁目) に左寄せして特殊キーワードを指定するか、 または FROM フィールド (26 から 32 桁目) および TO フィールド (33 から 39 桁目) にフィールドの位置を指定します。

```
DCL-F MYFILE DISK(*EXT) INFDS(FILEFBK);

DCL-DS FILEFBK;
  FILE          *FILE;                  // File name
  OPEN_IND      IND        POS(9);      // File open?
  EOF_IND       IND        POS(10:      // File at eof?
  STATUS        *STATUS;                // Status code
  OPCODE        *OPCODE;                // Last opcode
  ROUTINE       *ROUTINE;               // RPG Routine
  LIST_NUM      CHAR(8)    POS(30);     // Listing line
  SPCL_STAT     ZONED(5)   POS(38);     // SPECIAL status
  RECORD        *RECORD;                // Record name
  MSGID         CHAR(7)    POS(46);     // Error MSGID
  SCREEN        *SIZE;                  // Screen size
  NLS_IN        *INP;                   // NLS Input?
  NLS_OUT       *OUT;                   // NLS Output?
  NLS_MODE      *MODE;                  // NLS Mode?
END-DS;
```

図 *34.* ファイル・フィードバック情報を持つ *INFDS* の自由形式でのコーディング例

```
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++Comments++++++++++
FMYFILE    IF   E             DISK    INFDS(FILEFBK)
DName+++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++Comments++++++++++
DFILEFBK          DS
D FILE              *FILE                                                  * File name
D OPEN_IND               9      9N                                         * File open?
D EOF_IND               10     10N                                         * File at eof?
D STATUS            *STATUS                                                * Status code
D OPCODE            *OPCODE                                                * Last opcode
D ROUTINE           *ROUTINE                                               * RPG Routine
D LIST_NUM               30     37                                         * Listing line
D SPCL_STAT              38     42S 0                                      * SPECIAL status
D RECORD            *RECORD                                                * Record name
D MSGID                  46     52                                         * Error MSGID
D SCREEN            *SIZE                                                  * Screen size
D NLS_IN            *INP                                                   * NLS Input?
D NLS_OUT           *OUT                                                   * NLS Output?
D NLS_MODE          *MODE                                                  * NLS Mode?
```

図 *35.* ファイル・フィードバック情報を持つ *INFDS* の固定形式でのコーディング例

注 : キーワードはラベルではないので、サブフィールドのアクセスに使用する ことはできません。 記入項目が短い場合には、右側にブランクが埋め込まれます。

オープン・フィードバック情報

ファイル情報データ構造の 81 から 240 桁目には、オープン・フィードバック情 報が入れられます。 INFDS と関連したファイルがオープンされている時にはいつ でも、RPG によってファイル・オープン・フィードバック域の内容が INFDS のオ ープン・フィードバック・セクションにコピーされます。 これには、複数のメン バーが処理されるファイルの読み取り操作の結果としてオープンされたメンバーが 含まれます。

オープン・フィードバック域の内容、およびそれらのフィールドがどのファイル・タイプの場合に有効かについては、IBM i Information Center に説明があります。

INFDS オープン・フィードバックの例

オープン・フィードバック・セクションのフィールドが入る INFDS を指定す るためには、次の記入を行うことができます。

- ファイル仕様書でファイル情報データ構造の名前と一緒に INFDS キ ーワードを指定します。
- 定義仕様書で使用したいファイル情報データ構造およびサブフィールドを指定 します。
- IBM i Information Center のデータベースおよびファイル・システムのカテゴリーにある情報を使用して、INFDS に入れるフィールドを決めます。 INFDS のオープン・フィードバック・セクションのサブフィー

ルドの開始位置と長さを計算するには、Information Center に示されているオフセット、データ・タイプ、および長さを使用し、次の計算を行います。

開始位置 = 81 + オフセット
文字の長さ = 長さ (バイト数)

例えば、印刷装置ファイルのオーバーフロー行番号の場合は、Information Center には次のように示されています。

オフセット = 107
データ・タイプは 2 進数
長さ = 2 バイト

したがって、

開始位置 = 81 + 107 = 188
RPG データ・タイプは整数
長さ = 5 桁

以下の例のサブフィールド OVERFLOW を参照してください。

```
DCL-F MYFILE PRINTER(132) INFDS(OPNFBK);

DCL-DS OPNFBK;
  ODP_TYPE      CHAR(2)    POS(81);     // ODP Type
  FILE_NAME     CHAR(10)   POS(83);     // File name
  LIBRARY       CHAR(10)   POS(93);     // Library name
  SPOOL_FILE    CHAR(10)   POS(103);    // Spool file name
  SPOOL_LIB     CHAR(10)   POS(113);    // Spool file lib
  SPOOL_NUM_OLD INT(5)     POS(123);    // Spool file num
  RCD_LEN       INT(5)     POS(125);    // Max record len
  KEY_LEN       INT(5)     POS(127);    // Max key len
  MEMBER        CHAR(10)   POS(129);    // Member name
  TYPE          INT(5)     POS(147);    // File type
  ROWS          INT(5)     POS(152);    // Num PRT/DSP rows
  COLUMNS       INT(5)     POS(154);    // Num PRT/DSP cols
  NUM_RCDS      INT(10)    POS(156);    // Num of records
  SPOOL_NUM     INT(10)    POS(160);    // 6 digit Spool Nbr
  ACC_TYPE      CHAR(2)    POS(160);    // Access type
  DUP_KEY       CHAR(1)    POS(162);    // Duplicate key?
  SRC_FILE      CHAR(1)    POS(163);    // Source file?
  VOL_OFF       INT(5)     POS(184);    // Vol label offset
  BLK_RCDS      INT(5)     POS(186);    // Max rcds in blk
  OVERFLOW      INT(5)     POS(188);    // Overflow line
  BLK_INCR      INT(5)     POS(190);    // Blk increment
  FLAGS1        CHAR(1)    POS(196);    // Misc flags
  REQUESTER     CHAR(10)   POS(197);    // Requester name
  OPEN_COUNT    INT(5)     POS(207);    // Open count
  BASED_MBRS    INT(5)     POS(211);    // Num based mbrs
  FLAGS2        CHAR(1)    POS(213);    // Misc flags
  OPEN_ID       CHAR(2)    POS(214);    // Open identifier
  RCDFMT_LEN    INT(5)     POS(216);    // Max rcd fmt len
  CCSID         INT(5)     POS(218);    // Database CCSID
  FLAGS3        CHAR(1)    POS(220);    // Misc flags
  NUM_DEVS      INT(5)     POS(227);    // Num devs defined
END-DS;
```

図 *36.* オープン・フィードバック情報を持つ *INFDS* のコーディング例

入出力フィードバック情報

ファイル情報データ構造の 241 から 366 桁目は、入出力フィードバック情報用に 使用されます。 次の場合に、ファイル共通入出力フィードバック域の内容は、RPG によって INFDS の入出力フィードバック・セクションにコピーされます。

- POST 命令の存在がファイルに影響を及ぼす場合:
  - そのファイルに対する POST の後だけ。
- 上記以外の場合、
  - そのファイルに関してブロック化が活動状態で なければ、各入出力操作後。
  - そのファイルに関してブロック化が活動状態であれば、データのブロックの 読み取りまたは書き出しのための、データ管理機能に対する入出力要求の後。

詳しくは、849 ページの『POST (転記)』を参照してください。

入出力フィードバック域の内容の説明は、Information Center にあります。

INFDS 入出力フィードバックの例

入出力フィードバック・セクションのフィールドが入る INFDS を指定するために は、次の記入を行うことができます。

- ファイル仕様書でファイル情報データ構造の名前と一緒に INFDS キ ーワードを指定します。
- 定義仕様書で使用したいファイル情報データ構造およびサブフィールドを指定 します。
- Information Center の情報を使用して、INFDS に入れるフィールドを決めます。 INFDS の入出力フィードバック・セクションのサブフィールドの開始位置と長さを計算するには、Information Center に示されているオフセット、データ・タイプ、および長さを使用し、次の計算を行います。

  開始位置 = 241 + オフセット
  文字の長さ = 長さ (バイト数)

  例えば、ファイルの装置クラスの場合は、Information Center には次のように示されています。

  オフセット = 30
  データ・タイプは文字
  長さ = 2

  したがって、

  開始位置 = 241 + 30 = 271

  以下の例のサブフィールド DEV_CLASS を参照してください。

```
DCL-F MYFILE WORKSTN INFDS(MYIOFBK);

DCL-DS MYIOFBK;
                                        // 241-242 not used
  WRITE_CNT     UNS(10)    POS(243);    // Write count
  READ_CNT      UNS(10)    POS(247);    // Read count
  WRTRD_CNT     UNS(10)    POS(251);    // Write/read count
  OTHER_CNT     INT(10)    POS(255);    // Other I/O count
  OPERATION     CHAR(1)    POS(260);    // Current operation
  IO_RCD_FMT    CHAR(10)   POS(261);    // Rcd format name
  DEV_CLASS     CHAR(2)    POS(271);    // Device class
  IO_PGM_DEV    CHAR(10)   POS(273);    // Pgm device name
  IO_RCD_LEN    INT(10)    POS(283);    // Rcd len of I/O
END-DS;
```

図 *37.* 入出力フィードバック情報を持つ *INFDS* のコーディング例

装置固有のフィードバック情報

ファイル情報データ構造の装置固有のフィードバック情報は INFDS の 367 桁目 から始まり、装置固有の入出力フィードバック情報が入れられます。

装置固有のフィードバック情報が必要な時の INFDS の長さは、ファイルの 装置タイプと DISK ファイルがキー付きかどうかの 2 つの要因によって 異なります。 最 小の長さは 528 ですが、一部のファイルにはそれより長い INFDS が必要です。

- ワークステーション・ファイルの場合には、 INFDS は、すべてのタイプの表示装置または ICF ファイルに関して 241 桁目から開始される装置固有のフィードバック情報を十分に保留できる長さとなります。 たとえば、最長の装置固有のフィードバック情報に 390 バイトが必要な場合には、ワークステーション・ファイルの INFDS は 630 バイト (240+390=630) の長さになります。
- 外部記述 DISK ファイルの場合は、INFDS には少なくとも 401 桁目から開始 されるファイルに最長のキーを十分に保留できる長さが必要になります。

データベース・ファイル、印刷装置ファイル、ICF ファイルおよび表示装置ファイルの、装置フィードバックの内容および長さについて詳しくは、IBM i Information Center の「データベースおよびファイル・システム」のカテゴリーにあります。

次の場合に、ファイルの装置固有の入出力フィードバック域の内容は RPG に よって INFDS の装置固有のフィードバック・セクションにコピーされます。

- POST 命令の存在がファイルに影響を及ぼす場合:
  - そのファイルに対する POST の後だけ。
- 上記以外の場合、
  - そのファイルに関してブロック化が活動状態で なければ、各入出力操作後。
  - そのファイルに関してブロック化が活動状態であれば、データのブロックの 読み取りまたは書き出しのための、データ管理機能に対する入出力要求の後。

注 :

1. 各キー付き入力操作の後では、キー・フィールドのみが更新されます。
2. 各キーなし入力操作の後では、相対レコード番号のみが更新されます。

詳しくは、849 ページの『POST (転記)』を参照してください。

INFDS 装置固有のフィードバックの例

装置固有のフィードバック・セクションのフィールドが入る INFDS を指定するた めには、次の記入を行うことができます。

- ファイル仕様書でファイル情報データ構造の名前と一緒に INFDS キ ーワードを指定します。
- 定義仕様書で使用したいファイル情報データ構造およびサブフィールドを指定 します。
- Information Center の情報を使用して、INFDS に入れるフィールドを決めます。 INFDS の入出力フィードバック・セクションのサブフィールドの開始位置と長さを計算するには、Information Center に示されているオフセット、データ・タイプ、および長さを使用し、次の計算を行います。

  開始位置 = 367 + オフセット
  文字の長さ = 長さ (バイト数)

  例えば、データベース・ファイルの相対レコード番号の場合、Information Center には次のように示されています。

  オフセット = 30
  データ・タイプは 2 進数
  長さ = 4

  したがって、

  開始位置 = 367 + 30 = 397
  RPG データ・タイプは整数
  長さ = 10 桁

以下の例の DBFBK データ構造のサブフィールド DB_RRN を参照してください。

```
DCL-F MYFILE PRINTER(132) INFDS(PRTFBK);

DCL-DS PRTFBK;
  CUR_LINE      INT(5)     POS(367);    // Current line num
  CUR_PAGE      INT(10)    POS(369);    // Current page cnt
  // If the first bit of PRT_FLAGS is on, the spooled file has been
  // deleted.  Use TESTB X'80' or TESTB '0' to test this bit.
  PRT_FLAGS     CHAR(1)    POS(373);    // Print Flags
  PRT_MAJOR     CHAR(2)    POS(401);    // Major ret code
  PRT_MINOR     CHAR(2)    POS(403);    // Minor ret code
END-DS;
```

図 *38.* 印刷装置固有のフィードバック情報を持つ *INFDS* のコーディング例

```
DCL-F MYFILE DISK(*EXT) INFDS(DBFBK);

DCL-DS DBFBK;
  FDBK_SIZE     INT(10)    POS(367);    // Current line num
  JOIN_BITS     INT(10)    POS(371);    // JFILE bits
  LOCK_RCDS     INT(5)     POS(377);    // Nbr locked rcds
  POS_BITS      CHAR(1)    POS(385);    // File pos bits
  DLT_BITS      CHAR(1)    POS(384);    // Rcd deleted bits
  NUM_KEYS      INT(5)     POS(387);    // Num keys (bin)
  KEY_LEN       INT(5)     POS(393);    // Key length
  MBR_NUM       INT(5)     POS(395);    // Member number
  DB_RRN        INT(10)    POS(397);    // Relative-rcd-num
  KEY           CHAR(2000) POS(401);    // Key value (max size 2000)
END-DS;
```

図 *39.* データベース固有のフィードバック情報を持つ *INFDS* のコーディング例

```
DCL-F MYFILE WORKSTN(*EXT) INFDS(ICFFBK);

DCL-DS ICFFBK;
  ICF_AID       CHAR(1)    POS(369);    // AID byte
  ICF_LEN       INT(10)    POS(372);    // Actual data len
  ICF_MAJOR     CHAR(2)    POS(401);    // Major ret code
  ICF_MINOR     CHAR(2)    POS(403);    // Minor ret code
  SNA_SENSE     CHAR(8)    POS(405);    // SNA sense rc
  SAFE_IND      CHAR(1)    POS(413);    // Safe indicator
  RQSWRT        CHAR(1)    POS(415);    // Request write
  RMT_FMT       CHAR(10)   POS(416);    // Remote rcd fmt
  ICF_MODE      CHAR(8)    POS(430);    // Mode name
END-DS;
```

図 *40. ICF* 固有のフィードバック情報を持つ *INFDS* のコーディング例

```
DCL-F MYFILE WORKSTN(*EXT) INFDS(DSPFBK);

DCL-DS DSPFBK;
  DSP_FLAG1     CHAR(2)    POS(367);    // Display flags
  DSP_AID       CHAR(1)    POS(369);    // AID byte
  CURSOR        CHAR(2)    POS(370);    // Cursor location
  DATA_LEN      INT(10)    POS(372);    // Actual data len
  SF_RRN        INT(5)     POS(376);    // Subfile rrn
  MIN_RRN       INT(5)     POS(378);    // Subfile min rrn
  NUM_RCDS      INT(5)     POS(380);    // Subfile num rcds
  ACT_CURS      CHAR(2)    POS(382);    // Active window cursor location
  DSP_MAJOR     CHAR(2)    POS(401);    // Major ret code
  DSP_MINOR     CHAR(2)    POS(403);    // Minor ret code
END-DS;
```

図 *41.* 表示装置固有のフィードバック情報を持つ *INFDS* のコーディング例

属性入手フィードバック情報

ファイル情報データ構造の属性入手フィードバック情報は、INFDS の 241 桁 目から始まり、表示装置または ICF セッション (ワークステーション・ファイルと関連した装 置) に関する情報が入れられます。 属性入手フィードバック情報の終了位置は、属性入手データ管理操作によって戻 されたデータの長さによって異なります。 属性入手データ管理操作は、演算項目 1 にプログラム装置を指定した POST が使用された時に実行されます。

属性入手データの内容および長さについて詳しくは、Information Center にあります。

INFDS 属性入手フィードバックの例

属性入手フィードバック・セクションのフィールドが入る INFDS を指定する ためには、次の記入を行うことができます。

- ファイル仕様書でファイル情報データ構造の名前と一緒に INFDS キ ーワードを指定します。
- 定義仕様書で使用したいファイル情報データ構造およびサブフィールドを指定 します。
- Information Center の情報を使用して、INFDS に入れるフィールドを決めます。 INFDS の属性入手フィードバック・セクションのサブフィールドの開始位置と長さを計算するには、Information Center に示されているオフセット、データ・タイプ、および長さを使用し、次の計算を行います。

  開始位置 = 241 + オフセット
  文字の長さ = 長さ (バイト数)

  例えば、ファイルの装置タイプの場合は、Information Center には次のように示されています。

  オフセット = 31
  データ・タイプは文字
  長さ = 6

  したがって、

  開始位置 = 241 + 31 = 272

  以下の例のサブフィールド DEV_TYPE を参照してください。

```
DCL-F MYFILE WORKSTN INFDS(DSPATRFBK);

DCL-DS DSPATRFBK;
  PGM_DEV       CHAR(10)   POS(241);    // Program device
  DEV_DSC       CHAR(10)   POS(251);    // Dev description
  USER_ID       CHAR(10)   POS(261);    // User ID
  DEV_CLASS     CHAR(1)    POS(271);    // Device class
  DEV_TYPE      CHAR(6)    POS(272);    // Device type
  REQ_DEV       CHAR(1)    POS(278);    // Requester?
  ACQ_STAT      CHAR(1)    POS(279);    // Acquire status
  INV_STAT      CHAR(1)    POS(280);    // Invite status
  DATA_AVAIL    CHAR(1)    POS(281);    // Data available
  NUM_ROWS      INT(5)     POS(282);    // Number of rows
  NUM_COLS      INT(5)     POS(284);    // Number of cols
  BLINK         CHAR(1)    POS(286);    // Allow blink?
  LINE_STAT     CHAR(1)    POS(287);    // Online/offline?
  DSP_LOC       CHAR(1)    POS(288);    // Display location
  DSP_TYPE      CHAR(1)    POS(289);    // Display type
  KBD_TYPE      CHAR(1)    POS(290);    // Keyboard type
  CTL_INFO      CHAR(1)    POS(342);    // Controller info
  COLOR_DSP     CHAR(1)    POS(343);    // Color capable?
  GRID_DSP      CHAR(1)    POS(344);    // Grid line dsp?
  // The following fields apply to ISDN.
  ISDN_LEN      INT(5)     POS(385);    // Rmt number len
  ISDN_TYPE     CHAR(2)    POS(387);    // Rmt number type
  ISDN_PLAN     CHAR(2)    POS(389);    // Rmt number plan
  ISDN_NUM      CHAR(40)   POS(391);    // Rmt number
  ISDN_SLEN     INT(5)     POS(435);    // Rmt sub-address length
  ISDN_STYPE    CHAR(2)    POS(437);    // Rmt sub-address type
  ISDN_SNUM     CHAR(40)   POS(439);    // Rmt sub-address
  ISDN_CON      CHAR(1)    POS(480);    // Connection
  ISDN_RLEN     INT(5)     POS(481);    // Rmt address len
  ISDN_RNUM     CHAR(32)   POS(483);    // Rmt address
  ISDN_ELEN     INT(5)     POS(519);    // Extension len
  ISDN_ETYPE    CHAR(1)    POS(521);    // Extension type
  ISDN_ENUM     CHAR(40)   POS(522);    // Extension num
  ISDN_XTYPE    CHAR(1)    POS(566);    // X.25 call type
END-DS;
```

図 *42.* 表示装置ファイル属性入手フィードバック情報を持つ *INFDS* のコーディング例

```
DCL-F MYFILE WORKSTN INFDS(ICFATRFBK);

DCL-DS ICFATRFBK;
  PGM_DEV       CHAR(10)   POS(241);    // Program device
  DEV_DSC       CHAR(10)   POS(251);    // Dev description
  USER_ID       CHAR(10)   POS(261);    // User ID
  DEV_CLASS     CHAR(1)    POS(271);    // Device class
  DEV_TYPE      CHAR(1)    POS(272);    // Device type
  REQ_DEV       CHAR(1)    POS(278);    // Requester?
  ACQ_STAT      CHAR(1)    POS(279);    // Acquire status
  INV_STAT      CHAR(1)    POS(280);    // Invite status
  DATA_AVAIL    CHAR(1)    POS(281);    // Data available
  SES_STAT      CHAR(1)    POS(291);    // Session status
  SYNC_LVL      CHAR(1)    POS(292);    // Synch level
  CONV_TYPE     CHAR(1)    POS(293);    // Conversation typ
  RMT_LOC       CHAR(10)   POS(294);    // Remote location
  LCL_LU        CHAR(8)    POS(302);    // Local LU name
  LCL_NETID     CHAR(8)    POS(310);    // Local net ID
  RMT_LU        CHAR(8)    POS(318);    // Remote LU
  RMT_NETID     CHAR(8)    POS(326);    // Remote net ID
  APPC_MODE     CHAR(8)    POS(334);    // APPC Mode
  LU6_STATE     CHAR(1)    POS(345);    // LU6 conv state
  LU6_COR       CHAR(8)    POS(346);    // LU6 conv correlator
  // The following fields apply to ISDN.
  ISDN_LEN      INT(5)     POS(385);    // Rmt number len
  ISDN_TYPE     CHAR(2)    POS(387);    // Rmt number type
  ISDN_PLAN     CHAR(2)    POS(389);    // Rmt number plan
  ISDN_NUM      CHAR(40)   POS(391);    // Rmt number
  ISDN_SLEN     INT(5)     POS(435);    // sub-addr len
  ISDN_STYPE    CHAR(2)    POS(437);    // sub-addr type
  ISDN_SNUM     CHAR(40)   POS(439);    // Rmt sub-address
  ISDN_CON      CHAR(1)    POS(480);    // Connection
  ISDN_RLEN     INT(5)     POS(481);    // Rmt address len
  ISDN_RNUM     CHAR(32)   POS(483);    // Rmt address
  ISDN_ELEN     CHAR(2)    POS(519);    // Extension len
  ISDN_ETYPE    CHAR(1)    POS(521);    // Extension type
  ISDN_ENUM     CHAR(40)   POS(522);    // Extension num
  ISDN_XTYPE    CHAR(1)    POS(566);    // X.25 call type
```

```
  // The following information is available only when program was started
  // as result of a received program start request. (P_ stands for protected)
  TRAN_PGM      CHAR(64)   POS(567);    // Trans pgm name
  P_LUWIDLN     CHAR(1)    POS(631);    // LUWID fld len
  P_LUNAMELN    CHAR(1)    POS(632);    // LU-NAME len
  P_LUNAME      CHAR(17)   POS(633);    // LU-NAME
  P_LUWIDIN     CHAR(6)    POS(650);    // LUWID instance
  P_LUWIDSEQ    INT(5)     POS(656);    // LUWID seq num
  // The following information is available only when a protected conversation
  // is started on a remote system.  (U_ stands for unprotected)
  U_LUWIDLN     CHAR(1)    POS(658);    // LUWID fld len
  U_LUNAMELN    CHAR(1)    POS(659);    // LU-NAME len
  U_LUNAME      CHAR(17)   POS(660);    // LU-NAME
  U_LUWIDIN     CHAR(6)    POS(677);    // LUWID instance
  U_LUWIDSEQ    INT(5)     POS(683);    // LUWID seq num
END-DS;
```

図 *43. ICF* ファイル属性入手フィードバック情報を持つ *INFDS* のコーディング例

ブロック化の考慮事項

INFDS の入出力固有のフィードバックのフィールドおよび INFDS の装置固有のフ ィードバック情報セクションのフィールドは、レコードがブロック化および非ブ ロック化されているファイルに対する操作のたびには更新されません。 フィードバック情報が更新されるのは、レコードのブロックが RPG プログ ラムとオペレーティング・システムの間で転送された時のみです。 しかし、データベース・ファイル で入力のブロック化を実行している場合には、INFDS の データベース・フィードバック・セクションの相対レコード番号お よびキーの値は次の場合に更新されます。

- すべての入出力操作時 (プログラム内の POST 命令の存在がファイルに影響を及ぼさない場合)
- ファイルに対する POST の実行後のみ (プログラム内の POST 命令がファイルに影響を及ぼす場合)

849 ページの『POST (転記)』を参照してください。

更新済みの有効なフィードバック情報は、CL コマンドの OVRDBF (データベース・ファイル一時変更) に SEQONLY(*NO) を指定することによって得られます。 ファイル一時変更コマンドを使用した場合には、ILE RPG コンパイラーはファイル中のレコードのブロック化も非ブロック化も実行しません。

RPG でのレコードのブロック化および非ブロック化について詳しくは、*Rational Development Studio for i: ILE RPG* プログラマーの手引きを参照してください。

ファイル状況コード

サブフィールド位置 *STATUS に入るコードで 99 より大きいものは、例外/エラー条件と見なされます。 状況コードが 99 より大きい場合に は、73 から 74 桁目にエラー標識が指定されていれば、その標識がオンに設定され ます。'E' 拡張が指定されていれば、%ERROR 組み込み関数が 1 を戻すように 設定されます。そうでない場合は、ファイル例外/エラー処理サブルーチンが 制御を受け取ります。 位置 *STATUS は各ファイル操作の後に更新されます。

例外/エラーに関する情報を入手するために、%STATUS 組み込み関数を使用 することができます。 この関数は、プログラムまたはファイル状況に関して設定された最新の値を戻します。 ファイルが指定されている場合、%STATUS は、 指定されたファイルに関する INFDS *STATUS フィールドに入っている値を戻します。

ファイル情報データ構造のサブフィールド位置 *STATUS には、以下の表のコード が入れられます。

表 *64.* 通常のコード

コード	Device[1]	RC[2]	条件
00000			例外/エラーなし。
00002	W	n/a	機能キーを使用した表示の終了。
00011	W、D、SQ	11xx	読み取り時のファイルの終わり (入力)。
00012	W、D、SQ	n/a	CHAIN、SETLL、または SETGT 命令でのレコード不在条件。
00013	W	n/a	WRITE 命令でサブファイルがいっぱい。

注 : [1]"装置"とは、条件が適用される装置のことです。 次の略語が使用されます。 P = PRINTER; D = DISK; W = WORKSTN; SP = SPECIAL; SQ = 順次。 RC の欄のメジャー/マイナー戻りコードは、ワークステーション・ファイルにのみ適用されます。[2] メジャー/マイナー戻りコードを記述するために mmnn の形式が使用されています。mm がメジャー戻りコードで、nn がマイナー戻りコードです。

表 *65.* 例外/エラー・コード

コード	Device[1]	RC[2]	条件
01011	W、D、SQ	n/a	定義されていないレコード・タイプ (入力レコードがレコード識別標識と一致 しない)。
01012	D	n/a	キーの数が %KDS に対して無効。
01021	W、D、SQ	n/a	すでに存在しているレコードを書き出そうとした (使用中のファイルに固有の キーがあり、キーが重複した場合、あるいは相対レコード番号を重複してサブファイルに書き出そうとした場合)。
01022	D	n/a	ファイル・メンバーで参照制約エラーが検出された。
01023	D、SQ	n/a	ファイル操作が実行される前のトリガー・プログラムのエラー。
01024	D、SQ	n/a	ファイル操作が実行された後のトリガー・プログラムのエラー。
01031	W、D、SQ	n/a	突き合わせフィールドの順序の誤り。
01041	n/a	n/a	配列/テーブルのロード順序のエラー。

表 65. 例外/エラー・コード (続き)

コード	Device[1]	RC[2]	条件
01042	n/a	n/a	配列/テーブルのロード順序のエラー。 代替照合順序が使用されます。
01051	n/a	n/a	配列/テーブル・ファイルの項目が多すぎる。
01061	n/a	n/a	ファイル・パラメーターに関連付けられている変数のエラー処理
01071	W、D、SQ	n/a	数値順序エラー。
01121[4]	W	n/a	印刷キーの DDS キーワードに標識がない。
01122[4]	W	n/a	前ページ・キーの DDS キーワードに標識がない。
01123[4]	W	n/a	次ページ・キーの DDS キーワードに標識がない。
01124[4]	W	n/a	CLEAR キーの DDS キーワードに標識がない。
01125[4]	W	n/a	HELP キーの DDS キーワードに標識がない。
01126[4]	W	n/a	HOME キーの DDS キーワードに標識がない。
01201	W	34xx	入力でレコードの不一致が検出された。
01211	すべて	n/a	クローズされているファイルへの入出力操作。
01215	すべて	n/a	すでにオープン済みのファイルへ OPEN が出された。
01216[3]	すべて	はい	暗黙の OPEN/CLOSE 命令でのエラー。
01217[3]	すべて	はい	明示の OPEN/CLOSE 命令でのエラー。
01218	D、SQ	n/a	レコードがすでにロックされている。
01221	D、SQ	n/a	前に読み取りを行わないで更新操作を行おうとした。
01222	D、SQ	n/a	参照制約エラーのためにレコードを割り振ることができない。
01231	SP	n/a	SPECIAL ファイルでのエラー。
01235	P	n/a	PRTCTL のスペースまたはスキップ記入項目にエラー。
01241	D、SQ	n/a	レコード番号が見付からない (レコード・アドレス・ファイルに指定されたレコード番号が処理中のファイル にない)。
01251	W	80xx 81xx	永続的な入出力エラーが起こった。
01255	W	82xx 83xx	セッションまたは装置エラーが起こった。 回復の可能性があります。
01261	W	n/a	入手できる装置の最大数を超えようとした。
01271	W	n/a	使用できない装置を入手しようとした。
01281	W	n/a	入手されていない装置への操作。
01282	W	0309	制御付きオプションによるジョブの終了。
01284	W	n/a	単一装置ファイルの 2 番目の装置を入手できない。
01285	W	0800	すでに入手済みの装置を入手しようとした。
01286	W	n/a	共用ファイルを SAVDS または IND オプションによりオープンしようとした。
01287	W	n/a	応答標識が IND 標識と上書きしている。

表 65. 例外/エラー・コード (続き)

コード	Device[1]	RC[2]	条件
01299	W、D、SQ	はい	その他の入出力エラーが検出された。
01331	W	0310	ワークステーション・ファイルからの READ で待ち時間を超過した。

注 :

1. "装置"とは、条件が適用される装置のことを意味します。 次の略語が使用されます。 P = PRINTER; D = DISK; W = WORKSTN; SP = SPECIAL; SQ = 順次。 RC の欄のメジャー/マイナー戻りコードは、ワークステーション・ ファイルにのみ適用されます。
2. メジャー/マイナー戻りコードを記述するために mmnn の形式が使用されていま す。mm がメジャー戻りコードで、nn がマイナー戻りコードです。
3. オープンまたはクローズ操作の時点で発生したエラーは、メジャー/マイナーの 戻りコード値に関係なく、1216 または 1217 の *STATUS 値になります。
4. 特別な処理については、114 ページの図 15 を参照してください。

次の表は、ワークステーション・ファイルだけを使用している プログラムで起こったエラーをマッピングする *STATUS 値に対するメジャー/マイナー戻りコードを示しています。 メジャー/マイナー戻りコードについて詳しくは、Information Center を参照してください。

ﾒｼﾞｬｰ	マイナー	*STATUS
00、02	すべて	00000
03	すべて (09、10 を除く)	00000
03	09	01282
03	10	01331
04	すべて	01299
08	すべて	01285[1]
11	すべて	00011
34	すべて	01201
80、81	すべて	01251
82、83	すべて	01255

注 :

1. 戻りコード・フィールドは、1285、1261、または 1281 の *STATUS 値の場合に は、データ管理機能を呼び出す前にこれらの条件が検出されるので、更新 されません。 これらのエラーのモニターには、対応するメジャー/マイナー戻りコード値で はなく、*STATUS 値を検査しなければなりません。

ファイル例外/エラー処理サブルーチン (INFSR)

ファイル例外/エラーの後で制御を受け取ることができる、ユーザー作成の RPG IV サブルーチンを識別するには、このファイルでの例外/エラーの発生時に制御を受け取るサブルーチンの名前を持つ INFSR キーワードをファイル仕様書に指定してください。 サブルーチン名は、このファイルの例外/エラーについてはプログラム例外/エラー処理サブルーチンに制御が与えられることを指示する *PSSR とすることができます。

例外/エラーが暗黙の (1 次または 2 次) ファイル操作で起こった場合、あるいは 73 から 74 桁目に標識が指定されておらず、(E) 拡張を持たず、かつそのエラーを処理できる MONITOR グループの監視グループ内にもない、明示のファイル操作で起こった場合には、ファイル例外/エラー処理サブルーチン (INFSR) が制御を受け取ります。 ファイル例外/エラー処理サブルーチンは、EXSR 命令コードによって実行すること もで

きます。 RPG IV の命令は、すべてファイル例外/エラー処理サブルーチンの中で使用することができます。 BEGSR 命令の演算項目 1 および EXSR 命令の演算項目 2 には、制御 を受け取るサ ブルーチンの名前 (ファイル仕様書で INFSR キーワードによって指定された のと同じ名前) が入っていなければなりません。

注 : キーワード MAIN または NOMAIN が制御仕様書に指定されている場合、あるいはサブプロシージャーがファイルにアクセスする場合には、INFSR キーワードを指定することはできません。 プロシージャー内のファイルに関するエラーを処理する場合、入出力操作ごとにエラーを処理するには (E) 拡張を使用し、複数の操作についてエラーを処理するには MONITOR グループを使用します。 MONITOR グループの ON-ERROR セクションからサブプロシージャーを呼び出して、さらに詳細なエラー処理を実行することもできます。

ENDSR 命令はファイル例外/エラー処理サブルーチン中の最後の指定であり、 次のように指定しなければなりません。

位置指定
記入

6
C

7 から 11
ブランク

12 から 25
サブルーチン内の GOTO 指定で使用するラベルを入れることができます。

26-35
ENDSR

36 から 49
任意指定の記入項目で、サブルーチンの処理後に制御が戻される地点を指定 します。 この記入項目は 6 桁の文字フィールド、リテラル、または配列要素でな ければならず、その値によって以下の戻り点の 1 つが指定されます。

注 : 戻り点をリテラルとして指定する場合には、アポストロフィで囲む必要が あります。 名前付き定数として指定する場合には、その定数は文字で、 先行ブランクのない戻り点だけが含まれていることが必要です。 フィールド または配列要素の中に指定する場合には、フィールドまたは配列要素 の中で値を左寄せしなければなりません。

***DETL**
明細行の先頭に続きます。

***GETIN**
入力レコードの入手ルーチンに続きます。

***TOTC**
合計演算の先頭に続きます。

***TOTL**
合計行の先頭に続きます。

***OFL**
オーバーフロー行の先頭に続きます。

***DETC**
明細演算の先頭に続きます。

***CANCL**
プログラムの処理を取り消します。

ブランク
RPG IV のデフォルトのエラー処理プログラムへ制御を戻します。 演算項目 2 が ブランクの値および演算項目 2 が指定されていない場合でも、これ が適用されます。 サブルーチンが EXSR 命令によって呼び出され、演算項目 2 が ブランクであった場合には、次の順次命令に制御が戻されます。 ブランクは実行時にのみ有効です。

50 から 76
ブランク。

ファイル例外/エラー処理サブルーチンの指定にあたっては、以下の点に留意し てください。

- プログラマーは、EXSR 命令の演算項目 2 にサブルーチンの名前を指定するこ とによって、ファイル例外/エラー処理サブルーチンを明示的に呼び出すことがで きます。
- ファイル例外/エラー処理サブルーチンの ENDSR 命令が実行された後に、RPG IV 言語では、演算項目 2 に指定されたフィールドまたは配列要素がブランクにリセットされます。 したがって、サブルーチンの処理中にプログラマーがこのフィールドに値を入れなかった場合、サブルーチンが EXSR 命令によって呼び出されたのでなければ、そのサブルーチンの処理後に RPG IV のデフォルトのエラー処理プログラムが制御を受け取ります。 演算項目 2 はブランクに設定される ので、プログラマーは、発生した例外/エラーに最も適したサブルーチンの中の戻り点を指定することができます。 サブルーチン が EXSR 命令によって呼び出され、ENDSR 命令の演算項目 2 がブラ ンクの場合には、その ENDSR 命令の後の次の順次命令に制御が戻されます。 ファ イル例外/エラー処理サブルーチンでは、複数のファイルのエラーを処理すること ができます。
- ファイルの例外/エラーがプログラムの開始時または終了時に発生した場合、RPG IV のデフォルトのエラー処理プログラムに制御が渡され、ユーザー作成のファイル例外/エラーまたはサブルーチン (INFSR) へは渡されません。
- ファイル例外/エラーが起こった場合にはいつでもファイル例外/エラー処理サブルーチンが制御を受け取ることができるので、エラーのあるファイルについ て入出力操作が処理されていた場合には、サブルーチンの実行中に例外/エラー が起こることがあります。 サブルーチンの実行中にすでにエラーになっているフ ァイルで例外/エラーが起こった場合には、サブルーチンが再び呼び出されます。この結果、プログラマーがこの問題を避けるようにサブルーチンをコーディングしていなけ れば、プログラムのループとなります。 このようなプログラム・ループを避ける 1 つの方法として、初回のスイッチをサブルーチンで設定する方法があります 。 このサブルーチンを初めて通るのではない場合には、停止標識をオンに設定して、次のような RETURN 命令を出してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C* If INFSR is already handling the error, exit.
C     ERRRTN        BEGSR
C     SW            IFEQ      '1'
C                   SETON                                        H1
C                   RETURN
C* Otherwise, flag the error handler.
C                   ELSE
C                   MOVE      '1'           SW
C                     :
C                     :
C                     :
C                   ENDIF
C* End error processing.
C                   MOVE      '0'           SW
C                   ENDSR
```

図 *44.* 初回のスイッチの設定

注 : 入出力エラーが起こった後は、ファイルの処理を続行することができなく なることがあります。 続行するためには、そのファイルに対して CLOSE 命令を出 してから、OPEN 命令を出すことが必要な場合があります。

プログラム例外/エラー

プログラム例外/エラーの例としては、ゼロによる除算、負数の SQRT、無効な 配列指標、CALL でのエラー、呼び出されたプログラムからのエラー戻り、および ストリング命令での範囲外の開始位置または長さなどがあります。 これらは、以下 のいずれかの方法によって処理することができます。

- 一部の命令コードに、命令コード拡張 'E' を指定することができま す。 この拡張を、命令の開始前に指定すれば、%ERROR および %STATUS 組み込み関数 がゼロを戻すように設定することができます。 例外/エラーが操作中に発生した 場合、その操作後、%ERROR 命令は '1' を戻し、%STATUS はプログラム状 況を戻します。 任意指定のプログラム状況データ構造は、例外/エラー情報によ って更新されます。 %ERROR および %STATUS をテストすることによって、 とるべき処置を決定することができます。
- 一部の命令コードの演算仕様書の 73 から 74 桁目に標識を 指定することができます。 この標識は、指定された命令の処理中に例外/エラーが起こった場合にオンに設定されます。 任意指定のプログラム状況データ構造は、例外/エラー情報によって更新されます。 標識をテストすることによって、とるべき処置を判別するこ とができます。
- ON-ERROR グループを使用して、MONITOR ブロックの中で処理されるステートメントのエラーを処理することができます。 ステートメントの処理の際にエラーが発生すると、適切な ON-ERROR グループに制御が渡ります。
- 例外が起こった時に制御を受け取ることになるユーザー定義の ILE 例外処理 プログラムを作成することができます。 詳しくは、*Rational Development Studio for i: ILE RPG* プログラマーの手引きを参照してください。
- プログラム例外/エラー処理サブルーチンを指定 することができます。 BEGSR 命令の演算項目 1 に *PSSR を記入して、この サブルーチンを指定してください。 プログラム例外/エラーに関する情報は、プログラム状況データ構造を介して提供されます。これは、自由形式定義では PSDS キーワードで、または固定形式定義では 23 桁目の S で指定されています。 また、プログラムまたは ファイル状況に関して設定された最新の値を戻す %STATUS 組み込み関数も 使用することができます。
- 標識、'E' 拡張、MONITOR ブロック、またはプログラム例外/エラー処理サブルーチンがない場合には、プログラム例外/エラーは RPG IV のデフォルトのエラー処理プログラムによって処理されます。

プログラム状況データ構造

プログラム状況データ構造 (PSDS) を定義して、プログラム例外/エラー情報を RPG IV プログラムに対して使用可能にすることができます。 PSDS はメイン・ソース・セクションに定義されていなければなりません。したがって、1 つのモジュールにつき PSDS は 1 つだけです。

データ構造は、自由形式定義 では PSDS キーワードによって PSDS として定義され、固定形式定義では 23 桁目の S によって定義されます。 PSDS には事前定義サブフィールドが含まれ、発生したプログラム例外/エラーに 関する情報が提供されます。 PSDS 内のサブフィールドの位置は、特殊なキーワードによってか、または事前に 定義された開始位置および終了位置によって定義されます。 サブフィールドにアクセスするためには、各サブフィールドに名前を割り当てま す。 キーワードは、26 から 39 桁目に左寄せして指定しなければなりません。

PSDS からの情報は、定様式ダンプによっても提供されます。 しかし、PSDS がコーディングされていなかったり、PSDS の長さにそれらのフィ ールドが含まれていなかった場合には、PSDS のフィールドに関する情報は定様式 ダンプに含まれていないことがあります。 例えば、PSDS の長さが 275 バイトのみの場合、時刻および日付または実行中のプログラムの情報は 276 バイトから始まるため、それらの情報はダンプに *N/A* (適用されません) と表示されます。 詳細については、768 ページの『DUMP (プログラム・ダンプ)』を参照 してください。

ヒント:

PSDS の 80 バイトより後を埋めるための情報の一部は入手にコストが かかるため、PSDS を指定しないか、あるいは PSDS を 80 バイトより 長くしないことによって、LR をオンにした呼び出しパフォーマンスを改善できる可能性があります。

164 ページの表 66 にデータ構造のサブフィールドのレイアウト、およびサブ フィールドの事前に定義された開始および終了位置が示されているので、これを 使用してこのデータ構造内の情報にアクセスすることができます。

表 66. プログラム状況データ構造の内容

FROM (Pos. 26 から 32)	終了位置 (Pos. 33 から 39)	様式	長さ	キーワード	情報
1	10	文字	10	*PROC	CRTRPGMOD を使用してモジュールをコンパイルした場合には、作成されたモジュールの名前になります。 CRTBNDRPG を使用してプログラムを作成した場合には、作成されたプログラムの名前になります。サイクル・メイン・モジュールの場合には、メイン・プロシージャーの名前になります。
11	15	ゾーン 10 進数	5、0	*STATUS	状況コード。 これらのコードの説明については、168 ページの『プログラム状況コード』を参照してください。
16	20	ゾーン 10 進数	5、0		前の状況コード。
21	28	文字	8		RPG IV ソース・リスト行番号またはステートメント番号。 ソース・リスト 行番号は、OPTION(*SRCSTMT) が OPTION(*NOSRCSTMT) の代わりに 指定されている場合、ソース・リスト・ステートメント番号に 置き換えられます。 そのステートメント番号がルート・ソース・メンバーに適用される 場合、ステートメント番号全体が組み込まれます。 ステートメント番号が 6 桁を超える場合 (すなわち、ゼロ以外のソース ID を組み込んでいる場合)、8 バイト・フィードバック域の最初の 2 桁は、ステートメン ト番号の残りの部分が 354 から 355 桁目に保管されることを示す "+" になります。

表 66. プログラム状況データ構造の内容 (続き)

FROM (Pos. 26 から 32)	終了位置 (Pos. 33 から 39)	様式	長さ	キーワード	情報
29	36	文字	8	*ROUTINE	例外またはエラーが起こった RPG IV ルーチンの名前。 このサブフィールドは RPG IV ルーチンの開始時点、または *STATUS サブフィールドがゼロ以外の値によって更新される場合にのみ、プログラム呼び出しの後に更新されます。 次の名前によってルーチンが識別されます。 ***INIT** プログラム初期化 ***DETL** 明細行 ***GETIN** 入力レコードの入手 ***TOTC** 合計演算 ***TOTL** 合計行 ***DETC** 明細演算 ***OFL** オーバーフロー行 ***TERM** プログラム終了 ***ROUTINE** 呼び出されたプログラムまたはプロシージャーの名前 (最初の 8 文字) **注 :** 通常の RPG IV サイクルを使用していない場合には、*ROUTINE は有効でありません。 プログラムが通常の RPG IV サイクルからはみ出してしまうような論理では、*ROUTINE が正しい値を反映しない場合があります。
37	39	ゾーン 10 進数	3、0	*PARMS	呼び出し側プログラムからこのプログラムに渡されたパラメーターの数。 この値は、%PARMS によって戻り値と同じです。 使用可能な情報がない場合には、-1 が戻されます。
40	42	文字	3		例外タイプ (オペレーティング・システムの例外の場合には CPF、またはマシン例外の場合には MCH)。
43	46	文字	4		例外番号。 CPF 例外の場合には、このフィールドには CPF メッセージ番号が 入れられます。 マシン例外の場合には、マシンの例外番号が入れられます。
47	50	文字	4		予約済み

表 66. プログラム状況データ構造の内容 (続き)

FROM (Pos. 26 から 32)	終了位置 (Pos. 33 から 39)	様式	長さ	キーワード	情報
51	80	文字	30		メッセージ用の作業域。 この区域は、ILE RPG コンパイラーが内部で使用するための専用区域です。 情報の編成は、常に一定であるとは限りません。 この区域は、ユーザーが表示する ことができます。
81	90	文字	10		プログラムが入っているライブラリーの名前。
91	170	文字	80		検索された例外データ。 位置 *STATUS に 09999 が入っている場合に、CPF メッ セージがこのサブフィールドに入れられます。
171	174	文字	4		通知される RNX9001 例外の原因となった例外の識別。
175	184	文字	10		ファイル操作が最後に行われたファイルの名前 (エラーが起こった場合にのみ 更新されます)。 この情報には、常に完全なファイル名が 含まれています。
185	190	文字	6		未使用。
191	198	文字	8		ジョブがシステムに入力された日付 (*DATE 形式)。 バッチ・ジョブが夜間処理として 投入された場合、真夜中を過ぎた時間帯に実行 されるバッチ・ジョブは翌日の日付になります。 この値は、ジョブ日付から 導出され、年は 4 桁に拡張されます。 この値によって表される日付は、270 から 275 桁目で表される日付と同じ日付です。
199	200	ゾーン 10 進数	2,0		4 桁の年の先頭の 2 桁。 *YEAR の先頭の 2 桁と同じ。 このフィールド は、270 から 275 桁目の日付の世紀部分に適用されます。 たとえば、日付が 1999-06-27 の場合、UDATE は 990627 で、この世紀フィールド は 19 になります。 270 から 275 桁目の値とこのフィールドの値を合わせると、191 から 198 桁目にある値の 結合情報になります。 **注 :** この世紀フィールドは、276 から 281 桁目、または 288 から 293 桁目の 日付には適用されません。
201	208	文字	8		ファイル操作が最後に行われたファイルの名前 (エラーが起こった場合にのみ 更新されます)。 長いファイル名が使用されている場合、このファイル名は切り捨て られます。 長いファイル名の情報については、175 から 184 桁目を参照してください。

表 66. プログラム状況データ構造の内容 (続き)

FROM (Pos. 26 から 32)	終了位置 (Pos. 33 から 39)	様式	長さ	キーワード	情報
209	243	文字	35		最後に使用されたファイルの状況情報。 この情報には、状況コード、RPG IV 命令コード、RPG IV ルーチン名、ソース・リスト行番号またはステートメント番号、およびレコード名が 含まれます。 これはエラーが起こった場合にのみ更新されます。 **注 :** 命令コード名は INFDS の *OPCODE と同じ形式になっています。 ソース・リスト 行番号は、OPTION(*SRCSTMT) が OPTION(*NOSRCSTMT) の代わりに 指定されている場合、ソース・リスト・ステートメント番号に 置き換えられます。 そのステートメント番号がルート・ソース・メンバーに適用される 場合、ステートメント番号全体が組み込まれます。 ステートメント番号が 6 桁を超える場合 (すなわち、ゼロ以外 のソース ID を組み込んでいる場合)、8 バイト・フィードバック域の 最初の 2 桁は、ステートメント番号の残りの部分が 356 から 357 桁目に保管される ことを示す "+" になります。
244	253	文字	10		ジョブ名。
254	263	文字	10		ユーザー・プロファイルからのユーザー名。
264	269	ゾーン 10 進数	6、0		ジョブ番号。
270	275	ゾーン 10 進数	6、0		プログラムの実行がシステムで開始された日付 (UDATE 形式) (UDATE はこの 日付から導出されます)。 UDATE の説明について は、77 ページの『ユーザー日付の特殊語』を参照してください。 これは、一般に「ジョブ日付」と 呼ばれています。 この値によって表される日付は、191 から 198 桁目で表される日付と同じ日付です。
276	281	ゾーン 10 進数	6、0		プログラム実行の日付 (UDATE 形式のシステム日付)。 この値の年の部分 が 40 から 99 の間の場合、この日付は 1940 から 1999 の間になります。 それ以外の場合、この日付は 2000 から 2039 の間になります。 199 から 200 桁目の「世紀」値は、このフィールドには適用されません。
282	287	ゾーン 10 進数	6、0		プログラム実行の時刻 (hhmmss 形式)。
288	293	文字	6		プログラムがコンパイルされた日付 (UDATE 形式)。この値の年の部分 が 40 から 99 の間の場合、この日付は 1940 から 1999 の間になります。 それ以外の場合、この日付は 2000 から 2039 の間になります。 199 から 200 桁目の「世紀」値は、このフィールドには適用されません。
294	299	文字	6		プログラムがコンパイルされた時刻 (hhmmss 形式)。

表 66. プログラム状況データ構造の内容 (続き)

FROM (Pos. 26 から 32)	終了位置 (Pos. 33 から 39)	様式	長さ	キーワード	情報
300	303	文字	4		コンパイラーのレベル。
304	313	文字	10		ソース・ファイル名。
314	323	文字	10		ソース・ライブラリー名。
324	333	文字	10		ソース・ファイル・メンバー名。
334	343	文字	10		プロシージャーが入っているプログラム。
344	353	文字	10		プロシージャーが入っているモジュール。
354	355	Binary	2		ソース ID は、21 から 28 桁目からのステートメント番号と突き合わせ られます。
356	357	Binary	2		ソース ID は、228 から 235 桁目からのステートメント番号と突き合わせ られます。
358	367	文字	10		現在のユーザー・プロファイル名。
368	371	整数	10,0		外部エラー・コード
372	379	整数	20,0		XML-INTO または DATA-INTO によって設定される要素
380	395	文字	16		内部ジョブ ID。
396	403	文字	8		システム名。
404	429	文字	50		未使用。

プログラム状況コード

サブフィールド位置 *STATUS に入るコードで 99 より大きいものは、例外/エラー条件と見なされます。 状況コードが 99 より大きい場合には、73 から 74 桁目に エラー標識が指定されていれば、その標識がオンに設定されます。'E' 拡張が指定されていれば、%ERROR 組み込み関数が '1' を戻すように設定されるか、または MONITOR ブロックの中の適切な ON-ERROR グループに制御が渡されます。そうでない場合は、プログラム例外/エラー処理サブルーチンが制御を受け取ります。 位置 *STATUS は、例外/エラーが起こった時に更新されます。

%STATUS 組み込み関数は、プログラムまたはファイル状況に関して設定された最新の値を戻します。

プログラム状況データ構造のサブフィールド位置 *STATUS には、次のコードが入 れられます。

通常のコード

コード
条件

00000
例外/エラーは起こっていない。

00001
呼び出されたプログラムから LR 標識がオンになって戻った。

00050
変換の結果、置換が行われる。

例外/エラー・コード

コード
条件

00100
ストリング命令の範囲外の値。

00101
負の平方根。

00102
ゼロによる除算。

00103
中間結果が結果を入れるだけ大きくない。

00104
浮動アンダーフロー。 中間値が小さ過ぎて、中間結果フィールドに 入りません。

00105
文字の中にある、数値変換関数に対して無効な文字。

00112
日付、時刻、またはタイム・スタンプの値が正しくない。

00113
日付オーバーフローまたはアンダーフロー。 (たとえば、日付演算の結果 が *HIVAL より大きいかまたは *LOVAL より小さい数になるとき。)

00114
日付が 4 文字の年から 2 文字の年にマップされ、日付の範囲が 1940 から 2039 に ない日付マッピング・エラー。

00115
可変長フィールドの現在の長さが正しくない。

00120
テーブルまたは配列の順序が違っている。

00121
配列指標が正しくない。

00122
OCCUR が範囲外。

00123
プログラムの初期化ステップでリセットしようとした。

00124
変動次元配列の要素の数が無効である。

00125
ストリング・オペランドが無効です。

00202
呼び出されたプログラムまたはプロシージャーが正常に実行されず、停止標 識 (H1 から H9) はオンでない。

00211
呼び出し側プログラムまたはプロシージャーのエラー。

00222
ポインターまたはパラメーター・エラー。

00231
呼び出されたプログラムまたはプロシージャーから停止標識がオンになって 戻った。

00232
このプログラムで停止標識がオンになった。

00233
RETURN 命令の実行時に停止標識がオンになった。

00299
RPG IV 定様式ダンプが失敗した。

00301
メソッドの呼び出しでクラスまたはメソッドが見付からなかったか、あるいはメソッドの呼び出しでエラー。

00302
Java ネイティブ・メソッドへの入り口において、Java 配列から RPG パラメーターへの変換時にエラー。

00303
RPG ネイティブ・メソッドからの出口において、RPG パラメーターから Java 配列への変換でエラー。

00304
Java メソッドの呼び出しの準備中に、RPG パラメーターから Java 配列への変換でエラー。

00305
Java メソッド後の Java 配列から RPG パラメーターまたは戻り値への変換でエラー。

00306
RPG 戻り値から Java 配列への変換でエラー。

00333
DSPLY 命令のエラー。

00351
XML 文書の構文解析時エラー。

00352
%XML の無効オプション。

00353
XML 文書が RPG 変数に一致しない。

00354
XML 構文解析の準備時エラー。

00355
DATA-GEN 命令または DATA-INTO 命令で、プログラムまたはプロシージャーを使用できない。

00356
DATA-INTO 命令の文書が RPG 変数と一致しない。

00356
DATA-INTO 命令の文書が RPG 変数と一致しない。

00357
DATA-INTO 命令のパーサーがエラーを検出した。

00358
DATA-INTO 命令でパーサーによって提供された情報にエラーがあった。

00359
DATA-INTO 命令または DATA-GEN 命令のプログラムまたはプロシージャーの実行中にエラーが発生した。

00361
DATA-GEN の生成プログラムの RPG 変数にあるデータを準備しているときにエラーが発生した。

00362
DATA-GEN 命令で構文解析プログラムによって提供された情報にエラーがあった。

00363
DATA-GEN 命令の生成プログラムがエラーを検出した。

00364
DATA-GEN 命令の出力ファイルまたは出力変数の処理中にエラーが発生した。

00365
DATA-GEN 命令のシーケンスでエラーが発生した。

00401
IN/OUT で指定されたデータ域が見付からない。

00402
事前開始でないジョブに対しては *PDA は無効。

00411
データ域のタイプまたは長さが一致しない。

00412
データ域が出力用にロックされていない。

00413
IN/OUT 命令のエラー。

00414
ユーザーにデータ域を使用する権限がない。

00415
ユーザーにデータ域を変更する権限がない。

00421
UNLOCK 命令のエラー。

00425
記憶割り振りに必要な長さが範囲外。

00426
記憶域管理操作中にエラーが起こった。

00431
データ域は別のプログラムによってすでにロック済みである。

00432
データ域は同じ処理内のプログラムによってロック済みである。

00450
文字フィールド全体がシフトアウトおよびシフトイン文字で囲まれてい ない。

00451
2 つの CCSID 間の変換はサポートされない。

00452
2 つの CCSID 間で変換できない文字があった。

00453
2 つの CCSID 間の変換中にエラーが発生した。

00501
分類順序検索の障害。

00502
分類順序変換の障害。

00802
コミットメント制御は活動状態になっていない。

00803
ロールバック操作が正常に実行されなかった。

00804
COMMIT 命令でエラーが起こった。

00805
ROLBK 命令でエラーが起こった。

00907
10 進数データ・エラー (数字または符号が無効)。

00970
プログラムの生成に使用されたコンパイラーのレベル番号が、RPG IV の実行時サブルーチンのレベル番号と一致しない。

09998
ILE RPG コンパイラーまたは実行時サブルーチンの内部障害。

09999
システム・ルーチン内のプログラム例外。

PSDS の例

プログラム内で PSDS を指定するためには、使用したいプログラム状況データ構 造およびサブフィールドを定義仕様書にコーディングします。

```
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++Comments++++++++++
DMYPSDS          SDS
D PROC_NAME         *PROC                                          * Procedure name
D PGM_STATUS        *STATUS                                        * Status code
D PRV_STATUS             16     20S 0                              * Previous status
D LINE_NUM               21     28                                 * Src list line
num
D ROUTINE           *ROUTINE                                       * Routine name
D PARMS             *PARMS                                         * Num passed parms
D EXCP_TYPE              40     42                                 * Exception type
D EXCP_NUM               43     46                                 * Exception number
D PGM_LIB                81     90                                 * Program library
D EXCP_DATA              91    170                                 * Exception data
D EXCP_ID               171    174                                 * Exception Id
D DATE                  191    198                                 * Date (*DATE fmt)
D YEAR                  199    200S 0                              * Year (*YEAR fmt)
D LAST_FILE             201    208                                 * Last file used
D FILE_INFO             209    243                                 * File error info
D JOB_NAME              244    253                                 * Job name
D USER                  254    263                                 * User name
D JOB_NUM               264    269S 0                              * Job number
D JOB_DATE              270    275S 0                              * Date (UDATE fmt)
D RUN_DATE              276    281S 0                              * Run date (UDATE)
D RUN_TIME              282    287S 0                              * Run time (UDATE)
D CRT_DATE              288    293                                 * Create date
D CRT_TIME              294    299                                 * Create time
D CPL_LEVEL             300    303                                 * Compiler level
D SRC_FILE              304    313                                 * Source file
D SRC_LIB               314    323                                 * Source file lib
D SRC_MBR               324    333                                 * Source file mbr
D PROC_PGM              334    343                                 * Pgm Proc is in
D PROC_MOD              344    353                                 * Mod Proc is in
```

図 *45. PSDS* のコーディング例

注 : キーワードはラベルではないので、サブフィールドのアクセスに使用する ことはできません。 記入項目が短い場合には、右側にブランクが埋め込まれます。

プログラム例外/エラー処理サブルーチン

プログラム例外/エラーが発生したときに制御を受け取るユーザー作成の RPG IV サブルーチンを識別するには、サブルーチンの BEGSR 命令の演算項目 1 に *PSSR を指定してください。 命令コードの 73 から 74 桁目に標識が指定されていない場合、命令に (E) 拡張がない場合、そのステートメントがそのエラーを処理できる MONITOR グループ内にない場合、あるいは命令コードで予期されていない例外 (すなわち、SCAN 命令時の配列指標エラー) が起こった場合には、プログラム例外/エラーが起こった時点でこのサブルーチンに制御が移されます。 さらに、サブルーチンは EXSR 命令から呼び出すこともできます。ファイル仕様書の INFSR キーワードに *PSSR を指定して、ファイル例外/エラーが発生した場合に制御を受け取ることができます。

どの RPG IV 命令コードも、プログラム例外/エラー処理サブルーチンの中で使用することができます。ENDSR 命令はサブルーチンに対する最後の指定でなけ ればならず、ENDSR 命令の演算項目 2 の記入項目には、そのサブルーチンの実行 後の戻り点が指定されます。演算項目 2 における有効な記入項目の説明については、160 ページの『ファイル例外/エラー処理サブルーチン (INFSR)』 を参照してください。

プログラム例外/エラー処理サブルーチンを指定する場合には、以下の点に留意 してください。

- EXSR 命令の演算項目 2 に *PSSR を指定することによって、*PSSR サブルー チンを明示的に呼び出すことができます。
- *PSSR サブルーチンの ENDSR 命令の実行後に、RPG IV 言語によって、演算項目 2 に指定されたフィールド、サブフィールド、または配列要素がブランクにリセットされます。 これによって、発生した例外/エラーに最も適したサブルーチン内の戻り点を 指定することができます。 サブルーチンの終わりで演算項目 2 にブランクが入っていた場合には、RPG IV のデフォルトのエラー処理プログラムが制御を受け取り、サブルーチンが EXSR または CASxx 命令によって呼び出されていた場合には、制御はその EXSR または ENDCS の後の次の順次命令に戻されます。
- ファイル例外/エラー以外のものが起こった場合には、常にプログラム例外/エラー処理サブルーチンが制御を受け取ることになるので、サブルーチンの実行中 も例外/エラーが起こることがあります。 サブルーチンの実行中に例外/エラー が起こった場合には、そのサブルーチンが再び呼び出されます。プログラマーが この問題を避けるようにサブルーチンをコーディングしていなければ、これがプ ログラム・ループの原因となります。
- CRTBNDRPG または CRTRPGMOD コマンドのいずれかで OPTIMIZE(*FULL) を使 用した場合には、例外処理時に参照するすべてのフィールドを、フィールドに対 する定義仕様書の NOOPT キーワードによって宣言しなければなりません。 これにより、ユーザー・プログラムで実行する時に、例外処理時に参照されるフ ィールドが確実に現在の値を持つことになります。
- *PSSR をサブプロシージャーの中で定義し、各サブプロシージャーにはその 独自の *PSSR を入れることができます。 サブプロシージャー内の *PSSR は、そのサブプロシージャーに対してローカルな ものであることに注意してください。 サブプロシージャーに同じ例外ルーチンを共用させたい場合には、各 *PSSR で共 用プロシージャーを呼び出すことが必要です。

ファイルに関する一般的な考慮事項

この章は、以下の事項についてさらに詳しく説明しています。

- グローバル・ファイルとローカル・ファイル
- ファイル・パラメーター
- オープン・アクセス・ファイル
- ファイルに関連付けられる変数
- 複数ファイル処理
- 突き合わせフィールド
- 代替照合順序
- ファイル変換

ファイル名に関する規則

コンパイル時:

- ファイルがプログラム記述である場合、ファイルは存在していなくてもかまいません。
- ファイルが外部記述である場合、ファイルは存在している必要がありますが、IBM i システム一時変更コマンドを使用して、その名前を IBM i システムに対して定義済みのファイルに関連付けることも、システムに対して定義済みのファイルを EXTDESC キーワードを使用して示すこともできます。
- RPG プログラム内のファイル名が 10 文字より長い場合は、EXTFILE キーワードを指定する必要があります。 ファイルが外部記述である場合、EXTDESC キーワードを指定する必要があります。

実行時:

- EXTFILE キーワード、EXTMBR キーワードのいずれかまたは両方を使用している場合、RPG はこれらのキーワードで名前を指定されたファイルをオープンします。
- それ以外の場合、RPG は、RPG プログラム内のファイルの名前と同じ名前のファイル をオープンします。 このファイル (または一時変更されるファイル) は、ファイルがオープンされるときに存在している必要があります。
- RPG が開くファイルに対して IBM i システムの一時変更コマンドが使用されていた場合、この一時変更が効力を持ち、開かれる実際のファイルはこの一時変更によって決まります。 一時変更とこのキーワードの相互作用について詳しくは、370 ページの『EXTFILE(ファイル名 | *EXTDESC)』 を参照してください。

実行時に、USROPN キーワードによる定義がされていないファイルがオープンされる場合には、それらはファイル仕様書に指定されたのと逆の順序でオープンされます。 RPG IV 装置名によって、関連したファイルについて処理することのできる操作が定義されます。

ファイル装置

ファイルの装置は、自由形式ファイル定義では 装置タイプ・キーワードによって指定され、 固定形式ファイル定義では 装置記入項目によって指定されます。

RPG IV 装置名によって、関連したファイルについて実行することができる ILE RPG 機能が定義されます。 一部の機能は、 特定の ILE RPG 装置名にのみ有効です (例えば、WORKSTN 装置に対する EXFMT 命令など)。

注 : 自由形式ファイルに装置タイプ・キーワードの指定がなく、 LIKEFILE キーワードも指定されていない場合、装置は DISK(*EXT) にデフォルト設定されます。

ファイルの装置タイプ

PRINTER
このファイルは印刷装置ファイル、すなわち、印刷装置へ送ることのできる 制御文字の入ったファイルです。

ディスク
このファイルはディスク・ファイルです。 この装置は、順次およびランダム 処理の読み取り/書き込み機能をサポートします。 これらのファイルは、分散データ管理機能 (DDM) によってリモート・システムでアクセ スすることができます。

WORKSTN
このファイルはワークステーション・ファイルです。 入出力はディスプレイまたは ICF ファイルを通じて処理されます。

SPECIAL
このファイルは特殊ファイルです。 入力または出力は、ユーザー提供のプログラムによってアクセスされる装置上で 行われます。 プログラムの名前は、PGMNAME キーワードに対するパラメーターとして指定しな ければなりません。 このプログラムによって使用される、オプション・コード・パラメーターおよび 状況コード・パラメーターを含めたパラメーター・リストが作成されます。 このファイルは固定長の 非ブロック化形式でなければなりません。 詳しくは、381 ページの『PLIST(PLIST 名)』 および 381 ページの『PGMNAME(プログラム名)』 を参照してください。

SEQ
このファイルは順次編成ファイルです。 実際の装置は CL コマンドまたはファイル記述に指定され、ファイル名によって アクセスされます。

グローバル・ファイルとローカル・ファイル

RPG IV モジュールでは、モジュール内のすべてのプロシージャーで使用できるグローバル・ファイル、または単一のプロシージャーでのみ使用できるローカル・ファイルを定義することができます。 グローバル・ファイルは、制御仕様書と定義仕様書の間にあるメイン・ソース・セクションで定義します。 グローバル・ファイルは、1 次ファイル、2 次ファイル、テーブル・ファイル、または全プロシージャー・ファイルとして使用できます。 ローカル・ファイルは、プロシージャー仕様書と定義仕様書の間にあるサブプロシージャー内で定義します。 ローカル・ファイルは、全プロシージャー・ファイルとしてのみ使用できます。 グローバル・ファイルのフィールド・データを処理できるように、入力仕様および出力仕様を定義することができます。

入力仕様および出力仕様はサブプロシージャーではサポートされていないので、ローカル・ファイルに対するすべての入力操作および出力操作は、データ構造または %FIELDS を使用して行う必要があります。

オープン・アクセス・ファイル

オープン・アクセス・ファイルは、オペレーティング・システムではなく、ユーザー作成のプログラムまたはプロシージャーによってすべての操作が処理されるファイルです。 このプログラムまたはプロシージャーは、「オープン・アクセス・ハンドラー」、または単純に「ハンドラー」と呼ばれます。

例えば、オープン・アクセス・プリンター・ファイルのハンドラーは、 ファイルのオープン、ファイルへのデータ書き込み、およびファイルのクローズが 行われるときに制御権を獲得します。 その後、ハンドラーは、いつオーバーフローが起こるのかを決定します。

オープン・アクセス・ファイルは、ファイル定義に HANDLER キーワードを指定することによって定義されます。 プログラム記述ファイルまたは外部記述ファイルのどちらでもかまいませんが、 特定のハンドラーは、サポートするファイルのタイプに関して独自の制限を設定することがあります。

ファイルは、特定のハンドラーがそのファイルを必要としない限り、実行時には必要ありません。

HANDLER キーワード以外には、オープン・アクセス・ファイルは、HANDLER キーワードがない場合と同じように 使用されます。

HANDLER キーワードの例については、372 ページの『HANDLER(プログラムまたはプロシージャー{:通信域})』 を参照してください。

オープン・アクセス・ハンドラーの例については、177 ページの『オープン・アクセス・ハンドラーの例』の例を参照してください。

オープン・アクセス・ハンドラーの作成については、トピック 『*Rational Open Access: RPG* 版』を参照してください。

オープン・アクセス・ハンドラーの探索

オープン・アクセス・ハンドラーは、ファイルがオープンされるときに システム上で探索されます。 ファイルがクローズされるまで、後続のすべての 命令に対して、同じハンドラーが呼び出されます。

例えば、 キーワード *HANDLER(handlerName)* を指定して定義されたファイルがあるとします。

このファイルが開かれるときに変数 *handlerName* の値が「MYPGM」であり、プログラム MYPGM がライブラリー MYLIB 内で見つかった場合、ファイルを開くためにプログラム MYLIB/MYPGM が呼び出されます。

入力命令の前に変数 *handlerName* の値 が「MYSRVPGM(myProc)」に変更されても、その影響は ありません。入力命令を処理するためにプログラム MYLIB/MYPGM が呼び出されます。

オープン・アクセス・ハンドラー

オープン・アクセス・ハンドラーは、オープン・アクセス・ファイルに対するすべての命令 の処理を担当します。

ハンドラーは、ファイルがオープンされるとき、クローズされるとき、 および、ファイルに対する入出力命令が行われるときに呼び出されます。

ハンドラーに渡されるパラメーター

ハンドラーには単一のパラメーターが渡されます。ファイル QOAR/QRPGLESRC 内のコピー・メンバー QRNOPENACC には、データ構造テンプレート *QrnOpenAccess_T* が含まれていて、それを LIKEDS キーワードで使用することでハンドラー内のパラメーターを定義できます。

ハンドラー・パラメーターには、他の構造をポイントする多くのポインター・サブフィールドが含まれています。 コピー・ファイルには、他にもデータ構造テンプレートが含まれていて、 これらの他の構造を定義するために使用できます。 例えば、キーワード LIKEDS(QrnPrtctl_T) を使用して定義される構造の 基底ポインターとして *prtctl* サブフィールドを使用できます。

コピー・ファイルには、ハンドラー内で使用できる名前付き定数もいくつか含まれています。 例えば、*QrnOperation_OPEN* など、名前が *QrnOperation_* で始まる 名前付き定数がいくつかあり、ハンドラー・パラメーターのサブフィールド *rpgOperation* とともに使用することができます。

サブフィールドは、必要な命令を実行するためにハンドラーが必要とするすべての情報を ハンドラーに提供します。 例えば、出力命令の場合、ファイルに書き込むデータを受け取ります。

また、ハンドラーは、サブフィールドを使用することで、命令の後に RPG によって必要とされるすべての情報を 戻すこともできます。 例えば、入力命令の場合、ハンドラーは入力データを戻したり、 ファイルの終わりに達したかどうかの情報を戻したりすることができます。

ハンドラーが RPG プログラマーと直接通信する必要がある場合、 ハンドラーの作成者は RPG プログラマーに、HANDLER キーワードの通信域 パラメーターを 指定するように依頼できます。 RPG プログラムおよびハンドラーの作成者は、通信域パラメーターの定義がハンドラーと RPG プログラムで 同じであるように注意しなければなりません。 通常、ハンドラー作成者がコピー・ファイル内にテンプレート・データ構造を用意し、 それを RPG プログラマーが使用して通信域パラメーターを定義できるようにします。

注 : 通信域はユーザー域 とも呼ばれます。 ハンドラー・パラメーター内の *userArea* サブフィールド は、RPG プログラム内で HANDLER キーワードに指定される通信域パラメーターを指す ポインターです。

ハンドラーは、ハンドラーのすべての呼び出しで使用可能な状態情報を保持する必要が ある場合、ハンドラー・パラメーター内 の *stateInfo* ポインター・サブフィールドを使用できます。 ある 1 回のハンドラー呼び出し中にこのサブフィールドにハンドラーがポインター値を 入れると、同じ値が、その特定のファイルに対するそれ以降のすべてのハンドラー呼び出しで 使用可能になります。 通常、ハンドラーは OPEN 命令を処理している間に状態情報のための記憶域を 割り振り、CLOSE 命令の処理中にその記憶域を割り振り解除します。

ハンドラーでのエラー

ハンドラーが未処理例外で失敗した場合、RPG 命令はその命令に関連する状況コードで失敗 します。 例えば、命令が OPEN または CLOSE の場合、エラー状況は 1216 または 1217 のいずれかです。 他の命令の場合の状況は 1299 です。

ハンドラーがエラーを検出した場合、RPG プログラムに障害を知らせる手段として次の 2 つの方法があります。

- 例外メッセージを送信する。これによって、ハンドラーは未処理の例外で終了します。 このメッセージはジョブ・ログに記録され、後続の RPG エラー・メッセージ はこのエラー・メッセージを参照します。

  この方法の利点は、ハンドラーは例外メッセージが送信されるとすぐに 終了し、したがって、ハンドラーは命令が失敗したかどうかを追跡する必要がないことです。
- ハンドラー・パラメーターの *rpgStatus* サブフィールド を適切な RPG 状況コードに設定する。 これは、診断メッセージをジョブ・ログに送信するのにも役立ちます。

  この方法の利点は、正確な状況コードをハンドラーが選択できること です。 例えば、WORKSTN に対する命令に関連付けられたいくつかの状況コードが あります。

オープン・アクセス・ハンドラーの作成については、トピック 『Rational Open Access: RPG 版』を参照してください。

オープン・アクセス・ハンドラーの例

注 : この例の説明では、オープン・アクセス・ファイルに関連した部分についてのみ詳しく説明されています。 IFS ファイルのオープン、書き込み、およびクローズを行うコードや、 例外を通知するコードなど、説明なしで示されているコードもあります。

この例では、ハンドラーを使用することによって、RPG プログラマーが統合ファイル・システムのストリーム・ファイル を読み取ることができます。

このハンドラーの作成者は、RPG プログラムとハンドラーとの間で通信域として 使用されるデータ構造のテンプレートが入っているコピー・ファイルも用意しています。 このデータ構造は、ファイルのパスと、ハンドラーがファイルを作成する方法を制御するための他のオプション を定義します。 コピー・ファイルには、ハンドラー・プログラムの名前が入っている 名前付き定数 *IFSHDLRS_read_handler* も含まれています。

```
/IF DEFINED(IFSHDLRS_COPIED)
/EOF
/ENDIF
/DEFINE IFSHDLRS_COPIED

DCL-DS ifshdlrs_info_t QUALIFIED TEMPLATE;
   path VARCHAR(5000);
   createCcsid INT(10);
   append IND;
END-DS;

DCL-C ifshdlrs_write_handler 'IFSHDLRS/WRITEHDLR';
```

オープン・アクセス・ファイルを定義する RPG プログラムを以下に示します。

プログラムの以下の局面に注意してください。

1. ハンドラー用に用意されたコピー・ファイル。
2. ハンドラーと直接通信するために使用される通信域データ構造。 このデータ構造は、ハンドラーに渡されるパラメーター中の *userArea* サブフィールド によってポイントされます。
3. オープン・アクセス・ファイル。 この例のハンドラーは、プログラム記述ファイルと外部記述ファイルの両方をサポートします。 このプログラム例では、次のソース・ファイルから定義された外部記述ファイル が使用されています。

   ```
   A          R STREAMFMT
   A            LINE       32740A         VARLEN
   ```

4. HANDLER キーワード。
   a. HANDLER キーワードの最初のパラメーターは、ハンドラー・プログラムまたはプロシージャーを定義する、コピー・ファイルからの 名前付き定数です。
   b. HANDLER キーワードの 2 番目のパラメーターは、 通信域データ構造です。
5. このプログラムは、ハンドラーが必要とする追加情報で通信域を セットアップし、次に、ファイルをオープンします。
6. 2 つのレコードをファイルに書き込みます。

```
CTL-OPT DFTACTGRP(*NO) ACTGRP(*NEW);

/copy IFSHDLRS/SRC,RPG  1
DCL-DS ifs_info LIKEDS(ifshdlrs_info_t);  2
DCL-F streamfile DISK(*EXT) USAGE(*OUTPUT)  3
                 EXTDESC('MYLIB/MYSTMF')
                 HANDLER(ifshdlrs_write_handler  4a
                       : ifs_info)               4b
                 USROPN;

ifs_info.path = '/home/mydir/myfile.txt';  5
ifs_info.createCcsid = 0;  // job CCSID
ifs_info.append = *ON;
OPEN streamfile;

line = 'Hello';  6
WRITE streamFmt;
line = 'world!';
WRITE streamFmt;
*inlr = '1';
```

以下の例はハンドラーを示しています。

- 178 ページの『制御ステートメント、コピー・ファイル、およびグローバル定義』
- 178 ページの『メイン・ハンドラー・プロシージャー』
- 180 ページの『ファイルをオープンするプロシージャー』
- 181 ページの『ファイルをクローズするプロシージャー』
- 181 ページの『出力バッファーを使用してファイルに書き込むプロシージャー』
- 182 ページの『名前-値情報を使用してファイルに書き込むプロシージャー』
- 182 ページの『ファイルに 1 行を書き込むプロシージャー』
- 183 ページの『例外を送信するプロシージャー』
- 183 ページの『"errno" に関連する例外を送信するプロシージャー』
- 184 ページの『"errno" の値を取得するプロシージャー』

制御ステートメント、コピー・ファイル、およびグローバル定義

1. テンプレートのデータ構造 *state_t* は、 ハンドラーへのすべての呼び出しでハンドラーが必要とする情報を定義します。 この例では、ハンドラーはオープン・ファイルのディスクリプターを追跡する必要があります。

```
CTL-OPT DFTACTGRP(*NO) ACTGRP(*CALLER)
        MAIN(writeHdlr);

/COPY IFSHDLRS/SRC,RPG
/COPY QOAR/QRPGLESRC,QRNOPENACC
/COPY QSYSINC/QRPGLESRC,IFS

DCL-S descriptor_t INT(10) TEMPLATE;
DCL-DS state_t QUALIFIED template;  1
   descriptor LIKE(descriptor_t);
END-DS;
```

メイン・ハンドラー・プロシージャー

1. プロシージャー・インターフェースは、オープン・アクセス・ハンドラーに毎回渡される パラメーターを定義しています。*QrnOpenAccess_T* テンプレートは、ソース・ファイル QOAR/QRPGLESRC 内にあるコピー・ファイル QRNOPENACC 内に定義されています。

2. いくつかの基底データ構造が定義されています。 これらのデータ構造の基底ポインターは、ハンドラー・パラメーター中のポインターから設定されます。
   - データ構造 *state* は、どのハンドラー呼び出しでも ハンドラーが必ず必要とする情報を保持します。 基底ポインター *pState* は、ハンドラー・パラメーター 中の *stateInfo* サブフィールドから設定されます。
   - データ構造 *ifsInfo* は、通信域パラメーターです。 基底ポインター *pIfsInfo* を ハンドラー・パラメーター中の *userArea* サブフィールド から設定することによって、*ifsInfo* データ構造 は、RPG プログラムで HANDLER キーワードの 2 番目のパラメーターとして指定 された *ifs_info* データ構造と同じ記憶域を参照するようになります。
   - データ構造 *namesValues* は、 ファイル内の外部記述フィールドを記述します。 基底ポインターは、ハンドラー・パラメーターの *namesValues* サブフィールド から設定されます。
3. *state* および *ifsInfo* の基底ポインターが設定されます。
4. OPEN 命令の場合、ハンドラーは以下を行います。
   - *state* データ構造用の記憶域を割り振り、 ポインターをハンドラー・パラメーターの *stateInfo* サブフィールド に割り当てます。 後続の命令のためにハンドラーが呼び出されるとき には、*stateInfo* サブフィールドに同じポインター値が 保持されているため、ハンドラーは状態情報にアクセスすることができます。
   - ファイルをオープンし、戻されたファイル記述子を状態情報データ構造に保存します。
   - ファイルが RPG プログラム内に外部記述されている場合、 ハンドラー・パラメーター内の *useNamesValues* 指標サブフィールドをオンに設定します。
     - そのサブフィールドがオンの場合、出力命令用のデータは 各フィールドについての情報を格納する配列に入って提供されます。
     - そのサブフィールドがオフの場合、出力命令用のデータは、 *OUTPUT 外部記述データ構造と同じレイアウトのデータ構造として提供されます。
5. WRITE 命令の場合、ハンドラーは、 ハンドラー・パラメーターの *useNamesValues* サブフィールド に基づいて、プロシージャーのうちの 1 つを呼び出してファイルに書き込みます。
6. CLOSE 命令の場合、ハンドラーはファイルをクローズします。
7. 他の命令の場合、ハンドラーは例外を通知します。 この例のハンドラーは、OPEN 命令、WRITE 命令、および CLOSE 命令のみをサポートします。

```
DCL-PROC writeHdlr;
   DCL-PI *N EXTPGM;  1
      parm LIKEDS(QrnOpenAccess_T);
   END-PI;

   DCL-S stackOffsetToRpg INT(10) INZ(2);
   DCL-S errnoVal INT(10);

   DCL-DS state LIKEDS(state_t) BASED(pState); 2
   DCL-DS ifsInfo LIKEDS(ifshdlrs_info_t) BASED(pIfsInfo);
   DCL-DS namesValues LIKEDS(QrnNamesValues_T)
                      BASED(parm.namesValues);

   pState = parm.stateInfo; 3
   pIfsInfo = parm.userArea;

   SELECT;
   WHEN parm.RpgOperation = QrnOperation_OPEN; 4
      pState = %ALLOC(%SIZE(state_t));
      parm.stateInfo = pState;

      state.descriptor = openFile (ifsInfo
                               : stackOffsetToRpg + 1);

      IF parm.externallyDescribed;
         parm.useNamesValues = '1';
      ENDIF;
   WHEN parm.RpgOperation = QrnOperation_WRITE; 5
      IF parm.useNamesValues;
         writeFileNv (state.handle
                     : namesValues
                     : stackOffsetToRpg + 1);
      ELSE:
         writeFileBuf (state.handle
                      : parm.outputBuffer
                      : parm.outputBufferLen
                      : stackOffsetToRpg + 1);
      ENDIF;
   WHEN parm.RpgOperation = QrnOperation_CLOSE; 6
      closeFile (state.handle
                : stackOffsetToRpg + 1);
      state.descriptor = -1;
      DEALLOC(N) pState;
   OTHER; 7
      sendException ('Unexpected operation '
                     + %CHAR(parm.RpgOperation)
                   : stackOffsetToRpg + 1);
      // Control will not return here
   ENDSL;

END-PROC writeHdlr;
```

ファイルをオープンするプロシージャー

1. ファイルをオープンできなかった場合、プロシージャーは 例外メッセージを送信します。 これによってハンドラーは失敗し、 それが原因で RPG プログラム内の OPEN 命令は失敗します。

```
DCL-PROC openFile;
   DCL-PI *n LIKE(descriptor_t);
      ifsInfo LIKEDS(ifshdlrs_info_t) CONST;
      stackOffsetToRpg INT(10) VALUE;
   END-PI;
   DCL-C JOB_CCSID 0;
   DCL-S openFlags INT(10);
   DCL-S descriptor LIKE(descriptor_t);

   openFlags = O_WRONLY
             + O_CREAT + O_TEXT_CREAT + O_TEXTDATA
             + O_CCSID + O_INHERITMODE;
   IF ifsInfo.append;
      openFlags += O_APPEND;
   ELSE:
      openFlags += O_TRUNC;
   ENDIF;

   descriptor = open(ifsInfo.path
                   : openFlags
                   : 0
                   : ifsInfo.createCcsid
                   : JOB_CCSID);
   IF descriptor < 0; 1
      errnoException ('Could not open ' + ifsInfo.path + '.'
                    : getErrno ()
                    : stackOffsetToRpg + 1);
      // Control will not return here
   ENDIF;

   return descriptor;
END-PROC openFile;
```

ファイルをクローズするプロシージャー

```
DCL-PROC closeFile;
   DCL-PI *n;
      descriptor LIKE(descriptor_t) VALUE;
      stackOffsetToRpg INT(10) VALUE;
   END-PI;
   DCL-S rc INT(10);

   rc = close (descriptor);
   IF rc < 0;
      errnoException ('Error closing file.'
                    : getErrno ()
                    : stackOffsetToRpg + 1);
      // Control will not return here
   ENDIF;
END-PROC closeFile;
```

出力バッファーを使用してファイルに書き込むプロシージャー

```
DCL-PROC writeFileBuf;
   DCL-PI *n;
      descriptor LIKE(descriptor_t) VALUE;
      pBuf pointer VALUE;
      bufLen INT(10) VALUE;
      stackOffsetToRpg INT(10) VALUE;
   END-PI;

   writeLine (descriptor : pBuf : bufLen
             : stackOffsetToRpg + 1);
END-PROC writeFileBuf;
```

名前-値情報を使用してファイルに書き込むプロシージャー

1. 名前-値情報には、外部記述形式の各フィールドについての情報の配列が含まれています。 この例のハンドラーでは、 外部記述形式のフィールドの数およびタイプに関して独自の制限があります。
   a. ハンドラーは、フィールドが 1 つのみであることを検証します。
   b. 次に、ハンドラーは、それが英数字フィールドであることを検証します。
2. *nv.field(1).value* は、最初のフィールド内のデータを指すポインターです。 フィールドが可変長フィールドである場合、このポインターはフィールドのデータ部分を指します。*nv.field(1).valueLenBytes* は、データの長さを保持します。

```
DCL-PROC writeFileNv;
   DCL-PI *n;
      descriptor LIKE(descriptor_t) VALUE;
      nv LIKEDS(QrnNamesValues_T);
      stackOffsetToRpg INT(10) VALUE;
   END-PI;

   IF nv.num > 1; 1a
      sendException ('Only one field supported.'
                     : stackOffsetToRpg + 1);
      // Control will not return here
   ELSE:
      IF nv.field(1).dataType <> QrnDatatype_Alpha 2b
      AND nv.field(1).dataType <> QrnDatatype_AlphaVarying;
         sendException ('Field ' + nv.field(1).externalName
                        + 'must be Alpha or AlphaVarying type.'
                       : stackOffsetToRpg + 1);
         // Control will not return here
      ENDIF;
   ENDIF;

   writeLine (descriptor : nv.field(1).value : nv.field(1).valueLenBytes
              : stackOffsetToRpg + 1);
END-PROC writeFileNv;
```

ファイルに 1 行を書き込むプロシージャー

```
DCL-PROC writeLine;
   DCL-PI *n;
      descriptor LIKE(descriptor_t) VALUE;
      pBuf pointer VALUE;
      bufLen INT(10) VALUE;
      stackOffsetToRpg INT(10) VALUE;
   END-PI;
   DCL-S lineFeed CHAR(1) INZ(STREAM_LINE_FEED);
   DCL-S bytesWritten INT(10);

   bytesWritten = write (descriptor : pbuf : bufLen);
   IF bytesWritten < 0;
      errnoException ('Could not write data.'
                      : getErrno ()
                      : stackOffsetToRpg + 1);
      // Control will not return here
   ELSE:
      bytesWritten = write (descriptor : %ADDR(lineFeed) : 1);
      IF bytesWritten < 0;
         errnoException ('Could not write line-feed.'
                         : getErrno ()
                         : stackOffsetToRpg + 1);
         // Control will not return here
      ENDIF;
   ENDIF;
END-PROC writeLine;
```

例外を送信するプロシージャー

```
DCL-PROC sendException;
   DCL-PI *n;
      msg VARCHAR(2000) CONST;
      stackOffsetToRpg INT(10) VALUE;
   END-PI;
   DCL-DS msgFile qualified;
      *n CHAR(10) INZ('QCPFMSG');
      *n CHAR(10) INZ('*LIBL');
   END-DS;
   DCL-DS errorCode;
      bytesProvided INT(10) INZ(0);
      bytesAvailable INT(10);
      msgId CHAR(7);
      *n CHAR(1);
   END-DS;
   DCL-S key CHAR(4);
   DCL-PR QMHSNDPM EXTPGM;
      msgId CHAR(7) CONST;
      msgFile LIKEDS(msgFile) CONST;
      msgData CHAR(1000) CONST;
      dataLen INT(10) CONST;
      msgType CHAR(10) CONST;
      callStackEntry CHAR(10) CONST;
      callStackOffset INT(10) CONST;
      msgKey CHAR(4) CONST;
      errorCode LIKEDS(errorCode);
   END-PR;

   QMHSNDPM ('CPF9898' : msgFile : msg : %LEN(msg)
           : '*ESCAPE' : '*' : stackOffsetToRpg
           : key : errorCode);
END-PROC sendException;
```

"errno" に関連する例外を送信するプロシージャー

```
DCL-PROC errnoException;
   DCL-PI *n;
      msg VARCHAR(2000) CONST;
      errnoVal INT(10) VALUE;
      stackOffsetToRpg INT(10) VALUE;
   END-PI;
   DCL-S errnoMsg VARCHAR(200);
   DCL-S pErrnoMsg pointer;
   DCL-PR strerror pointer extproc(*dclcase);
      errnoVal INT(10) VALUE;
   END-PR;

   pErrnoMsg = strError (errnoVal);
   IF pErrnoMsg <> *null;
      errnoMsg = ' ' + %STR(pErrnoMsg);
   ENDIF;
   errnoMsg += ' (errno = ' + %CHAR(errnoVal) + ')';

   sendException (msg + errnoMsg
                : stackOffsetToRpg + 1);
END-PROC errnoException;
```

"errno" の値を取得するプロシージャー

```
DCL-PROC getErrno;
   DCL-PI *n INT(10) END-PI;
   DCL-PR getErrnoPtr pointer extproc('__errno') END-PR;
   DCL-S pErrno pointer static INZ(*null);
   DCL-S errno INT(10) BASED(pErrno);

   IF pErrno = *null;
      pErrno = getErrnoPtr();
   ENDIF;

   return errno;
END-PROC getErrno;
```

オープン・アクセス・ハンドラーの作成については、『Rational Open Access: RPG 版』のトピックを参照してください。

ファイル・パラメーター

RPG プログラムおよびプロシージャーへのプロトタイプ呼び出しを使用して、ファイルをパラメーターとして渡すことができます。 LIKEFILE キーワードを使用して、プロトタイプおよびプロシージャー・インターフェース定義用のファイル・パラメーターを定義することができます。 呼び出し先のプログラムまたはプロシージャーは、ファイル・パラメーターの定義に使用された元のファイルで有効だった、あらゆる操作を実行することができます。

注 : RPG ファイル・パラメーターは、C や C++ などの言語で記述されたファイル・パラメーターに使用される形式とは関連のない形式になっています。RPG で使用されるファイル・パラメーターを、他の言語で使用されるファイル・パラメーターと置き換えることはできません。つまり、RPG ファイル・パラメーターを受け取ることになっている RPG プロシージャーには、C ファイルを渡すことができません。同様に、RPG ファイルを C プログラムに渡すこともできません。

ファイル・パラメーターを渡すプログラムの例については、185 ページの『ファイルを渡す方法および関連付けられている変数を含むデータ構造を渡す方法の例』を参照してください。

ファイルに関連付けられる変数

ファイル仕様書のキーワードを使用して、複数の変数をファイルに関連付けることができます。 例えば、INFDS キーワードを使うと、ファイル情報データ構造がファイルに関連付けられます。 このデータ構造は、ファイル操作時に RPG によって、ファイルの現在の状態に関する情報を使用して更新されます。 SFILE キーワードを使用すると、記述するレコードの相対レコード番号に設定する数値変数が定義されます。

ファイルをパラメーターとして渡すと、呼び出し先プロシージャーのファイル・パラメーターが、呼び出し元プロシージャーで関連付けられているのと同じ物理変数に、継続的に関連付けられるようになります。呼び出し先プロシージャーは、ファイル・パラメーターで関連付けられている変数にアクセスすることができます。 ただし、実際にアクセスできるのは RPG コンパイラーのみになります。 これによって、呼び出し先プロシージャーがファイル・パラメーターに関する操作を実行した時に、関連付けられている変数を RPG コンパイラーで処理できるようになります。 ファイル・パラメーターに対するファイル操作で、関連付けられている変数の値が必要になる場合には、関連付けられている変数の現行値が使用されます。 ファイル・パラメーターに対するファイル操作によって、関連付けられている変数の内容が変更されると、新しい値によって、関連付けられている変数がただちに更新されます。 ファイル・パラメーターを渡しても、関連付けられている変数に対して、呼び出し先プロシージャーが直接アクセスできるようにはなりません。関連付けられている変数に呼び出し先プロシージャーがアクセスできるのは、その変数がグローバル変数である場合や、追加パラメーターとして呼び出し先プロシージャーに変数が渡されている場合のみです。

ヒント: ファイル・パラメーターを別のプロシージャーに渡して、関連付けられている変数にそのプロシージャーがアクセスできるようにする必要がある場合には、関連付けられている変数ごとにサブフィールドを指定してデータ構造を定義し、そのデータ構造を追加パラメーターとしてプロシージャーに渡します。 186 ページの図 46 を参照してください。 以下の表は、変数をファイルに関連付けるために使用できるキーワードの一覧です。

表 67. ファイル仕様書のキーワード (変数の関連付け用)

キーワード	使用法	記述
COMMIT	入力	コミットメント制御のためにファイルをオープンするかどうかを指示する際に、RPG プログラマーが設定します。
DEVID	入力/フィードバック	特定のデバイスに対するファイル操作を指示する際に、RPG プログラマーが設定します。 直前のファイル操作で使用されたデバイスを示す際に、RPG コンパイラーが設定します。
EXTFILE	入力	オープンする外部ファイルを指示する際に、RPG プログラマーが設定します。
EXTIND	入力	ファイルを使用するかどうかを制御する際に、プログラムが呼び出される前にアプリケーション開発者が設定します。
EXTMBR	入力	オープンする外部メンバーを指示する際に、RPG プログラマーが設定します。
INDDS	入出力 (Input/Output)	ファイル操作で使用される出力可能標識の設定を、RPG プログラマーが行います。 入力可能標識の設定を、システムが操作時に行います。
INFDS	入力	ファイルの現在の状態を表す際に、RPG コンパイラーが設定します。
PRTCTL	入力/フィードバック	印刷装置ファイルを制御する際に、RPG プログラマーがスペースおよびスキップ・フィールドを設定します。
RECNO	入力/フィードバック	印刷装置ファイルの現在行を示す際に、RPG コンパイラーが設定します。
SAVEDS	すべて	サブファイル・レコードに書き込まれる相対レコード番号を示す際に、RPG プログラマーが設定します。
SFILE	入力/フィードバック	入力操作によってサブファイル・コードに取り出された相対レコード番号を示す際に、RPG コンパイラーが設定します。
SLN	入力	表示装置ファイルのレコード形式の開始行を示す際に、RPG プログラマーが設定します。

ファイルを渡す方法および関連付けられている変数を含むデータ構造を渡す方法の例

関連付けられた変数を保持するためのデータ構造をファイルに定義する方法、ファイルおよびデータ構造をパラメーターとしてプロシージャーに渡す方法、ならびにプロシージャー内のパラメーターの使用方法を、以下の例で説明します。

```
 * The /COPY file has template definitions for the File and Associated Variables

 /if defined(FILE_DEFINITIONS)
 // Template for the "INFILE" file type
Finfile_t  if   e             disk    template block(*yes)
F                                     extdesc('MYLIB/MYFILE')
 /eof
 /endif
 /if defined(DATA_DEFINITIONS)
 // Template for the associated variables for an INFILE file
D infileVars_t    ds                  qualified template
D    filename                  21a
D    mbrname                   10a
 // Prototype for a procedure to open an INFILE file
D open_infile     pr
D    theFile                          likefile(infile_t)
D    kwVars                           likeds(infileVars)
D                                     options(*nullind)
 /eof
 /endif
```

図 *46. /COPY* ファイル *INFILE_DEFS*

```
P myproc          b // Copy in the template and prototype definitions
 /define FILE_DEFINITIONS
 /COPY INFILE_DEFS
 /undefine FILE_DEFINITIONS

 /define DATA_DEFINITIONS
 /COPY INFILE_DEFS
 /undefine DATA_DEFINITIONS
 // Define the file using LIKEFILE, to enable it to be passed as
 // a parameter to the "open_infile" procedure.

 // Define all the associated variables as subfields of a data
 // structure, so that all the associated variables can be
 // passed to the procedure as a single parameter
Ffile1                                likefile(infile_t)
F                                     extfile(file1Vars.filename)
F                                     extmbr(file1Vars.mbrname)
F                                     usropn

D file1Vars       ds                  likeds(infileVars_t)

  /free
        open_infile (file1 : file1Vars);

        . . .
```

図 *47.* ファイル・パラメーターを渡す呼び出し元プロシージャー

```
 // Copy in the template and prototype definitions
 /define FILE_DEFINITIONS
 /COPY INFILE_DEFS
 /undefine FILE_DEFINITIONS

 /define DATA_DEFINITIONS
 /COPY INFILE_DEFS
 /undefine DATA_DEFINITIONS
P open_infile     b
 // The open_infile procedure has two parameters
 //  - a file
 //  - a data structure containing all the associated variables for the file
D open_infile     pi
D    theFile                         likefile(infile_t)
D    kwVars                          likeds(infileVars)
  /free
        // The %OPEN(filename) built-in function reflects the
        // current state of the file
        if not %open(theFile);
           // The called procedure modifies the calling procedure's "file1Vars"
           // variables directly, through the passed parameter
           kwVars.extfile = 'LIB1/FILE1';
           kwVars.extmbr = 'MBR1';
           // The OPEN operation uses the file1Vars subfields in the
           // calling procedure to open the file, opening file LIB1/FILE1(MBR1)
           open theFile;
        endif;
        . . .
```

図 *48.* ファイル・パラメーターを使用する呼び出し先プロシージャー

全手順ファイル

全手順ファイルは、自由形式 DCL-F ステートメントを使用して定義されるか、 または、ファイル記述仕様書の 18 桁目の F によって識別されます。

ファイルに対するすべての入力命令、更新命令、および出力命令は、 演算命令によって制御されます。

1 次/2 次の複数ファイル処理

RPG IV プログラムでは、1 つの 1 次入力ファイルと 1 つ以上の 2 次入力ファイルを処理することを、突き合わせフィールドを使用するかどうかに関係なく、複数ファイル処理と呼びます。 突き合わせフィールドの 内容に基づいて 2 つ以上のファイルからレコードを選択することは、突き合わせレコ ードによる複数ファイル処理として知られています。 複数ファイル処理 は、1 次/2 次ファイルとして指定された 外部記述またはプログラム記述入力ファイルで使用することができます。

突き合わせフィールドを用いない複数ファイル処理

複数ファイル処理で突き合わせフィールドを使用しない場合には、レコードは一度 に 1 つのファイルから選択されます。 1 つのファイルからのレコードがすべて 処理されると、次のファイルのレコード が選択されます。 ファイルは次の順序で選択されます。

1. 指定されていれば、1 次ファイル
2. ファイル仕様書に記述されている順序で 2 次ファイル

突き合わせフィールドを用いた複数ファイル処理

複数ファイル処理で突き合わせフィールドを使用した場合には、突き合わせフィールド の内容に応じて、処理するレコードがプログラムによって選択されます。 最初のサイクルの始めでは、プログラムがすべての 1 次/2 次 入力ファイルから 1 つずつレコードを読み取って、レコード内の突き合わせフィール ドを比較します。 レコードが昇順であれば、突き合わせフィールドが最低のレコード がプログラムによって選択されます。 レコードが降順であれば、プログラムは最高の突き合わせフィール ドのレコードを選択します。

レコードがファイルから選択されると、プログラムはそのファイルから次のレ コードを読み取ります。 次のプログラム・サイクルの始めに、新しいレコードが 選択されるのを待機している読み取り域内の他のレコードと比較され、1 つのレコ ードが処理のために選択されます。

突き合わせフィールドのないレコードもファイルに含めることができます。 このよ うなレコードは、突き合わせフィールドのあるレコードより前に、処理のために選択 されます。 比較されているレコードの中に突き合わせフィールドのないレコードが 2 つ以上あっ た場合には、それらのレコードが入っていたファイルの優先順位によって選択さ れるレコードが決まります。 ファイルの優先順位は次のとおりです。

1. 指定されていれば、1 次ファイル
2. ファイル仕様書に記述されている順序で 2 次ファイル

1 次ファイルのレコードが 1 つまたは複数の 2 次レコー ドと一致した場合には、MR (突き合わせレコード) 標識がオンに設定されます。 MR 標識は、一致したレコードの明細時処理から、そのレコードに続く合計時まで 、オンのままです。 この標識を使用して、選択されたレコードに対する演算または 出力命令を条件付 けすることができます。 一致したレコードを 1 つ選択する必要があった場合には、それらの レコードが入 っていたファイルの優先順位によって選択されるレコードが決まります。

110 ページの図 13 は、複数ファイル処理の論理フローを示しています。

1 つの入力ファイルだけに突き合わせフィールドがあるように定義し、他の入力フ ァイルには突き合わせフィールドがないようなプログラムを作成することができます。 突き合わせフィールドのないファイルは、前述のファイルの優先順位に完全に従って 処理されます。 突き合わせフィールドのあるファイルは最後に処理され、そのファイ ルについては順序検査が行われます。

突き合わせフィールド値 *(M1* から *M9)* の割り当て

入力仕様の 65 から 66 桁目で突き合わせフィールド値 (M1 から M9) をフィールドに割 り当てる場合には、以下の点を考慮してください。

- 突き合わせフィールド の指定があるすべてのレコード・タイプについて順序検査 が実行されます。 すべての突き合わせフィールドは、すべて昇順か降順の同じ順序で なければなりません。 M1 から M9 の割り当てられたフィールドの内容が正しい順序であるかどうかが検査さ れます。 順序にエラーがあった場合には、RPG IV 例外/エラー処理ルーチンが制御を受け取ることになります。 プログラムの処理が続行される場合には、同じファイルから 次のレコードが読み 取られます。
- プログラムで使用されるすべてのファイルに突き合わせフィールドが必要なわけ ではありません。 また、1 つのファイルの中ですべてのレコード・タイプ に突き合わせ フィールドが必要なわけでもありません。 しかし、ファイルの突き合わせが行わ れることになっている場合には、2 つのファイルからの少なくとも 1 つのレコー ド・タイプに突き合わせフィールドがなければなりません。
- 突き合わせに使用されるすべてのレコード・タイプに同じ突き合わせフィールド値を 指定しなければなりません。190 ページの図 49 を参照してください。
- 同じ突き合わせフィールド値 (M1 から M9) を持つ日付、時刻、およびタイム・スタンプ 突き合わせフィールドは、同じタイプ (例えば、すべて日付) でなければなりませ んが、形式は異なっていても差し支えありません。
- 同じ突き合わせフィールド値 (M1 から M9) を持つすべての文字、図形、または数値 突き合わせフィールドは、同じ長さおよびタイプでなければなりません。 突き合わせフィールド にパック形式のデータが入っている場合には、突き合わせフィール ドの長さとしてゾーン 10 進数の長さ (2 X パックされた長さ - 1) が使用 されます。 桁数が同じであれば、1 つのレコードのパック形式フィールドを別のレコードの ゾーン 10 進数フィールドと突き合わせることも有効です。 パック形式フィールドの長さは常に 奇数であるため、この長さは常に奇数でなけ ればなりません。
- 異なる突き合わせフィールドのレコード位置はオーバーラップしていてもかまい ませんが、すべてのフィールドの合計長が 256 文字を超えていてはなりません。
- 1 つのレコード・タイプに複数の突き合わせフィールドが指定された場合には、すべてのフィールドが結合されて、1 つの連続したフィールドとして取り扱われます (190 ページの図 49 を参照)。 フィールドは、突き合わせフィールド値の降順 (M9 から M1) に従って結合されます。
- 突き合わせフィールド値を 1 つのレコードの中で繰り返すことはできません。
- 突き合わせフィールドが 1 つでも数値として記述されていた場合には、 同じ突き合わ せフィールド値 (M1 から M9) が与えられたすべての突き合わせフィールドは数値と見なされます。

- 小数点以下の桁数がある数値フィールドが突き合わされる場合には、その小 数点以下の桁数はないものとして取り扱われます。 例えば、3.46 は 346 と等 しいと見なされます。
- 数値突き合わせフィールドでは、数字部分だけが比較されます。 負のフィールド であっても、数値フィールドの符号は無視されるので、正と見なされます。 した がって、-5 は +5 と一致します。
- 日付および時刻フィールドは、比較の前に *ISO 形式に変換されます。
- 図形データは、16 進数値によって比較されます。
- 複数の突き合わせフィールドが使用された場合には、MR 標識がオンに設定され る前に、常にすべての突き合わせフィールドが一致していることが必要です。 例えば、突き合わせフィールド値 M1、M2、および M3 が指定された場合には、 1 次レコードからのこれら 3 つのすべてのフィールドが、2 次 レコードからの 3 つの突き合わせフィールドのすべてと一致していなければなりません。M1 および M2 フィールドによって指定されたフィールドのみが一致しても、MR 標識はオンに設定されません (190 ページの図 49 を参照)。
- UCS-2 フィールドは、突き合わせフィールドとして使用できません。
- 突き合わせフィールドを先読みフィールドおよび配列に使用することはできません。
- レコードの突き合わせ操作では、フィールド名は無視されます。 したがって、同 じ突き合わせレベルが割り当てられている異なるレコード・タイプからのフィールド は、同じ名前であっても差し支えありません。
- プログラムに代替照合順序またはファイル変換が定義された場合には、文字 フィールドの突き合わせは指定された代替順序に従って行われます。
- ヌル値可能フィールド、ALTSEQ(*NONE) によって定義された文字フィールド、 および 2 進数、浮動、整数、ならびに符号なし (入力仕様 の 36 桁目 B、F、I、または U) には突き合わせフィールド値を 割り当てていることはできません。
- フィールドとレコードの関連標識がない突き合わせフィールドは、その標識があ る突き合わせフィールドより前に記述する必要があります。 フィールドとレコードの 関連標識を突き合わせフィールドと併用する場合には、フィールドとレコードの関連 標識がこのファイルのレコード識別標識と同じでなければならず、突き合わせフィー ルドはフィールドとレコードの関連標識に従ってグループ化されていなければな りません。
- フィールド・レコードの関連標識がないフィールドについて 突き合わせ値 (M1 から M9) を指定する場合には、使用するすべての突き合わせ値を、一度は、フィールドと レコードの関連標識を使用せずに指定しなければなりません。 すべての突き合わせフィールドが すべてのレコードに共通していない場合には、ダミ ー突き合わせフィールドを使用することが必要です。外部記述ファイルの場合には、フィールドとレコードの関連標識は無効です (191 ページの図 50 を参照してください)。
- 突き合わせフィールドは、制御レベル標識 (L1 から L9) からは独立しています。
- 複数ファイル処理が指定されていても、LR 標識がオンに設定された場合に は、プログラムは複数ファイル処理ルーチンを回避します。

190 ページの図 49 は、突き合わせフィールドの指定方法の例です。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++++
 * The files in this example are externally described (E in position
 * 22) and are to be processed by keys (K in position 34).
FMASTER    IP   E           K DISK
FWEEKLY    IS   E           K DISK
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IRcdname+++....Ri..................................................
I..............Ext-field+..................Field+++++++++L1M1..PlMnZr....
 *                          MASTER FILE
IEMPMAS        01
I                                          EMPLNO          M1
I                                          DIVSON          M3
I                                          DEPT            M2
IDEPTMS        02
I                                          EMPLNO          M1
I                                          DEPT            M2
I                                          DIVSON          M3
 *                          WEEKLY FILE
IWEEKRC        03
I                                          EMPLNO          M1
I                                          DIVSON          M3
I                                          DEPT            M2
```

図 *49.* 値がすべて一致する突き合わせフィールド

レコードの突き合わせに 3 つのファイルが使用されます。 すべてのファイルに 3 つ の突き合わせフィールドが指定され、突き合わせるべきフィールドを指示するために 、すべて同じ値 (M1、M2、M3) を使用しています。 MR 標識がオンに設定される のは、EMPMAS か DEPTMS のいずれかのファイルの 3 つのすべてのフィールドが、 WEEKRC ファイルからの 3 つのすべてのフィールドと同じである場合だけです。

各ファイル中の 3 つの突き合わせフィールドは結合され、次のように降順で編成さ れた 1 つの突き合わせフィールドとして取り扱われます。

DIVSON
M3

DEPT
M2

EMPLNO
M1

突き合わせフィールドが入力仕様に指定された順序は、各突き合わせフィールドの編 成には影響しません。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC..............................
I.....................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
IDISK      AB  01    1 C1
I         OR   02    1 C2
I         OR   03    1 C3
I                                 1   10 0EMPNO           M1
I                                11   15 0DUMMY           M2
I                                11   15 0DEPT            M202
I                                16   20 0DEPT            M203
```

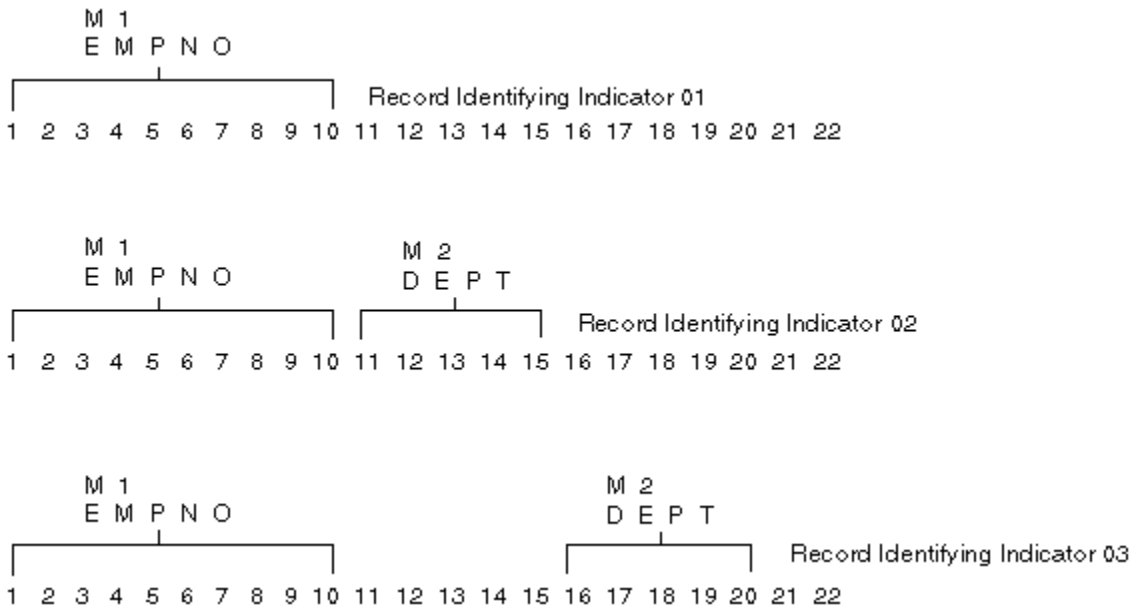


図 *50. ダミー M2* フィールドによる突き合わせフィールド

入力ファイル には、3 つの異なるレコード・タイプがあります。 この 3 つのす べての 1 から 10 桁目に突き合わせフィールドが入っています。 それらの 2 つには、2 番 目の突き合わせフィールドがあります。 M1 はすべてのレコード・タイプにあるので、 67 から 68 桁目にフィールドとレコードの関連を記入せずに指定することができます。 フィールドとレコードの関連を記入せずに 1 つの突き合わせ値 (M1 から M9) を指定する 場合には、すべての突き合わせ値を、一度は、フィールドとレコードの関連を記入せずに指定しなければなりません。 M1 の値はフィールドとレコードの関連なしに指定さ れているので、M2 の値も一度はフィールドとレコードの関連なしに指定されてい なければなりません。 M2 フィールドはすべてのレコード・タイプにあるわけでは ないので、ダミーの M2 フィールドを次に指定することが必要です。 ダミー・フ ィールドには任意の固有名を与えることができますが、指定された長さは、実際 の M2 フィールドの長さと等しくなければなりません。 次に、フィールドとレコ ードの関連の指定によって M2 フィールドが見付かったレコード・タイプに、そ の M2 フィールドが関連付けられます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++++
FPRIMARY   IPEA F   64        DISK
FFIRSTSEC  IS A F   64        DISK
FSECSEC    IS A F   64        DISK

*...1....+....2....+....3....+....4....+....5....+....6....+....7...
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC..............................
I.....................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
IPRIMARY   AA  01    1 CP    2NC
I                                2    3  MATCH           M1
 *
I          BB  02    1 CP    2 C
I                                2    3  NOM
 *
IFIRSTSEC  AB  03    1 CS    2NC
I                                2    3  MATCH           M1
 *
I          BC  04    1 CS    2 C
I                                2    3  NOM
 *
ISECSEC    AC  05    1 CT    2NC
I                                2    3  MATCH           M1
 *
I          BD  06    1 CT    2 C
I                                2    3  NOM
```

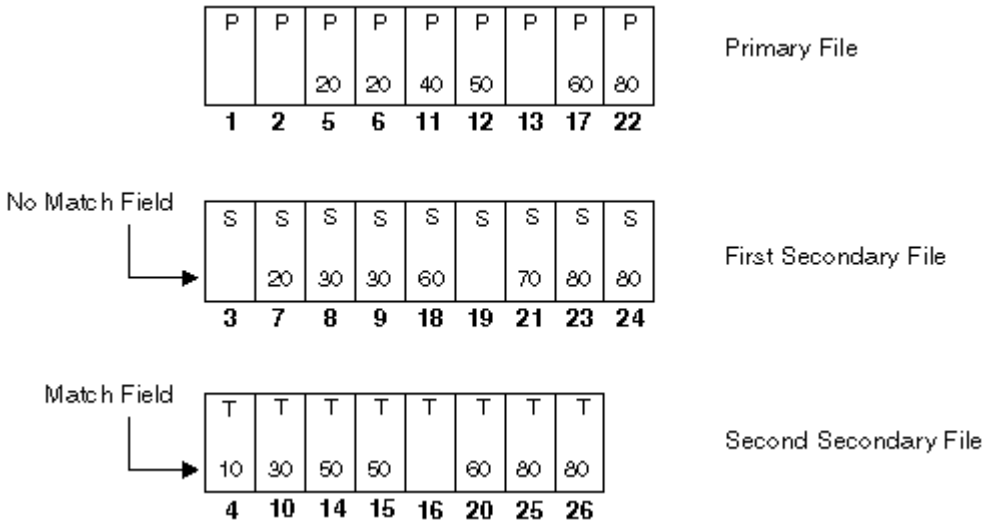
図 *51. 3* つのディスク・ファイルの突き合わせフィールドの指定

突き合わせレコードの処理

2 つ以上のファイルのレコードの突き合わせは、以下の方法で処理されます。

- 1 次ファイルのレコードと 2 次ファイルのレコードが 一致した場合には、常に 1 次ファイルが先に処理されます。 その後で 、一致した 2 次ファイルが処理されます。 選択されたばかりのレコー ド・タイプを識別するレコード識別標識は、レコードが処理される時点でオンに なっています。 この標識は、行われる処理のタイプを制御するためによく使用されます。
- 昇順ファイルのレコードが一致しない場合には、常に突き合わせフィールドの内 容が最低のレコードが先に処理されます。 降順ファイルのレコードが一致しない 場合には、常に突き合わせフィールドの内容が最高のレコードが先に処理されます。
- 突き合わせフィールドの指定のないレコードは、前のレコードの直後に続けて処 理されます。 MR 標識はオフになっています。 このレコード・タイプがファイル中の最初のもの である場合には、1 次ファイルの中のものでなくても先に処理されま す。
- レコードの突き合わせによって、1 次レコードは一致した 2 次レコードの前に処理されるので、1 次レコードからのデータをそ の一致した 2 次レコードに入れることができます。 しかし、2 次レコード から一致した 1 次レコードへのデー タの転送は、先読みフィールドが指定されている場合にだけ実行することができ ます。

193 ページの図 52 および 194 ページの図 53 は、3 つのファイルから処理対象レコードを選択する方法を示しています。



上記の 3 つのディスク・ファイルのレコードは太字で示した順序で選択されます。

図 *52. 3* つのディスク・ファイルからレコードを選択する通常の方法

表 *68. 3* つのディスク・ファイルからレコードを選択する通常の方法

サイクル	処理されるファイル	オンになる標識	標識が設定される理由
1	PRIMARY	02	突き合わせフィールドの指定がない
2	PRIMARY	02	突き合わせフィールドの指定がない
3	最初の 2 次	04	突き合わせフィールドの指定がない
4	2 番目の 2 次	05	2 番目の 2 次が低く、1 次と一致しない
5	PRIMARY	01、MR	1 次が最初の 2 次と一致する
6	PRIMARY	01、MR	1 次が最初の 2 次と一致する
7	最初の 2 次	03、MR	最初の 2 次が 1 次と一致する
8	最初の 2 次	03	最初の 2 次が低く、1 次と一致しない
9	最初の 2 次	03	最初の 2 次が低く、1 次と一致しない
10	2 番目の 2 次	05	2 番目の 2 次が低く、1 次と一致しない
11	PRIMARY	01	1 次が低く、2 次と一致しない
12	PRIMARY	01、MR	1 次が 2 番目の 2 次と一致する
13	PRIMARY	02	突き合わせフィールドの指定がない
14	2 番目の 2 次	05、MR	2 番目の 2 次が 1 次と一致する
15	2 番目の 2 次	05、MR	2 番目の 2 次が 1 次と一致する
16	2 番目の 2 次	06	突き合わせフィールドの指定がない
17	PRIMARY	01、MR	1 次が両方の 2 次ファイルと一致する
18	最初の 2 次	03、MR	最初の 2 次が 1 次と一致する
19	最初の 2 次	04	突き合わせフィールドの指定がない
20	2 番目の 2 次	05、MR	2 番目の 2 次が 1 次と一致する
21	最初の 2 次	03	最初の 2 次が低く、1 次と一致しない

表 68. 3 つのディスク・ファイルからレコードを選択する通常の方法 (続き)			
サイクル	**処理されるファイル**	**オンになる標識**	**標識が設定される理由**
22	PRIMARY	01、MR	1 次が両方の 2 次ファイルと一致する
23	最初の 2 次	03、MR	最初の 2 次が 1 次と一致する
24	最初の 2 次	02、MR	最初の 2 次が 1 次と一致する
25	2 番目の 2 次	05、MR	2 番目の 2 次が 1 次と一致する
26	2 番目の 2 次	05、MR	2 番目の 2 次が 1 次と一致する

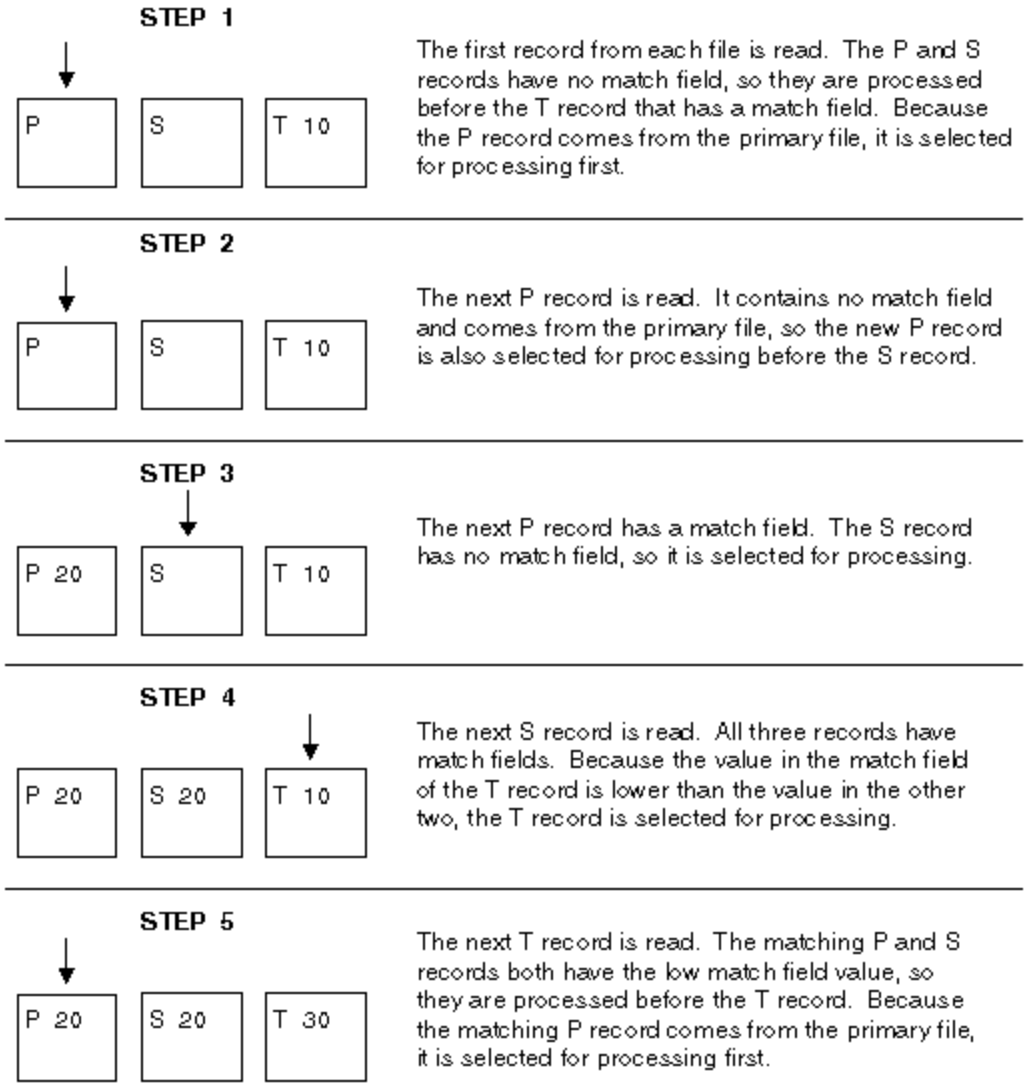


図 *53. 3* つのディスク・ファイルからレコードを選択する通常の方法 - パート *1*

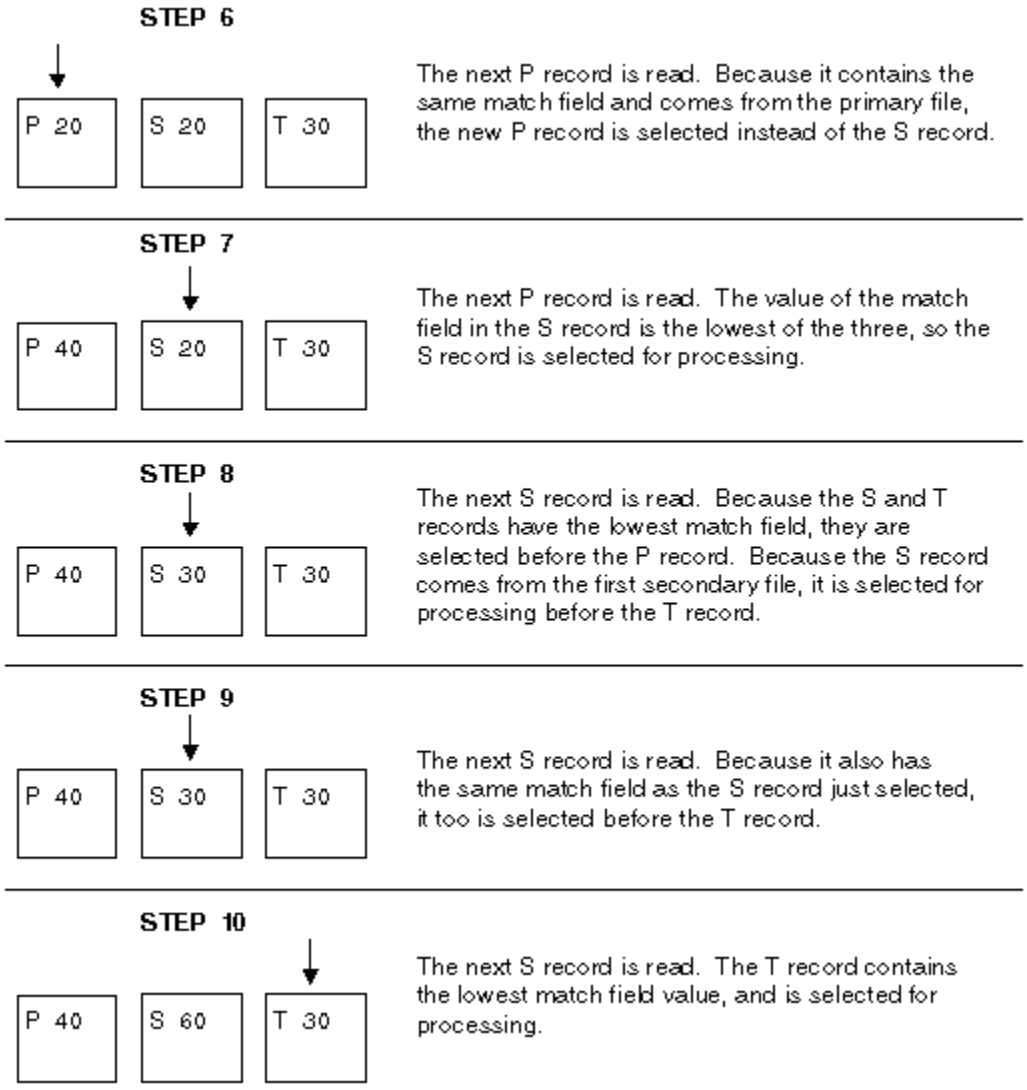


図 *54. 3* つのディスク・ファイルからレコードを選択する通常の方法 - パート *2*

ファイル変換

ファイル変換機能は、文字に使用されている 8 ビット・コードを、それぞれ別 の 8 ビット・コードに変換します。 ファイル変換の使用は、次の一方または両方 を指示することになります。

- 入力データで使用されている文字をシステム・コードに変換する必要がある。
- 出力データをシステム・コードから別のコードに変換する必要がある。 入力データの変換は、どのフィールド選択が行われるより前に実行されます。 出力データの変換は、すべての編集が行われた後に実行されます。

ファイル変換の指定にあたっては、以下の点に留意してください。

- ファイル変換は、配列またはテーブル・ファイルのデータについて指定する ことができます (ファイル仕様書の 18 桁目に T)。
- 入出力共用、入力、または更新ファイルのデータにファイル変 換を使用し、 用意されているファイル変換テーブルに従って、入力および出力時に変換するこ とができます。 ファイル変換を使用して更新ファイルのデータを変換する場合 には、次のレコー ドが読み取られる前に各レコードを書き出さなければなりません。
- キーによりアクセスされるファイルに対して演算項目 1 に検索引数を指定し た入出力命令 (CHAIN、READE、READPE、SETGT、または SETLL など) の場合には 、ファイルがアクセスされる前に検索引数が変換されます。

- レコード・アドレス・ファイルと処理されているファイル (処理されている ファイルが限界範囲内で順次処理される場合) の両方にファイル変換を指定した 場合には、レコード・アドレス・ファイル中のレコードがそのファイルについて 指定されたファイル変換に従って変換され、次に、処理されているファイル中の レコードがそのファイルについて指定されたファイル変換に従って変換されます。
- ファイル変換は、1 バイト単位でのみ適用されます。
- 入力レコードおよび出力レコードのすべてのバイトが変換されます。
- サブプロシージャーで定義されたローカル・ファイルについては、ファイル変換はサポートされていません。

ファイル変換の指定

ファイル変換を指定するためには、制御仕様書で FTRANS キーワードを使用して ください。 これらの変換をシステムへの入力用の正しいレコード様式に書き換えなければな りません。 ファイル変換テーブル・レコードと呼ばれるこれらのレコードは、代替照合順序 レコード、あるいはコンパイル時にロードされる配列およびテーブルより前にな ければなりません。 これらのレコードの前には、1 から 3 桁目に **b (b = ブランク) または 1 から 8 桁目に **FTRANS の入ったレコードがなければなりません。 このレコードの残りの桁は、注記に使用することができます。

1 つのファイルまたはすべてのファイルの変換

ファイル変換テーブル・レコードは、次のように形式設定されていなければな りません。

レコード位置指定	立入り
1 から 8 (すべてのファイルを変換する場合)	*FILESbb (b はブランクを表します) を記入して、すべての ファイルを変換することを指示します。 11 から 12 桁目から始まるファイル変換 テーブル・レコードを完成させます。 *FILESbb を指定した場合には、 他のファイル変換テーブルをプログラムに指定することはできません。
1 から 8 (特定のファイルを変換する場合)	変換するファイルの名前を記入します。 11 から 12 桁目から始まるファイル変換 テーブル・レコードを完成させます。 特定のファイルを変換する場合に は、1 から 8 桁目に *FILESbb の入力は 行いません。
9 から 10	ブランク
11 ～ 12	入力から、または出力へ変換する文字の 16 進数値を記入します。
13 ～ 14	RPG IV 言語が処理する内部文字に相当する 16 進数値を記入します。 この 文字は、入力では 11 から 12 桁目の文字に置き換わり、出力では 11 から 12 桁目の文 字によって置き換えられます。
15 から 18 19 から 22 23 から 26 ... 77 から 80	15 桁目から始まる 4 桁のグループは、すべて 11 から 14 桁目と同じように使用 されます。 各グループの最初の 2 桁には、置き換えられる文字の 16 進数値を記 入します。 後の 2 桁には、置き換わる文字の 16 進数値を記入します。

レコードは、最初のブランク項目によって終了します。 1 つのファイル変換テ ーブルに 1 つまたは複数のレコードを使用することができます。 テーブルを定義 するために複数のレコードが必要な場合には、すべてのレコード に同じファイル名を記入しなければなりません。 ファイル名を変えて、複数の変 換テーブルの区切りに使用することができます。 *FILES レコードにより、ファイ ル仕様書の 18 桁目の T によって指定されたテーブルおよび配列も含め、す べてのファイルが同じテーブルによって変換されます。

```
     HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      * In this example all the files are translated
     H FTRANS
     FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++
     FFILE1     IP   F   10        DISK
     FFILE2     IS   F   10        DISK
     FFILE3     IS   F   10        DISK
     FFILE4     IS   F   10        DISK
**FTRANS
*FILES    81C182C283C384C4
```

```
     HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      * In this example different translate tables are used and
      * FILE3 is not translated.
     H FTRANS
     FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++
     FFILE1     IP   F   10        DISK
     FFILE2     IS   F   10        DISK
     FFILE3     IS   F   10        DISK
     FFILE4     IS   F   10        DISK
**FTRANS
FILE1     8182
FILE2     C1C2
FILE4     81C182C283C384C4
```

複数ファイルの変換

複数のファイルについて同じファイル変換テーブルが必要であるが、すべてのフ ァイルについて必要というわけではない場合には、2 つのタイプのレコードを指 定することが必要です。 最初のタイプのレコードは、テーブルを使用するファイ ルを指定し、2 番目のタイプのレコードは、テーブルを指定します。 これらのレ コードの各タイプについて、複数のレコードを指定することができます。 ファイ ル名を変えて、複数の変換テーブルの区切りに使用することができます。

ファイルの指定

ファイル変換テーブル・レコードは、次のように形式設定されていなければな りません。

レコード位置指定	立入り
1 から 7	*EQUATE
8 から 10	これらの桁はブランクのままにしておきます。
11 から 80	変換するファイル (複数も可) の名前を記入します。 複数のファイルを変換す る場合には、ファイル名をコンマで区切らなければなりません。

追加のファイル名は、後にコンマが続いていないファイル名が見付かるまで、こ のテーブルと関連付けられます。 ファイル名を 2 つのレコードに分割することは できません。ファイル名の後のコ ンマは、ファイル名と同じレコードになければなりません。 *EQUATE を使用して 作成することができるファイル変換テーブルは 1 つだけです。

テーブルの指定

ファイル変換テーブル・レコードは、次のように形式設定されていなければな りません。

レコード位置指定	立入り
1 から 7	*EQUATE
8 から 10	これらの桁はブランクのままにしておきます。
11 ～ 12	入力から、または出力へ変換する文字の 16 進数値を記入します。
13 ～ 14	RPG IV 言語が処理する内部文字に相当する 16 進数値を記入します。 この 文字は、入力では 11 から 12 桁目の文字に置き換わり、出力では 11 から 12 桁目の文 字によって置き換えられます。
15 から 18 19 から 22 23 から 26 ... 77 から 80	15 桁目から始まる 4 桁のグループは、すべて 11 から 14 桁目と同じように使用 されます。 各グループの最初の 2 桁には、置き換えられる文字の 16 進数値を記 入します。 後の 2 桁には、置き換わる文字の 16 進数値を記入します。

レコードは、最初のブランク・レコードの位置で終了します。 記入項目の数が 80 桁を超えた場合には、1 から 10 桁目を次のレコードにコピーし、11 から 80 桁目に は前と同様に変換文字の対を続けてください。 1 つのファイルのテーブル・レコ ードは、すべて 1 つにまとめて保存してください。

ファイル変換テーブルを記述するレコードの前には、1 から 3 桁目 に **b (b = ブランク) または **FTRANS の入ったレコードが なければなりません。 このレコードの残りの桁は、注記に使用することができます。

```
     HKeywords+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      * In this example several files are translated with the
      * same translation table.  FILE2 is not translated.
     H FTRANS
     FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++
     FFILE1     IP   F   10        DISK
     FFILE2     IS   F   10        DISK
     FFILE3     IS   F   10        DISK
     FFILE4     IS   F   10        DISK
**FTRANS
*EQUATE   FILE1,FILE3,FILE4
*EQUATE   81C182C283C384C485C586C687C788C889C98ACA8BCB8CCC8DCD8ECE8F
*EQUATE   91D192D2
```

定義

変数、定数、プロトタイプ、およびプロシージャー・インターフェースの定義

この部は、ソースにコーディングすることができる各種の定義についての情報 を提供しています。 ここでは、以下について説明します。

- 次のものの定義方法
  - 独立フィールド、配列、およびテーブル
  - 名前付き定数
  - データ構造およびそのサブフィールド
  - プロトタイプ
  - プロトタイプ・パラメーター
  - プロシージャー・インターフェース
- 各定義タイプの定義方法と定義の有効範囲および記憶域
- データ・タイプおよびデータ形式
- 数値フィールドの編集

ファイルの定義方法については、349 ページの『ファイル仕様書』および「*IBM Rational Development Studio for i: ILE RPG* プログラマーの手引き」のファイルの定義に関する章も参照してください。

データおよびプロトタイプの定義

ILE RPG では、以下の項目を定義することができます。

- データ構造、データ構造サブフィールド、独立フィールド、および名前付き 定数などのデータ項目。 配列およびテーブルは、データ構造サブフィールドかまたは独立フィールドとし て定義することができます。
- プロトタイプ、プロシージャー・インターフェース、およびプロトタイプ・ パラメーター

この章には、以下のトピックに関する情報が示されています。

- 定義タイプ、有効範囲、 および記憶域を含む一般的な考慮事項
- 独立フィールド
- 固定情報
- データ構造
- プロトタイプ、パラメーター、およびプロシージャー・インターフェース

一般的な考慮事項

各項目は、定義仕様書を使用して定義します。 定義は、モジュールまたはプログラムの中と、サイクル・メイン・ソース・セクションおよびサブプロシージャーの中の、2 つの場所に表すことができます。 (**メイン・ソース・セクション**は、モジュール内の H、F、D、I、C、および O の各仕様の最初のセットによって構成され、独立型プログラムまたはサイクル・メイン・プロシージャーにある仕様と対応しています。) 定義がどこにあるかによって、定義できる内容とその定義の有効範囲の両方が決 まります。 定義のタイプを、次のように 24 から 25 桁目に指定してください。

記入
　定義タイプ

ブランク
　データ構造サブフィールドまたはパラメーター定義

C
　名前付き固定情報

DS
データ構造

PI
プロシージャー・インターフェース

PR
Prototype

S
独立フィールド

データ構造、プロトタイプ、およびプロシージャー・インターフェースの定義は、 24 から 25 桁目がブランクでない最初の定義仕様書、または定義仕様書でない最初の指定で終わります。

```
     *----------------------------------------------------------------*
     *  Global Definitions
     *----------------------------------------------------------------*
     D String          S               6A   INZ('ABCDEF')
     D Spcptr          S                *
     D SpcSiz          C                     8
     D DS1             DS                    OCCURS(3)
     D  Fld1                           5A   INZ('ABCDE')
     D  Fld1a                          1A   DIM(5) OVERLAY(Fld1)
     D  Fld2                           5B 2 INZ(123.45)
     D Switch          PR
     D   Parm                          1A
     ...
     *----------------------------------------------------------------*
     *  Local Definitions
     *----------------------------------------------------------------*
     P Switch          B
     D Switch          PI
     D   Parm                          1A
     * Define a local variable.
     D Local           S               5A      INZ('aaaaa')
     ...
     P                 E
```

図 *55.* サンプル定義

定義の有効範囲

定義がどこにあるかによって、有効範囲が異なります。 **有効範囲**は、名前が認識されるソース行の範囲のことを表します。有効範囲には、201 ページの図 56 に示されているように、グローバルとローカルの 2 つのタイプがあります。

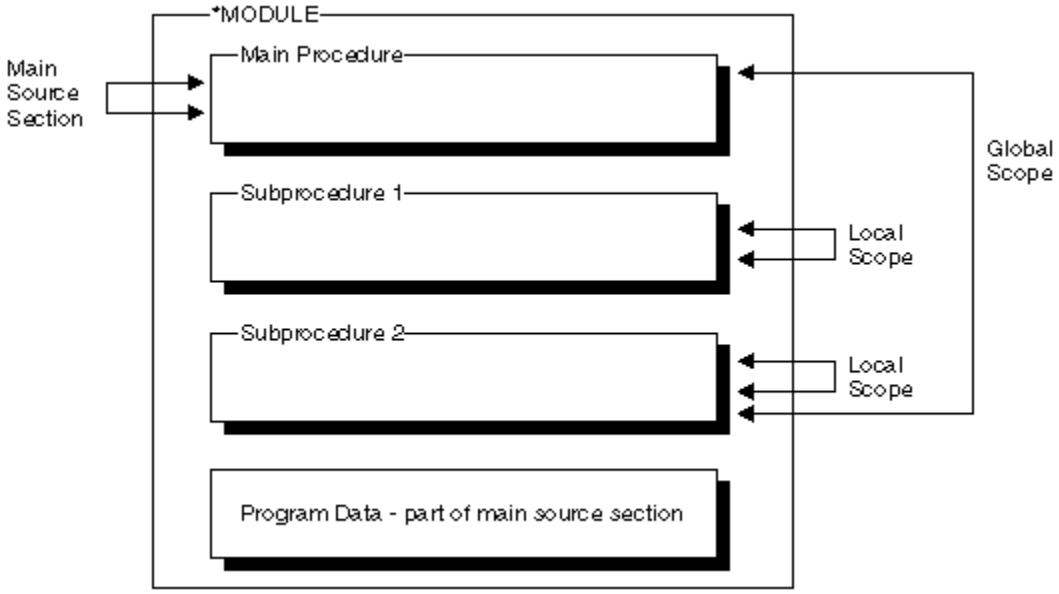


図 56. 定義の有効範囲

一般に、メイン・ソース・セクションに定義されたすべての項目はグローバルであり、したがって、モジュール全体で認識されます。 **グローバル定義**とは、モジュール内のサイクル・メイン・プロシージャーと任意のサブプロシージャーの両方で使用することができる定義のことです。 それらはエクスポートすることもできます。

他方、サブプロシージャー内の項目はローカルです。 **ローカル定義**は、そのサブプロシージャーの内部でのみ認識される定義です。 項目がグローバル項目と同じ名前で定義された場合には、サブプロシージャー内部でのその名前に対する参照ではローカル定義が使用されます。

ただし、以下の例外に注意してください。

- サブルーチン名およびタグ名は、それらが定義されているプロシージャーに対してのみ認識されます。サイクル・メイン・プロシージャーで定義されているサブルーチンまたはタグの名前が、ここに含まれます。
- 入力および出力仕様に指定されたすべてのフィールドは、グローバル・フィールドになります。 たとえば、サブプロシージャーで WRITE 命令のようなレコード様式を使用する命令が実行される場合には、そのレコード様式フィールドと同じ名前を持つローカル定義があっても、グローバル・フィールドが使用されます。

場合によっては、グローバル定義とローカル定義を混合させることができます。 たとえば、KLIST および PLIST はグローバルにもローカルにもすることができます。 グローバル KLIST および PLIST と関連付けられたフィールドには、グローバル・フィールドだけが入れられます。 ローカル KLIST および PLIST と関連付けられたフィールドには、グローバルとローカルの両方のフィールドを入れることができます。サブプロシージャー内部での KLIST と KFLD の動作について詳しくは、95 ページの『定義の有効範囲』を参照してください。

定義の記憶域

ローカル定義には自動記憶域が使用されます。 **自動記憶域**は、プロシージャーに対する呼び出しの間だけ存在する記憶域です。 自動記憶域の変数は、呼び出しの間にまたがってその値が保管されることはありません。

他方、グローバル定義には静的記憶域が使用されます。 **静的記憶域**は、プログラムまたはプロシージャーのすべての呼び出しの間にわたって、メモリー内に常に位置を占めている記憶域です。 ここには、呼び出しの間にまたがってその値が保存されます。

ローカル・フィールド定義に静的記憶域を使用することを指示するためには、STATIC キーワードを指定してください。この場合には、プロシージャーに対する各呼び出しでのその値が保存されることになります。STATIC キーワードを指定した場合には、モジュールの初期化時に項目が初期化されます。

サイクル・モジュールにおいて、グローバル定義のための静的記憶域は RPG サイクルに従うため、前回の呼び出し終了時に LR がオンになっていた場合、次回のサイクル・メイン・プロシージャーの呼び出し時には値が変更されます。 ただし、ローカル静的変数については、サイクル・メイン・プロシージャーの LR が原因で再初期化されることはありません。

ヒント:

自動記憶域の使用により、実行時にプログラムで必要とされる記憶域の量が削減さ れます。 自動記憶域はプロシージャーの実行の間だけ割り振られるため、記憶域は大幅に 削減されます。 これに対して、プログラムと関連したすべての静的記憶域は、その静的記憶域を 使用するプロシージャーが呼び出されていなくても、プログラムが開始された時 に割り振られます。

独立フィールド

個別の作業フィールドは、独立フィールドによって定義することができます。 独立フィールド には、以下の特性があります。

- 明記可能な内部データ・タイプを持つ。
- 配列、テーブル、またはフィールドとして定義することができる。
- 絶対バイト位置によってではなく、データ長によって定義される。

独立フィールドの詳細については、次を参照してください。

- 229 ページの『配列およびテーブルの使用』
- 246 ページの『データ・タイプおよびデータ形式』
- 412 ページの『定義仕様書のキーワード』

変数の初期化

データは、定義仕様書の 442 ページの『INZ{(初期値)}』 キーワードによって初期化することができます。 初期値を INZ キーワードのパラメーターとして指定するか、あるいはパラメータ ーのないキーワードを指定してデフォルトの初期値を使用してください。 INZ キーワードを使用して表すには初期化が複雑すぎる場合には、初期化 サブルーチンの中でそのデータをさらに初期化することができます。

各種データ・タイプのデフォルトの初期値については、246 ページの『データ・タイプおよびデータ形式』で説明されています。 配列の初期化については、229 ページの『配列およびテーブルの使用』を参照してください。

プログラムの実行中にデータを再初期化するためには、CLEAR および RESET 命 令を使用してください。

CLEAR 命令コードは、レコード様式または変数 (フィールド、サブフィールド、 標識、データ構造、配列、またはテーブル) をそのデフォルトの値に設定します。 レコード様式、データ構造、または配列内のすべてのフィールドは、それらが宣 言された順に消去されます。

RESET 命令コードは、変数をそのリセット値に復元します。 グローバル変数のリセット値は、初期化サブルーチンが呼び出された後に、RPG IV サイクルの初期化ステップの終了時にそのグローバル変数に格納されていた値です。

初期化サブルーチンを使用してグローバル変数に初期値を割り当て、後で RESET を使用して、変数をこの値に設定し戻すことができます。 これは、初期化ステップの一部として自動的に実行される初期化サブルーチ ンに対してだけ適用されます。

ローカル変数の場合のリセット値は、サブプロシージャーが初めて呼び出された 時の、演算が開始される前のその変数の値です。

定数

定数のタイプには、リテラルと名前付き定数が あります。 これらは、以下の任意の場所に指定することができます。

- 演算項目 1
- 演算項目 2

- 演算仕様書の拡張演算項目 2
- 制御仕様書でキーワードに対するパラメーターとして
- 組み込み関数に対するパラメーターとして
- 出力仕様のフィールド名、定数、または編集語フィールドに
- 配列指標として
- WORKSTN 出力仕様に形式名として
- 定義仕様書にキーワードと一緒に

リテラル

リテラルは、プログラム中で参照することのできる自己定義定数です。 リテラルは、RPG IV データ・タイプのいずれかに含めることができます。

文字リテラル

文字リテラルを指定する場合の規則は次のとおりです。

- 文字リテラルでは、任意の文字の組み合わせを使用することができます。 これには DBCS 文字が含まれます。 DBCS 文字は、シフトアウトおよびシフトイン文字で囲み、偶数のバイト数に なっていなければなりません。 ブランクの挿入も有効です。
- アポストロフィの間に文字がない文字リテラルも使用 することができます。 例については、207 ページの図 58 を参照してください。
- 文字リテラルはアポストロフィ (') で囲む必要があります。
- リテラルの一部として必要なアポストロフィは、2 重のアポストロフィで表 されます。 例えば、リテラル O'CLOCK は'O''CLOCK'とコーディングされます。
- 文字リテラルは文字データとだけ互換性があります。
- 標識リテラルは、'1' (オン) または '0' (オフ) の いずれかが入れられる、1 バイトの文字リテラルです。

16 進数リテラル

16 進数リテラルを指定する場合の規則は次のとおりです。

- 16 進数リテラルは、次の形式をとります。

```
X'x1x2...xn'
```

  ここで、`X'x1x2...xn'` には、 文字 A から F、a から f、および 0 から 9 しか含めることができません。
- アポストロフィで囲んでコーディングされたリテラルの長さは偶数でなけれ ばなりません。
- 文字の対はそれぞれ 1 バイトを定義します。
- 16 進数リテラルは、ENDSR の演算項目 2 および編集語として以外は、文字リテ ラルがサポートされているところであればどこでも指定できます。
- ビット命令 BITON、BITOFF、TESTB で使用される場合を除き、16 進数リテラル の意味は対応する文字リテラルと同じです。 ビット命令の場合は、演算項目 2 には 1 バイトを表す 16 進数リテラルが含まれることがあります。 16 進数リテラルの規則と意味は、文字リテラルの場合と同じです。
- 16 進数リテラルに単一引用符の 16 進数値が含まれている場合には、文字リテ ラルと異なり、2 度指定する必要はありません。 例えば、リテラル `A'B` は `'A''B'` と指定されますが、16 進の場合は `X'C17DC2'` であり、`X'C17D7DC2'` ではありません。
- 通常、16 進数リテラルは文字データとだけ互換性が あります。 ただし、16 桁以下の 16 進数字を含む 16 進数リテラルは、数値式の 中で使用されるときや、数値変数が INZ キーワードを使用して初期化される ときは、符号なし数値として扱うことができます。

数字リテラル

数値リテラルを指定する場合の規則は次のとおりです。

- 数値リテラルは、数字 0 から 9 の任意の組み合わせによって構成されます。 小数点または符号を含めることができます。

- 符号 (+ または -) がある場合には、左端の文字としなければなりません。 符号なしリテラルは、正数として取り扱われます。
- 数値リテラルにブランクを表すことはできません。
- 数値リテラルはアポストロフィ (') で囲まれません。
- 数値リテラルは数値フィールドと同様に使用されます。ただし、その値を数 値リテラルに割り当てることはできません。
- 10 進数区切り記号はコンマまたはピリオドのいずれかにすることができます。

浮動形式の数値リテラルは、異なった方法で指定されます。 浮動リテラルの形式は次のとおりです。

```
     <mantissa>E<exponent>

Where

     <mantissa> is a literal as described above with 1 to 16 digits
     <exponent> is a literal with no decimal places, with a value
                between -308 and +308
```

- 浮動リテラルは正規化する必要がありません。 つまり、 小数部を書く場合に、小数点の左側に必ず 1 桁設ける必要がないということ です。 (さらに、小数点を指定する必要もありません。)
- 小文字の **e** を **E** の代わりに使用する ことができます。
- ピリオド (「.」) またはコンマ (「,」) のいずれかを小数点として使用できます。
- 浮動リテラルは、浮動データ・タイプを認めない命令の 中以外なら、数値定数が使用できる場所ならどこでも使用する ことができます。 たとえば、浮動リテラルは、配列指標などの、小数点以下の桁数がゼロである 数値リテラルが予想されている場所では使用できません。
- 浮動リテラルは、正規の数値リテラルと同じ連結規則に従います。 このリテラルは、その中のどこで分割してもかまいません。

次のリストは、有効な浮動リテラルをいくつか例証したものです。

```
    1E1                = 10
    1.2e-1             = .12
    -1234.9E0          = -1234.9
    12e12              = 12000000000000
    +67,89E+0003       = 67890  (the comma is the decimal point)
```

次のリストは、無効な浮動リテラルをいくつか例証したものです。

```
    1.234E                          <--- no exponent
    1.2e-                           <--- no exponent
    -1234.9E+309                    <--- exponent too big
    12E-2345                        <--- exponent too small
    1.797693134862316e308           <--- value too big
    179.7693134862316E306           <--- value too big
    0.0000000001E-308               <--- value too small
```

日付リテラル

日付リテラルは形式 D'xx-xx-xx' をとります。ここで、

- D は、リテラルが日付タイプであることを指示します。
- xx-xx-xx は、制御仕様書で指定された形式の有効な時刻です (区切り記号 を含む)。
- xx-xx-xx はアポストロフィで囲まれます。

時刻リテラル

時刻リテラルは形式 T'xx:xx:xx' をとります。ここで、

- T は、リテラルが時刻タイプであることを指示します。
- xx:xx:xx は制御仕様書で指定された形式の有効な時刻 です (区切り記号を含む)。
- xx:xx:xx はアポストロフィで囲まれます。

タイム・スタンプ・リテラル

タイム・スタンプ・リテラルの形式は Z'yyyy-mm-dd-hh.mm.ss' であり、 オプションで、その後に 1 個のピリオドとゼロ桁から 12 桁の 秒の小数部を続けることができます (Z'yyyy-mm-dd-hh.mm.ss.frac')。 各部の意味は次のとおりです。

- Z は、リテラルがタイム・スタンプ・タイプであることを指示します。
- yyyy-mm-dd は、有効な日付 (年-月-日) です。
- hh.mm.ss は、有効な時刻 (時.分.秒) です。
- frac は、ゼロ桁から 12 桁で、秒の小数部を表します。
- 秒の小数部はオプションであり、6 桁より少ない桁の小数部が指定された場合、 タイム・スタンプ・リテラルは秒の小数部が 6 桁になるようにゼロを追加して埋められます。

例えば、Z'2014-12-03-11.41.52' には秒の小数部が 6 桁あり、Z'2014-12-03-11.41.52.000000' と等価です。 Z'2014-12-03-11.41.52.123' には秒の小数部が 3 桁あり、Z'2014-12-03-11.41.52.123000' と等価です。 Z'2014-12-03-11.41.52.1234567' には秒の小数部が 7 桁あり、
Z'2014-12-03-11.41.52.123456789012' には秒の小数部が 12 桁あります。

図形リテラル

図形リテラルは形式 G'oK1K2i' をとります。ここで、

- G は、リテラルが図形タイプであることを指示します。
- o はシフトアウト文字です。
- K1K2 は偶数バイト (ゼロの場合もある) で、シフトアウトまたは シフトイン文字を含みません。
- i はシフトイン文字です。
- oK1K2i はアポストロフィで囲まれます。

UCS-2 リテラル

UCS-2 リテラルは U'Xxxx... という形式をとります。ここで、

- U は、リテラルがタイプ UCS-2 であることを指示します。
- 各 UCS-2 リテラルは、そのリテラル内の UCS-2 文字ごとに 4 バイトずつ必要と します。 そのリテラルの各 4 バイトは、それぞれ 1 個の 2 バイト UCS-2 文字を 表します。
- UCS-2 リテラルは UCS-2 データとだけ互換性があります。

UCS-2 リテラルは、モジュールのデフォルトの UCS-2 CCSID 内にあると想定されます。

リテラルおよびコンパイル時データの *CCSID*

制御ステートメントに CCSID(*EXACT) が指定されている場合、次のようになります。

- 文字リテラルの CCSID はコンパイルの CCSID です。
- グラフィック・リテラルの CCSID は、コンパイルの CCSID に関連する DBCS CCSID です。

コンパイルの CCSID は、ソース・ファイルを読み取るために使用される CCSID です。 これは、コマンドの TGTCCSID パラメーターによって指定されます。 TGTCCSID パラメーターは TGTCCSID(*SRC) にデフォルトで設定されています。 これは、1 次ソース・ファイルの CCSID に関連付けられた EBCDIC CCSID です。 TGTCCSID パラメーターについて詳しくは、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」の CRTBNDRPG コマンドの説明を参照してください。

CCSID(*EXACT) が指定されていない場合、次のようになります。

- 文字リテラルの CCSID は、実行時のジョブ CCSID に関連する混合バイト CCSID です。 文字リテラルに X'0E' が含まれる場合、CCSID(*CHAR) キーワードの有無にかかわらず、 コンパイラーは常にこの文字をシフトアウト文字として扱います。
- グラフィック・リテラルの CCSID は、制御ステートメントのキーワード CCSID(*GRAPH) で指定される CCSID です。 CCSID(*GRAPH) キーワードが指定されない場合、 グラフィック・リテラルは CCSID なしになります。

UCS-2 リテラルの CCSID は、制御ステートメントのキーワード CCSID(*UCS2) で指定される CCSID です。このキーワードが指定されない場合、UCS-2 リテラルは CCSID 13488 になります。

16 進リテラルの CCSID は 65535 または *HEX です。

コンパイル時データの CCSID は、リテラルの CCSID と同じです。

リテラルの定義例

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
H DATFMT(*ISO)
 * Examples of literals used to initialize fields
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D.................................Keywords+++++++++++++++++++++++++++++
D DateField       S               D   INZ(D'1988-09-03')
D NumField        S              5P 1 INZ(5.2)
D CharField       S             10A   INZ('abcdefghij')
D UCS2Field       S              2C   INZ(U'00610062')
 * Even though the date field is defined with a 2-digit year, the
 * initialization value must be defined with a 4-digit year, since
 * all literals must be specified in date format specified
 * on the control specification.
D YmdDate         S               D   INZ(D'2001-01-13')
D                                     DATFMT(*YMD)
 * Examples of literals used to define named constants
D DateConst       C                   CONST(D'1988-09-03')
D NumConst        C                   CONST(5.2)
D CharConst       C                   CONST('abcdefghij')

 * Note that the CONST keyword is not required.
D Upper           C                   'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

 * Note that the literal may be continued on the next line
D Lower           C                   'abcdefghijklmn-
D                                     opqrstuvwxyz'

 * Examples of literals used in operations
C                   EVAL      CharField = 'abc'
C                   IF        NumField > 12
C                   EVAL      DateField = D'1995-12-25'
C                   ENDIF
```

図 *57.* 名前付き定数の定義

長さがゼロのリテラルの使用例

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D.....................................Keywords++++++++++++++++++++++++++++++
 * The following two definitions are equivalent:
D varfld1         S              5    INZ VARYING
D varfld2         S              5    INZ('') VARYING
 * Various fields used by the examples below:
D blanks          S             10    INZ
D vblanks         S             10    INZ('   ') VARYING
D fixfld1         S              5    INZ('abcde')
 * VGRAPHIC and VUCS2 are initialized with zero-length literals.
D vgraphic        S             10G   INZ(G'oi') VARYING
D vucs2           S             10C   INZ(U'') VARYING
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq++++
 * The following statements do the same thing:
C                   eval      varfld1 = ''
C                   clear                 varfld1
 * Moving '' to a variable-length field using MOVE(P) or MOVEL(P)
 * sets the field to blanks up to the fields current length.
C                   move(p)   ''            varfld1
C                   movel(p)  ''            varfld1

 * Moving '' to a fixed-length field has no effect in the following
 * examples:  (The rightmost or leftmost 0 characters are changed.)
C                   move      ''            fixfld1
C                   movel     ''            fixfld1

 * The following comparisons demonstrate how the shorter operand
 * is padded with blanks:
C                   eval      *in01 = (blanks = '')
 * *in01 is '1'

C                   eval      *in02 = (vblanks = '')
 * *in02 is '1'

C                   eval      *in03 = (varfld2 = blanks)
 * *in03 is '1'

C                   eval      *in04 = (varfld2 = vblanks)
 * *in04 is '1'

C                   eval      *in05 = (%len(vgraphic)=0)
 * *in05 is '1'

 C                  eval      *in06 = (%len(vucs2)=0)
 * *in06 is '1'
```

図 *58.* 長さがゼロの文字、図形、および *UCS-2* リテラル

名前付き定数

定数に名前を与えることができます。 この名前は、プログラムの実行中に変更することはできない特定の値を 表します。 名前付き数値定数は、事前定義された精度を持っていません。 これらの 実際の精度は、指定された文脈の中で定義されます。

名前付き定数の定義例については、206 ページの図 57 を参照してください。 名前付き定数の値は、定義仕様書のキーワード・セクションに指定 されます。 ただし、キーワード CONST の指定はオプションです。 たとえば、'ab' の値を定数に割り当てているためには、キーワード・セクション に CONST('ab') か 'ab' のどちらかを指定することができます。

表意定数

形象定数 *BLANK/*BLANKS、*ZERO/*ZEROS、*HIVAL、*LOVAL、*NULL、*ALL'x..'、*ALLG'oK1K2i'、*ALLU'XxxxYyyy'、*ALLX'x1..'、および *ON/*OFF は、長さなしで指定することができる暗黙のリテラルです。形象定数の暗黙の長さおよび小数点以下の桁数は、関連したフィールドの長さおよび小数点以下の桁数と同じになるからです。(例外については、次のセクションの 209 ページの『表意定数に関する規則』 を参照してください。)

形象定数は演算仕様書の演算項目 1 および演算項目 2 の中で指定することが できます。 以下は、形象定数の予約語および暗黙の値を示しています。

予約語
暗黙の値

***BLANK/*BLANKS**
すべてブランク。 文字、図形、または UCS-2 フィールドの場合にだけ有効です。 文字の場合の値は ' ' (ブランク) または X'40'、 図形の場合は X'4040'、UCS-2 の場合は X'0020' です。

***ZERO/*ZEROS**
文字/数値フィールド: すべて ゼロ。 値は '0' または X'F0' です。 **数値浮動フィールドの 場合**: 値は '0 E0'。

***HIVAL**
文字、図形、または UCS-2 フィールド: システムで最高の照合文字 (16 進 FF)。 **数値フィールド**: 対応するフィールドに可能な 最大値 (適用可能な場合は正符号付き)。 **浮動フィールドの場合**: *HIVAL (4 バイト浮動の場合) = 3.402 823 5E38 (/x'7F7FFFFF'/) *HIVAL (8 バイト浮動の場合) = 1.797 693 134 862 315 E308 (/x'7FEFFFFFFFFFFFFF'/) **日付、時刻、およびタイム・スタンプ・フィールド**: 日付、時刻、およびタイム・スタンプ・データの *HIVAL 値については、273 ページの『日付データ・タイプ』、275 ページの『時刻データ・タイプ』、および 277 ページの『タイム・スタンプ・データ・タイプ』を参照してください。

***LOVAL**
文字、図形、または UCS-2 フィールド: システムで最低の照合文字 (16 進ゼロ)。 **数値フィールド**: 可能な最小値 (適用可能な 場合は負符号付き)。 **浮動フィールドの場合**: *LOVAL for 4-byte float = -3.402 823 5E38 (/x'FF7FFFFF'/) *LOVAL for 8-byte float = -1.797 693 134 862 315 E308 (/x'FFEFFFFFFFFFFFFF'/) **日付、時刻、およびタイム・スタンプ・フィールド**: 日付、時刻、およびタイム・スタンプ・データの *LOVAL 値については、273 ページの『日付データ・タイプ』、275 ページの『時刻データ・タイプ』、および 277 ページの『タイム・スタンプ・データ・タイプ』を参照してください。

***ALL'x..'**
文字/数値フィールド: 文字ストリング x.. は関連したフィールドの長さと等し くなるまで循環して反復されます。 フィールドが数値フィールドの場合には、ストリング内の すべての文字が数値 (0 から 9) でなければなりません。 *ALL'x..' が数値定数として使用された場合、符号または小数点を指定することができません。

注 : 浮動形式の数値フィールドで *ALL'x..' を使用することはできません。

注 : 数値整数または符号なしフィールドの場合には、その値が対応するフィール ドに可能な最大値より大きくなることはありません。 たとえば、対応するフィー ルドが 5 桁の整数フィールドである場合には、*ALL'95' は値 9595 を表します。 95959 は、5 桁の符号付き整数に可能な最大値より大きくなるからです。

***ALLG'oK1K2i'**
図形フィールド: 図形ストリング K1K2 は、 関連したフィールドの長さと等しくなるまで、周期的に反復されます。

***ALLU'XxxxYyyy'**
UCS-2 フィールド: 形式 *ALLU'XxxxYyyy' の形象定数は、形式が 'XxxxYyyyXxxxYyyy...' で、長さが *ALLU'XxxxYyyy' 定数に関連したフィールドの長さによって決まるリテラルを示します。 この定数内の 各 2 バイト文字は、4 桁の 16 進数字によって表されます。 たとえば、*ALLU'0041' は UCS-2 'A' の反復ストリングを 表します。

***ALLX'x1..'**
文字フィールド: 16 進数リテラル X'x1..' は関連したフィールドの長さと等し くなるまで循環して反復されます。

***NULL**
ヌル値は基底ポインター、プロシージャー・ポインター、またはオブジェクトに有効です。

***ON/*OFF**
*ON はすべて 1 ('1' または X'F1') です。*OFF はすべてゼロ ('0' または X'F0') です。 どちらも有効なのは、文 字フィールドの場合だけです。

表意定数に関する規則

形象定数の使用にあたっては、以下の規則に留意してください。

- MOVE および MOVEL 命令によって、文字形象定数を数値フィールドに 転送することができます。 形象定数は、まず、数値フィールド のサイズを持つゾーン数値として拡張され、必要な場合にはパックまたは 2 進数値に 変換されてから、受動数値フィールドに記憶されます。 この定数の各文字の数字部分が有効でなければな りません。 そうでない場合には、10 進数データ・エラーが起こります。
- 形象定数は基本項目と見なされます。 MOVEA の場合を除いて、形象定数は配 列と関連して使用されるかのように働きます。 たとえば、MOVE *ALL'XYZ' ARR があります。

  ARR が 4 バイトの文字要素を持つ場合には、各要素に 'XYZX' が入ります。
- MOVEA は、特殊な場合と見なされます。 この定数は、指定された配列の部分 と等しい長さで生成されます。 例:
  - MOVEA *BLANK ARR(X)

    要素 X から始まり、ARR の残りの部分にはブランクが入ります。
  - MOVEA *ALL'XYZ' ARR(X)

    ARR には 4 バイトの文字要素があります。 要素の境界は、文字の MOVEA 命 令では常にそうですが、無視されます。要素 X から始まり、配列の残りには「XYZXYZXYZ...」が入ります。

MOVEA の結果は上記の MOVE の例の結果とは異なることに注意してください。

- 代替照合順序の指定がある場合には、形象定数が該当する長さに設定/リセットされた後で、通常の照合順序を変更することができます。
- 移動命令の MOVE および MOVEL では、形象定数の *ALL'X..'、 *ALLG'oK1K2i'、 および *ALLX'x1..' を転送するときは同じ結果になります。 ストリングは、 関連したフィールドの長さとストリングの長さが同じになるまで 、文字単位で (左から順に) 周期的に反復されます。
- 演算項目の一方が形象定数ではない限り、形象定数は比較命令の中で使用す ることができます。
- 形象定数 *BLANK/*BLANKS は、MOVE 命令ではゼロとして数値フィールドへ 転送されます。

データ構造

ILE RPG コンパイラーによって、記憶域の区域およびその区域内のサブフィールドと呼ばれるフィールドのレイアウトを定義することができます。 記憶域のこの区域は、**データ構造**と呼ばれます。

自由形式でデータ構造を定義するには、 DCL-DS 命令コードの後にデータ構造名とキーワードを指定します。 固定形式でデータ構造を定義するには、定義仕様書の 24 から 25 桁目に DS を 指定します。

データ構造は次の目的で使用することができます。

- 種々のデータ形式を使用して同じ内部域を複数回定義する。
- レコードを定義するのと同じ方法でデータ構造およびそのサブフィールドを 定義する。
- データのセットの複数オカレンスを定義する。
- 不連続のデータを連続した内部記憶場所の中にグループ化する。
- データ構造の名前を使用してすべてのサブフィールドをグループとして操 作する。
- その名前を使用して個別のサブフィールドを操作する。

さらに、次の 4 つの特殊なデータ構造があり、それぞれが 特定の目的を持っています。

- データ域データ構造 (自由形式定義では DTAARA キーワードの *AUTO パラメーター によって識別され、固定形式定義では 23 桁目の U によって識別される)。
- ファイル情報データ構造 (ファイル仕様書 の INFDS キーワードによって識別される)。
- プログラム状況データ構造 (自由形式定義 では PSDS キーワードで識別され、固定形式定義では 23 桁目の S によって識別される)。
- 標識データ構造 (ファイル仕様書 の INDDS キーワードによって識別される)。

データ構造は、プログラム記述または外部記述とすることができます。 ただし、標識データ構造は プログラム記述のみです。

あるデータ構造を、LIKEDS キーワードを使用して別のデータ構造と同じように定義することができます。

プログラム記述データ構造は、自由形式定義では EXT または EXTNAME キーワード がないことによって識別され、固定形式定義では 22 桁目のブランクによって識別されます。 プログラム記述データ構造のサブフィールド定義は、データ構造定義の直後に続 いていなければなりません。

外部記述データ構造 (自由形式定義 では EXT キーワードまたは EXTNAME キーワードによって識別され、固定形式定義では 22 桁目の E によって識別される) のサブフィールド記述は外部記述ファイル に入っています。 コンパイル時に、ILE RPG コンパイラーは、外部名を使用してデータ構造サブフィールドの外部記述を見つけ、それを抜き出します。 EXTNAME キーワード が指定されていない場合、外部ファイル名にはデータ構造の名前が使用されます。

注 : 外部記述にサブフィールドについて指定されたデータ形式は、コンパイラーによ ってそのサブフィールドの内部形式として使用されます。 これは、外部記述ファイルの取り扱われ方と異なっています。

外部サブフィールドの名前は、キーワード EXTFLD を使用してプログラム中で変 更することができます。 キーワード PREFIX を使用して、EXTFLD によって名前が変更されていない外部サ ブフィールド名に接頭部を追加することができます。 ファイル名が、外部ファイル名を使用したデータ構造の定義時に EXTNAME キーワ ードに指定されたパラメーターと同じであっても、データ構造サブフィールドは 、ファイル仕様書に指定された PREFIX キーワードの影響を受けないことに注 意してください。 追加のサブフィールドは、外部サブフィールドのリストの直後にプログラム記述 サブフィールドを指定することによって、外部記述データ構造に追加することが できます。

外部記述データ構造は LIKEREC キーワードを使用して定義することもできます。

データ構造に CCSID(*EXACT) または CCSID(*NOEXACT) キーワードを指定することによって、外部記述データ構造の英数字サブフィールドの CCSID を制御できます。 CCSID(*EXACT) を指定すると、英数字サブフィールドの CCSID はファイル内のフィールド と同じ CCSID になります。 CCSID(*NOEXACT) を指定するか、または、データ構造に対して CCSID キーワードを指定しない 場合、英数字サブフィールドの CCSID は、英数字定義のデフォルト CCSID になります。 英数字 CCSID について詳しくは、419 ページの『CCSID(*EXACT | *NOEXACT)』 および 325 ページの『CCSID(*CHAR : *JOBRUN | *JOBRUNMIX | *UTF8 | *HEX | 番号)』 を参照してください。

データ構造名の修飾

キーワード QUALIFIED は、データ構造のサブフィールドが修飾表記を使用して参照されることを示します。 これは、データ構造のサブフィ ールドが、データ構造名と、その後にピリオドとサブフィールド名を指定することによってアクセスを許可します (例 DS1.FLD1)。 QUALIFIED キーワードを使用しない場合、サブフィールド名は未修飾のままとなります (例 FLD1)。 QUALIFIED が使用される場合、サブフィールド名は次のいずれか 1 つを使用して指定することができます。

- 「単純修飾名」は形式が「A.B」となる名前です。 単純修飾名は、ファイルおよび定義仕様書、入力仕様および出力仕様のフィールド名項目、固定形式演算仕様書の演算項目 1、演算項目 2、および結果フィールド項目 (つまり dsname.subf) のキーワードに対する引数として使用できます。 完全修飾名の要素間にはスペースを入れることができますが、単純修飾名にはスペースを入れることができません。
- 「完全修飾名」は、任意のレベルの数に修飾と指標付けが行われている名前です (例えば「A(X).B.C(Z+17)」)。完全修飾名は、ほとんどの自由形式の演算仕様書またはすべての拡張演算項目 2 記入項目で使用できます。 これには、自由形式演算でコード化されている命令コード CLEAR および DSPLY が含まれます。

さらに、任意レベルの指標付けおよび修飾を使用できます。 たとえば、プログラマーは ds(x).subf1.s2.s3(y+1).s4 を 式の中のオペランドとしてコーディングできます。 QUALIFIED キーワードの使用について詳しくは、475 ページの『QUALIFIED』 を参照してください。

自由形式演算仕様書でコーディングされる場合、完全修飾名は命令コード CLEAR および DSPLY の結果フィールド・オペランドとして 指定することができます。 式は、命令コード DSPLY の演算項目 1 および演算項

目 2 オペランドとして使用できますが (自由形式 演算仕様書でコーディング)、オペランドが完全修飾名よりも複雑である場合には、式は括弧で囲む必要があります。

QUALIFIED キーワードは、ネストされたデータ構造定義には使用されません。 ネストされたデータ構造は自動的に修飾されます。400 ページの『ネストされたデータ構造サブフィールド』を参照してください。

配列データ構造

「配列データ構造」は、DIM キーワードを使用して定義されるデータ構造です。 配列データ構造は、 複数オカレンス・データ構造と似ていますが、指標は配列の場合と同様に明示的に指定されます。

「キー付き配列データ構造」とは、1 つのサブフィールドが検索キーまたはソート・キーとして識別された配列データ構造です。 配列データ構造には (*) という指標が付き、その後にキー・サブフィールドの指定が続きます。

配列データ構造は、881 ページの『SORTA (配列の分類)』 命令コードを使用してソートできます。

- 配列は、*DS(*).KEY* 構文を指定することにより、サブフィールドの 1 つをキーとして使用してソートできます。
- 配列は、%FIELDS を指定して必須サブフィールドをリストすることにより、複数のサブフィールドをキーとして使用してソートできます。

配列データ構造は、%LOOKUP 組み込み関数を使用して検索できます。配列は、サブフィールドの 1 つをキーとして使用して検索されます。

%MAXARR または %MINARR 組み込み関数を使用することにより、そのサブフィールドの 1 つに対する最大値または最小値を持つ配列データ構造の要素を見つけることができます。

任意のサブフィールドを選択して、特定の SORTA 命令コードまたは %LOOKUP 組み込み関数のキーにすることができます。 例えば、ORDERS データ構造配列にサブフィールド ID および PRICE がある場合は、ID サブフィールドをキーとして使用してデータ構造をソートしてから、PRICE サブフィールドをキーとして使用してデータ構造をソートすることができます。

例えば、配列データ構造 FAMILIES にスカラー・サブフィールド NAME および NUM_CHILDREN、および配列サブフィールド CHILDREN があるとします。

```
DCL-DS families QUALIFIED DIM(10);
   name VARCHAR(25);
   num_children INT(10);
   DCL-DS children DIM(5);
      name VARCHAR(25);
      age INT(10);
   END-DS;
END-DS;
```

1. NAME をキーとする配列データ構造として FAMILIES データ構造を使用するには、FAMILIES(*).NAME と指定します。

   ```
   SORTA families(*).name;
   IF %LOOKUP('Smith' : families(*).name);
      ...
   ENDIF;
   ```

2. NUM_CHILDREN だけでなく、NUM_CHILDREN 値が同じである場合のために NAME によっても FAMILIES 配列をソートするには、NUM_CHILDREN を最初にリストし、NAME を 2 番目にリストして、%FIELDS を指定します。

   ```
   SORTA FAMILIES %FIELDS(NUM_CHILDREN : NAME);
   ```

3. 1 つの FAMILIES 要素の CHILDREN 配列サブフィールドを AGE サブフィールドでソートするには、FAMILIES(I).CHILDREN(*).AGE を使用します。

   ```
   FOR i = 1 to %ELEM(families);
      SORTA families(i).children(*).age;
   ENDFOR;
   ```

4. 最初の CHILDREN 要素の AGE によって FAMILIES 配列を検索するには、FAMILIES(*).CHILDREN(1).NAME を使用します。

```
family = %LOOKUP(10 : families(*).children(1).age);
```

配列データ構造の詳細な例については、以下を参照してください。

- SORTA 命令コード
- %LOOKUP 組み込み関数
- %MAXARR および %MINARR 組み込み関数

注 :

1. キーワード DIM は QUALIFIED として定義されるデータ構造に使用できます。
2. キーワード DIM をデータ構造または LIKEDS サブフィールドにコーディングする場合、配列キーワード CTDATA、FROMFILE、および TOFILE は使用できません。 さらに、次のデータ構造キーワードは配列データ構造では使用できません。
   - DTAARA
   - OCCURS
3. LIKEDS(Y) を使用して定義されているデータ構造 X の場合、データ構造 Y がキーワード DIM で定義されると、データ構造 X は 配列データ構造として定義されません。
4. X が配列データ構造 DS のサブフィールドである場合、修飾名で X が参照されるときには、配列指標を指定する必要があります。 さらに、配列指標は、キー付き配列データ構造のコンテキスト内を除いて、* でない場合があります。 完全修飾名の式においては、配列指標は一番右の名前についてのみ省略 (または * を指定) することができます。
5. 以下に、無効なキー付き配列データ構造式を使用したステートメントの例をいくつか示します。 TEAMS という配列データ構造があり、スカラー・サブフィールド MANAGER と配列データ構造サブフィールド EMPS が含まれているとします。

```
DCL-DS teams QUALIFIED DIM(10);
   manager VARCHAR(25);
   DCL-DS emps DIM(20);
      name VARCHAR(25);
      salary PACKED(7 : 2);
   END-DS;
END-DS:
```

   a. TEAMS は配列データ構造であるため、以下のステートメントは無効です。 非配列キー・サブフィールドが指定されなければなりません。

```
   SORTA TEAMS;
   SORTA TEAMS(*);
```

   b. TEAMS(1).EMPS は配列データ構造であるため、以下のステートメントは無効です。 非配列キー・サブフィールドが指定されなければなりません。

```
   SORTA TEAMS(1).EMPS;
   SORTA TEAMS(1).EMPS(*);
```

   c. TEAMS(*).EMPS(*) では、2 つの異なるソート対象配列が指定されているため、このステートメントは無効です。 (*) は 1 つしか指定できません。

```
   SORTA TEAMS(*).EMPS(*).NAME;
```

   d. 修飾名のすべての配列に指標が必要なため、以下のステートメントは無効です。 TEAMS と EMPS の両方のサブフィールドに指標が必要です。 一方には、(*) という指標が付いていなければなりません。

```
   SORTA TEAMS(*).EMPS.NAME;
   SORTA TEAMS.EMPS(*).NAME;
```

e. 少なくとも 1 つの配列に (*) という指標が付く必要があるため、以下のステートメントは無効です。 TEAMS(1).EMPS(1).NAME はスカラー値です。

```
SORTA TEAMS(1).EMPS(1).NAME;
```

プロトタイプまたはプロシージャー・インターフェースにおける データ構造パラメーターの定義

プロトタイプされたパラメーターをデータ構造として定義するには、まず通常のデータ構造を定義することによってパラメーターのレイアウトを定義する必要があります。 その後で、プロトタイプされたパラメーターを、LIKEDS キーワードを使ってデータ構造として定義することができます。 パラメーターのサブフィールドを使用するには、そのサブフィールドを「パラメーター名.サブフィールド」のようにパラメーター名で修飾して指定します。 以下に例を示します。

```
 * PartInfo is a data structure describing a part.
D  PartInfo       DS                  QUALIFIED
D  Manufactr                     4
D  Drug                          6
D  Strength                      3
D  Count                         3  0
 * Procedure "Proc" has a parameter "Part" that is a data
 * structure whose subfields are the same as the subfields
 * in "PartInfo".  When calling this procedure, it is best
 * to pass a parameter that is also defined LIKEDS(PartInfo)
 * (or pass "PartInfo" itself), but the compiler will allow
 * you to pass any character field that has the correct
 * length.
D Proc            PR
D  Part                             LIKEDS(PartInfo)
P Proc            B
 * The procedure interface also defines the parameter Part
 * with keyword LIKEDS(PartInfo).
 * This means the parameter is a data structure, and the subfields
 * can be used by specifying them qualified with "Part.", for
 * example "Part.Strength"
D Proc            PI
D  Part                             LIKEDS(PartInfo)
C                   IF        Part.Strength > getMaxStrength (Part.Drug)
C                   CALLP     PartError (Part : DRUG_STRENGTH_ERROR)
C                   ELSE
C                   EVAL      Part.Count = Part.Count + 1
C                   ENDIF
P Proc            E
```

データ構造サブフィールドの定義

自由形式でサブフィールドを定義するには、サブフィールドの 名前を指定してその後にキーワードを続けるか、または、 DCL-SUBF を指定してその後にサブフィールド名とキーワードを続けるか、または DCL-DS を指定して、ネストされたデータ構造サブフィールドを定義します。

固定形式でサブフィールドを定義するには、定義仕様書の定義タイプ 記入項目 (24 から 25 桁目) にブランクを指定します。 サブフィールド定義 (複数も可) は、データ構造定義の直後に続いていなければ なりません。 自由形式では、サブフィールド定義 は END-DS ステートメントで終わります。 固定形式では、サブフィールド定義は、定義タイプ記入項目がブランク以外の定義仕様書 が見つかるか、別の仕様タイプが見つかったときに終わります。

固定形式では、サブフィールドの名前は 7 から 21 桁目に記入されます。 ソース仕様の読みやすさを向上させるために、サブフィールド名を字下げし、それら がサブフィールドであることを視覚的に表示することができます。

データ構造が QUALIFIED キーワードを指定して定義されている場合、サブフィールド名はユーザーのプログラムの中の他の名前と同じであっても構いません。 サブフィールド名は、使用される時点でそのサブフィールドを所有するデータ構造によって修飾されます。

また、LIKE キーワードを使用して、サブフィールドを既存の項目と類似のものと して定義することができます。 このようにして定義された場合のサブフィールドの長さとデータ・タイプは、 その基礎になった項

目と同じになります。 同様に、LIKEDS キーワードまたは LIKEREC キーワードを使用してサブフィールドをデータ構造として定義することもできます。 LIKE キーワードの使用例については、445 ページの『LIKE キーワードを使用してデータを定義する例』 を参照してください。

キーワード LIKEDS はサブフィールド定義に指定できます。 指定される場合、 サブフィールドは、独自のサブフィールドのセットを持つデータ構造として定義されます。 データ構造 DS がサブフィールド S1 を持ち、このサブフィールドが サブフィールド S2 を持つデータ構造と同じに定義されている場合、プログラマーは式 DS.S1.S2 を使用して S2 を参照する必要が あります。

注 :

1. キーワード LIKEDS と LIKEREC は、QUALIFIED データ構造内でのみサブフィールドに使用できます。
2. ネストされたデータ構造は QUALIFIED データ構造内でのみ使用できます。400 ページの『ネストされたデータ構造サブフィールド』 を参照してください。
3. DIM キーワードを LIKEDS および LIKEREC キーワードと一緒に使用できます。

前に定義されたサブフィールドの記憶域は、OVERLAY キーワードを使用し、別の サブフィールドの記憶域でオーバーレイすることができます。 このキーワード は、後の方のサブフィールド定義に指定されます。 OVERLAY キーワードの使用例については、219 ページの図 63 を参照してください。

サブフィールドの長さの指定

サブフィールドの長さは、絶対 (位取り) 表記法、長さ表記法を使用して指定することができます。また、その長さを暗黙指定することもできます。

自由形式
長さは、データ・タイプ・キーワードのパラメーターとして指定されます。

固定形式での絶対表記
定義仕様書の「開始位置」(26 から 32 桁目) と「終了位置/長さ」(33 から 39 桁 目) の両方の記入項目に値を指定してください。

固定形式での長さ表記
「終了位置/長さ」(33 から 39 桁目) の記入項目に値を指定して ください。 「開始位置」の記入項目はブランクです。

暗黙の長さ
サブフィールドが 1 つ以上の OVERLAY キーワードの 最初のパラメーターで指定される場合、このサブフィールドはタイプまたは長さの情報を 指定しないで定義することができます。 この場合、タイプは文字になり、長さはオーバーレイされるサブフィールドによって決まります。

また、ポインター、日付、時刻、およびタイム・スタンプなどのように、 固定長になっているデータ・タイプもあります。 これらのタイプの場合、長さは暗黙指定されますが、固定形式で指定することもできます。

自由形式定義で POS キーワードが指定されていないか、 または、固定形式定義で開始位置が指定されていない場合、 サブフィールドは、前に定義されたすべてのサブフィールドの最大終了位置よりも大きい開始位置になるように 位置決めされます。各表記法の例については、217 ページの『データ構造の例』を参照してください。

データ構造サブフィールドの位置合わせ

サブフィールドの位置合わせが必要な場合があります。 場合によっては、それが自動的に行われることがありますが、手操作で行わ なければならないこともあります。

たとえば、長さ表記法を使用し、基底ポインターまたはプロシージャー・ポイン ター・タイプのサブフィールドを定義している場合には、コンパイラーが必要に 応じて埋め込みを行うので、サブフィールドは確実に正しく位置合わせされます。

浮動、 整数、または符号なしサブフィールドを定義している場合には、 実行時のパフォーマンスを改善するために位置合わせが 必要になる場合があります。 長さ表記法を使用してサブフィールドを定義する場合には、データ構造定義にキーワード ALIGN を 指定することによって、浮動、整数または符号なしサブフィールド を自動的に位置合わせすることができます。 ただし、以下の例外に注意してください。

- ファイル情報データ構造またはプログラム状況データ構造の場合には、 ALIGN キーワードを使用することはできません。
- キーワード OVERLAY を使用して定義されたサブフィールドは、データ構造に キーワード ALIGN が指定されていても、自動的には位置合わせされ ません。 この場合には、サブフィールドを手操作で位置合わせすることが必要です。

自動位置合わせでは、フィールドが次の境界で位置合わせされることになります。

- 5 桁の整数または符号なしサブフィールドの場合は 2 バイト
- 10 桁の整数または符号なしサブフィールドの場合または 4 バイト の浮動サブフィールドの場合は 4 バイト
- 20 桁の整数または符号なしサブフィールドの場合は 8 バイト
- 8 バイトの浮動サブフィールドの場合は 8 バイト
- ポインター・サブフィールドの場合は 16 バイト

フィールドを手操作で位置合わせしている場合には、それらが同じ境界で位置合わせ されることを確認してください。 ((桁 - 1) mod n) = 0 の場合には、開始位置は n バイト境界にあります。 (「x mod y」の値は、整数算術で x を y で除算した余りです。 これは、X DIV Y の後の MVR 値と同じです。)

215 ページの図 59 は、一連のバイトを示し、位置合わせに使用される種々の境界を表しています。

図 *59.* データ位置合わせの境界

上のバイト順序については、次の点に注意してください。

- ((1-1) mod 16) = 0 であるので、1 桁目は 16 バイト境界上にあります。
- ((13-1) mod 4) = 0 であるので、13 桁目は 4 バイト境界上にあります。
- ((7-1) mod 4) = 2 であるので、7 桁目は 4 バイト境界上にありません。

ネストされたデータ構造の初期化

キーワード INZ(*LIKEDS) は LIKEDS サブフィールドで使用できます。 LIKEDS サブフィールドは、対応するデータ構造と完全に同じ ように初期化されます。

キーワード INZ は LIKEDS サブフィールドで使用できます。 LIKEDS サブフィールドのネストされたすべてのサブフィールドは、デフォルト値に初期化されます。 これは、 さらに深くネストされた LIKEDS サブフィールドにも適用されますが、ネストされた LIKEDS サブフィールドで INZ(*LIKEDS) が指定されているものには 適用されません。

キーワード INZ が主なデータ構造定義でコーディングされている場合、キーワード INZ は明示的な初期化なしに、データ構造のすべてのサブフィールドに暗黙指定されます。 これには、LIKEDS サブフィールドが含まれます。

特殊なデータ構造

特殊なデータ構造として、次のものがあります。

- データ域データ構造
- ファイル情報データ構造 (INFDS)
- プログラム状況データ構造
- 標識のデータ構造

データ域データ構造およびプログラム状況データ構造をサブプロシージャーの中に定義することはできないことに 注意してください。

データ域データ構造

データ域データ構造は、自由形式定義では DTAARA キーワードの *AUTO パラメーター によって識別され、固定形式定義では 23 桁目の U によって識別されます。

これは、コンパイラーに対して、 プログラムの初期化時に同じ名前のデータ域を読み込み、ロックする必要があ り、プログラムの終了時には同じデータ域を書き出し、アンロックする必要があ ることを指示します。ロックは内部データ域には適用されません (216 ページの『内部データ域 (LDA)』 を参照)。 データ域データ構造は、他のすべてのデータ構造と同様に、タイプ文字を持って います。 また、データ域データ構造に読み込まれるデータ域は文字でなければなりません。 *DTAARA DEFINE 命令コードまたは DTAARA キーワードを使用して ILE RPG プログラムの中でデータ域の名前を変更した場合を除いて、データ域とデータ域データ構造は同じ名前である必要があります。

暗黙の読み込みおよび書き出しが行われるデータ域に対するデータ域命令 (IN、 OUT、および UNLOCK) を指定することができます。 これらの命令でデータ域データ構造を使用する前に、*DTAARA DEFINE 命令の結果 のフィールドの中か、または DTAARA キーワードによってそのデータ域データ 構造名を指定しなければなりません。

*ENTRY PLIST の PARM 命令の結果のフィールドにデータ域データ構造を指定する ことはできません。

内部データ域 *(LDA)*

コンパイラーは、以下のシチュエーションで内部データ域を使用します。

- 自由形式定義で、名前の付いていないデータ構造の DTAARA キーワード がパラメーターなしで指定されている。
- 固定形式定義で、データ域データ構造の名前がブランク になっている (23 桁目に U が入っている定義仕様書の 7 から 21 桁目)。

内部データ域に名前を指定するためには、演算項目 2 に *LDA、および結果のフィ ールドに名前あるいは定義仕様書に DTAARA(*LDA) を持つ *DTAARA DEFINE 命令 を使用してください。

ファイル情報データ構造

プログラム中の各ファイルについて (ファイル仕様書でキーワード INFDS に よって定義される) ファイル情報データ構造を指定することができます。 これによって、ファイル例外/エラーが起こった場合の状況情報が提供されます。 ファイル情報データ構造は 1 つのファイルにのみ使用することができます。 ファイル情報データ構造には、ファイル例外/エラーが起こった場合に情報を提 供する事前定義のサブフィールドが含まれています。ファイル情報データ構造およびそのサブフィールドの説明については、146 ページの『ファイル情報データ構造』を参照してください。

プログラム状況データ構造

プログラム状況データ構造は、自由形式定義では PSDS キーワードで、固定形式定義では 23 桁目の S で示され、プログラム例外/エラー情報をプログラムに提供します。プログラム状況データ構造およびその事前定義のサブフィールドの説明については、163 ページの『プログラム状況データ構造』を参照してください。

標識のデータ構造

標識データ構造は、ファイル仕様書の INDDS キーワードに よって識別されます。 この構造は、ファイルのデータ管理機能との間で渡される、 条件付け標識および応答標識を記憶するために使用します。 標識データ構造は、デフォルトにより、すべてゼロ (複数の '0') に初期化されます。

このデータ構造を定義するための規則は次のとおりです。

- 外部記述であってはなりません。
- 標識サブフィールドまたは固定長文字サブフィールドのみを持つことができます。
- 複数オカレンス・データ構造として定義することができます。
- このデータ構造の %SIZE は 99 を戻します。 複数オカレンス・データ構造の 場合、%SIZE(ds:*ALL) は 99 の倍数を戻します。 長さを指定した場合は 99 に なります。
- サブフィールドには、全長が 99 を超えない限り、標識の配列を 入れることができます。

データ構造の例

以下の例は、データ構造の各種の使用法およびそれらの定義方法を示しています。

例	記述
217 ページの図 60	フィールドを分割するデータ構造の使用
218 ページの図 61	フィールドをグループ化するデータ構造の使用
218 ページの図 62	データ構造に対するキーワード QUALIFIED、LIKEDS、および DIM の使用、および完全修飾サブフィールドのコーディングの 方法
219 ページの図 63	絶対表記法および長さ表記法によるデータ構造
219 ページの図 64	外部記述データ構造の名前の変更および初期化
220 ページの図 65	外部データ構造内のすべてのフィールドの名前を変更する PREFIX の使用
220 ページの図 66	複数オカレンス・データ構造の定義
221 ページの図 67	データ構造サブフィールドの位置合わせ
222 ページの図 68	*LDA データ域データ構造の定義
222 ページの図 69	データ域データ構造の使用 (1)
223 ページの図 70	データ域データ構造の使用 (2)
223 ページの図 71	標識データ構造の使用
224 ページの図 72	複数回繰り返し標識データ構造の 使用

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D.....................................Keywords++++++++++++++++++++++++++++++
 *
 * Use length notation to define the data structure subfields.
 * You can refer to the entire data structure by using Partno, or by
 * using the individual subfields Manufactr, Drug, Strength or Count.
 *
D Partno          DS
D  Manufactr                     4
D  Drug                          6
D  Strength                      3
D  Count                         3  0
D
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC.................................
I.......................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr......
 *
 *  Records in program described file FILEIN contain a field, Partno,
 *  which needs to be subdivided for processing in this program.
 *  To achieve this, the field Partno is described as a data structure
 *  using the above Definition specification
 *
IFILEIN    NS  01   1 CA    2 CB
I                                  3   18  Partno
I                                 19   29  Name
I                                 30   40  Patno
```

図 *60.* フィールドを分割するデータ構造の使用

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D.....................................Keywords++++++++++++++++++++++++++++++
 *
 *  When you use a data structure to group fields, fields from
 *  non-adjacent locations on the input record can be made to occupy
 *  adjacent internal locations. The area can then be referred to by
 *  the data structure name or individual subfield name.
 *
D Partkey         DS
D  Location                     4
D  Partno                       8
D  Type                         4
D
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC..............................
I.....................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr......
 *
 *  Fields from program described file TRANSACTN need to be
 *  compared to the field retrieved from an Item_Master file
 *
ITRANSACTN NS  01   1 C1    2 C2
I                                3   10  Partno
I                               11   16 0Quantity
I                               17   20  Type
I                               21   21  Code
I                               22   25  Location
I
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  Use the data structure name Partkey, to compare to the field
 *  Item_Nbr
 *
C                   :
C     Partkey       IFEQ      Item_Nbr                              99
C                   :
C*
```

図 *61.* フィールドをグループ化するデータ構造の使用

```
     D CustomerInfo     DS                        QUALIFIED BASED(@)
     D    Name                               20A
     D    Address                            50A

     D ProductInfo      DS                        QUALIFIED BASED(@)
     D    Number                              5A
     D    Description                        20A
     D    Cost                                9P 2

     D SalesTransaction...
     D                  DS                        QUALIFIED
     D    Buyer                                   LIKEDS(CustomerInfo)
     D    Seller                                  LIKEDS(CustomerInfo)
     D    NumProducts                        10I 0
     D    Products                                LIKEDS(ProductInfo)
     D                                            DIM(10)

      /free
          TotalCost = 0;
          for i = 1 to SalesTransation. Numproducts;
              TotalCost = TotalCost + SalesTransaction.Products (i).Cost;
              dsply SalesTransaction.Products (i).Cost;
          endfor;
          dsply ('Total cost is ' + %char(TotalCost));
      /end-free
```

図 *62.* データ構造に対するキーワード *QUALIFIED*、*LIKEDS*、および *DIM* の使用

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D.................................Keywords++++++++++++++++++++++++++++++
 *
 *  Define a program described data structure called FRED
 *  The data structure is composed of 5 fields:
 *   1.   An array with element length 10 and dimension 70(Field1)
 *   2.   A field of length 30 (Field2)
 *   3/4. Divide Field2 in 2 equal length fields (Field3 and Field4)
 *   5.   Define a binary field over the 3rd field
 *  Note the indentation to improve readability
 *
 *
 *  Absolute notation:
 *
 *  The compiler will determine the array element length (Field1)
 *  by dividing the total length (700) by the dimension (70)
 *
D FRED            DS
D  Field1                1    700    DIM(70)
D  Field2              701    730
D   Field3             701    715
D    Field5            701    704B 2
D   Field4             716    730
 *
 *  Length notation:
 *
 *  The OVERLAY keyword is used to subdivide Field2
 *
D FRED            DS
D  Field1                     10    DIM(70)
D  Field2                     30
D   Field3                    15    OVERLAY(Field2)
D    Field5                    4B 2 OVERLAY(Field3)
D   Field4                    15    OVERLAY(Field2:16)
```

図 *63.* 絶対表記法および長さ表記法によるデータ構造

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D.................................Keywords++++++++++++++++++++++++++++++
 *
 *  Define an externally described data structure with internal name
 *  FRED and external name EXTDS and rename field CUST to CUSTNAME
 *  Initialize CUSTNAME to 'GEORGE' and PRICE to 1234.89.
 *  Assign to subfield ITMARR the DIM keyword.
 *  The ITMARR subfield is defined in the external description as a
 *  100 byte character field.  This divides the 100 byte character
 *  field into 10 array elements, each 10 bytes long.
 *  Using the DIM keyword on an externally described numeric subfield
 *  should be done with caution, because it will divide the field into
 *  array elements (similar to the way it does when absolute notation
 *  is used for program described subfields).
 *
D Fred          E DS                  EXTNAME(EXTDS)
D   CUSTNAME    E                     EXTFLD(CUST) INZ('GEORGE')
D   PRICE       E                     INZ(1234.89)
D   ITMARR      E                     DIM(10)
```

図 *64.* 外部記述データ構造の名前の変更および初期化

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D.....................................Keywords++++++++++++++++++++++++++++++
D
D extds1        E DS                  EXTNAME (CUSTDATA)
D                                     PREFIX (CU_)
D   Name        E                     INZ ('Joe's Garage')
D   Custnum     E                     EXTFLD (NUMBER)
D
 *
 *  The previous data structure will expand as follows:
 *  -- All externally described fields are included in the data
 *     structure
 *  -- Renamed subfields keep their new names
 *  -- Subfields that are not renamed are prefixed with the
 *     prefix string
 *
 *  Expanded data structure:
 *
D EXTDS1        E DS
D   CU_NAME     E               20A   EXTFLD (NAME)
D                                     INZ ('Joe's Garage')
D   CU_ADDR     E               50A   EXTFLD (ADDR)
D   CUSTNUM     E                9S0  EXTFLD (NUMBER)
D   CU_SALESMN  E                7P0  EXTFLD (SALESMN)
```

図 *65.* 外部データ構造内のすべてのフィールドの名前を変更する *PREFIX* の使用

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D.....................................Keywords++++++++++++++++++++++++++++++
 *
 *  Define a Multiple Occurrence data structure of 20 elements with:
 *  -- 3 fields of character 20
 *  -- A 4th field of character 10 which overlaps the 2nd
 *     field starting at the second position.
 *
 *  Named constant 'Max_Occur' is used to define the number of
 *  occurrences.
 *
 *  Absolute notation (using begin/end positions)
 *
D Max_Occur       C                   CONST(20)
D
DDataStruct       DS                  OCCURS (Max_Occur)
D field1                   1     20
D field2                  21     40
D  field21                22     31
D field3                  41     60
 *
 *  Mixture of absolute and length notation
 *
D DataStruct      DS                  OCCURS(twenty)
D  field1                         20
D  field2                         20
D   field21               22      31
D  field3                 41      60
```

図 *66.* 複数オカレンス・データ構造の定義

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
 * Data structure with alignment:
D MyDS            DS                  ALIGN
 *  Properly aligned subfields
 *    Integer subfields using absolute notation.
D    Subf1                33    34I 0
D    Subf2                37    40I 0
 *    Integer subfields using length notation.
 *    Note that Subf3 will go directly after Subf2
 *    since positions 41-42 are on a 2-byte boundary.
 *    However, Subf4 must be placed in positions 45-48
 *    which is the next 4-byte boundary after 42.
D    Subf3                       5I 0
D    Subf4                      10I 0
 *    Integer subfields using OVERLAY.
D    Group               101   120A
D      Subf6                     5I 0 OVERLAY (Group: 3)
D      Subf7                    10I 0 OVERLAY (Group: 5)
D      Subf8                     5U 0 OVERLAY (Group: 9)
 *  Subfields that are not properly aligned:
 *    Integer subfields using absolute notation:
D    SubfX1               10    11I 0
D    SubfX2               15    18I 0
 *    Integer subfields using OVERLAY:
D    BadGroup            101   120A
D      SubfX3                    5I 0 OVERLAY (BadGroup: 2)
D      SubfX4                   10I 0 OVERLAY (BadGroup: 6)
D      SubfX5                   10U 0 OVERLAY (BadGroup: 11)
 *    Integer subfields using OVERLAY:
D    WorseGroup          200   299A
D      SubfX6                    5I 0 OVERLAY (WorseGroup)
D      SubfX7                   10I 0 OVERLAY (WorseGroup: 3)
 *
 * The subfields receive warning messages for the following reasons:
 * SubfX1 - end position (11) is not a multiple of 2 for a 2 byte field.
 * SubfX2 - end position (18) is not a multiple of 4 for a 4 byte field.
 * SubfX3 - end position (103) is not a multiple of 2.
 * SubfX4 - end position (109) is not a multiple of 4.
 * SubfX5 - end position (114) is not a multiple of 4.
 * SubfX6 - end position (201) is not a multiple of 2.
 * SubfX7 - end position (205) is not a multiple of 4.
```

図 *67.* データ構造サブフィールドの位置合わせ

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D.................................Keywords++++++++++++++++++++++++++++++
 *
 *  Define a data area data structure based on the *LDA.
 *
 *  Example 1:
 *  A data area data structure with no name is based on the *LDA.
 *  In this case, the DTAARA keyword does not have to be used.
 *
D               UDS
D  SUBFLD                 1    600A
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
 *  Example 2:
 *  This data structure is explicitly based on the *LDA using
 *  the DTAARA keyword.  Since it is not a data area data
 *  structure, it must be handled using IN and OUT operations.
 *
D LDA_DS          DS                  DTAARA(*LDA)
D  SUBFLD                 1    600A
...
C                   IN        LDA_DS
C                   OUT       LDA_DS
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
 *  Example 3:
 *  This data structure is explicitly based on the *LDA using
 *  the DTAARA keyword.  Since it is a data area data
 *  structure, it is read in during initialization and written
 *  out during termination.  It can also be handled using IN
 *  and OUT operations, since the DTAARA keyword was used.
 *
D LDA_DS          UDS                 DTAARA(*LDA)
D  SUBFLD                 1    600A
...
C                   IN        LDA_DS
C                   OUT       LDA_DS
```

図 *68. *LDA* データ域データ構造の定義*

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
HKeywords+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H DFTNAME(Program1)
H
 *
FFilename++IPEASF.....L.....A.Device+.Keywords++++++++++++++++++++++++++++
FSALESDTA  IF   E             DISK
 *
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D.................................Keywords++++++++++++++++++++++++++++++
 *
 *  This program uses a data area data structure to accumulate
 *  a series of totals.  The data area subfields are then added
 *  to fields from the file SALESDTA.
D  Totals         UDS
D    Tot_amount                  8 2
D    Tot_gross                  10 2
D    Tot_net                    10 2
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
CL0N01Factor1+++++++Opcode(E)+Factor2++++++++++++++++++++++++++++++++++++++++
 *
C                   :
C                   EVAL      Tot_amount = Tot_amount + amount
C                   EVAL      Tot_gross  = Tot_gross  + gross
C                   EVAL      Tot_net    = Tot_net    + net
```

図 *69. データ域データ構造の使用 (プログラム 1)*

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H DFTNAME(Program2)
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D.....................................Keywords+++++++++++++++++++++++++++++
 *
 *  This program processes the totals accumulated in Program1.
 *  Program2 then uses the total in the subfields to do calculations.
 *
D  Totals        UDS
D    Tot_amount                 8 2
D    Tot_gross                 10 2
D    Tot_net                   10 2
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
C                   :
C                   EVAL      *IN91 =  (Amount2 <> Tot_amount)
C                   EVAL      *IN92 =  (Gross2 <> Tot_gross)
C                   EVAL      *IN93 =  (Net2   <> Tot_net)
C                   :
```

図 *70.* データ域データ構造の使用 *(*プログラム *2)*

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
FFilename++IPEASFRLen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++
 * Indicator data structure "DispInds" is associated to file "Disp".
FDisp      CF   E             WORKSTN INDDS (DispInds)
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D.....................................Keywords++++++++++++++++++++++++++++++
 *
 * This is the indicator data structure:
 *
D DispInds        DS
 * Conditioning indicators for format "Query"
D    ShowName             21    21N
 * Response indicators for format "Query"
D    Exit                  3     3N
D    Return               12    12N
D    BlankNum             31    31N
 * Conditioning indicators for format "DispSflCtl"
D    SFLDSPCTL            41    41N
D    SFLDSP               42    42N
D    SFLEND               43    43N
D    SFLCLR               44    44N
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 * Set indicators to display the subfile:
C                   EVAL      SFLDSP = *ON
C                   EVAL      SFLEND = *OFF
C                   EVAL      SFLCLR = *OFF
C                   EXFMT     DispSFLCTL
 *
 * Using indicator variables, we can write more readable programs:
C                   EXFMT     Query
C                   IF        Exit or Return
C                   RETURN
C                   ENDIF
```

図 *71.* 標識データ構造の使用

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
FFilename++IPEASFRLen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++
 * Indicator data structure "ErrorInds" is associated to file "Disp".
FDisp      CF   E             WORKSTN INDDS (ERRORINDS)
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D.................................Keywords+++++++++++++++++++++++++++++++
D @NameOk         C                   0
D @NameNotFound   C                   1
D @NameNotValid   C                   2
D @NumErrors      C                   2
 *
 * Indicator data structure for ERRMSG:
 *
D ERRORINDS        DS                  OCCURS(@NumErrors)
 * Indicators for ERRMSG:
D   NotFound               1     1N
D   NotValid               2     2N
 *
 * Indicators for QUERY:
D    Exit                  3     3N
D    Refresh               5     5N
D    Return               12    12N
 *
 * Prototype for GetName procedure (code not shown)
D GetName          PR            10I 0
D    Name                        50A    CONST
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
C                   DOU       Exit or Return
C                   EXFMT     QUERY
 * Check the response indicators
C                   SELECT
C                   WHEN      Exit or Return
C                   RETURN
C                   WHEN      Refresh
C                   RESET             QUERY
C                   ITER
C                   ENDSL
 *
 * Check the name
C                   EVAL      RC = GetName(Name)
 *
 * If it is not valid, display an error message
C                   IF        RC <> @NameOk
C     RC            OCCURS    ErrorInds
C                   EXFMT     ERRMSG
C                   ENDIF
C                   ENDDO
 ...
C     *INZSR        BEGSR
 *
 * Initialize the occurrences of the ErrorInds data structure
C     @NameNotFound OCCUR     ErrorInds
C                   EVAL      NotFound = '1'
C     @NameNotValid OCCUR     ErrorInds
C                   EVAL      NotValid = '1'
C                   ENDSR
```

図 72. 複数回繰り返し標識データ構造の 使用

プロトタイプおよびパラメーター

お勧めのプログラムおよびプロシージャーの呼び出し方法は、プロトタイプ呼び出しを使用することです。プロトタイプ呼び出しでは、 コンパイル時に呼び出しインターフェースをコンパイラーに検査させることができるからです。 サブプロシージャーをコーディングする場合には、 コンパイラーが呼び出しインターフェースをそのサブプロシージャーと突き合わせることができるように、 プロシージャー・インターフェース定義をコーディングすることが必要になります。

この節では、以下の各概念を定義する方法について説明します。

- 225 ページの『プロトタイプ』
- 226 ページの『プロトタイプ・パラメーター』

- 228 ページの『プロシージャー・インターフェース』.

プロトタイプ

プロトタイプは、呼び出しインターフェースの定義です。 これには、次の情報が含まれます。

- 呼び出しが結合 (プロシージャー) と動的 (プログラム) のいずれであるか
- プログラムまたはプロシージャー (外部名) の検索方法
- パラメーターの数および特質
- 渡す必要があるパラメーター、およびオプションで渡されるパラメーター
- 操作記述子を渡す必要があるかどうか
- 戻り値があった場合のそのデータ・タイプ (プロシージャーの場合)

「多重定義プロトタイプ」と呼ばれる特別なタイプのプロトタイプがあります。 これにより、呼び出し操作のために多重定義プロトタイプの名前を使用して、複数の異なる候補プロトタイプの 1 つを呼び出すことができます。 詳しくは、470 ページの『OVERLOAD(プロトタイプ 1 { : プロトタイプ 2 ...})』を参照してください。

プロトタイプは明示的に定義することも、暗黙的に定義することもできます。 異なる RPG モジュールからプロシージャーが呼び出される場合には、 呼び出し元モジュールと、プロシージャーを定義するモジュールの両方にプロトタイプが明示的に指定されていなければなりません。 プロシージャーが同じモジュール内でのみ呼び出される場合には、 明示的にプロトタイプを定義することも、それを省略することもできます。 プロトタイプが省略されると、コンパイラーは、プロシージャー・インターフェースからそれを暗黙的に定義します。

別のモジュールで定義されたプロシージャーを呼び出すモジュールでは、呼び出しを行うプログラムまたはプロシージャーの定義仕様書 にプロトタイプが含まれていなければなりません。 このプロトタイプは、プログラムまたはプロシージャーを正しく呼び出し、呼び出 し元が確実に正しいパラメーターを渡せるように、コンパイラーによって使用さ れます。

デフォルトでは、コンパイラーに、メイン・プロシージャーまたはエクスポートされたプロシージャーのプロトタイプは必要ありません。 REQPREXP コマンド・パラメーターまたは制御仕様書キーワードを使用して、メイン・プロシージャーまたはエクスポート・プロシージャーにプロトタイプが指定されていない場合のコンパイル時に、警告メッセージまたはエラー・メッセージが発行されるようにすることができます。 344 ページの『REQPREXP(*NO | *WARN | *REQUIRE)』を参照してください。

プロトタイプ定義に適用される規則は、次のとおりです。

- プロトタイプには名前が付いている必要があります。 キーワード EXTPGM または EXTPROC をプロトタイプ定義に指定した場合には、プ ログラムまたはプロシージャーに対するすべての呼び出しで、そのキーワードにつ いて指定された外部名が使用されます。 どちらのキーワードも指定されていない場合、外部名が大文字でプロトタイプ名になります。
- 自由形式では、DCL-PR 命令コードを指定し、その後にプロトタイプ名とキーワードを 続けます。固定形式では、定義タイプ記入項目 (24 から 25 桁目) に PR を指定 します。 その他のパラメーター定義は、この PR の指定の直後に続けなければなりません。 自由形式では、プロトタイプ定義 は END-PR ステートメントで終わります。固定形式では、プロトタイプ定義は、24 から 25 桁目 がブランク以外の最初の定義仕様か、非定義仕様によって終わります。
- 呼び出しインターフェースと関係した次の任意のキーワードを指定してください。

  EXTPROC(名前)
  呼び出しは、キーワードによって指定された外部名を使用する結合プロシージ ャー呼び出しとなります。

  EXTPGM(名前)
  呼び出しは、キーワードによって指定された外部名を使用する外部プログラム 呼び出しとなります。

  OPDESC
  プロトタイプに記述されたパラメーターによって操作記述子が渡されます。

RTNPARM
戻り値がパラメーターとして処理されます。 特に戻り値が大きい場合に、プロシージャー呼び出し時のパフォーマンスが向上する可能性があります。

- 戻り値 (それがあった場合) は PR 定義に指定されます。 戻り値の長さおよびデータ・タイプを指定してください。 さらに、戻り値について次のキーワードを指定することができます。

DATFMT(形式)
戻り値は、キーワードによって指定された日付の形式を持ちます。

DIM(N)
戻り値は、要素が N の配列またはデータ構造です。

LIKEDS(データ構造名)
戻り値はデータ構造です。 (プロシージャーを呼び出す場合、戻り値のサブフィールドを参照することはできません。)

LIKEREC(名前 {,タイプ })
戻り値は、指定されたレコード様式名と同様に定義されているデータ構造です。

注 : プロシージャーを呼び出す場合、戻り値の サブフィールドを参照することはできません。

LIKE(名前)
戻り値は、キーワードによって指定された項目と同じに定義されます。

PROCPTR
戻り値は、プロシージャー・ポインターです。

TIMFMT(形式)
戻り値は、キーワードによって指定された時刻の形式を持ちます。

VARYING{(2|4)}
文字、図形、または UCS-2 戻り値は可変長形式です。

これらのキーワードについては、412 ページの『定義仕様書のキーワード』を参照してください。226 ページの図 73 は、数値入力パラメーターを受け入れて、文字ストリン グを戻すサブプロシージャー CVTCHR のプロトタイプを示しています。 戻り値と関連した名前はないことに注意してください。 この理由から、プログラムのデバッグ時にその内容を表示することはできません。

```
     * The returned value is the character representation of
     * the input parameter NUM, left-justified and padded on
     * the right with blanks.
    D CVTCHR          PR            31A
    D   NUM                         31P 0   VALUE
     * The following expression shows a call to CVTCHR.  If
     * variable rrn has the value 431, then after this EVAL,
     * variable msg would have the value
     *    'Record 431 was not found.'
    C                   EVAL      msg = 'Record '
    C                               + %TRIMR(CVTCHR(RRN))
    C                               + ' was not found '
```

図 *73. CVTCHR* のプロトタイプ

エクスポートされるサブプロシージャー用またはメイン・プロシージャー用のプロトタイプを書いている場合は、プロトタイプを /COPY ファイルに入れ、そのプロトタイプを、呼び出し側とそのプロシージャーを定義するモジュールの両方のソース・ファイルにコピーします。 このコーディング技法では、呼び出し元もプロシージャー自身もすべてが同一のプロトタイプを使用するため、両者にとってパラメーター・チェックの利点を最大に生かすことができます。

プロトタイプ・パラメーター

プロトタイプ呼び出しインターフェースがパラメーターの受け渡しと関係している場合には、 PR または PI の指定の直後にパラメーターを定義しなければなりません。 パラメーター定義仕様書では、そのタイプの定義に適用される次のキーワードを 使用することができます。

ASCEND
配列は昇順です。

DATFMT(形式)
日付パラメーターは (形式) と同じ形式を持ちます。

DESCEND
配列は降順です。

DIM(N)
パラメーターは、要素が N の配列またはデータ構造です。

LIKE(名前)
パラメーターは、キーワードによって指定された項目と同様に定義されます。

LIKEREC(名前 {,タイプ })
パラメーターは、指定されているレコード様式名のフィールドと同じサブフィールドを持つデータ構造です。

LIKEDS(データ構造名)
パラメーターは、LIKEDS キーワードの中で識別されるサブフィールドと同じサブフィールドを持つデータ構造です。

LIKEFILE(ファイル名)
このパラメーターはファイル、つまり ファイル名 で指定されるファイルか、または LIKEFILE キーワードを介してファイル名 に関連付けられるファイルのいずれかとなります。

PROCPTR
パラメーターはプロシージャー・ポインターです。

TIMFMT(形式)
時刻パラメーターは (形式) と同じ形式を持ちます。

VARYING{(2|4)}
文字、図形、または UCS-2 パラメーターは可変長形式です。

これらのキーワードについて詳しくは、412 ページの『定義仕様書のキーワード』を参照してください。

パラメーター定義仕様書では、パラメーターを渡す方法を指定する次のキーワー ドも使用することができます。

CONST
パラメーターは、読み取り専用参照によって渡されます。 CONST によって定義されたパラメーターは、呼び出されたプログラムまたはプロ シージャーによって変更されてはなりません。 このパラメーター受け渡し方式によって、リテラルおよび式を渡すことができます。

NOOPT
パラメーターは、呼び出されたプログラムまたはプロシージャーで最適化さ れません。

OPTIONS(opt1 { : opt2 { : opt3 { : opt4 { : opt5 } } } })

ここで、opt1 ... opt5 には *NOPASS、*OMIT、*VARSIZE、*STRING、 *TRIM、または *RIGHTADJ を指定することができます。 たとえば、**OPTIONS(*VARSIZE : *NOPASS)** のようになります。

次のパラメーター受け渡しオプションを指定します。

***NOPASS**
パラメーターを渡す必要はありません。 1 つのパラメーターに OPTIONS(*NOPASS) を指定した場合には、それに続くすべ てのパラメーターにも OPTIONS(*NOPASS) を指定しなければなりません。

***OMIT**
この参照パラメーターについて、特殊値 *OMIT が渡されることがあります。

***VARSIZE**
パラメーターには、定義で指示されたより少ないデータしか含まれていない ことがあります。 このキーワードが有効なのは、参照によって渡された文字パラメーター、図形パラメーター、UCS-2 パラメーターまたは配列の場合だけです。 呼び出されるプログラムまたはプロシージャーには、渡されたパラメーターの長 さを判別するなんらかの手段がなければなりません。

注 : 固定長フィールドの場合に、このキーワードを省略すると、パラメーターには、 定義で指示された量以上のデータが含まれることがあります。 可変長フィールドの場合、パラメーターは、定義で宣言された 最大長を持っていなければなりません。

***STRING**
ヌル文字で終了するストリングとして文字値を渡します。 このキーワードが 有効なのは、値によってまたは読み取り専用参照によって渡された基底ポインター・パラメーターの場合だけです。

***TRIM**
パラメーターは受け渡しの前にトリミングされます。 このオプションは、値によってまたは読み取り専用参照によって渡された、文字、UCS-2、 またはグラフィック・パラメーターの場合に有効です。 また、OPTIONS(*STRING) がコーディングされているポインター・パラメーターの場合にも有効です。

注 : ポインター・パラメーターで OPTIONS(*STRING : *TRIM) が指定されている場合には、 ポインターが直接渡される場合にも値がトリミングされます。 ポインターが指しているヌル終了ストリングは、 ブランクがトリミングされ、新しいヌル終止符が末尾に追加されて、一時ファイルにコピーされ、 その一時ファイルのアドレスが渡されます。

***RIGHTADJ**
CONST パラメーターまたは VALUE パラメーター の場合、*RIGHTADJ は、図形パラメーター、UCS-2 パラメーター、また は文字パラメーターの値が右寄せされることを示します。

ヒント : オプション *NOPASS、*OMIT、および *VARSIZE を渡すパラメーターの場合、 そのプロシージャーのプログラマーは、これらのオプションが確実に処理されるようにする必要があります。 たとえば、OPTIONS(*NOPASS) がコーディングされていて、パラメーターを**渡さない**ようにした場合、プロシージャーは、そのパラメーターにアクセスする前にそれが渡されていることをチェックする必要があります。 コンパイラーは、これについては一切チェックしません。

VALUE
パラメーターは値によって渡されます。

上記のキーワードについては、412 ページの『定義仕様書のキーワード』を参照してください。 プロトタイプ・パラメーターについて詳しくは、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」のプログラムおよびプロシージャーの呼び出しに関する章を参照してください。

プロシージャー・インターフェース

プロトタイプ・プログラムまたはプロシージャーに呼び出しパラメーターまたは戻り値がある場合は、 メイン・ソース・セクション (サイクル・メイン・プロシージャーの場合) 内またはサブプロシージャー・セクション内に、 プロシージャー・インターフェース定義が定義される必要があります。 プロトタイプが定義されている場合、**プロシージャー・インターフェース定義**は、プロシージャーの定義の中のプロトタイプ情報を繰り返します。 それ以外の場合には、コンパイラーがプロトタイプを暗黙的に定義できるようにする情報をプロシージャー・インターフェースが提供します。 プロシージャー・インターフェースは、プロシージャーへの入り口パラメーターを宣言し、プロシージャーの内部 定義を外部定義 (プロトタイプ) と確実に整合させるために使用されます。

プロシージャー・インターフェース定義に適用される規則は、次のとおりです。

- プロシージャー・インターフェースの名前は任意指定です。 指定される場合は、対応するプロトタイプ定義の名前と一致している必要があります。
- 自由形式定義では、DCL-PI を指定することによって、プロシージャー・インターフェース定義を 開始します。 固定形式定義では、「定義タイプ」記入項目 (24 から 25 桁目) に PI を 指定します。 プロシージャー・インターフェース定義は、定義仕様書のどこにでも指定するこ とができます。 サイクル・メイン・プロシージャーでは、プロトタイプが 指定される場合は、プロシージャー・インターフェースの前に、それが参照するプロトタイプがなければなりません。 プロシージャー・インターフェースは、プロシージャーが値を戻す 場合、またはプロシージャーにパラメーターがある場合は必須であり、それ以外の場合は任意指定です。
- その他のパラメーター定義は、このプロシージャー・インターフェースの指定の直後に続けなければなりません。

- 自由形式のプロシージャー・インターフェースは END-PI で終わる必要があります。これは、DCL-PI ステートメント の末尾に指定するか、パラメーターの後にある別のステートメントとして指定します。
- プロシージャー・インターフェースの名前を指定する必要はありません。 自由形式定義では、名前を指定しないことを示すために *N を使用します。
- プロシージャー・インターフェースの名前を指定する場合は、プロシージャーの名前と同じでなければなりません。 サイクル・メイン・プロシージャーのプロシージャー・インターフェースであり、名前を 指定する場合は、前に指定されたプロトタイプの名前と同じでなければなりません。
- パラメーター名を指定しなければなりませんが、それがプロトタイプに指定 された名前と一致している必要はありません。
- データ・タイプ、長さ、および次元を含むパラメーターのすべての属性が、 対応するプロトタイプ定義の中のものと正確に一致していなければなりません。
- パラメーターがデータ構造であることを示すには、 LIKEDS キーワードを使用して、別のデータ構造と同じサブフィールドを持つパラメーターを定義します。
- PI 指定およびパラメーター指定に設定されたキーワードは、プロトタイプが明示的に指定されていれば、プロトタイプに指定されたものと一致していなければなりません。
- プロトタイプが指定されない場合、EXTPGM キーワードまたは EXTPROC キーワードをプロシージャー・インターフェースに指定できます。

ヒント：

モジュールに、別のモジュールに定義されているプロトタイプ・プログラムまたはプロシージャーへの呼び出しが 含まれている場合、呼び出すプログラムおよびプロシージャーごとにプロトタイプ定義が なければなりません。 必要なコーディングを最小にする 1 つの方法は、共用プロトタイプを /COPY ファ イルに保管することです。

プロトタイプ・プログラムまたはプロシージャーを他のユーザーに提供する場合 には、必ずプロトタイプも (/COPY ファイルに入れて) 提供してください。

配列およびテーブルの使用

配列およびテーブルは、どちらも以下の点が同じデータ・フィールド (要素) の 集まりです。

- フィールド長
- データ・タイプ
  - 文字
  - 数値
  - データ構造
  - 日付
  - 時刻
  - タイム・スタンプ
  - ｸﾞﾗﾌｨｯｸ
  - 基底ポインター
  - プロシージャー・ポインター
  - UCS-2
- 様式
- 小数点以下の桁数 (数字の場合)

配列とテーブルは次の点で異なります。

- 特定の配列要素はその位置によって参照することができます。
- 特定のテーブル要素はその位置によって参照することができません。

- 配列名自体は配列中のすべての要素を参照します。
- テーブル名は常に最後の 798 ページの『LOOKUP (テーブルまたは配列要素の検索)』 命令で見つかった要素を参照します。

注 : サブプロシージャーには、実行時配列だけを定義することができます。 テーブル、実行時前の配列、およびコンパイル時配列はサポートされていません。 サブプロシージャーで実行時前の配列またはコンパイル時配列を使用す る場合は、メイン・ソース・セクションで定義する必要があります。

次の項では、配列のコーディング方法、配列要素の初期値の指定方法、配列の値 の変更方法、および配列の使用に関する特別な考慮事項について説明しています。 さらにその次の項では、テーブルに関する同じ情報について説明しています。

配列

配列には、次の 3 つのタイプがあります。

- 実行時配列 は、ユーザー・プログラムの実行中にそのプログラム によってロードされます。
- コンパイル時配列 は、ユーザー・プログラムの作成時にロードさ れます。 初期データはユーザー・プログラムの永続部分になります。
- 実行時前の配列 は、ユーザー・プログラムの実行開始の時点で、入 力操作、演算操作、または出力操作が処理される前に、配列ファイルからロード されます。

実行時配列の場合の配列の定義およびロードに関する基本事項が説明されます。 コンパイル時および実行時前の配列の定義およびロードの場合には、これらの基本 事項に加えて、いくつか追加の仕様が使用されます。

配列名および指標

配列全体を参照するためには、配列名を単独で使用します。 配列の個々の要素を参照するためには、(1) 配列名、(2) そのあとに左括弧、 (3) そのあとに指標、(4) そのあとに右括弧という指定をします。 たとえば、AR(IND) のようにします。 指標は、配列内の (1 から始まる) 要素の位置を示すものであり、数または数の入っているフィールドのいずれかです。 指標は、配列内の (1 から始まる) 要素の位置を示すものであり、数字か数字が 入っているフィールドのいずれかです。

配列名および指標を指定する場合には、次の規則が適用されます。

- 配列名は、固有の記号名でなければなりません。
- 指標は、小数点以下の桁数のないゼロより大きい数字フィールドまたは固定 情報でなければなりません。
- 拡張演算項目 2 フィールドの式の中で配列が指定された場合には、指標は小 数点以下の桁数のない数字を戻す式とすることができます。
- 実行時に、ゼロ、負、または配列内の要素の数より大きい値を持つ指標を使 用してユーザー・プログラムが配列を参照した場合には、エラー/例外ルーチン がプログラムの制御を受け取ります。

配列の基本仕様

配列は定義仕様書に定義します。 以下は、すべての配列に共通の基本仕様です。

- DIM キーワードを使用して配列の項目の数を指定します。
- すべてのスカラー・フィールドの場合と同様に、長さ、データ形式、および 小数点以下の桁数を指定します。 明示的な「開始位置」および「終了位置」記入項目 (固定形式でサブフィールドを定義する場 合)、または明示的な「長さ」記入項目を指定することができます。あるいは、 LIKE キーワードを使用して属性を定義することができます。属性は、 プログラムの他の場所で指定することもできます。
- 分類順序を指定する必要がある場合には、ASCEND または DESCEND キーワー ドを使用してください。

231 ページの図 74 は、配列の基本仕様の例を示しています。

実行時配列のコーディング

配列の基本仕様以外の指定を行わない場合には、実行時配列 を定義 したことになります。 キーワード ALT、CTDATA、EXTFMT、FROMFILE、PERRCD、 および TOFILE は実行時配列に使用することはできないことに注意してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
DARC              S              3A   DIM(12)
```

図 *74.* 実行時配列を定義する配列の基本仕様

実行時配列のロード

定義仕様書の INZ キーワードを使用して、実行時配列に初期値を割り当てることが できます。 入力仕様または演算仕様書によって実行時配列に初期値を割り当てることもで きます。 この 2 番目の方式を使用すれば、データを他のタイプの配列に入れることもでき ます。

たとえば、演算仕様書を使用して、MOVE 命令で配列の各要素 (または選択した要 素) に 0 を入れることができます。

入力仕様を使用して、ファイルからのデータを配列に埋め込むことができます。 以下の項では、ファイルのレコードからこのデータを検索する際の詳細が示され ています。

注 : 日付および時刻の実行時データは同じ形式になっていなければならず、ロー ドしている日付配列または時刻配列と同じ区切り記号が使用されていなければな りません。

ファイルからレコードを *1* つ読み込んで実行時配列をロードする

データベース・ファイルから読み込まれる入力レコードに、配列全体に関するすべての情報が含まれていれば、その配列を 1 回の入力操作でロードすることができます。データベース・レコード内のフィールドのうち、配列に対応するものがデータベース・レコード内で連続して配置されている場合、単一の入力仕様を使用して、その配列をロードすることができます (231 ページの図 75 を参照)。 この入力仕様では、データベース・レコードにおける配列全体の位置を定義します。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
DINPARR           S             12A   DIM(6)
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC.............................
I.....................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
IARRFILE   AA  01
I                                 1   72  INPARR
```

図 *75.* 連続した要素を持つ実行時配列の使用

データベース・レコード内のフィールドのうち、配列に対応するものがデータベース・レコード内に分散している場合は、複数の入力仕様を使用して、その配列をロードしなければなりません。 232 ページの図 76 の例では、すべての配列要素のデータがデータベース・レコードに含まれていますが、データベース・レコード内のデータは配列要素ごとにブランクで区切られています。 単一の要素に対応するデータベース・レコード内の位置を、それぞれの入力仕様で定義します。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
DARRX             S             12A   DIM(6)
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC..............................
I.....................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
IARRFILE   AA  01
I                                1   12  ARRX(1)
I                               14   25  ARRX(2)
I                               27   38  ARRX(3)
I                               40   51  ARRX(4)
I                               53   64  ARRX(5)
I                               66   77  ARRX(6)
```

図 76. 要素が散在した実行時配列の定義

ファイルから複数のレコードを読み込んで実行時配列をロードする

データベース・ファイルの単一のレコードからだけでは配列のデータを取得できない場合、データベース・ファイルから複数のレコードを読み込んで、配列をロードしなければなりません。 それぞれのレコードによって、配列の要素 (単一または複数) にデータを提供します。 ILE RPG プログラムは、レコードを一度に 1 つずつ処理します。 したがって、配列全体の処理が行われるのは、配列情報の入っているレコードがすべて読み取られて、その情報が配列要素に転送された後になります。 配列全体がプログラムに読み込まれるまで、演算および出力操作を抑制すること が必要な場合があります。

例えば、ファイル ARRFILE2 の各レコードの桁 1 から 12 に、ある配列要素に関する情報が含まれているものとします。 このような場合には、変数指標を使用して、その配列要素の入力仕様をコーディングすることができます。 232 ページの図 77 にあるように、レコードが読み込まれる前に、プログラムによって指標が設定されます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
DARRX             S             12A   DIM(6)
DN                S              5P 0 INZ(1)
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC..............................
I.....................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr....
IARRFILE2  AA  01
I                                  1   12  ARRX(N)
CL0N01Factor1+++++++Opcode&ExtFactor2;+++++++Result++++++++Len++D+HiLoEq
C                   IF        N = %ELEM(ARR) * The array has been loaded
 ..... process the array
 * Set the index to 1 to prepare for the next complete array
C                   EVAL      N = 1
C                   ELSE * Increment the index so the next input operation will fill
 * the next array element
C                   EVAL      N = N + 1
C                   ENDIF
```

図 77. ファイルからの配列のロード *(1* レコードにつき *1* 要素*)*

同一の外部記述フィールドから配列をロードする

外部記述データベース・ファイルの入力レコードに、複数のフィールドがまったく同じように定義されている場合には、データ構造を定義して、それらのフィールドをまるで 1 つの配列であるかのように処理することができます。 以下の 3 つのケースを検討してみます。

1. フィールドがレコード内で連続しており、かつレコードの先頭にある場合

```
     A          R REC
     A            FLD1           5P 0
     A            FLD2           5P 0
     A            FLD3           5P 0
     A            OTHER         10A
```

この場合は、外部記述データ構造を使用し、配列を追加サブフィールドとして定義します。 次に OVERLAY キーワードを指定して、その配列をフィールドにマッピングします。

```
FMYFILE    IF   E             DISK
D myDS          E DS                    EXTNAME(MYFILE)
D   fldArray                            LIKE(FLD1) DIM(3)
D                                       OVERLAY(myDs)

C                   READ      MYFILE
C                   FOR       i = 1 to %ELEM(fldArray)
C*                     ... process fldArray(i)
C                   ENDFOR
```

2. フィールドがレコード内で連続しており、かつレコードの先頭にはない場合

```
A          R REC
A            OTHER1        10A
A            ... more fields
A            FLD1          15A
A            FLD2          15A
A            FLD3          15A
A            OTHER2        10A
```

この場合は、外部記述データ構造を使用し、配列を独立フィールドとして定義します。 次に、BASED キーワードを使用して、その配列をフィールドにマッピングし、先頭フィールドのアドレスを指す基底ポインターを初期化します。

```
FMYFILE    IF   E             DISK
D myDS          E DS                    EXTNAME(MYFILE)

D fldArray        S                     LIKE(FLD1) DIM(3)
D                                       BASED(pFldArray)
D pFldArray       S                *    INZ(%addr(FLD1))

C                   READ      MYFILE
C                   FOR       i = 1 to %ELEM(fldArray)
C*                     ... process fldArray(i)
C                   ENDFOR
```

3. フィールドがレコード内で連続していない場合

```
A          R REC
A            OTHER1        10A
A            FLD1            T         TIMFMT(*ISO)
A            FLD2            T         TIMFMT(*ISO)
A            OTHER2        10A
A            FLD3            T         TIMFMT(*ISO)
A            OTHER3        10A
```

この場合は、プログラム記述データ構造を定義し、いかなるタイプの情報も定義せずに、その配列に使用するフィールドを列挙します。 次に、OVERLAY キーワードを使用して、配列をフィールドにマッピングします。

```
FMYFILE    IF   E             DISK
D myDS            DS
D   FLD1
D   FLD2
D   FLD3
D   fldArray                            LIKE(FLD1) DIM(3)
D                                       OVERLAY(myDs)
C                   READ      MYFILE
C                   FOR       i = 1 to %ELEM(fldArray)
C*                     ... process fldArray(i)
C                   ENDFOR
```

実行時配列の順序付け

実行時配列の順序は検査されません。 SORTA (配列の分類) 命令を処理する場合に は、配列は、その配列を定義する定義仕様書に指定された順序 (ASCEND または DESCEND キーワード) に分類されます。 順序が指定されない場合には、配列は昇順に分類されます。 LOOKUP 命令で高位 (演算仕様書の 71 から 72 桁目) または低位 (演算仕様書の 73 から 74 桁目) の標識を 使用する場合には、配列の順序を指定しなければなりません。

コンパイル時配列のコーディング

コンパイル時配列は、配列の基本仕様およびキーワード CTDATA を使用して指定 されます。 さらに、定義仕様書で次のように指定することができます。

- PERRCD キーワードを使用して入力レコード中の配列項目の数。 このキーワードが指定されない場合には、項目の数のデフォルトの値として 1 が使用 されます。
- EXTFMT キーワードを使用して外部データ形式。 使用できる値は、L (先行符号)、R (後書き符号)、および S (ゾーン 10 進数) だけです。 EXTFMT キーワードは浮動コンパイル時配列には使用できません。
- LR がオンでプログラムが終了した時に配列が書き出されるファイル。 これは 、TOFILE キーワードを使用して指定します。

コンパイル時配列の例については、234 ページの図 78 を参照してください。

コンパイル時配列のロード

コンパイル時配列 の場合には、配列ソース・データをプログラム・ソース・メン バーのレコードの中に入力します。 **ALTSEQ、**CTDATA、および **FTRANS キーワードを使用した場合には、ソース・レ コードの後の任意の場所に配列データを入力することができます。 それらのキーワードを使用しない場合には、コンパイル時配列およびテーブルが 定義仕様書に定義されている順序で、ソース・レコード、および任意の代替照合順序 またはファイル変換レコードの後に配列データが続いていなければなりません。 このデータは、プログラムのコンパイル時に配列にロードされます。 前の呼び出し が LR がオフで終了した場合を除き、プログラムが新しいデータで再コンパイル されるまで、配列はプログラムを呼び出すたびに常に当初の値と同じままです。

コンパイル時配列は、個別にかまたは (ALT キーワードによって) 交互形式で記 述することができます。 交互形式とは、入力レコード上で 1 つの配列の要素が別の配列の要素と混ざり合 っていることを意味します。

配列ソース・レコードに関する規則

配列ソース・レコードについては、次の規則があります。

- 各レコードの最初の配列項目は、1 桁目から始めなければなりません。
- すべての要素は同じ長さで、中間にスペースを入れずに各要素が続いていな ければなりません。
- レコード全体に項目を埋め込む必要はありません。そうでない場合には、項目の後にブランクまたは注記を含めることができます (234 ページの図 78 を参照)。
- 定義仕様書に指定された配列の要素の数が指定された項目の数より大きい場合 には、残りの要素には指定されたデータ・タイプのデフォルトの値によって埋め込みが 行われます。

```
      DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++
      DARC              S               3A   DIM(12) PERRCD(5) CTDATA
**CTDATA ARC
48K16343J64044HComments can be placed here
12648A47349K346Comments can be placed here
50B125         Comments can be placed here
```

48K	163	43J	640	44H	125	48A	473	49K	346	50B	125

This is the compile-time array, ARC.

図 *78.* 注記を持つ配列ソース・レコード

- 最後のレコード以外の各レコードには、定義仕様書で PERRCD キーワードに よって指定された項目の数が入っていなければなりません。 最後のレコードでは、未使用の項目はブランクでなければならず、その未使用の 項目の後にブランクまたは注記を含めることができます。
- 各項目の全体が 1 つのレコードの中に入っていなければなりません。 1 つの項目を 2 つのレコードに分割することはできません。したがって、単一の 項目の長さは 100 文字の最大長 (ソース・レコードのサ

イズ) に制限されています。 配列が使用され、交互形式で記述されている場合には、対応する要素が同じレコ ード上になければならず、それらの合計の長さが 100 文字を超えることはできま せん。

- 日付および時刻のコンパイル時配列の場合には、データは同じ形式になって いなければならず、ロードされている日付または時刻配列と同じ区切り記号が使 用されなければなりません。
- 配列データは、次の 2 つの方法の 1 つで指定することができます。

  1. **CTDATA 配列名: 配列のデータはコンパイル時データ・セクションのどこ でも指定することができます。
  2. **b: (b=ブランク) 配列のデータは定義仕様書に指定された順序どおりに指定 しなければなりません。

  1 つのプログラムには、これらの手法を 1 つだけ使用することができます。
- 配列は昇順 (ASCEND キーワード)、降順 (DESCEND キーワード)、または順 序指定なし (キーワードを指定しない) にすることができます。
- 昇順または降順の文字配列の場合には、制御仕様書に ALTSEQ(*EXT) が指定さ れた時に、代替照合順序が順序検査に使用されます。 コンパイル時に実際の照合順序 が不明の場合 (たとえば、制御仕様書またはコマンド・パラメーター に SRTSEQ(*JOBRUN) が指定された場合) には、実行時に代替照合順序テーブルが検索され、*INIT での初期化時に検査が行われます。 そうでない場合には、検査はコンパイル時に行われます。
- 図形および UCS-2 配列は、代替照合順序とは無関係 に、16 進数値によって分類されます。
- 定義仕様書の EXTFMT キーワードに L または R が指定された場合には、各要素に符号 (+ または -) が含まれている必要があります。L が指定され、要素サイズが 2 の配列は、次の例に示されているように、ソース・データ中に 3 桁を必要とします。

```
*....+....1....+....2....+....3....+....4....+....5....+....6....+....*
      DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++
      D UPDATES                        2 0  DIM(5) PERRCD(5) EXTFMT(L) CTDATA
**CTDATA UDPATES
+37-38+52-63-49+51
```

- 浮動コンパイル時データは、浮動リテラルまたは数値リテラルとして ソース・レコード内に指定されます。 4 バイト浮動として定義された配列の場合、各要素ごとに 14 桁必要 です。また、8 バイト浮動として定義された配列の場合は、各要素ごと に 23 桁必要です。
- 図形データは、シフトアウトおよびシフトイン文字で囲まなければなり ません。 図形データの複数の要素が (中間に図形データ以外のデータを持たない) 単一の レコードに含まれている場合には、レコードには 1 セットのシフトアウトおよ びシフトイン文字だけが必要です。 図形配列を図形配列以外の配列と一緒に交互形式で定義する場合には、図形 データをシフトインおよびシフトアウト文字で囲まなければなりません。 2 つの配列を交互形式で定義する場合には、各レコードに 1 セットのシフトインおよびシフトアウト文字だけが必要です。

実行時前の配列のコーディング

実行時前の配列の場合には、配列の基本仕様に加えて、以下の仕様書またはキーワ ードもコーディングすることができます。

定義仕様書では、次のものを指定することができます。

- FROMFILE キーワードを使用して、配列入力データが入っているファイルの名 前。
- TOFILE キーワードを使用して、プログラムの終了時に配列が書き出されるフ ァイルの名前。
- PERRCD キーワードを使用して、入力レコード当たりの要素の数。
- EXTFMT キーワードを使用して、数値配列データの外部形式。
- ALT キーワードを使用して、交互形式。

注 : 定義される配列が 10 桁を超える場合には、整数および符号なし形式を指定することはできません。

ファイル仕様書では、配列入力データが入っているファイルについて 18 桁 目に T を指定することができます。

配列のコーディング例

236 ページの図 79 は、2 つの実行時前の配列、コンパイル時配列、および実行時配列に必要な定義仕様書を示しています。

```
*....+....1....+....2....+....3....+....4....+....5....+....6....+....*
      HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++
      H DATFMT(*USA) TIMFMT(*HMS)
      D*ame++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++
       * Run-time array.  ARI has 10 elements of type date. They are
       * initialized to September 15, 1994.  This is in month, day,
       * year format using a slash as a separator as defined on the
       * control specification.
      DARI              S               D   DIM(10) INZ(D'09/15/1994')
       *
       * Compile-time arrays in alternating format.  Both arrays have
       * eight elements (three elements per record).  ARC is a character
       * array of length 15, and ARD is a time array with a predefined
       * length of 8.
      DARC              S             15    DIM(8) PERRCD(3)
      D                                     CTDATA
      DARD              S               T   DIM(8) ALT(ARC)
       *
       * Prerun-time array.  ARE, which is to be read from file DISKIN,
       * has 250 character elements (12 elements per record).  Each
       * element is five positions long.  The size of each record
       * is 60 (5*12). The elements are arranged in ascending sequence.
      DARE              S              5A   DIM(250) PERRCD(12) ASCEND
      D                                     FROMFILE(DISKIN)
       *
       * Prerun-time array specified as a combined file.  ARH is written
       * back to the same file from which it is read when the program
       * ends normally with LR on.  ARH has 250  character elements
       * (12 elements per record).  Each elements is five positions long.
       * The elements are arranged in ascending sequence.
      DARH              S              5A   DIM(250) PERRCD(12) ASCEND
      D                                     FROMFILE(DISKOUT)
      D                                     TOFILE(DISKOUT)
**CTDATA ARC
Toronto        12:15:00Winnipeg       13:23:00Calgary        15:44:00
Sydney         17:24:30Edmonton       21:33:00Saskatoon      08:40:00
Regina         12:33:00Vancouver      13:20:00
```

図 *79.* 種々のタイプの配列の定義仕様書

実行時前の配列のロード

実行時前の配列 の場合には、配列入力データをファイルに入力してくだ さい。 このファイルは、プログラム記述の順次ファイルでなければなりません。 初期化時に、入力、演算、または出力命令が処理される前に、配列にファイル から初期値がロードされます。 このファイルを変更することによって、プログラムをコンパイルし直さずに、プログラムに対する次回の呼び出し時の配列の初期値を変更する ことができます。 こ のファイルは到着順で読み取られます。 実行時前の配列データに関する規則は、各レコードの長さに制約がない点を除いて 、コンパイル時配列データに関する規則と同じです。 234 ページの『配列ソース・レコードに関する規則』 を参照してください。

文字配列の順序検査

ALTSEQ(*NONE) によって定義されていない文字配列の順序検査には、 次の 2 つの依存関係があります。

1. ALTSEQ 制御仕様書キーワードの指定の有無、およびそれが指定された 場合の指定方法。
2. 配列がコンパイル時配列と実行時前の配列のいずれであるか。

以下の表は、順序検査が行われる時点を示しています。

制御仕様書の指定	SORTA、LOOKUP、および順序検査への ALTSEQ の使用	コンパイル時配列の順序検査の時点	実行時前の配列の順序検査の時点
ALTSEQ(*NONE)	いいえ	コンパイル時刻	ランタイム
ALTSEQ(*SRC)	いいえ	コンパイル時刻	ランタイム
ALTSEQ(*EXT) (コンパイル時に判明)	はい	コンパイル時刻	ランタイム
ALTSEQ(*EXT) (実行時にだけ判明)	はい	ランタイム	ランタイム

注 : RPG III との互換性のために、SORTA および LOOKUP では、ALTSEQ(*SRC) に よる代替照合順序は使用されません。 代替照合順序を使用してこれらの命令を実行したい場合には、ユーザーの代替順 序を含むテーブル (オブジェクト・タイプ *TBL) をシステム上に定義することが できます。 次に、制御仕様書で ALTSEQ(*SRC) を ALTSEQ(*EXT) に変更し、 作成コマンドの SRTSEQ キーワードまたはパラメーターに ユーザー・テーブルの名前を指定することができます。

配列の初期化

実行時配列

実行時配列の各要素を同じ値に初期化するためには、定義仕様書に INZ キー ワードを指定してください。 配列がデータ構造サブフィールドとして定義されている場合には、データ構造初期化のオーバーラップに関する一般規則が適用されます (初期化はフィール ドがデータ構造内で宣言されている順序で実行されます)。

コンパイル時配列および実行時配列

コンパイル時配列または実行時前の配列の初期値は他の手段 (コンパイル時データ または入力ファイルからのデータ) を通じて割り当てられるので、これらの配列 に INZ キーワードを指定することはできません。 コンパイル時配列または実行時前の配列がグローバルに初期化されたデータ構造 に現れた場合には、それはグローバルな初期化には含まれません。

注 : コンパイル時配列は、データがプログラムの後に宣言された順序で初期化 され、実行時前の配列は、それらの配列がデータ構造内で宣言された順序と無関係 に、その初期化ファイルの宣言の順序で初期化されます。 実行時前の配列は、コンパイル時配列の後で初期化されます。

サブフィールドの初期化がコンパイル時または実行時前の配列とオーバーラップ する場合には、配列の初期化が優先されます。すなわち、データ構造内でフィ ールドが宣言されている順序とは無関係に、配列はサブフィールドの後で初期化されます。

関連した配列の定義

交互配列の定義で ALT キーワードを使用することによって、2 つのコンパイル時 配列または 2 つの実行時前の配列を交互形式 でロードすることができます。 1 次配列の名前を ALT キーワードのパラメーターとして指定します。 このような配列のデータを記憶するレコードでは、最初の配列の最初の要素の後 に 2 番目の配列の最初の要素が続き、最初の配列の 2 番目の要素の後に 2 番目の 配列の 2 番目の要素が続き、最初の配列の 3 番目の要素の後に 3 番目の配列の 3 番目の要素が続き、以下同様となります。 対応する要素は、同じレコードに現 れていなければなりません。 主要配列定義の PERRCD キーワードによって、レコード当たりの対応する対の数、 すなわち、単一の項目としてカウントする要素のそれぞれの対が指定されます。 主要配列と交互配列の両方に EXTFMT を指定することができます。

238 ページの図 80 は、交互形式の ARRA と ARRB の 2 つの配列を示してい ます。

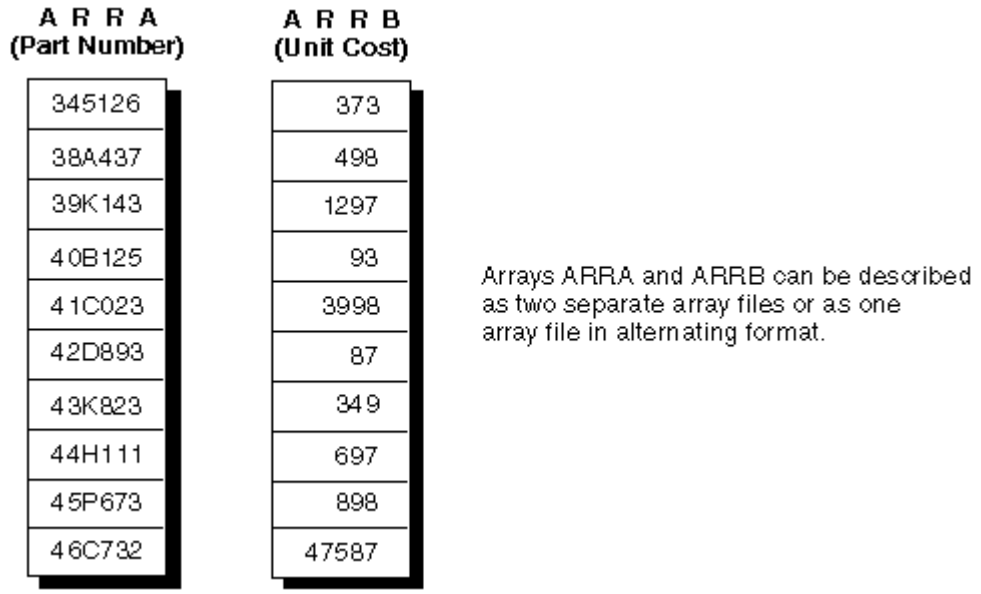


図 *80.* 交互形式の配列と交互形式でない配列

2 つの別個の配列ファイルとして記述した時には、ARRA および ARRB のレコード は次のようになります。

このレコードの 1 から 60 桁目には ARRA の項目が入ります。

ARRA entry	ARRA entry	ARRA entry	ARRA entry	ARRA entry	ARRA entry	ARRA entry	ARRA entry	ARRA entry	ARRA entry
1	7	13	19	25	31	37	43	49	55 . . . .

図 *81. 2* つの別個の配列ファイルの配列レコード

このレコードの 1 から 50 桁目には ARRB の項目が入ります。

ARRB entry	ARRB entry	ARRB entry	ARRB entry	ARRB entry	ARRB entry	ARRB entry	ARRB entry	ARRB entry	ARRB entry
1	6	11	16	21	26	31	36	41	46 . . . .

図 *82. 1* つの配列ファイルの配列レコード

交互形式の 1 つの配列ファイルとして記述した場合に、ARRA および ARRB のレコ ードは次のようになります。 最初のレコードの 1 から 55 桁目には、交互形式の ARRA および ARRB の項目が含まれます。 2 番目のレコードの 1 から 55 桁目には、交互形式の ARRA および ARRB の 項目が含まれます。

ARRA entry	ARRB entry	ARRA entry	ARRB entry	ARRA entry	ARRB entry	ARRA entry	ARRB entry	ARRA entry	ARRB entry
1	1	7	6	13	11	19	16	25	21 . . . .

図 *83.* 交互形式の *1* つの配列ファイルの配列レコード

```
      DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++
      DARRA             S              6A   DIM(6) PERRCD(1) CTDATA
      DARRB             S              5  0 DIM(6) ALT(ARRA)
      DARRGRAPHIC       S              3G   DIM(2) PERRCD(2) CTDATA
      DARRC             S              3A   DIM(2) ALT(ARRGRAPHIC)
      DARRGRAPH1        S              3G   DIM(2) PERRCD(2) CTDATA
      DARRGRAPH2        S              3G   DIM(2) ALT(ARRGRAPH1)
**CTDATA ARRA
345126  373
38A437  498
39K143 1297
40B125   93
41C023 3998
42D893   87
**CTDATA ARRGRAPHIC
ok1k2k3iabcok4k5k6iabc
**CTDATA ARRGRAPH1
ok1k2k3k4k5k6k1k2k3k4k5k6i
```

配列の検索

配列の検索には、以下を使用することができます。

- LOOKUP 命令コード
- %LOOKUP 組み込み関数
- %LOOKUPLT 組み込み関数
- %LOOKUPLE 組み込み関数
- %LOOKUPGT 組み込み関数
- %LOOKUPGE 組み込み関数

LOOKUP 命令コードの詳細については、以下の資料を参照してください。

- 240 ページの『指標を用いた配列の検索』
- 239 ページの『指標を用いない配列の検索』
- 798 ページの『LOOKUP (テーブルまたは配列要素の検索)』

%LOOKUPxx 組み込み関数について詳しくは、657 ページの『%LOOKUPxx (配列要素の検索)』を参照してください。

指標を用いない配列の検索

指標を使用しないで配列を検索する場合には、結果の標識の状況 (オンまたはオ フ) を使用して、特定の要素が配列に存在しているかどうかを判別してください。 指標を使用しない配列の検索は、あるフィールドが配列要素のリストにあるかど うかを判別するための入力データの妥当性検査に使用することができます。 一般 に、等しい条件の LOOKUP が要求されます。

演算仕様書の演算項目 1 に検索引数 (指定の配列中で一致するものが検索される データ) を指定して、配列名を演算項目 2 に入れてください。

演算項目 2 には、検索する配列の名前を指定します。 少なくとも 1 つの結果の標 識を指定しなければなりません。 同じ LOOKUP 命令で、高低の両方に指定をしてはなりません。 配列が 順序 (ASCEND または DESCEND キーワード) どおりになっていない場合には、結果の標識を 高または低に指定してはなりません。 制御レベルおよび条件付け標識 (7 から 11 桁目に指定) も使用することができます。 結果の標識を使用することはできません。

検索は、配列の先頭から開始され、配列の終わりかまたは検索の条件が満たされ た時に終了します。 行われた検索のタイプ (等しい、高い、低い) を満たす配列要素が見付かった時にはい つでも、結果の標識がオンに設定されます。

240 ページの図 84 は、指標を使用しない配列の LOOKUP の例を示しています。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++
FARRFILE   IT   F    5        DISK
F*
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++
DDPTNOS           S              5S 0 DIM(50) FROMFILE(ARRFILE)
D*
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C* The LOOKUP operation is processed and, if an element of DPTNOS equal
C* to the search argument (DPTNUM) is found, indicator 20 is set on.
C     DPTNUM        LOOKUP    DPTNOS                                  20
```

図 *84.* 指標を用いない配列に対する *LOOKUP* 命令

部門番号を含んでいる ARRFILE は、ファイル仕様書で、配列ファイルの指定 (18 桁目に T) のある入力ファイル (17 桁目に I) として定義されています。 このファイルはプログラム記述ファイル (22 桁目に F) であり、各レコードの長 さは 5 桁 (27 桁目に 5) です。

定義仕様書において、ARRFILE は配列 DPTNOS を含んでいるものとして定義され ます。 この配列には 50 の要素が入っています (DIM(50))。 各項目は、小数点以 下の桁数 (41 から 42 桁目) がゼロで、長さ (33 から 39 桁目) が 5 桁です。 各レコー ドに 1 つの部門番号を入れることができます (PERRCD のデフォルトの値として 1 が使 用されます)。

配列データ構造の検索

%LOOKUP 組み込み関数を使用して、配列データ構造のサブフィールドの 1 つをキーとして、配列データ構造を検索することができます。

配列データ構造の検索について詳しくは、657 ページの『%LOOKUPxx (配列要素の検索)』 を参照してください。

指標を用いた配列の検索

LOOKUP 検索条件を満たす要素を見付けるためには、配列中の特定の要素から検索 を開始してください。 このタイプの検索を実行するためには、指標を用いない配列の場合と同様に、演 算仕様書に指定をしてください。 しかし、演算項目 2 には、検索する配列の名前 と、その後に検索を開始する要素の番号が入った (小数点以下の桁数のない) 数 値フィールドを括弧で囲んで続けてください。 この数値定数または数値フィールドは、配列中の特定の要素を指し示すため 、指標と呼ばれます。 指標は、検索条件を満たした要素番号によって更新されたり、あるいは検索が正 常に実行されなかった場合には 0 に設定されます。

数値定数を指標として使用して、1 以外の要素から始まる検索条件を満たす 要素の存在をテストすることができます。

指標を用いない配列に適用されるこれ以外の規則は、すべて、指標を用いた配列 にも適用されます。

241 ページの図 85 は、指標を使用した配列の LOOKUP を示しています。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++
FARRFILE   IT   F   25            DISK
F*
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++
DDPTNOS           S              5S 0 DIM(50) FROMFILE(ARRFILE)
DDPTDSC           S             20A   DIM(50) ALT(DPTNOS)
D*
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C* The Z-ADD operation begins the LOOKUP at the first element in DPTNOS.
C                   Z-ADD     1             X                 3 0
C* At the end of a successful LOOKUP, when an element has been found
C* that contains an entry equal to the search argument DPTNUM,
C* indicator 20 is set on and the MOVE operation places the department
C* description, corresponding to the department number, into DPTNAM.
C     DPTNUM        LOOKUP    DPTNOS(X)                              20
C* If an element is not found that is equal to the search argument,
C* element X of DPTDSC is moved to DPTNAM.
C                   IF        NOT *IN20
C                   MOVE      DPTDSC(X)     DPTNAM           20
C                   ENDIF
```

図 *85.* 指標を用いた配列に対する *LOOKUP* 命令

この例は、240 ページの図 84 の場合と同じ部門番号の配列 DPTNOS を示しています。 しかし、部門名称の交互配列 DPTDSC も定義されています。 DPTDSC の各要素の長さは 20 桁です。 ファイル中のデータでは配列全体を初期化するのに不十分な場合には、 DPTNOS の残りの要素にゼロが埋め込まれ、DPTDSC の残りの要素にはブランクが 埋め込まれます。

配列の使用

配列は入力仕様、出力仕様、または演算仕様書の中で使用することができます。

演算での配列の指定

演算仕様書では、配列全体または配列の個々の要素を指定することができます。 個々の要素は、フィールドの場合と同様に処理することができます。

OVERLAY キーワードによって定義された不連続の配列は、MOVEA 命令によって、 あるいは PARM 命令の結果フィールドの中で使用することはできません。

配列全体を指定するためには、配列名だけを使用してください。配列名は、演算 項目 1、演算項目 2、または結果のフィールドとして使用することができます。 配列 名を使用できる命令は次のとおりです。すなわち、ADD、Z-ADD、SUB、Z-SUB、 MULT、DIV、SQRT、ADDDUR、SUBDUR、EVAL、EXTRCT、MOVE、MOVEL、 MOVEA、MLLZO、MLHZO、MHLZO、MHHZO、DEBUG、XFOOT、LOOKUP、 SORTA、PARM、DEFINE、CLEAR、RESET、CHECK、CHECKR、および SCAN です。

配列名だけでは使用できず、配列要素だけで使用できる命令もいくつかあります。 これらの命令には、BITON、BITOFF、COMP、CABxx、TESTZ、TESTN、TESTB、 MVR、DO、DOUxx、DOWxx、DOU、DOW、IFxx、WHENxx、WHEN、IF、 SUBST、および CAT が含まれますが、これらに限定されているわけではありません。

指標のない配列名またはアスタリスクを指標として持つ配列名 (たとえば、 ARRAY または ARRAY(*)) を指定した場合には、配列の各要素について特定の命令 が反復されます。 これらの命令は、ADD、Z-ADD、EVAL、SUB、Z-SUB、 ADDDUR、SUBDUR、EXTRCT、MULT、DIV、SQRT、MOVE、MOVEL、 MLLZO、MLHZO、MHLZO、および MHHZO です。 指標のない配列名を指定した場合には、これらの命令に対して次の規則が適用さ れます。

- 演算項目 1、演算項目 2、および結果のフィールドが同数の要素を含む配列 の場合に、演算には各配列の最初の要素が使用され、次に、各配列の 2 番目の要 素が使用され、同様に配列中のすべての要素が処理されます。 配列が同じ項目数 でない場合には、要素の数が最も少ない配列の最後の要素が処理された時点で、 演算は終了します。 ADD、SUB、MULT、および DIV 命令の場合に演算項目 1 が指定されないと、演算項 目 1 は結果のフィールドと同じであると見なされます。

- 演算項目のどちらか 1 つがフィールド、リテラル、または形象定数で、もう 1 つの演算項目と結果のフィールドが配列であった場合には、演算は、短い方の 配列の各要素について一度だけ実行されます。 すべての演算で、同じフィールド 、リテラル、または形象定数が使用されます。
- 結果のフィールドは、常に配列でなければなりません。
- 命令コードが演算項目 2 だけを使用し (たとえば、Z-ADD、Z-SUB、SQRT、 ADD、SUB、MULT、または DIV では演算項目 1 が 指定されないことがある)、結果のフィールドが配列の場合には、演算は配列のすべての要素に対して一度だ け実行されます。 演算項目 2 が配列でない場合には、すべての命令で同じフィールドまたは固定情 報が使用されます。
- 処理される演算の回数が多いため、結果の標識 (71 から 76 桁目) を使用するこ とはできません。
- EVAL 式において、右側に指標を用いない配列を指定した場合、 左側にも指標を用いない配列を含める必要があります。

注 : EVAL 命令の中で使用した場合に、%ADDR(配列) および %ADDR(配列 (*)) は同じ意 味を持ちません。 詳しくは、614 ページの『%ADDR (変数のアドレスの検索)』 を参照してください。

EVAL または SORTA 命令をコーディングするときには、組み込み関数 %SUBARR(arr) を使用して、 その命令で使用される配列の一部を選択することができます。 詳しくは、695 ページの『%SUBARR (配列の部分の設定/入手)』 を参照してください。

配列の分類

881 ページの『SORTA (配列の分類)』 命令コードを使用することにより、配列または配列のセクションをソートすることができます。 配列は、定義仕様書でそ の配列に指定された順序によって異なる順序 (昇順または降順) に分類されま す。 配列の順序を指定しないと、順序はデフォルトで昇順になりますが、 命令拡張 'D' を指定することによって、降順でソートできます。

配列の一部をキーとして使用した分類

OVERLAY キーワードを使用して、1 つの配列を別の配列にオーバーレイさせるこ とができます。 たとえば、氏名および給与が入れられる基本配列と 2 つのオーバーレイ配列 (1 つは氏名用で 1 つは給与用) を持つことができます。 次に、該当するオーバーレイ配列を分類することによって、基本配列を氏名かま たは給与のいずれかの別に分類することもできます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D                 DS
D Emp_Info                      50    DIM(500) ASCEND
D   Emp_Name                    45    OVERLAY(Emp_Info:1)
D   Emp_Salary                   9P 2 OVERLAY(Emp_Info:46)
D
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C
C* The following SORTA sorts Emp_Info by employee name.
C* The sequence of Emp_Name is used to determine the order of the
C* elements of Emp_Info.
C                   SORTA     Emp_Name
C* The following SORTA sorts Emp_Info by employee salary
C* The sequence of Emp_Salary is used to determine the order of the
C* elements of Emp_Info.
C                   SORTA     Emp_Salary
```

図 *86. OVERLAY* による *SORTA* 命令

配列データ構造のソート

SORTA 命令を使用して、配列データ構造のサブフィールドの 1 つをキーとして、配列データ構造をソートすることができます。

配列データ構造のソートについて詳しくは、881 ページの『SORTA (配列の分類)』 を参照してください。

配列の出力

ILE RPG の制御のもとでの配列全体の書き出しは、LR 標識がオンになっている場合に、プログラムの終わりでのみ行うことができます。 配列全体を書き出すことを指示するためには、定義仕様書の TOFILE キーワード によって出力ファイルの名前を指定してください。 このファイルは、ファイル仕様書で順次編成出力ファイルまたは入出力共用 ファイルとして記述されていなければなりません。 このファイルが入出力共用ファイルであって、物理ファイルとして外部記述され ている場合には、プログラム開始時に配列に読み込まれた情報は、プログラム終 了時の配列中の情報で置き換えられます。 論理ファイルでは、予測できない結果となることがあります。

配列全体を (出力仕様を使用して) 出力レコードに書き出す場合には、レコー ドの他のフィールドと一緒に配列を次のように記述してください。

- 出力仕様の 30 から 43 桁目には、定義仕様書で使用した配列名を入れなければ なりません。
- 出力仕様の 47 から 51 桁目には、配列の最後の要素が終わりとなるレコード位 置を入れなければなりません。編集コードが指定されている場合には、この終了位置にブランク桁および編集コードによるすべての拡張部分が組み込まれている必要があります (この章で次に示される「配列全体の編集」を参照)。

出力標識 (21 から 29 桁目) を指定することができます。 ゼロ抑制 (44 桁目)、後で消去 (45 桁目)、およびデータ形式 (52 桁目) の記 入項目は配列のすべての要素に適用されます。

配列全体の編集

配列全体に対して編集を指定した場合には、配列のすべての要素が編集されます。 各種の要素について異なる編集が必要な場合には、それらの要素を個々に参照し てください。

編集コードが配列全体に対して指定されている (44 桁目) 場合には、配列中の要 素の間に 2 つのブランクが自動的に挿入されます。すなわち、配列中の最初の要 素を除くすべての要素の左側はブランクになります。編集語が指定された場合に は、ブランクは挿入されません。 挿入されるブランクは、すべて編集語に含まれ ていなければなりません。

配列全体の編集は、出力仕様においてのみ有効です。%EDITC または %EDITW 組み込み関数の場合には有効ではありません。

動的にサイズ指定される配列の使用

ある配列で必要な要素の数が実行時までわからない場合には、 最大サイズでその配列を定義しておいて、その配列のサブセットをプログラムで使用することができます。

そのためには、1 つの命令で配列のすべての要素を操作する場合に、%SUBARR 組み込み関数を使用してどの要素を使用するのかを制御します。 また、%LOOKUP 組み込み関数を使用して、 配列の一部を検索することもできます。

```
 * Define the "names" array as large as you think it could grow
D names           S             25A   VARYING DIM(2000)
 * Define a variable to keep track of the number of valid elements
D numNames        S             10I 0 INZ(0) * Define another array
D temp            S             50A   DIM(20)
D p               S             10I 0
 /free
     // set 3 elements in the names array
     names(1) = 'Friendly';
     names(2) = 'Rusty';
     names(3) = 'Jerome';
     names(4) = 'Tom';
     names(5) = 'Jane';
     numNames = 5;

     // copy the current names to the temporary array
     // Note: %subarr could also be used for temp, but
     //       it would not affect the number of elements
     //       copied to temp
     temp = %subarr(names : 1 : numNames);

     // change one of the temporary values, and then copy
     // the changed part of the array back to the "names" array
     temp(3) = 'Jerry';
     temp(4) = 'Harry';     // The number of elements actually assigned will be the
     // minimum of the number of elements in any array or
     // subarray in the expression.  In this case, the
     // available sizes are 2 for the "names" sub-array,
     // and 18 for the "temp" subarray, from element 3
     // to the end of the array.
     %subarr(names : 3 : 2) = %subarr(temp : 3);
     // sort the "names" array
     sorta %subarr(names : 1 : numNames);

     // search the "names" array
     // Note: %SUBARR is not used with %LOOKUP.  Instead,
     //       the start element and number of elements
     //       are specified in the third and fourth
     //       parameters of %LOOKUP.
     p = %lookup('Jane' : names : 1 : numNames);
```

図 *87.* 動的にサイズ指定される配列の使用例

テーブル

次の相違点を除けば、配列についての説明はテーブルにも当てはまります。

アクティビティー
相違点

定義
テーブル名は、文字 TAB で始まる固有の記号名でなければなりません。

ロード中
テーブルは、コンパイル時および実行時前にだけロードすることができます。

テーブル要素の使用および変更
テーブルの要素は、一度に 1 つだけが活動状態になります。 活動中の要素を参照するために、テーブル名が使用されます。 テーブルに指標を指定することはできません。

検索
テーブルに対する LOOKUP 命令の指定は異なります。 さまざまな組み込み関数が、テーブルの検索に使用されます。

注 : テーブルをサブプロシージャーの中で定義することはできません。

テーブルの検索には、以下を使用することができます。

- LOOKUP 命令コード

- %TLOOKUP 組み込み関数
- %TLOOKUPLT 組み込み関数
- %TLOOKUPLE 組み込み関数
- %TLOOKUPGT 組み込み関数
- %TLOOKUPGE 組み込み関数

LOOKUP 命令コードの詳細については、以下の資料を参照してください。

- 245 ページの『1 つのテーブルでの LOOKUP』
- 245 ページの『2 つのテーブルでの LOOKUP』
- 798 ページの『LOOKUP (テーブルまたは配列要素の検索)』

%TLOOKUPxx 組み込み関数について詳しくは、702 ページの『%TLOOKUPxx (テーブル要素の検索)』 を参照してください。

1 つのテーブルでの LOOKUP

単一のテーブルを検索するためには、演算項目 1、演算項目 2、および少なくと も 1 つの結果の標識が指定されなければなりません。 条件付け標識 (7 から 11 桁目 に指定される) も使用することができます。

行われている検索のタイプ (等しい、高い、低い) を満たすテーブル要素が見付かった 場合にはいつでも、そのテーブル要素がテーブルの現行要素になります。 検索が正常に行われなかった場合には、前の現行要素がそのまま現行要素とし て残ります。

最初の正常な LOOKUP が行われるまでは、最初の要素が現行要素になります。

検索の結果は、結果の標識に反映されます。 この標識がオンの場合には、検索が 正常に行われたことを反映し、検索条件を満たした要素が現行要素になります。

2 つのテーブルでの LOOKUP

2 つのテーブルが検索で使用された場合に、実際に検索されるのは 1 つだけです。 検索条件 (高い、低い、等しい) が満たされた時には、対応する要素が使用可能になりま す。

演算項目 1 には検索引数が含まれ、演算項目 2 には検索するテーブルの名前が含 まれていなければなりません。 結果のフィールドは、データも使用可能になっているテーブル の名前を示していなければなりません。 結果の標識も使用する必要があります。 必要な場合には、制御レベルおよび条件付け標識を 7 から 11 桁目に指定すること ができます。

使用される 2 つのテーブルは、項目数が同じでなければなりません。 検索される テーブルに 2 番目のテーブルより多くの要素が含まれている場合には、検索条件 を満たすことは可能です。 しかし、2 番目のテーブルには、検索テーブルで見付 かった要素と対応する要素はない場合があります。 望ましくない結果が起こることがあります。

注 : 正常な LOOKUP が行われる前に LOOKUP 以外の命令にテーブル名を指定し た場合には、テーブルはその最初の要素に設定されます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C* The LOOKUP operation searches TABEMP for an entry that is equal to
C* the contents of the field named EMPNUM. If an equal entry is
C* found in TABEMP, indicator 09 is set on, and the TABEMP entry and
C* its related entry in TABPAY are made the current elements.
C     EMPNUM        LOOKUP    TABEMP        TABPAY                 09
C* If indicator 09 is set on, the contents of the field named
C* HRSWKD are multiplied by the value of the current element of
C* TABPAY.
C                   IF        *IN09
C     HRSWKD        MULT(H)   TABPAY        AMT             6 2
C                   ENDIF
```

図 *88.* 等しい項目の検索

LOOKUP 命令で見付かったテーブル要素の指定

テーブル名が LOOKUP 以外の命令で使用された場合にはいつでも、テーブル名は 、実際には正常に行われた最後の検索によって取り出されたデータを参照する ことになります。 したがって、テーブル名をこの方法で指定する場合には、テー ブルからの要素を演算命令で使用することができます。

LOOKUP 命令で演算項目 1 としてテーブル名を使用する場合には、現行要素が 検索引数として使用されます。 この方法では、テーブルからの要素それ自体を検索引数とすることができます。

テーブルは、LOOKUP 命令以外の命令で結果のフィールドとして使用することもで きます。 この場合には、現行要素の値は演算仕様書によって変更されます。この方法では、テーブルの内容を演算命令によって変更することができます (246 ページの図 89 を参照)。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C     ARGMNT        LOOKUP    TABLEA                                 20
C* If element is found multiply by 1.5
C* If the contents of the entire table before the MULT operation
C* were 1323.5, -7.8, and 113.4  and the value of ARGMNT is -7.8,
C* then the second element is the current element.
C* After the MULT operation, the entire table now has the
C* following value:  1323.5, -11.7, and 113.4.
C* Note that only the second element has changed since that was
C* the current element, set by the LOOKUP.
C                   IF        *IN20
C     TABLEA        MULT      1.5           TABLEA
C                   ENDIF
```

図 *89. LOOKUP* 命令で見付かったテーブル要素の指定

データ・タイプおよびデータ形式

この章では、RPG IV がサポートするデータ・タイプおよびその特別な特性について説明しています。 サポートされているデータ・タイプは次のとおりです。

- 280 ページの『基底ポインター・データ・タイプ』
- 285 ページの『プロシージャー・ポインター・データ・タイプ』

さらに、一部のデータ・タイプでは、異なるデータ形式を使用することができます。 この章は、内部データ形式と外部データ形式の相違点、形式、およびそれらの指 定方法について説明しています。

内部形式および外部形式

数値フィールド、文字フィールド、日付フィールド、およびタイム・スタンプ・フィールドには、外部形式とは 別に内部形式があります。 **内部形式**とは、データがプログラムに記憶される方法のことです。 **外部形式**とは、データがファイルに記憶される方法のことです。

内部形式は、次の場合に知っておく必要があります。

- 参照によるパラメーターの受け渡し
- データ構造内でのサブフィールドのオーバーレイ

さらに、演算操作の実行時パフォーマンスが重要である場合には、数値フィール ドの内部形式を考慮することが必要なこともあります。 詳しくは、「558 ページの『パフォーマンスに関する考慮事項』」を参照してください。

数値および日付/時刻データ・タイプについては、デフォルトの値の内部形式および外 部形式があります。 内部形式は、定義仕様書の特定のフィールドに指定することができます。 同様に、外部形式は、対応する入力または出力仕様のプログラム記述フィールド に指定することができます。

外部記述ファイルのフィールドの場合には、外部データ形式はデータ記述仕様書 の 35 桁目に指定されます。 1 つの例外を除き、外部記述フィールドの外部形式を 変更することはできません。 制御仕様書に EXTBININT を指定した場合には、小数点 以下の桁数がないすべての 2 進フィールドは整数の外部形式を持つものとして 取り扱われます。

外部記述データ構造のサブフィールドの場合には、外部記述に指定されたデータ 形式が、コンパイラーによってそのサブフィールドの内部形式として使用されま す。

内部形式

数値独立フィールドのデフォルトの内部形式は、パック 10 進数です。 数値データ構造サブフィールドのデフォルトの内部形式は、ゾーン 10 進です。 異なる内部形式を指定するためには、そのフィールドまたはサブフィールド用の 定義仕様書の 40 桁目に所要の形式を指定してください。

日付フィールド、時刻フィールド、およびタイム・スタンプ・フィールドのデフォルト形式は *ISO です。 一般に、とくに外部形式タイプが混合の場合には、デフォルトの ISO 内部形 式の使用をお奨めします。

日付フィールド、時刻フィールド、およびタイム・スタンプ・フィールドの場合、制御仕様書で DATFMT および TIMFMT キーワード を使用して、必要であればプログラム中のすべての 日付/時刻フィールドについて、デフォルトの内部形式を変更することができます。 DATFMT または TIMFMT キーワードを定義仕様書で使用すれば、個別の 日付/時刻フィールドのデフォルトの内部形式を一時変更することができます。

外部形式

プログラム記述ファイルに数値フィールド、文字フィールド、または日時フィールドがある場合には、その外部形式を指定できます。

外部形式は、フィールドの処理方法には影響 しません。 しかし、指定された内部形式によっては、演算操作のパフォーマンスを向上させ ることができます。 詳しくは、「558 ページの『パフォーマンスに関する考慮事項』」を参照してください。

次の表は、プログラム記述フィールドの外部形式の指定方法を示して います。 各形式タイプの詳細については、この章の以下の該当する項を参照してください。

表 69. 外部形式を指定する記入項目および位置

フィールドのタイプ	仕様書	使用する位置
入力	入力	36 桁目

表 69. 外部形式を指定する記入項目および位置 (続き)		
フィールドのタイプ	**仕様書**	**使用する位置**
出力	出力	52 桁目
配列またはテーブル	定義	EXTFMT キーワード

数値フィールドの場合の外部形式の指定

247 ページの表 69 のどのフィールドについても、次の有効な外部数値形式の 1 つを指定してください。

B
Binary

F
float

I
整数

L
先行符号

P
パック 10 進数

R
後書き符号

S
ゾーン 10 進数

U
符号なし

浮動数値データのデフォルトの外部形式は、外部表示表現と呼ばれます。 4 バイト浮動データの形式は次のとおりです。

```
   +n.nnnnnnnE+ee,
where + represents the sign (+ or -)
      n represents digits in the mantissa
      e represents digits in the exponent
```

8 バイト浮動データの形式は次のとおりです。

```
   +n.nnnnnnnnnnnnnnnE+eee
```

4 バイト浮動値は 14 桁を占め、8 バイト浮動値は 23 桁を占める ことを覚えておいてください。

浮動以外の数値データの場合、デフォルトの外部形式はゾーン 10 進です。 コンパイル時配列およびテーブルの外部形式は、ゾーン 10 進、先行符号、お よび後書き符号形式でなければなりません。

浮動コンパイル時配列およびテーブルの場合、コンパイル時データ は、数値リテラルまたは浮動リテラルとして指定されます。 4 バイト浮動配列の各要素のソース・レコード内には 14 桁必要です。 また、8 バイト浮動配列の各要素の場合は 23 桁必要です。

入力仕様、 演算仕様書、または出力仕様に定義され、定義仕様書に対応する 定義のない非浮動数値フィールドは、パック 10 進形式で内部的に 保管されます。

文字フィールド、図形フィールド、または *UCS-2* フィールドの場合の外部形式の指定

247 ページの表 69 の入力フィールドおよび出力フィールドには、次の有効外部データ形式を指定してください。

A
文字 (文字および標識データの場合に有効)

N
標識 (文字および標識データの場合に有効)

G
図形 (図形データの場合に有効)

C
UCS-2 (UCS-2 データの場合に有効)

EXTFMT キーワードを使用すると、UCS-2 形式で配列またはテーブルにデータを 指定できます。

可変長文字、図形、または UCS-2 データの場合、入力仕様では 31 から 34 桁目 に、出力仕様では 53 から 80 桁目に *VAR データ属性を指定します。

日付/時刻フィールドの場合の外部形式の指定

プログラム記述ファイル内に、日付フィールド、時刻フィールド、およびタイム・スタンプ・フィールドがある場合、フィールドの外部形式を指定しなければなりません。 ファイル記述仕様 で DATFMT キーワードおよび TIMFMT キーワードを使用することによって、プログラム記述ファイル内の すべての日付フィールド、時刻フィールド、およびタイム・スタンプ・フィールドに対してデフォルトの外部形式を指定できます。 特定のフィールドについても、外部形式を指定することができます。 入力仕様の 31 から 34 桁目に所要の形式を指定してください。 出力仕様の 53 から 80 桁目 に、適切なキーワードおよび形式を指定してください。

各形式タイプの詳細については、この章の以下の該当する項を参照してください。

文字データ・タイプ

文字データ・タイプは文字値を表し、次のいずれかの形式になります。

A
文字 (「英数字」とも呼ばれる)

N
標識

G
グラフィック

C
UCS-2

文字データには、指定された形式に従って、1 つまたは複数 の 1 バイトまたは 2 バイト文字が含まれています。 文字フィールド、図形フィールド、および UCS-2 フィールドも、固定長、可変長の どちらの形式にもすることができます。 次の表は、いろいろな文字データ・タイプ形式を要約したものです。

警告 : UTF-8、UTF-16、または SBCS/DBCS が混在する CCSID などの一部の CCSID では、文字数は 1 バイトまたは 2 バイトの数よりも少なくなることがあります。 例えば、CCSID(*UTF8) で定義された可変長の英数字変数の現行値が 2 文字であり、一方の文字が 3 バイトで、もう一方の文字が 2 バイトである場合、値の長さは 2 ではなく 5 と見なされます。

%SUBST 命令コード、または %SUBST、%LOWER、および %UPPER 組み込み関数などのストリング操作の開始位置および長さオペランドは、英数字データのバイト単位で測定され、グラフィック・データおよび UCS-2 データの場合は、2 バイト単位で測定されます。 開始位置および長さのオペランドによって表されるサブストリングが原因でサブストリングの最初または最後の文字が分割されることのないようにすることは、RPG プログラマーの責任です。 例えば、UTF-8 ストリング「abcdó」は 5 文字ですが、最後の文字は 2 バイトであるため、バイト数は 6 バイトです。 %SUBST(string:1:5) は、最後の文字「ó」が完全ではないため無効です。

異なるデータ・タイプまたは CCSID とデータを比較するときに生じる予期しない結果

警告 : Unicode または ASCII での比較は EBCDIC での比較とは異なります。 例えば、Unicode では「1」は「A」より小さいですが、EBCDIC では大きくなります。 比較するデータの CCSID を制御するには、%CHAR または %UCS2 組み込み関数を使用して、オペランドのタイプを制御します。

不要な CCSID 変換が行われているかどうかを検査するには、制御ステートメントで CCSIDCVT(*LIST) を指定し、リストの CCSIDCVT セクションを使用して、どの CCSID 変換がモジュールで実行されているかを確認できます。

例えば、以下のプログラムの場合:

1. *fld_ebcdic* はジョブ CCSID を持つ文字フィールドです。これは常に EBCDIC CCSID です。
2. *fld_ucs2* は UCS-2 フィールドです。
3. *fld_ebcdic* と「A」の比較に CCSID の変換は必要ありません。
4. DSPLY メッセージには「A」が「0」より小さいことが示されています。
5. *fld_ucs2* と「A」の比較には CCSID の変換が必要です。 第 2 オペランド「A」は *fld_ucs2* の CCSID に変換されるため、比較は UCS-2 データを使用して行われます。
6. DSPLY メッセージには「A」が「0」より大きいことが示されています。

```
CTL-OPT CCSIDCVT(*LIST);

DCL-S fld_ebcdic CHAR(10) INZ('A');   // 1
DCL-S fld_ucs2 UCS2(10) INZ('A');     // 2

IF fld_ebcdic < '0';                  // 3
   DSPLY '1: "A" < "0"';              // 4
ELSE;
   DSPLY '1: "A" > "0"';
ENDIF;

IF fld_ucs2 < '0';                    // 5
   DSPLY '2: "A" < "0"';
ELSE;
   DSPLY '2: "A" > "0"';              // 6
ENDIF;

return;
```

文字データのタイプ

文字データ・タイプ	パック 10 進数フィールド	CCSID
文字	固定長または可変長の、1 つまたは複数 の **1 バイト**文字	文字形式のデータの CCSID については、251 ページの『文字形式データの CCSID』 を参照してください。
標識	固定長の 1 つの **1 バイト**文字	標識データの CCSID については、252 ページの『標識形式のデータの CCSID』 を参照してください。
ｸﾞﾗﾌｨｯｸ	固定長または可変長の、1 つまたは複数 の **2 バイト**文字	65535 (*HEX) または EBCDIC 2 バイト・コード化体系 (x'1200') の CCSID
UCS-2	固定長または可変長の、1 つまたは複数 の **2 バイト**文字	13488 または UCS-2 エンコード・スキーム (X'7200') の CCSID (例えば、1200 (*UTF16))

文字データの CCSID については、261 ページの『文字データ、図形データおよび UCS-2 データの間の変換』 を参照してください。

文字形式

固定長文字形式は、1 バイトまたはそれ以上の 固定した長さを持ちます。

可変長文字形式については、253 ページの『可変長の文字形式、図形形式および UCS-2 形式』を参照してください。

文字フィールドを定義するには、 自由形式定義に CHAR または VARCHAR キーワードを指定するか、 または、固定形式仕様書のデータ・タイプ記入項目に A を指定 します。 パラメーターが文字フィールドである場合には、 定義仕様書の LIKE キーワードを使用して文字フィールドを定義することもできます。

デフォルトの初期化値はブランクです。 ブランクは、EBCDIC データの場合は x'40'、ASCII または UTF-8 データの場合は x'20' です。

文字フィールドのデフォルト CCSID は、 制御ステートメントで CCSID(*CHAR) キーワードを 使用するか、または /SET 指示を使用して指定できます。 定義ステートメント CCSID キーワードを使用して明示的に CCSID を指定することもできます。

文字形式データの CCSID

以下のうち少なくとも 1 つが真の場合、文字データには明示的または暗黙的に指定された CCSID があると見なされます。

- 制御ステートメントに CCSID(*EXACT) が指定されている。
- 制御ステートメントに *JOBRUN 以外の CCSID で CCSID(*CHAR) が指定されている。
- 有効である /SET ステートメントに CCSID(*CHAR) が指定されている。
- 文字サブフィールドが含まれているデータ構造に対して CCSID(*EXACT) または CCSID(*NOEXACT) が指定されている。
- 文字項目の定義に CCSID キーワードが指定されている。

文字データには明示的または暗黙的に指定された CCSID がないと見なされる場合、次のようになります。

- CCSID(*CHAR:*JOBRUN) が有効である場合、データの CCSID はジョブ CCSID です。
- それ以外の場合、CCSID は、ジョブ CCSID に関連した混合グラフィック CCSID であると想定されます。

警告 : CCSID(*CHAR:*JOBRUN) が指定されていて、 制御ステートメントのキーワード CCSID(*EXACT) が指定されていない場合、 明示的に指定された CCSID を持たない英数字フィールドは、 RPG コンパイラーが入力バッファーまたは出力バッファーとプログラム・フィールド との間で CCSID 変換を実行するかどうかを判断するときに、既知の CCSID を持っていないと 見なされます。 詳しくは、262 ページの『入出力命令中の CCSID 変換』を参照してください。

文字データの CCSID が 65535 (*HEX) の場合、このデータが別の CCSID の文字データと一緒に使用されるときには CCSID 変換は実行されず、 このデータがグラフィック・データまたは UCS-2 データと一緒に使用されるときには CCSID 変換は許可されません。 以下の場合は、文字項目の CCSID は 65535 です。

- 項目が 16 進リテラルである。
- CCSID(*HEX) または CCSID(65535) が、項目の定義にある CCSID キーワードで明示的または暗黙的に 指定されている。
- デフォルト文字 CCSID が 65535 である。デフォルト文字 CCSID の設定方法については、324 ページの『CCSID 制御キーワード』を参照してください。
- 項目が、CCSID オペランドに *HEX が指定された %CHAR 組み込み関数の結果である。
- 項目が、外部記述データ構造または LIKEREC キーワードを使用して定義された データ構造内にあるサブフィールドであり、データ構造の定義に CCSID(*EXACT) が指定されていて、外部フィールドのタイプが 16 進である。

標識形式

標識形式は特殊なタイプの文字データです。 標識はすべてが 1 バイトの長さで、 文字値 '0' (オフ) および '1' (オン) だけを入れることができます。 これは、一般に、命令の結果を示すか、あるいは命令の処理を 条件付け (制御) するために使用されます。 標識のデフォルトの値は '0' です。

標識フィールドを定義するには、 自由形式定義で IND キーワードを指定するか、または、 固定形式仕様書のデータ・タイプ記入項目に N を 指定します。 パラメーターが標識フィールドである場合には、 定義仕様

書の LIKE キーワードを使用して標識フィールドを定義することもできます。 標識フィールドはファイル仕様書の COMMIT キーワードによって暗黙的にも 定義されます。

定義済み RPG IV 標識 (*INxx) の特殊なセットも使用することができます。これらの標識の説明については、118 ページの『RPG IV 標識』を参照してください。

標識変数の定義に関する規則は次のとおりです。

- 標識は独立フィールド、サブフィールド、プロトタイプ・パラメーター、および プロシージャーの戻り値として定義することができます。
- 標識変数が実行時前またはコンパイル時の配列またはテーブルとして定義されて いる場合、初期化データは '0' と '1' のみから構成されていなければなりません。

  注 : 実行時に標識に '0' または '1' 以外の値が含まれている場合、 結果は予測不可能です。
- キーワード INZ が指定されている場合、値は '0'、*OFF、'1'、 または *ON のいずれかでなければなりません。
- キーワード VARYING は標識フィールドに指定することはできません。

標識変数の使用に関する規則は次のとおりです。

- 標識フィールドのデフォルトの初期化値は '0' です。
- 標識変数は、命令コード CLEAR によって '0' に設定されます。
- 標識変数は、「後で消去」機能が適用されると、'0' に設定されます。
- 標識の配列が MOVEA(P) 命令の結果として指定されている場合、 埋め込み文字は '0' になります。
- 標識は ALTSEQ(*NONE) によって暗黙的に定義されます。 これは、標識を含む 比較に代替照合順序が使用されないことを意味します。
- 外部キーが長さが 1 の文字である場合、標識をキー・フィールドとして 使用することができます。

標識形式のデータの CCSID

標識は CCSID(*JOBRUN) であると見なされます。

グラフィック形式

グラフィック形式は、各文字が 2 バイトで表される文字ストリング です (すべての文字は特定の 2 バイト文字セットの一部です)。

注 : 同様に 2 バイト文字を使用する UCS-2 (Unicode) 形式については、253 ページの『UCS-2 形式』を参照してください。

図形データとして定義されるフィールドには、シフトアウト (SO) またはシフトイン (SI) 文字は含まれません。 単一バイト文字と 2 バイトの図形データ の相違点を次の図に示します。

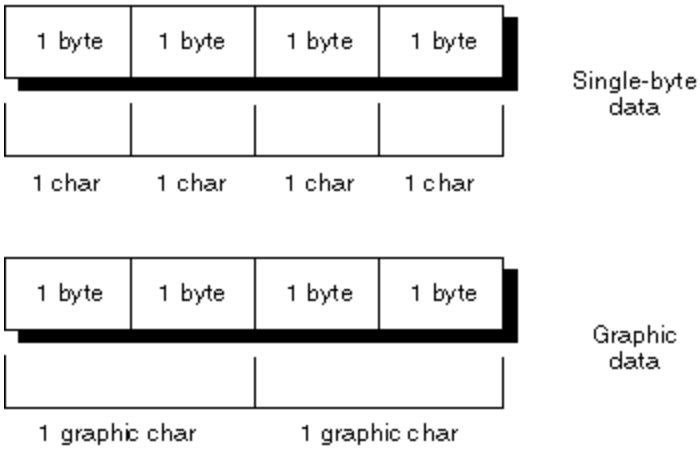


図 *90. SBCS* データと図形データの比較

図形フィールドの長さ (バイト数) はフィールド中の図形文字数の 2 倍です。

固定長の図形形式は、各文字が 2 バイトで表される、長さが一定の 文字ストリングです。

可変長グラフィック形式について詳しくは、253 ページの『可変長の文字形式、図形形式および UCS-2 形式』を参照してください。

図形フィールドを定義するには、 自由形式定義で GRAPH または VARGRAPH キーワードを指定するか、または、 固定形式仕様書のデータ・タイプ記入項目に G を指定します。 パラメーターが図形フィールドである場合には、 定義仕様書の LIKE キーワードを使用して図形フィールドを定義することもできます。

図形フィールドのデフォルト CCSID は、 制御ステートメントで CCSID(*GRAPH) キーワードを 使用するか、または /SET 指示を使用して指定できます。 定義ステートメント CCSID キーワードを使用して明示的に CCSID を指定することもできます。

注 : CCSID(*GRAPH:*IGNORE) が有効になっている場合、グラフィック定義に CCSID キーワードを明示的に 指定することはできません。 CCSID(*GRAPH:*IGNORE) は、以下のいずれかが真の場合、有効ではありません。

- 制御ステートメントにキーワード CCSID(*EXACT) が指定されている。
- 制御ステートメントにキーワード CCSID(*GRAPH) が *IGNORE 以外の値と一緒に指定されている。

図形データのデフォルトの初期化値は X'4040' です。 *HIVAL の値 は X'FFFF' であり、*LOVAL の値は X'0000' です。

注 : 本マニュアルの GRAPHIC リテラルの例は、無効な GRAPHIC リテラルです。 これらの例では、文字「o」が、シフトアウト文字を示し、文字「i」はシフトイン文字を示しています。 多くの場合、グラフィック・データは、D1D2 または AABB として表示さ れています。これらは無効な 2 バイト文字です。 通常 GRAPHIC リテラルは、DBCS 文字が入力される前および入力された後 に、自動的にシフトアウトおよびシフトイン文字を入力する DBCS 対応キーボードを使用して入力します。

UCS-2 形式

汎用文字セット (UCS-2) 形式は、各文字が 2 バイトで表される 文字ストリングです。 この文字セットは、多数の書き込み言語の文字を コーディングできます。

UCS-2 データとして定義されるフィールドには、シフトアウト (SO) または シフトイン (SI) 文字は含まれません。

UCS-2 フィールドの長さ (バイト数) はフィールド中の UCS-2 文字数の 2 倍です。

固定長の UCS-2 形式は、各文字が 2 バイトで表される、長さが設定された 文字ストリングです。

可変長 UCS-2 形式について詳しくは、253 ページの『可変長の文字形式、図形形式および UCS-2 形式』を参照してください。

UCS-2 フィールドを定義するには、 自由形式定義に UCS2 または VARUCS2 キーワードを指定するか、 または、固定形式仕様書のデータ・タイプ記入項目に C を指定します。 パラメーターが UCS-2 フィールドである場合 には、定義仕様書の LIKE キーワードを使用して UCS-2 フィールドを定義することもできます。

UCS-2 データのデフォルトの初期化値は X'0020' です。 *HIVAL の 値は X'FFFF'、*LOVAL は X'0000'、そして *BLANKS の値は X'0020' です。 文字、UCS-2 値、または図形値を使用して、UCS-2 フィールドの初期値を指定することができます。 リテラルのタイプが UCS-2 ではない場合は、コンパイラーによって暗黙的に UCS-2 へ変換されます。 例えば、UCS-2 フィールドを 'abc' という UCS-2 形式で初期化するには、INZ('abc')、INZ(%UCS2('abc'))、または INZ(U'006100620063') と指定してください。

UCS-2 形式について詳しくは、IBM i Information Center のトピック「グローバリゼーション」を参照してください。

可変長の文字形式、図形形式および UCS-2 形式

可変長文字フィールドには、宣言された最大長と、プログラムの実行時に変化する 現在の長さがあります。この長さは、文字形式の場合は 1 バイト単位で、図形形式および UCS-2 形式の場合は 2 バイト単位で測定されます。 可変長文字フィールドに割り振られる記憶域 は、そのフィールドに対して、VARYING、VARCHAR、VARGRAPH、または VARUCS2 キーワードがどのように 指定されているのかに基づいて、宣言

された最大長よりも 2 バイトまたは 4 バイト長くなります。 左端の 2 バイトまたは 4 バイトは符号なし整数フィールドで、文字、図形文字、または UCS-2 文字で現在の長さを表したものが組み込まれます。 実際のデータは、可変長フィールドの 3 番目または 5 番目のバイトから開始します。254 ページの図 91 は可変長文字フィールドの保管方法を示しています。

```
-----------------------------------
   | current  | character data |
   | length   |                |
-----------------------------------
     UNS(V)       CHAR(N)

 N = declared maximum length
 V = number of bytes specified for the length prefix

 V + N = total number of bytes
```

図 *91.* 可変長形式の 文字フィールド

符号なし整数の長さ接頭部は、2 バイトまたは 4 バイトの長さになります。 接頭部のサイズを指定するには、自由形式定義の VARCHAR、VARGRAPH、または VARUCS2 キーワード の 2 番目のパラメーターに 2 または 4 を指定するか、 固定形式仕様書の VARYING キーワードのパラメーターを 使用して VARYING(2) または VARYING(4) のように指定します。 VARCHAR、VARGRAPH、または VARUCS2 を 2 番目のパラメーターなしで指定するか、または、VARYING を パラメーターなしで指定すると、指定された長さが 1 から 65535 の範囲内の場合はサイズ 2 が 想定され、それ以外の場合はサイズ 4 が想定されます。

254 ページの図 92 は、可変長グラフィック・フィールドの保管方法を示しています。 UCS-2 フィールドも同様に保管 されます。

```
-----------------------------------
   | current  | graphic-data   |
   | length   |                |
-----------------------------------
     UNS(V)       CHAR(N)

 N = declared maximum length = number of double bytes
 V = number of bytes specified for the length prefix

 V + 2(N) = total number of bytes
```

図 *92.* 可変長形式の図形フィールド

注 : 現在の長さまで (現在の長さを含む) のデータのみが有効になります。

可変長フィールドを定義するには、自由形式定義に VARCHAR、VARGRAPH、または VARUCS2 キーワードを指定するか、 または、固定形式定義仕様書に A (文字)、 G (グラフィック)、または C (UCS-2) と、キーワード VARYING を 指定します。 また、パラメーターが可変長フィールドである場合には、 定義仕様書の LIKE キーワードを使用して定義することもできます。

外部可変長フィールドを参照するには、入力仕様 または出力仕様で、*VAR データ属性を 使用します。

可変長フィールドは、デフォルトで、ゼロの現在の長さを持つように 初期化されます。

可変長フィールドのデータ部のアドレスは、%ADDR(フィールド名:*DATA) を使用して取得することができます。

可変長フィールドの使用例については、以下の項を参照してください。

- 257 ページの『可変長フィールドの使用』
- 653 ページの『%LEN (長さの入手または設定)』
- 620 ページの『%CHAR (文字データへの変換)』
- 679 ページの『%REPLACE (文字ストリングの置換)』
- 614 ページの『%ADDR (変数のアドレスの検索)』

可変長項目の長さ接頭部のサイズ

可変長項目には、2 バイトまたは 4 バイトの接頭部が付けられ、 ここに項目の現在の長さが保管されます。接頭部のサイズは、VARCHAR、VARGRAPH、または VARUCS2 キーワード の 2 番目のパラメーターか、

VARYING キーワードの最初のパラメーターで指定されます。 パラメーターは 2 または 4 である必要があります。接頭部サイズが指定されない場合、指定された長さが 1 から 65535 までの場合はサイズ 2 が想定され、それ以外の場合はサイズ 4 が想定されます。 長さが 1 から 65535 までの定義に対しては、 任意の接頭部サイズを指定できます。 長さが 65535 を超える定義では、長さを格納するために 4 バイトが必要なため、 接頭部サイズ 2 を指定することはできません。

詳しくは、「253 ページの『可変長の文字形式、図形形式および UCS-2 形式』」を参照してください。

可変長の文字形式、図形形式、および *UCS-2* 形式に関する規則

可変長フィールドを定義する場合、次の規則が適用されます。

- 宣言するフィールド長は、1 バイト文字であれば 1 から 16773100 文字まで、2 バイト図形文字または UCS-2 文字であれば 1 から 8386550 文字までになります。
- 現在の長さは、0 からフィールドについて宣言した最大長までの任意の値を 定義することができます。
- フィールドはキーワード INZ を使用して初期化することができます。 初期値は指定した値になり、フィールドの初期長は、初期値の長さになります。 フィールドは初期化時にブランクで埋め込まれますが、ブランクは長さに含まれません。
- 長さ接頭部のサイズが異なる可変長フィールドは、参照パラメーターとして渡される場合を除き、完全互換です。
- プロトタイプ・パラメーターが可変長 として (VARYING、VARCHAR、VARGRAPH、または VARUCS2 キーワードを使用して) 定義されていて、 CONST キーワードも VALUE キーワードも指定されていない場合、渡されるパラメーターの長さ接頭部は、 プロトタイプ・パラメーターと同じサイズでなければなりません。 この規則は、OPTIONS(*VARSIZE) が指定されている場合でも適用されます。
- 位取り表記法を使用して定義されたサブフィールドを除くすべての場合において、フィールドの最大長 (文字単位) が長さ (LEN キーワード、または定義仕様書の 33 から 39 桁目の長さ記入項目によって指定) に含まれます。 ただし、この最大長には、2 バイトまたは 4 バイトの長さ接頭部は含まれません。
- 位取り表記法を使用して定義されたサブフィールドの場合、開始位置と終了位置を指定して定義されたサイズは、2 バイトまたは 4 バイトの長さ接頭部を含みます。 したがって、位取り表記法を使用して指定されたバイト数は、データを保持するために必要なバイト数よりも、2 バイトまたは 4 バイト長くなります。 VARYING(2) を指定した場合、データに必要なバイト数に 2 バイトを加算し、VARYING(4) を指定した場合には 4 バイトを加算します。 パラメーターなしで VARYING を指定した場合には、長さが 65535 以下であれば 2 バイトを加算し、長さが 65535 を超えていれば 4 バイトを加算します。 英数字のサブフィールドでは、サイズが 3 から 65537 であれば、長さが 1 から 65535 であることを表し、 UCS-2 および図形のサブフィールドでは、サイズが 5 から 131072 であれば、長さが 1 から 65535 であることを表します。

  注 : もっと簡単に可変長 サブフィールドを指定する方法は、長さ表記を使用し、さらに、自由形式定義で POS キーワードを使用するか、または固定形式定義 で OVERLAY キーワードを指定することによって、データ構造内でのサブフィールドの位置を 指定することです。
- キーワード VARYING はデータ構造に指定することはできません。
- 可変長の実行時前配列の場合、ファイル内の初期化データは、長さ接頭部を組み込まれて可変長形式で保管されます。
- 実行時前配列データはファイルから読み取られ、ファイルの最大レコード長は 32766 なので、可変長の実行時前配列の最大サイズは 32764 個の 1 バイト文字、または 16382 個の 2 バイト図形文字または UCS-2 文字になります。
- 可変長の配列またはテーブルは、コンパイル時データによって定義する ことができます。 データのフィールド内の後書きブランクは意味がありません。 データの長さは、 フィールド内のブランクでない最後の文字の位置になります。 コンパイル時データには長さ接頭部が保管されないので、 これは実行時前初期化とは異なります。
- *LIKE DEFINE は可変長フィールドのようなフィールドを定義するためには使用できません。

以下に示すのは、可変長文字フィールドの定義例です。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++++++
 * Standalone fields:
D var5            S              5A   VARYING
D var10           S             10A   VARYING INZ('0123456789')
D largefld_a      S          32767A   VARYING
D max_len_a       S              A   VARYING LEN(16773100)
      DCL-S free_form_10A VARCHAR(10);
 * Prerun-time array:
D arr1            S            100A   VARYING FROMFILE(dataf)
 * Data structure subfields:
D ds1             DS
 *   Subfield defined with length notation:
D   sf1_5                       5A   VARYING
D   sf2_10                     10A   VARYING INZ('0123456789')
 *   Subfield defined using positional notation: A(5)VAR
D   sf4_5              101     107A   VARYING
 *   Subfields showing internal representation of varying:
D   sf7_25                    100A   VARYING
D   sf7_len                     5I 0 OVERLAY(sf7_25:1)
D   sf7_data                  100A   OVERLAY(sf7_25:3)
 *   Free-form varying subfields:
    sf8_10 VARCHAR(10);
    sf9_13 VARCHAR(13 : 4);
 * Procedure prototype
D Replace         PR         32765A   VARYING
D   String                   32765A   CONST VARYING OPTIONS(*VARSIZE)
D   FromStr                  32765A   CONST VARYING OPTIONS(*VARSIZE)
D   ToStr                    32765A   CONST VARYING OPTIONS(*VARSIZE)
D   StartPos                     5U 0 VALUE
D   Replaced                     5U 0 OPTIONS(*OMIT)
```

図 *93.* 可変長文字および *UCS-2* フィールドの定義

以下に示すのは、可変長図形フィールドおよび UCS-2 フィールドの定義例です。

```
* .. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+...
DName++++++++++ETDsFrom+++To/L+++IDc.Functions+++++++++++++++++
 *---------------------------------------------------------------
 * Graphic fields
 *---------------------------------------------------------------
 * Standalone fields:
D GRA20           S             20G   VARYING
D MAX_LEN_G       S        8386550G   VARYING
      DCL-S free_form_10G VARGRAPH(10);
 * Prerun-time array:
D ARR1            S            100G   VARYING FROMFILE(DATAF)
 * Data structure subfields:
D DS1             DS
 *   Subfield defined with length notation:
D   SF3_20                     20G   VARYING
 *   Subfield defined using positional notation: G(10)VAR
D   SF6_10              11     32G   VARYING
 *---------------------------------------------------------------
 *   UCS-2 fields
 *---------------------------------------------------------------
D MAX_LEN_C       S        8386550C  VARYING
      DCL-S free_form_10C VARUCS2(10);
D FLD1            S              5C   INZ(%UCS2('ABCDE')) VARYING
D FLD2            S              2C   INZ(U'01230123') VARYING
D FLD3            S              2C   INZ(*HIVAL) VARYING
D DS_C            DS
D   SF3_20_C                   20C   VARYING
 *   Subfield defined using positional notation: C(10)VAR
D   SF_110_C            11     32C   VARYING
```

図 *94.* 可変長図形フィールドおよび *UCS-2* フィールドの定義

可変長フィールドの使用

可変長フィールドの長さの部分は、文字で測定されたフィールドの現在の長さを 表します。 文字フィールドの場合には、この長さはバイト単位での現在の長さも表します。 2 バイト・フィールド (図形および UCS-2) の場合、これは、2 バイト単位での フィールドの長さを表します。 たとえば、現在の長さが 3 である UCS-2 フィールドの長さは、3 個の 2 バイト文字 で、6 バイトということになります。

次のセクションでは、可変長フィールドを最大限活用する方法 と、異なる命令コードを使用するときに現在の長さを変更する方法について説明 します。

フィールドの長さの設定方法

可変長フィールドが INZ を使用して初期化されるとき、初期長は 初期化値の長さに設定されます。 たとえば、長さ 10 の文字フィールドが値 'ABC' に初期化される と、初期長は 3 に設定されます。

EVAL 命令では、可変長ターゲットの長さが変更されます。 たとえば、長さ 10 の文字フィールドに値 'XY' が割り当てられる と、長さは 2 に設定されます。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++++
D fld                     10A        VARYING
 * It does not matter what length 'fld' has before the
 * EVAL; after the EVAL, the length will be 2.
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq...
C                   EVAL      fld = 'XY'
```

DSPLY 命令では、可変長結果フィールドの長さが、ユーザーによって入力された 値の長さに変更されます。 たとえば、結果フィールドが長さ 10 の文字フィールドであり、ユーザーに よって入力された値が '12345' である場合、このフィールドの長さは、DSPLY 命令 によって 5 に設定されます。

CLEAR 命令では、可変長フィールドの長さが 0 に変更されます。

PARM 命令では、結果フィールドの長さが、演算項目 2 (指定されている場合) の中の フィールドの長さに設定されます。

固定形式命令の MOVE、MOVEL、CAT、SUBST および XLATE は、可変長結果フィールド の長さを変更しません。 たとえば、値 'XYZ' が、MOVE を使用して、現在の長さが 2 である長さ 10 の 可変長文字フィールドに移動された場合、このフィールドの長さ は変更されず、データが切り捨てられます。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++++
D fld                     10A        VARYING
      * Assume fld has a length of 2 before the MOVEL.
      * After the first MOVEL, it will have a value of 'XY'
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq...
C                   MOVEL     'XYZ'      fld
      * After the second MOVEL, it will have the value '1Y'
C                   MOVEL     '1'        fld
```

注 : MOVE と MOVEL について推奨される使用法は、EVAL とは対照的 に、一時的に長さを固定したいフィールドの値を変更する場合です。 たとえば、報告書を作成する場合に、その列のサイズが毎日異なって いて、ただし、プログラムの所定の実行時にはそれを固定する必要がある、という場合などです。

フィールドがファイル (入力仕様) から読み取られるとき、可変長フィールドの 長さは入力データの長さに設定されます。

出力仕様の「後で消去」機能は、可変長フィールドの長さを 0 に設定します。

可変長フィールドの長さは、EVAL 命令の左側の %LEN 組み込み関数を使用して、ユーザー自身で設定することができます。

フィールドの長さの使用法

可変長フィールドがその値に使用されるとき、その現在の長さが使用されます。 次の例では、「結果」は長さが 7 の固定長フィールドと想定されています。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++++
D fld                      10A        VARYING
 * For the following EVAL operation
 *    Value of 'fld'    Length of 'fld'   'result'
 *    --------------    ---------------   -----------
 *    'ABC'             3                 'ABCxxx '
 *    'A'               1                 'Axxx   '
 *    ''                0                 'xxx    '
 *    'ABCDEFGHIJ'      10                'ABCDEFG'
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq...
C                   EVAL      result = fld + 'xxx'
 * For the following MOVE operation, assume 'result'
 * has the value '.......' before the MOVE.
 *    Value of 'fld'    Length of 'fld'   'result'
 *    --------------    ---------------   -----------
 *    'ABC'             3                 '....ABC'
 *    'A'               1                 '......A'
 *    ''                0                 '.......'
 *    'ABCDEFGHIJ'      10                'DEFGHIJ'
C                   MOVE      fld           result
```

可変長フィールドを使用する理由

一時変数に可変長フィールドを使用すると、ストリング命令のパフォーマンスが 向上するとともに、ユーザーのコードを読みやすくします。これ は、フィールドの現在の長さを %SUBST の別の変数内に保管したり、余分のブランク を %TRIM を使用して無視する必要がないためです。

サブプロシージャーが、長さの異なるストリング・データを処理することを示している 場合、プロトタイプ・プロシージャーのパラメーターおよび戻り値 に可変長フィールドを使用すると、ユーザーの呼び出しとプロシージャーの、パフォーマンスと読みやすさの両方を向上させることができます。 サブプロシージャーの中で長さパラメーターを渡したり、CEEDOD を 使用して、パラメーターの実際の長さを知る必要はなくなります。

*CVTOPT(*VARCHAR) および CVTOPT(*VARGRAPHIC)*

ILE RPG コンパイラーは、外部記述ファイルまたはデータ構造からの可変長の文字フィールド、グラフィック・フィールド、または UCS-2 フィールドを、固定長の文字フィールドとして内部的に定義することができます。 可変長の文字フィールド、図形フィールド、および UCS-2 フィールドの固定長形式への変換は不要ですが、可変長フィールドがサポートされる前に作成されたプログラムをサポートするために、CVTOPT がこの言語に残されています。

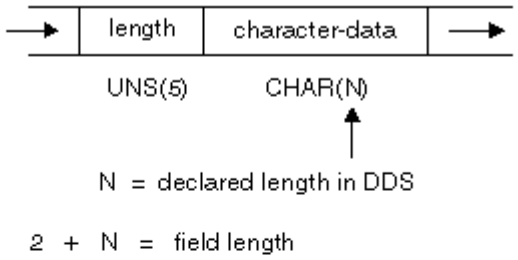
可変長フィールドを変換するには、CVTOPT 制御仕様書 キーワードまたはコマンド・パラメーターで、*VARCHAR (可変長文字フィールド) または *VARGRAPHIC (可変長図形フィールドまたは UCS-2 フィールド) を指定します。 *VARCHAR または *VARGRAPHIC を指定しないか、あるいは *NOVARCHAR または *NOVARGRAPHIC を指定した場合、可変長フィールドは固定長文字に変換されないので、ILE RPG プログラム内で可変長として使用することができます。

*VARCHAR または *VARGRAPHIC を指定 した場合には、次の条件が適用されます。

- 外部記述ファイルまたは外部記述データ構造から可変長フィールドを抜き出す場合には、ILE RPG プログラムの中でそれが固定長文字フィールドとして宣言されます。
- 1 バイト文字フィールドの場合には、宣言される ILE RPG フィールドの長さは DDS フィールドの長さに 2 バイトを加えたものになります。
- DBCS グラフィック・データ・フィールドの場合には、宣言される ILE RPG フィールドの長さは DDS フィールドの長さの 2 倍に 2 バイトを加えたものになります。
- ILE RPG フィールドの余分な 2 バイトには、可変長フィールドの現在の長さを表す符号なし整数が入ります。259 ページの図 95 は、可変長フィールドの ILE RPG フィールド長を示しています。

- 固定長文字フィールドとして定義された可変長図形フィールドの場合 には、長さは図形文字数の 2 倍になります。

Single-byte character fields:



Graphic data type fields:

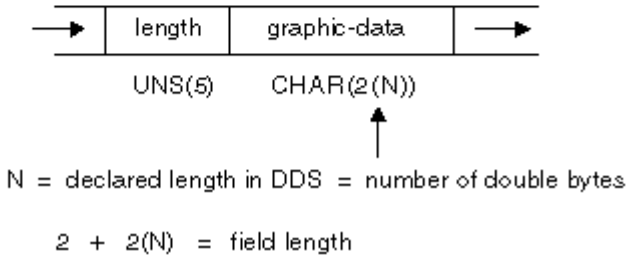


図 *95. ILE RPG* 変換された可変長フィールドのフィールド長

- ユーザーの ILE RPG プログラムでは、宣言された固定長フィールドに対する有効なすべての文字演算命令を実行することができます。 しかし、フィールドの 構造のために、フィールドがファイルに書き出される場合、フィールドの最初 の 2 バイトには有効な符号なし整数が入っていなければ なりません。 フィールドの最初の 2 バイトに正しくないフィールド長データが入っていた場合 には、出力操作で入出力例外エラーが起こります。
- 入力フィールドが外部記述入力ファイルからの可変長フィールドであった場 合には、制御レベル標識、突き合わせフィールド項目、およびフィールド標識を入力 仕様で使用することはできません。
- ファイルに可変長キー・フィールドが含まれている場合には、限界内順次処理を使用することはできません。
- キー順命令の演算項目 1 が外部記述ファイルの可変長キー・フィールドと対 応している場合には、キー順命令を使用することはできません。
- レコード中の一定のフィールドを選択的に出力することを選択し、可変長フィールドが出力仕様に指定されていないか、あるいは ILE RPG プログラム中で無視される場合には、ILE RPG コンパイラーは新たに追加するレコードの出力バッファーにデフォルトの値を入れます。 最初の 2 バイトのデフォルトの値は 0 で、残りのバイトのデフォルトの値はすべてブランクです。
- 変換済み可変長フィールドを変更したい場合には、現在の フィールド長が正しいことを確認して ください。 これを実行する 1 つの方法は次のとおりです。

  1. 可変長フィールド名を持つデータ構造をサブフィールド名として定義します。
  2. そのフィールドの始めにオーバーレイする 5 桁の符号なし 整数サブフィールドを定義し、3 桁目から始まるフィールドにオーバーレイ する N バイトの文字サブフィールドを定義します。
  3. フィールドを更新します。

  あるいは、左寄せされた別の可変長フィールドをフィールドの中に転送すること ができます。 以下は、ILE RPG プログラム中の変換済み可変長フィールドの変更方法の例です。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7....+..
A*
A*   File MASTER contains a variable-length field
A*
AAN01N02N03T.Name++++++Rlen++TDpBLinPosFunctions++++++++++++++++++++
A*
A          R REC
A            FLDVAR       100          VARLEN
```

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7....+.. *
 *
 *   Specify the CVTOPT(*VARCHAR) keyword on a control
 *   specification or compile the ILE RPG program with
 *   CVTOPT(*VARCHAR) on the command.
 *
HKeywords+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 *
H CVTOPT(*VARCHAR)
 *
 *   Externally described file name is MASTER.
 *
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++
 *
FMASTER    UF   E             DISK
```

```
 *
 *   FLDVAR is a variable-length field defined in DDS with
 *   a DDS length of 100.  Notice that the RPG field length
 *   is 102.
 *
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++
 *
D                 DS
D FLDVAR                  1    102
D   FLDLEN                       5U 0 OVERLAY(FLDVAR:1)
D   FLDCHR                     100    OVERLAY(FLDVAR:3)
```

```
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
 *
 * A character value is moved to the field FLDCHR.
 * After the CHECKR operation, FLDLEN has a value of 5.
C                   READ      MASTER                                 LR
C                   MOVEL     'SALES'       FLDCHR
C     ' '           CHECKR    FLDCHR        FLDLEN
C  NLR              UPDATE    REC
```

図 *96.* 可変長文字フィールドの変換

変換済み可変長図形フィールドを使用したい場合には、2 バイトの符号なし 整数フィールドをコーディングして、長さを保留し、長さ N の図形サブフィールドを コーディングしてフィールドのデータ部分を保留することができます。

```
 *
 *   Specify the CVTOPT(*VARGRAPHIC) keyword on a control
 *   specification or compile the ILE RPG program with
 *   CVTOPT(*VARGRAPHIC) on the command.
 *
 *   The variable-length graphic field VGRAPH is declared in the
 *   DDS as length 3.  This means the maximum length of the field
 *   is 3 double bytes, or 6 bytes.  The total length of the field,
 *   counting the length portion, is 8 bytes.
 *
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
 *
D                 DS
DVGRAPH                          8
D  VLEN                          4U 0 OVERLAY(VGRAPH:1)
D  VDATA                         3G   OVERLAY(VGRAPH:3)
```

```
 *
 *   Assume GRPH is a fixed-length graphic field of length 2
 *   double bytes.  Copy GRPH into VGRAPH and set the length of
 *   VGRAPH to 2.
 *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C*
C                   MOVEL     GRPH          VDATA
C                   Z-ADD     2             VLEN
```

図 97. 可変長図形フィールドの変換

文字データ、図形データおよび UCS-2 データの間の変換

注 : 図形 CCSID が無視される (制御ステートメントに CCSID(*GRAPH:*IGNORE) が指定されている か、または、CCSID(*GRAPH) も CCSID(*EXACT) も指定されていない) 場合、図形データ には CCSID がないと見なされて、図形データと UCS-2 データ間の変換、または、 図形データとジョブ CCSID 以外の CCSID を持つ英数字データ間の変換 はサポートされません。

文字データ、図形データ、および UCS-2 データの CCSID (コード化文字セット ID) は それぞれ異なっていてかまいません。 これらのデータ・タイプ間の変換はデータの CCSID によって行われます。 詳しくは、324 ページの『CCSID 制御キーワード』 および 418 ページの『CCSID 定義キーワード』 を参照してください。

データの *CCSID*

すべての文字データ、グラフィック・データ、および UCS-2 データにはコード化文字セット ID (CCSID) があり、これによってデータの意味が決まります。 例えば、UTF-8 CCSID 1208 ではデータ x'20' がブランクを意味しますが、EBCDIC CCSID 37 ではデータ x'40' が、UCS-2 CCSID 13488 では値 x'0020' がブランクを意味します。

ある CCSID から別の CCSID にデータを変換することができます。 例えば、CCSID 37 の EBCDIC データを持つ変数を UCS-2 CCSID 13488 の変数 に代入する場合、EBCDIC データは CCSID 13488 に変換される必要があります。 各タイプの有効な CCSID について詳しくは、418 ページの『CCSID 定義キーワード』を参照してください。

CCSID について詳しくは、URL http://www.ibm.com/systems/i/infocenter/ にある IBM i Information Center のグローバリゼーションに関するトピックを 参照してください。

変換

同じ命令の中でタイプまたは CCSID が異なる文字値、図形値、および UCS-2 値を使用する場合、 すべての値のタイプと CCSID が同じになるように変換が必要です。 変換は、変換組み込み関数 %CHAR、%UCS2、または %GRAPH を使用するか、MOVE または MOVEL 命令を 使用して、明示的に実行できます。ただし、以下のシナリオでは、必要なときにコンパイラーが暗黙的に変換を行います。

比較
比較は Unicode CCSID を使用して行われます。 一方または両方のオペランドがまだこの CCSID ではない場合、 オペランドはこの CCSID に一時的に変換されます。

代入
ソース値がターゲット値のタイプおよび CCSID に変換されます。

値および読み取り専用参照によって渡されるパラメーター
渡されたパラメーターは、プロトタイプ・パラメーターのタイプおよび CCSID に変換されます。

連結
一方のオペランドが 16 進リテラルの場合、他方のオペランド も文字タイプでなければなりません。この場合、16 進リテラル は他方のオペランドと同じ CCSID であると見なされます。 そうでない場合、オペランドは Unicode CCSID に 変換されます。

オペランドが Unicode CCSID に変換されるときには、 使用する Unicode CCSID を正確に判断するために、以下のルールが使用されます。

- いずれかのオペランドのタイプが文字の場合、 オペランドは UTF-8 に変換されます。
- それ以外の場合で、どちらのオペランドも UCS-2 ではない場合、 オペランドはモジュールの UCS-2 CCSID に変換されます。 モジュールの UCS-2 CCSID はデフォルトで 13488 に設定されますが、 制御ステートメントで CCSID(*UCS2) キーワードを 使用して、異なる CCSID に設定することができます。
- それ以外の場合で、一方のオペランドのみのタイプが UCS-2 の場合、他方のオペランドは UCS-2 オペランドの CCSID に変換されます。
- それ以外の場合で、UCS-2 オペランドの一方がモジュールの デフォルト UCS-2 CCSID である場合、その CCSID が使用されます。
- それ以外の場合、定義された長さが短いほうのオペランドが、長いほうのオペランドの CCSID に 変換されます。

警告 :

- 一方のオペランドが他方のオペランドの文字セットにない文字を含んでいる場合、 一部の CCSID 変換ではすべての文字を変換することはできないことがあります。 例えば、ユニコードのオペランドから、シングル文字セットを表す CCSID (例えば、EBCDIC CCSID) のオペランドに データを割り当てる場合、ユニコードのオペランド内の一部の文字が、異なる文字セットからのものであることがあります。 この場合、ターゲット・オペランドに置換文字が入れられます。 デフォルトでは、このシチュエーションの結果は、エラーではない状況コード 50 になります。 ただし、制御キーワード CCSIDCVT(*EXCP) を指定すると、 このシチュエーションの結果はエラー状況コード 00452 になります。
- 制御キーワード CCSIDCVT(*LIST) を使用して、モジュール内で実行される可能性のあるすべての CCSID 変換 のリストを取得できます。 このリストには、結果として置換文字になる可能性のある CCSID 変換についての警告も示されます。

置換文字および CCSIDCVT キーワードについて詳しくは、327 ページの『CCSIDCVT(*EXCP | *LIST)』 を参照してください。

入出力命令中の CCSID 変換

英数字フィールド、グラフィック・フィールド、または UCS2 フィールド が入力命令または出力命令で使用されている場合、フィールド内のデータと、ファイル用の入力バッファーまたは出力バッファー内のデータとの 間で、CCSID 変換が実行されることがあります。 入力命令または出力命令でフィールドが使用されるケースは次のとおりです。

- フィールドが、命令に使用される入力仕様または出力仕様に含まれている。
- フィールドが、キー付き命令の検索引数に含まれている。
- フィールドが、%FIELDS 組み込み関数によって指定されるフィールド・リスト中に含まれている。
- フィールドが、%KDS 組み込み関数で使用されるデータ構造のサブフィールドである。
- フィールドが、入力命令または出力命令の結果のデータ構造で使用されるデータ構造のサブフィールドである。

CCSID 変換はデータベースによって実行されることもあります。 その場合の変換は、レコードからのデータを入力バッファーに入れるとき、出力バッファーからの データをレコードに入れるとき、または、キー付き命令の 検索引数を解釈するときに、英数字フィールドおよびグラフィック・フィールドに対して行われます。

- ファイルに対して DATA(*NOCVT) が有効になっている場合、CCSID 変換は行われません。
- ジョブ CCSID が 65535 である場合、CCSID 変換は行われません。
- ジョブ CCSID が 65535 ではなく、ファイルに対して DATA(*NOCVT) が有効になっていない 場合、65535 以外の CCSID の英数字フィールドはジョブ CCSID に変換され、 65535 以外の CCSID のグラフィック・フィールドはジョブ CCSID に関連する DBCS CCSID に変換されます。 データベース・レベルでの CCSID 変換に影響する DATA キーワードについて詳しくは、366 ページの『DATA(*CVT | *NOCVT)』 を参照してください。

以下のすべての条件が真の場合、入力バッファーまたは出力バッファー内のデータ とプログラム・フィールドとの間で CCSID 変換が実行されます。

- プログラム・フィールドの CCSID が 65535 (*HEX) ではない。
- ファイル内のフィールドの CCSID が 65535 ではない。
- 入力バッファーまたは出力バッファー内のフィールドの CCSID が、フィールドの CCSID と異なる。
- プログラム・フィールドの CCSID が既知であるか、ファイルに対して DATA キーワードが有効になっている。 UCS-2 フィールドの CCSID は常に既知です。 グラフィック・フィールドの CCSID は、CCSID(*GRAPH:*IGNORE) が有効になっている場合を除いて、既知です。英数字フィールドの CCSID について詳しくは、261 ページの『データの CCSID』 を参照してください。ファイルに対して DATA キーワードが有効かどうかについて詳しくは、366 ページの『DATA(*CVT | *NOCVT)』 を参照してください。

警告 : ファイルに対して DATA キーワードが有効になっていないか、 または、英数字フィールドの CCSID が既知であると見なされない場合、RPG は、実行時にジョブ CCSID が 65535 で あると、プログラム内の英数字データの CCSID について誤った想定をすることがあります。

代替照合順序

代替照合順序は、1 バイト文字データにだけ適用されます。

各文字は、内部的に 16 進値で表され、この 16 進値に基づいて文字の 順序 (昇順または降順) が決められます。これは通常照合順序として知られてい るものです。 代替照合順序機能を使用することによって、通常照合順序を変更することができ ます。 また、この機能を使用して、2 つ以上の文字を等しいと見なすこともできます。

照合順序の変更

代替照合順序の使用は、文字の突き合わせフィールド (ファイル選択) および文字比 較の照合順序を変更することを意味します。 制御仕様書の ALTSEQ キーワードを 指定することによって、代替照合順序が使用されることを指定します。 代替照合順 序によって影響を受ける演算命令は、ANDxx、COMP、CABxx、CASxx、DOU、 DOUxx、DOW、DOWxx、IF、IFxx、ORxx、WHEN、および WHENxx です。 これは、図形または UCS-2 比較操作には適用されません。 LOOKUP および SORTA が影響を受けるのは、ALTSEQ(*EXT) を指定した場合だけです。 文字は、代替照合順序によって永続的に変更されるのではなく、フィールド 突き合わせまたは文字比較操作が完了するまで一時的に変更されます。

変数の定義仕様書で ALTSEQ(*NONE) キーワードを使用して、その変数が 他の文字データと比較されるときは、代替照合順序が定義されている場合でも、 必ず通常の照合順序を使用する必要があることを示します。

照合順序の変更は、LOOKUP および SORTA 命令 (ALTSEQ(*EXT) を指定しない場合) または形象定数 *HIVAL および *LOVAL に割り当てられた 16 進値には影響 しません。 しかし、照合順序の変更により、照合順序中の *HIVAL および *LOVAL の値に影響する可能性があります。 したがって、プログラム中で代替照合 順序を指定し、それによって *HIVAL および *LOVAL の値の順序に変化があった 場合には、望ましくない結果が起こることがあります。

外部照合順序の使用

SRTSEQ および LANGID コマンド・パラメーターまたは制御仕様書 キーワードの値を用いて代替照合順序を決定すべき場合は、 制御仕様書に ALTSEQ(*EXT) を指定してください。 たとえば、ALTSEQ(*EXT) を使用 し、SRTSEQ(*LANGIDSHR) および LANGID(*JOBRUN) を 指定した場合には、プログラムの実行時に、そのプログラム を実行中のユーザーの共用重みテーブルが代替照合順序として使用されま す。

ALTSEQ(*EXT) が指定されている場合には、LOOKUP および SORTA 命令が影響を受 けるので、コンパイル時文字配列およびテーブルは、代替照合順序を使用して順 序が検査されます。 実際の照合順序が実行時まで分からない場合には、その実行時 まで配列およびテーブルの順序を検査することはできません。 このことは、コン パイル時配列またはテーブルの順序が違っていることを知らせる実行時エラーを 受け取る可能性があることを意味しています。

ALTSEQ(*EXT) が指定された場合には、実行時前配列およびテーブルの順序検査も 代替照合順序を使用して行われます。

注 : 上記の説明は、定義仕様書で ALTSEQ(*NONE) を指定して定義 された配列およびテーブルには適用しません。

ソース仕様での代替照合順序の指定

代替照合順序を使用することを指定するためには、 制御仕様書で ALTSEQ(*SRC) キーワードを使用してください。 コンパイル時データ・セクションに **ALTSEQ、**CTDATA、 および **FTRANS キーワードを使用した場合には、ソース・レコードの後の 任意の場所に、代替照合順序データを入力することができます。 これらのキーワードを使用しない場合には、順序データはソース・レコードおよびフ ァイル変換レコードの後に続いていることが必要ですが、すべてのコンパイル時 配列データより前になければなりません。

2 つの連続した文字の間に文字を挿入する場合には、この挿入によって変更さ れるすべての文字を指定しなければなりません。 たとえば、A と B の間に円記号 (¥) を挿入する場合には、文字 B の前でその変更 を指定してください。

EBCDIC 文字セットについては、964 ページの『付録 B. EBCDIC 照合順序』を参照してください。

代替照合順序レコードの形式設定

照合順序に対する変更は、正しいレコード様式に書き換えて、システムに入力 できるようにしなければなりません。 代替照合順序は、次のように形式設定する 必要があります。

レコード位置指定	立入り
1 から 6	ALTSEQ (これは、通常照合順序が変更されていることをシステムに指示するものです。)
7 から 10	これらの桁はブランクのままにしておきます。
11 ～ 12	通常順序が変更されている文字の 16 進値を記入します。
13 ～ 14	通常順序が変更されている文字に置き換わる文字の 16 進値を記入します。
15 から 18 19 から 22 23 から 26 ... 77 から 80	15 桁目から始まる 4 桁のグループは、すべて 11 から 14 桁目と同じように使用 されます。 各グループの最初の 2 桁には、置き換えられる文字の 16 進値を記入します。 後の 2 桁には、置き換わる文字の 16 進値を記入します。

代替照合順序を記述しているレコードの前には、1 から 3 桁目 に **b (b = ブランク) の入ったレコードが存在する必要があります。 このレコードの残りの桁は、注記に使用することができます。

```
      HKeywords+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
     H ALTSEQ(*SRC)
     DFLD1             s               4A   INZ('abcd')
     DFLD2             s               4A   INZ('ABCD')
**
ALTSEQ    81C182C283C384C4
```

数字データ・タイプ

数字データ・タイプは、数値を表します。 固定形式の仕様書では、データ・タイプは 1 文字で 指定されます。 自由形式の仕様書では、データ・タイプはキーワードで指定されます。 数値データは、次の 1 つの形式となります。

B、BINDEC
265 ページの『2 進-10 進形式』

F、FLOAT
266 ページの『浮動形式』

I、INT
267 ページの『整数形式』

P、PACKED
268 ページの『パック 10 進数形式』

U、UNS
269 ページの『符号なし形式』

S、ZONED
270 ページの『ゾーン 10 進数形式』

数値フィールドのデフォルトの初期化値はゼロです。

2 進-10 進形式

2 進-10 進形式は、フィールドの左端ビットに符号 (正または負) が入っていて、 フィールドの残りのビットに数値が入っていることを意味します。 正数は符号ビットがゼロになり、負数は符号ビットが 1 で、2 の補数形式となります。 2 進フィールドは、1 から 9 桁の長さにすることができ、小数点以下の桁数付きと して定義することができます。 フィールドの長さが 1 から 4 桁であった場合には、コンパイラーは 2 バイトの 2 進フィールド長と見なします。 フィールドの長さが 5 から 9 桁であった場合には、コンパイラーは 4 バイトの 2 進フィールド長と見なします。

2 進-10 進形式の項目が保持できる値の範囲は 限定されています。 例えば、2 桁あって、小数部の桁数がゼロの 2 バイトの 2 進フィールド は、-99 から 99 までの値を保持できます。

注 : 整数データ型および符号なし整数データ型も 2 進形式ですが、 これらは全範囲の値を保持できます。 2 バイトの整数フィールドは、-32768 から 32767 までの値を保持できます。 2 バイトの符号なし整数フィールドは、0 から 65535 までの値を保持できます。

プログラム記述 *2* 進入力フィールドの処理

2 進形式で読み取られたすべての入力フィールドには、コンパイラーによって フィールド長 (桁数) が割り当てられます。 フィールドがプログラム内の別の場所で定義されていない場合には、 2 バイトの 2 進フィールドには長さ 4 が割り当てられ、4 バイトの 2 進フィールドには長さ 9 が割り当てられます。 これらの長さの制約事項のために、 2 バイトの 2 進フィールドに割り当てることができる最高の 10 進値は 9999 となり、 4 バイトの 2 進フィールドに割り当てることができる最高の 10 進値は 999 999 999 となります。 一般に、n 桁の 2 進フィールドには、nx9 の最大値を持たせることができます。 この説明は、小数点以下の桁数はないことを前提としています。

2 進形式の 2 バイト・フィールドはコンパイラーによって 1 から 4 桁の 10 進 フィールドに変換されるので、入力値が大きすぎることがあります。 その場合には、数値の左端桁が除去されます。 たとえば、4 桁の 2 進入力フィールドに 16 進 6000 の 2 進値が入っている場合には、 コンパイラーはこれを 10 進の 24 576 に変換します。 この 2 が除去され、結果は 4576 になります。 同様に、入力値が 2 進形式の 4 バ

イト・フィールドには大きすぎることがあります。 2 進フィールドの小数点以下の桁数がない (0) 場合には、2 進フィールド の代わりに整数フィールドを定義することによって、この変換の問題を避けるこ とができます。

注 : 2 進入力フィールドを突き合わせフィールドまたは制御フィールドとして定義する ことはできません。

外部記述 2 進入力フィールドの処理

2 進フィールドの桁数は、DDS 記述の長さと正確に同じで なければなりません。 例えば、2 進フィールドが 7 桁で小数点以下の桁数はないことを DDS 仕様に定義した場合には、RPG IV コンパイラーはこのデータを以下のように処理します。

1. フィールドは入力仕様で 4 バイトの 2 進フィールドとして定義されます。
2. RPG IV プログラム中でフィールドについてパック (7,0) フィールドが生成されます。

2 進フィールドの完全な情報を保存したい場合には、フィールドをデータ構造 内の 2 進サブフィールドとしてか、または独立した 2 進フィールドとして再 定義してください。

外部記述 2 進フィールドは、RPG IV の 2 進フィールドに使用できる範囲外の値を持つことがある点に注意してください。 外部記述 2 進フィールドの小数点以下の桁数がない (0) 場合には、この問題を 避けることができます。 そうするためには、外部記述 2 進フィールドを定義仕様書に定義 し、制御仕様書には EXTBININT キーワードを指定して ください。 これによって、外部記述フィールドの外部形式が符号付き整数の形式に変更され ます。

浮動形式

浮動形式は次の 2 つの部分から成り立ちます。

- 仮数
- 指数

浮動小数点フィールドの値は、この仮数に、10 の指数分のべき乗を 掛けて求められます。 たとえば、1.2345 が仮数で、指数が 5 である場合、浮動小数点の値は次のように なります。

```
   1.2345 * (10 ** 5) = 123450
```

浮動小数点フィールドを定義するには、自由形式での定義に FLOAT キーワード を指定するか、または、適切な仕様書のデータ・タイプ記入項目に F を 指定します。

小数点以下の桁はブランクのままにしておく必要があります。 ただし、 浮動小数点フィールドには小数点以下の桁があると見なされます。 この結果、配列指標、DO ループ指標などの、小数位のない数値が必要ないずれかの場所で、浮動変数が使用されない場合が生じます。

浮動小数点フィールドのデフォルトの初期化および CLEAR 値は 0E0 です。

浮動小数点フィールドの長さは、バイト数で定義されます。 この長さは、4 バイトまたは 8 バイトのいずれかに指定されます。 浮動小数点フィールドに使用できる値の範囲は次のとおりです。

4 バイトの浮動 (8 桁)
-3.4028235E+38 から -1.1754944E-38、0.0E+0、 +1.1754944E-38 から +3.4028235E+38

8 バイトの浮動 (16 桁)
-1.797693134862315E+308 から -2.225073858507201E-308、0.0E+0、
+2.225073858507201E-308 から +1.797693134862315E+308

注 : 浮動変数は、IBM i オペレーティング・システムによるサポートに従って IEEE 標準に準じています。 浮動変数は「科学的な」値を表すように意図されたものであるので、 浮動変数内に記憶された数値は、パック変数内の値と多少異なる値を 表す場合があります。 金額など、特定の数の小数位に正確な数値を表す必要がある 場合には、浮動を使用しないようにしてください。

浮動小数点フィールドの外部表示表現

外部表示表現の一般的な説明については、248 ページの『数値フィールドの場合の外部形式の指定』を参照してください。

浮動値の外部表示表現は次の場合に適用されます。

- データ形式記入項目をブランクにした浮動データの出力
- データ形式記入項目をブランクにした浮動データの入力
- コンパイル時および実行時前配列およびテーブルの 外部形式 (キーワード EXTFMT が省略された場合)
- 命令コード DSPLY を使用した浮動値の表示および入力
- ダンプ・リストへの浮動値の出力
- 組み込み関数 %EDITFLT の結果

出力

浮動値を出力するとき、外部表現は、浮動リテラルに似た形式 (ただし、 以下の点が異なります) を使用します。

- 値は、常に文字 E および仮数と指数の両方の符号を 付けて書き出されます。
- 値の長さは、14 文字または 23 文字です (それぞれ、4F およ び 8F の場合)。
- 値は正規化されます。 すなわち、小数点は最上位桁の直後に付けられます。
- 小数点は、制御仕様書キーワード DECEDIT のパラメーターに応じて、 ピリオドまたはコンマになります。

以下に浮動値の表示例をいくつか示します。

```
   +1.2345678E-23
   -8.2745739E+03
   -5.7227480274673 92E-123
   +1,2857638E+14             if DECEDIT(',') is specified
```

入力

浮動値を入力するとき、その値は浮動リテラルと同様に 指定されます。 この値は、そのフィールド内で正規化あるいは調整する必要はありません。 浮動値が配列/テーブル初期化データとして定義される 場合、それらの値は 14 文字または 23 文字の長さの フィールド (それぞれ 4F および 8F の場合) 内に 指定されます。

浮動フィールドについては、次の点に注意してください。

- 浮動サブフィールドのアクセスのパフォーマンスを改善するために、 浮動フィールドの位置合わせが必要になる場合があります。 定義仕様書に定義された浮動サブフィールドを位置合わせするには、ALIGN キーワードを使用できます。4 バイト浮動サブフィールドは 4 バイト境界に位置合わせされ、8 バイト浮動サブフィールドは 8 バイト境界に位置合わせされます。浮動サブフィールドの位置合わせについて詳しくは、414 ページの『ALIGN{(*FULL)}』 を参照してください。
- LIKE キーワードを浮動フィールドと類似のフィールドを定義するために 使用するときは、長さの調整はできません。
- 浮動入力フィールドを、突き合わせフィールドまたは制御フィールド として定義することはできません。

整数形式

整数形式は、次の 2 つの例外を除いて、2 進形式と 類似しています。

- 整数形式ではすべての範囲の 2 進値を使用することができます。
- 整数フィールドの小数点以下の桁数は常にゼロです。

整数フィールドを定義するには、自由形式での定義に INT キーワード を指定するか、または、適切な仕様書のデータ・タイプ記入項目に I を 指定します。 パラメーターが整数フィールドの場合には、 定義仕様書で LIKE キーワードを使用することによって、 整数フィールドを定義することもできます。

整数フィールドの長さは桁数によって定義され、3 桁、5 桁、10 桁、または 20 桁のいずれかの長さにすることができます。 3 桁のフィールドは 1 バイトの記憶域を占め、5 桁のフィールドは 2 バイト、10 桁のフ

ィールドは 4 バイト、20 桁のフィールドは 8 バイトをそれぞれ占めます。 整数フィールドに使用可能な値の範囲は、その長さによって異なります。

フィールド長
使用できる値の範囲

3 桁の整数
-128 から 127

5 桁の整数
-32768 から 32767

10 桁の整数
-2147483648 から 2147483647

20 桁の整数
-9223372036854775808 から 9223372036854775807

整数フィールドについては、次の点に注意してください。

- 整数フィールドへのアクセスのパフォーマンスを向上させるためには、整数フ ィールドの位置合わせが必要になる場合があります。 定義仕様書に定義された整数サブフィールドを位置合わせするためには、ALIGN キー ワードを使用することができます。

  2 バイトの整数サブフィールドは、2 バイト境界に位置合わせ され、4 バイト整数サブフィールドは 4 バイト境界に、さらに 8 バイト・サブフィールド は 8 バイト境界にそれぞれ位置合わせされます。整数サブフィールドの位置合わせについて詳しくは、414 ページの『ALIGN{(*FULL)}』 を参照してください。
- LIKE キーワードを使用して整数フィールドと類似のフィールドを定義する場合に は、「長さ」記入項目に桁数による長さの調整を含めることができます。 調整値は、調整後のフィールドの桁数が、3、5、10、または 20 になるようにする必要があります。
- 整数入力フィールドを突き合わせフィールドまたは制御フィールドとして定義するこ とはできません。

パック 10 進数形式

パック 10 進数形式は、記憶域の各バイト (最下位バイトは除く) に 2 つの 10 進を入れることができることを意味します。 最下位バイトには、その左端部分に 1 桁の数字が入り、右端部分には符号 (正ま たは負) が入ります。 使用される標準の符号は、正数には 16 進の F および負数には 16 進の D です。 パック 10 進数形式は、次のようになります。

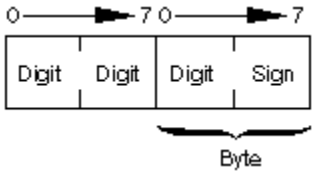


図 *98. パック 10* 進数形式

最下位バイトの符号部分は、数字部分に表された数値が正と負のいずれであるかを示します。272 ページの図 100 は、パック 10 進数形式での 10 進 21544 の表記を示しています。

パック *10* 進数フィールドの桁数の長さの判別

パック 10 進数フィールドの桁数による長さを見付けるためには、次の式を使用 してください。

```
Number of digits = 2n - 1,
...where n = number of packed input record positions used.
```

この式によって、パック 10 進数形式で表現できる最大桁数が得られます。上限 は 63 です。

パック・フィールドの長さには、最大で 32 バイトまで指定できます。269 ページの表 70 は、最高 63 桁の長さのゾーン 10 進数フィールドと同等のパック 10 進数フィールドを示しています。

表 70. 最高 63 桁の長さのゾーン 10 進数フィールドと同等のパック 10 進数フィールド	
ゾーン 10 進数 の長さ (桁)	**パック 10 進数フィールド で使用されるバイト数**
1	1
2、3	2
4、5	3
⋮	⋮
28、29	15
30、31	16
⋮	⋮
60、61	31
62、63	32

たとえば、パック 10 進数形式で読み取られた入力フィールドの長さが (入力ま たは定義仕様書に指定されたとおりの) 5 バイトであったとします。 このフィールドの桁数は 2(5) - 1、つまり 9 と等しくなります。 したがって、このフィールドが演算仕様書で使用される時には、結果のフィールドには 9 桁の長さが必要です。 定義仕様書の 473 ページの『PACKEVEN』 キーワードを使用して、桁数ではなく、開始位置および終了位置を使用してパック・サブフィールドを指定するときに必要な 2 つの使用可能なサイズのいずれかを指定することができます。

符号なし形式

符号なし整数形式は、値の範囲に負数は含まれない点を除いて、整数形式と類似 しています。 符号なし形式を使用する必要があるのは、負でない整数が必要とされる場合だけ です。

符号なしフィールドを定義するには、自由形式定義に UNS キーワード を指定するか、または、適切な仕様書のデータ・タイプ記入項目に U を 指定します。 パラメーターが符号なしフィールドである場合には、定義仕様書 の LIKE キーワードを使用して符号なしフィールドを定義する こともできます。

符号なしフィールドの長さは桁数によって定義され、3 桁、5 桁、10 桁、または 20 桁のいずれかの長さにすることができます。 3 桁のフィールドは 1 バイトの記憶域を占め、5 桁のフィールドは 2 バイト、10 桁のフィールドは 4 バイト、20 桁のフィールドは 8 バイトをそれぞれ占めます。 符号なしフィールドに使用可能な値の範囲は、その長さによって異なります。

フィールド長
　使用できる値の範囲

符号なしの 3 桁
　0 から 255

符号なしの 5 桁
　0 から 65535

符号なしの 10 桁
　0 から 4294967295

符号なしの 20 桁
　0 から 18446744073709551615

位置合わせについての情報も含めて、符号なしフィールドの使用に関するその他の考慮事項については、267 ページの『整数形式』を参照してください。

ゾーン 10 進数形式

ゾーン 10 進数形式は、記憶域の各バイトに 1 つの数字または 1 つの文字を入れ ることができることを意味します。 ゾーン 10 進数形式は、記憶域の各バイトには 1 桁または 1 文字を入れることがで きることを意味します。 ゾーン 10 進数形式では、記憶域の各バイトは、4 ビットのゾーン部分と 4 ビット の数字部分の 2 つの部分に分かれています。 ゾーン 10 進数形式は以下のようになります。 ゾーン 10 進数形式は次のようになります。

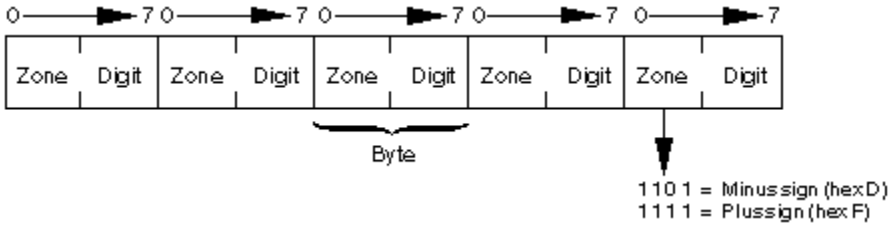


図 *99.* ゾーン *10* 進数形式

最下位バイトのゾーン部分は、10 進の符号 (正または負) を示します。 使用される標準の符号は、正数には 16 進の F および負数には 16 進の D です。 ゾーン 10 進数形式では、10 進の各桁にゾーン部分が入れられますが、最下位のゾーン部分のみは符号として働きます。272 ページの図 100 は、ゾーン 10 進数形式での数値 21544 の表記を示しています。

出力仕様の 40 から 43 桁目に終了位置をコーディングし、フィールドがパック形 式で出力される場合には、フィールド長の変更を考慮しなければなりません。 フィールドがパックされた後の長さを見付けるためには、次の式を使用してくだ さい。

$$\text{Field length} = \frac{n}{2} + 1$$

. . . where n = number of digits in the zoned decimal field.

(Any remainder from the division is ignored.)

ゾーン 10 進数形式には代替符号形式を指定することができます。 代替符号形式では、数値フィールドの直前または直後に + 符号または - 符号を置 くことができます。 プラス符号は 16 進の 4E、マイナス符号は 16 進の 60 になります。

代替符号形式を指定した時には、(入力仕様に指定する) フィールド長には符号 のための追加の桁を含めなければなりません。 たとえば、フィールドが 5 桁の長さで、代替符号形式を指定した場合には、6 桁 のフィールド長を指定しなければなりません。

数値形式の使用に関する考慮事項

数値フィールドの定義に際しては、以下の点に留意してください。

- 出力仕様の 47 から 51 桁目に終了位置をコーディングした場合には、出力フィー ルドが占有するバイト数の計算時には必ず外部形式を使用してください。 たとえば、5 桁のパック・フィールドは 3 バイトに記憶されますが、出力がゾー ン形式であった場合には、5 バイトを必要とします。 出力が整数形式の場合には、2 バイトしか必要とされません。
- 文字フィールドをゾーン形式の数値に転送した場合には、文字フィールドは ゾーン形式の正またはゾーン形式の負に固定されます。 その他のバイトのゾーン部分は、強制的に 'F' となります。 ただし、文字フィールドの 1 つのバイトの数字部分に有効数字が含まれていない 場合には、10 進データ・エラーが起こります。
- 数値フィールドが編集せずに書き出された場合には、符号は別個の文字として印 刷されず、数値の最後の桁に符号が組み込まれます。 これにより、意外な 結果が生ずる可能性があります。たとえば、-625 が書き出さ れる場合には、ゾーン 10 進数値は 62N を表す X'F6F2D5' になります。

フィールドの数値形式選択に関する考慮事項

次の場合には、フィールドに整数または符号なし形式を使用しなければなりません。

- 算術のパフォーマンスが重要な場合

  特定の算術演算では、使用される値が整数であることが重要な場合があります。 パフォーマンスが向上することがあるいくつかの例として、配列指標の計算および組み込み関数 %SUBST 用の引数があります。
- ILE C のような、整数データ・タイプをサポートする他の言語で作成されたルーチンと対話をしている場合。
- 整数として定義され、9999 または 999999999 を超える値が入れられること があるファイル・フィードバック域のフィールドを使用している場合。

次の場合には、フィールドにパック、ゾーン、および 2 進形式を 指定することが必要です。

- 通貨の値のように、暗黙の小数点以下の桁数がある値を使用している場合
- 19 桁を超える値を処理している場合
- フィールドの特定の桁数を確認することが重要な場合

次の場合には、フィールドに浮動形式を指定することが必要です。

- パックまたはゾーン値で表すことのできない非常に小さい値または 非常に大きい値 (あるいはその両方) を保持するために同じ変数が必要な場合。

ただし、16 桁を超える精度を必要とする場合には、 浮動形式を使用しないで ください。

注 : 算術演算が整数または符号なし形式を使用して実行され、とくに、整数の演算が 自由形式の式で行われた場合には、オーバーフローが起こる可能性がより高く なります。 これは、中間結果が十分なサイズの一時 10 進フィールドにではなく、整数ま たは符号なし形式で保存されるからです。

数値形式の表現

272 ページの図 100 は、各種の形式での 10 進 21544 の表記を示しています。

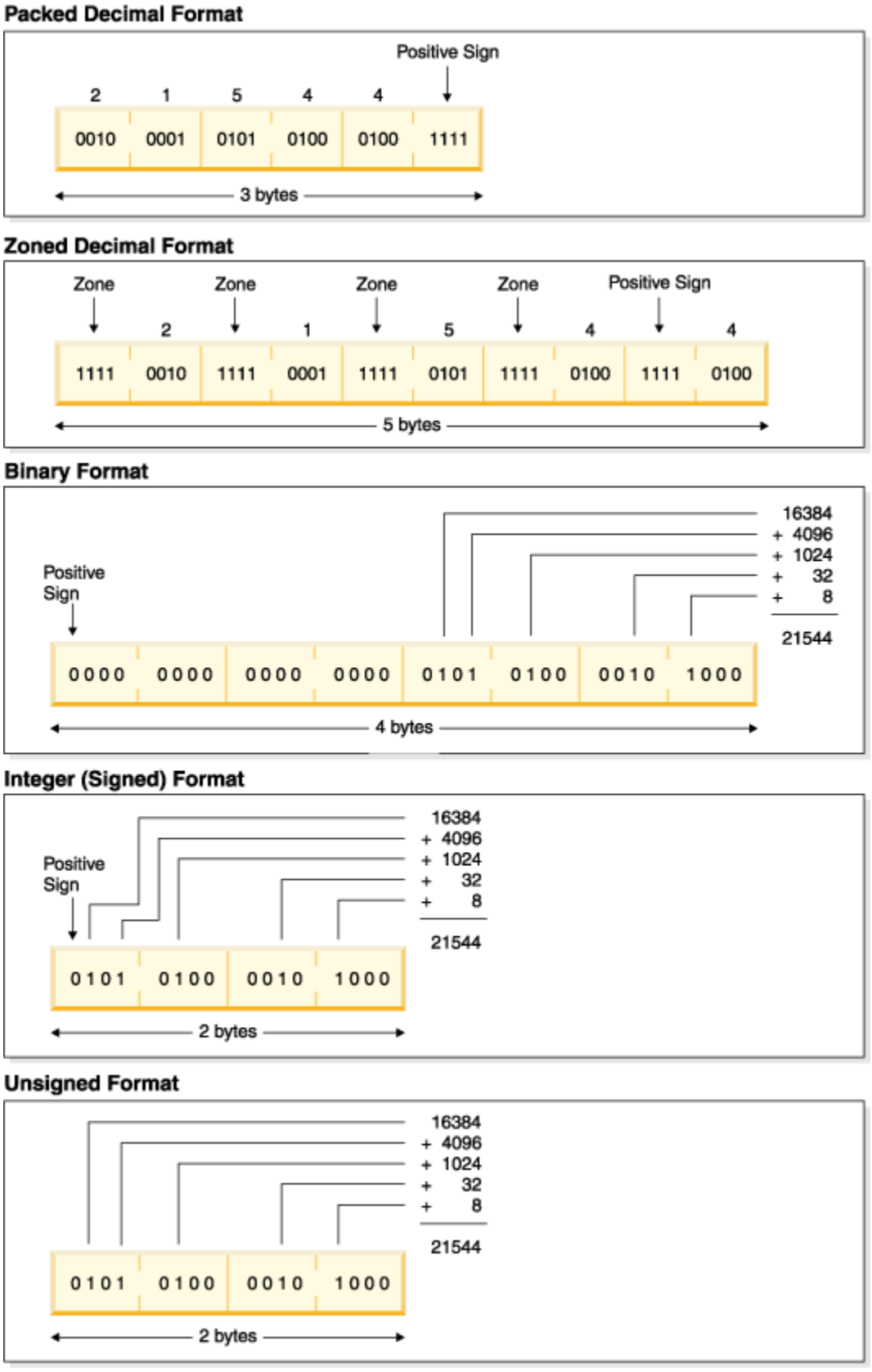


図 *100.* 各数値形式における数値 *21544* の表現

図の表現については、次の点に注意してください。

- 正の 2 進、整数、または符号なしの数の数値を入手するためには、オン (1) になっているビットの値を加算しますが、符号ビットは含めません (それが あった場合)。 符号なしの数の場合には、左端ビットも含めて、オンになっているビットの値を 加算してください。
- 値 21544 には最下位の 2 バイトのビットだけが使用されますが、それを 2 バ イトの 2 進フィールドに表すことはできません。 2 バイトの 2 進フィールドに保留できるのは 4 桁までであり、21544 は 5 桁です。

273 ページの図 101 は、整数形式の数値 -21544 を示しています。

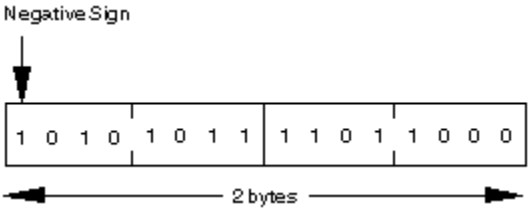


図 *101.* 数値 *-21544* の整数表現

日付データ・タイプ

日付フィールドは、事前に決定されたサイズおよび形式を持っています。 これらは定義仕様書で定義することができます。 すべての日付データに先行ゼロおよび後書きゼロが必要です。

比較または割り当てで使用される日付定数または変数は、同じ形式であるか、 または同じ区切り記号が使用されている必要はありません。 また、入力フィールド、出力フィールド、またはキー・フィールドのように入出 力命令に使用される日付は (必要な場合に) 命令に必要な形式に変換されます。

日付変数のデフォルトの内部形式は *ISO です。 このデフォルト内部形式は、制御仕様書キーワード DATFMT によって グローバルにオーバーライドすることができ、/SET 指示および /RESTORE 指示によって一時的に変更することができ、 また、定義仕様書キーワード DATE または DATFMT によって個々に設定することができます。

日付フィールドの内部日付形式および区切り記号を判別する時に使用される順位は次のとおりです。

1. 定義仕様書に指定された DATE または DATFMT キーワードから
2. /RESTORE 指示で元に戻されていない、DATFMT キーワードを含んでいる最新の /SET 指示から
3. 制御仕様書に指定された DATFMT キーワードから
4. *ISO

日付データ形式には、表現できる年の範囲により 3 つの種類があります。 これによって、操作の結果がターゲット・フィールドの有効範囲外の日付になった場合、 日付のオーバーフローまたはアンダーフロー条件が発生する可能性があります。 各形式および範囲は次のとおりです。

年の桁数	年の範囲
2 (*YMD、*DMY、*MDY、*JUL)	1940 から 2039
3 (*CYMD、*CDMY、*CMDY)	1900 から 2899
4 (*ISO、*USA、*EUR、*JIS、*LONGJUL)	0001 から 9999

274 ページの表 71 は、日付データおよびその区切り記号の RPG 形式を リストしています。

注 : 区切り記号「&」は、区切り記号としてブランクが使用されることを示します。 例えば、DATE(*YMD&) で定義された日付フィールド の区切り記号はブランクになります。

日付フィールドのコーディング方法の例については、以下の例を参照してください。

- 572 ページの『日付命令』
- 585 ページの『日付時刻データの転送』
- 711 ページの『ADDDUR (期間の加算)』
- 803 ページの『MOVE (転送)』

- 782 ページの『EXTRCT (日付/時刻/タイム・スタンプの抽出)』
- 887 ページの『SUBDUR (期間減算)』
- 893 ページの『TEST (日付/時刻/タイム・スタンプのテスト)』

表 71. 日付データ・タイプの *RPG* 定義日付形式および区切り記号

フォーマット名	説明	形式 (デフォルトの区切り記号)	有効な区切り記号	長さ	例
2 桁の年形式					
*MDY	月/日/年	mm/dd/yy	/ - . , &	8	01/15/96
*DMY	日/月/年	dd/mm/yy	/ - . , &	8	15/01/96
*YMD	年/月/日	yy/mm/dd	/ - . , &	8	96/01/15
*JUL	ユリウス	yy/ddd	/ - . , &	6	96/015
4 桁の年形式					
*ISO	国際標準化機構	yyyy-mm-dd	-	10	1996-01-15
*USA	IBM USA 標準規格	mm/dd/yyyy	/	10	01/15/1996
*EUR	IBM 欧州標準規格	dd.mm.yyyy	.	10	15.01.1996
*JIS	日本工業規格 (JIS) 西暦	yyyy-mm-dd	-	10	1996-01-15

274 ページの表 72 は、すべての RPG 定義日付形式の *LOVAL、*HIVAL、およ びデフォルトの値をリストしています。

表 72. 日付の値

フォーマット名	説明	*LOVAL	*HIVAL	デフォルト値
2 桁の年形式				
*MDY	月/日/年	01/01/40	12/31/39	01/01/40
*DMY	日/月/年	01/01/40	31/12/39	01/01/40
*YMD	年/月/日	40/01/01	39/12/31	40/01/01
*JUL	ユリウス	40/001	39/365	40/001
4 桁の年形式				
*ISO	国際標準化機構	0001-01-01	9999-12-31	0001-01-01
*USA	IBM USA 標準規格	01/01/0001	12/31/9999	01/01/0001
*EUR	IBM 欧州標準規格	01.01.0001	31.12.9999	01.01.0001
*JIS	日本工業規格 (JIS) 西暦	0001-01-01	9999-12-31	0001-01-01

MOVE、MOVEL、および TEST 命令でのみ使用されるフィールドの場合には、他の 形式もサポートされています。 このサポートは、すでに 3 桁の年形式および 4 桁の 年 *LONGJUL 形式である外部定義値との互換性のために用意されています。 このサポートは、*JOBRUN が指定された場合には、2 桁の年形式にも 適用されます。

*JOBRUN は、それが説明しているフィールドにジョブからの属性があることがわかって いる場合に使用してください。 たとえば、TIME 命令の 12 桁の数値結果は、ジョブ日付形式になります。

275 ページの表 73 は、MOVE、MOVEL または TEST 命令の演算項目 1 に 使用することができる、有効な外部定義日付形式をリストしています。

表 73. 外部定義の日付形式および区切り記号

フォーマット名	説明	形式 (デフォルトの区切り記号)	有効な区切り記号	長さ	例
2 桁の年形式					
*JOBRUN[1]	実行時に DATFMT、または DATSEP ジョブ値から 判別されます。				
3 桁の年形式 [2]					
*CYMD	世紀 年/月/日	cyy/mm/dd	/ - . , &	9	101/04/25
*CMDY	世紀 月/日/年	cmm/dd/yy	/ - . , &	9	104/25/01
*CDMY	世紀 日/月/年	cdd/mm/yy	/ - . , &	9	125/04/01
4 桁の年形式					
*LONGJUL	長形式のユリウス暦	yyyy/ddd	/ - . , &	8	2001/115

注 :

1. *JOBRUN は、DATFMT の実行時ジョブ属性が *MDY、*YMD、*DMY または *JUL だけ であるために、年の桁が 2 桁の文字日付または数値日付の場合にのみ有効です。
2. 世紀の文字 'c' の有効値は次のとおりです。

```
  'c'           Years
----------------------
  0          1900-1999
  1          2000-2099
  .              .
  .              .
  .              .
  9          2800-2899
```

分離文字

MOVE、MOVEL または TEST 命令で日付形式をコーディングする 場合は、文字フィールドには区切り記号はオプションです。 区切り記号がないことを指示するには、その形式の後にゼロを付けます。 区切り記号のない日付形式をコーディングするための方法について詳しくは、803 ページの『MOVE (転送)』、824 ページの『MOVEL (左につめて転送)』 および 893 ページの『TEST (日付/時刻/タイム・スタンプのテスト)』を参照してください。

初期化

実行時にシステム日付に日付フィールドを初期化するには、 定義仕様書に INZ(*SYS) を指定します。 実行時にジョブ日付に日付フィールドを初期化するには、定義仕様書に INZ(*JOB) を指定します。*SYS または *JOB は、エクスポートされたフィールドには使用できません。 また、日付フィールドは、リテラル、名前付き定数、あるいは 形象定数に初期化することもできます。

注 : 実行時初期化は、静的初期化の後に行われます。

時刻データ・タイプ

時刻フィールドは、事前に決定されたサイズおよび形式を持っています。 これらは定義仕様書で定義することができます。 すべての時刻データに先行ゼロおよび後書きゼロが必要です。

比較または割り当てで使用される時刻定数または変数は、同じ形式であるか、 または同じ区切り記号が使用されている必要はありません。 また、入力フィールド、出力フィールド、またはキー・フィールドのように入出 力命令に使用される時刻は (必要な場合に) 命令に必要な形式に変換されます。

時刻変数のデフォルトの内部形式は *ISO です。 このデフォルト内部形式は、制御仕様書キーワード TIMFMT によって グローバルにオーバーライドすることができ、/SET 指示および /RESTORE 指示によって

一時的に変更することができ、 また、定義仕様書キーワード TIME または TIMFMT によって個々に設定することができます。

時刻フィールドの内部時刻形式および区切り記号を判別する時に使用される順位 は次のとおりです。

1. 定義仕様書に指定された TIME または TIMFMT キーワードから
2. /RESTORE 指示で元に戻されていない、TIMFMT キーワードを含んでいる最新の /SET 指示から
3. 制御仕様書に指定された TIMFMT キーワードから
4. *ISO

時刻フィールドのコーディング方法の例については、以下の例を参照してください。

- 572 ページの『日付命令』
- 585 ページの『日付時刻データの転送』
- 711 ページの『ADDDUR (期間の加算)』
- 803 ページの『MOVE (転送)』
- 887 ページの『SUBDUR (期間減算)』
- 893 ページの『TEST (日付/時刻/タイム・スタンプのテスト)』

276 ページの表 74 は、サポートされている時刻形式およびその区切り記号をリストしています。

注 : 区切り記号「&」は、区切り記号としてブランクが使用されることを示します。 例えば、TIME(*HMS&) で定義された時刻フィールドの 区切り記号はブランクになります。

表 *74.* 時刻データ・タイプの時刻形式および区切り記号

RPG 形式名	記述	形式 (デフォルトの区切り記号)	有効な区切り記号	長さ	例
*HMS	時:分:秒	hh:mm:ss	: . , &	8	14:00:00
*ISO	国際標準化機構	hh.mm.ss	.	8	14.00.00
*USA	IBM 米国標準規格。 AM および PM を大文字および小文字の任意の組み合わせとすることができます。	hh:mm AM または hh:mm PM	:	8	02:00 PM
*EUR	IBM 欧州標準規格	hh.mm.ss	.	8	14.00.00
*JIS	日本工業規格 (JIS) 西暦	hh:mm:ss	:	8	14:00:00

276 ページの表 75 は、すべての時刻形式の *LOVAL、*HIVAL、およびデフォルトの値をリストしています。

表 *75.* 時刻の値

RPG 形式名	記述	*LOVAL	*HIVAL	デフォルト値
*HMS	時:分:秒	00:00:00	24:00:00	00:00:00
*ISO	国際標準化機構	00.00.00	24.00.00	00.00.00
*USA	IBM 米国標準規格。 AM および PM を大文字および小文字の任意の組み合わせとすることができます。	00:00 AM	12:00 AM	00:00 AM
*EUR	IBM 欧州標準規格	00.00.00	24.00.00	00.00.00
*JIS	日本工業規格 (JIS) 西暦	00:00:00	24:00:00	00:00:00

分離文字

MOVE、MOVEL または TEST 命令で時刻形式をコーディングする場合、 文字フィールドには区切り記号はオプションです。 区切り記号がないことを指示するには、その形式の後にゼロを付けます。区切り記号のない時刻形式をコーディングするための詳しい方法については、803 ページの『MOVE (転送)』を参照してください。

初期化

実行時に時刻フィールドをシステム時刻に初期化するには、定義仕様書に INZ(*SYS) を指定します。*SYS は、エクスポートされたフィールドには使用できません。 また、時刻フィールドは、実行時に、リテラル、名前付き定数または 形象定数に初期化することもできます。

注 : 実行時初期化は、静的初期化の後に行われます。

*JOBRUN

*JOBRUN の特殊値を、MOVE、MOVEL または TEST 命令の演算項目 1 に 使用することができます。 これは、説明されているフィールドの区切り記号 が、実行時ジョブ属性、TIMSEP に基づいていることを 示します。

タイム・スタンプ・データ・タイプ

タイム・スタンプ・フィールドは、事前に決定されたサイズおよび形式を持っています。 これらは定義仕様書で定義することができます。 タイム・スタンプ・データの形式は YYYY-MM-DD-hh-mm であり、 その後に任意指定の 1 から 12 桁の秒の小数部が続きます。

タイム・スタンプのデフォルトのサイズは 26 で、秒の小数部 6 桁を含みます。 自由形式のタイム・スタンプ定義では、 TIMESTAMP キーワードのパラメーターを使用して、秒の小数部の桁数を制御します。 固定形式のタイム・スタンプ定義の場合、タイム・スタンプ項目の長さとして、19、または 21 から 32 までの範囲内の値 を指定するか、または、「小数点以下の桁数」記入項目に秒の小数部の桁数を指定することができます。 次の例では、項目 *ts1* および *ts3* には、秒の小数部が 6 桁あります。 項目 *ts2* には、秒の小数部が 3 桁あります。 項目 *ts4* には、秒の小数部が 7 桁あります。 項目 *ts5* には、秒の小数部が 2 桁あります。

```
    DCL-S ts1 TIMESTAMP;
    DCL-S ts2 TIMESTAMP(3);
D ts3             S               Z
D ts4             S              27Z
D ts5             S               Z 2
```

タイム・スタンプ・リテラルには、0 から 12 の範囲内で秒の小数部の桁数を指定できます。 ただし、タイム・スタンプ・リテラルの秒の小数部は 6 から 12 桁です。 リテラルに秒の小数部として 6 桁より少ない桁しか指定されなかった場合、 タイム・スタンプ・リテラルは秒の小数部が 6 桁になるように右にゼロを追加して埋められます。 タイム・スタンプ・リテラルについて詳しくは、203 ページの『リテラル』を参照してください。

すべてのタイム・スタンプ・データに先行ゼロが必要です。

タイム・スタンプのデフォルトの初期化値は、0001 年 1 月 1 日 の午前 0 時 (0001-01-01-00.00.00、秒の小数部は 0 桁) です。 タイム・スタンプの *HIVAL 値 は、9999-12-31-24.00.00 で、秒の小数部は 0 桁です。 タイム・スタンプの *LOVAL 値は、0001-01-01-00.00.00 で、秒の小数部は 0 桁です。

タイム・スタンプのコーディング例については、以下を参照してください。

- 484 ページの『TIMESTAMP { (秒の小数部の桁数) }』
- 572 ページの『日付命令』
- 585 ページの『日付時刻データの転送』
- 711 ページの『ADDDUR (期間の加算)』

- 803 ページの『MOVE (転送)』
- 887 ページの『SUBDUR (期間減算)』

分離文字

MOVE、MOVEL または TEST 命令でタイム・スタンプ形式をコーディングする場合、 文字フィールドには区切り記号はオプションです。 区切り記号がないことを示すために、*ISO0 を指定することができます。区切り記号のない *ISO の使用例については、893 ページの『TEST (日付/時刻/タイム・スタンプのテスト)』を参照してください。

初期化

実行時にタイム・スタンプ・フィールドをシステム日付に初期化するには、定義仕様書に INZ(*SYS) を指定します。*SYS は、エクスポートされたフィールドには使用できません。 また、タイム・スタンプ・フィールドは、実行時に、リテラル、名前付き定数 または形象定数に初期化することもできます。

注 : 実行時初期化は、静的初期化の後に行われます。

オブジェクト・データ・タイプ

オブジェクト・データ・タイプによって Java オブジェクトを定義することができます。

自由形式定義では、最初のキーワードとして OBJECT キーワードを指定します。

```
DCL-S MyString OBJECT(*JAVA : 'java.lang.String');

DCL-PR bdcreate OBJECT EXTPROC(*JAVA : 'java.math.BigDecimal'
                               : *CONSTRUCTOR);
   val UCS2(100) CONST;
END-PR;
```

固定形式定義では、次のようにオブジェクト・データ・タイプ を指定します。 40 桁目にはデータ・タイプ O を指定します。 キーワード・セクションには CLASS キーワードを指定して、オブジェクトのクラスであることを指定します。

```
* Variable MyString is a Java String object.
D MyString        S               O   CLASS(*JAVA
D                                         :'java.lang.String')
D bdcreate        PR              O   EXTPROC(*JAVA
D                                           :'java.math.BigDecimal'
D                                           :*CONSTRUCTOR)
```

OBJECT キーワードと CLASS キーワードの両方とも、 最初のパラメーターには *JAVA を指定し、2 番目のパラメーターにはクラス名を 指定します。

オブジェクトが Java コンストラクターの戻りタイプである場合は、戻されたオブジェクトのクラスがメソッドのクラスと同じであるため、自由形式定義の OBJECT キーワードのパラメーターまたは固定形式定義の CLASS キーワードを指定する必要はありません。 代わりに、環境 *JAVA、クラス名、および プロシージャー名 *CONSTRUCTOR を指定した EXTPROC キーワードを指定します。

オブジェクトは、基底付きにはできません。 また、オブジェクトはデータ構造のサブフィールドにもなれません。

オブジェクトが配列またはテーブルの場合、実行時にロードされるのでなければなりません。 タイプが「オブジェクト」の実行時前とコンパイル時の配列とテーブルは許されません。

すべてのオブジェクトは、このオブジェクトがそのクラスのインスタンスと関連づけられていないことを示す *NULL に初期化されます。

オブジェクトの内容を変更するには、メソッドの呼び出しを使用する必要があります。 オブジェクトを使用して記憶域に直接アクセスすることはできません。

クラスは実行時に解決されます。 コンパイラーは、クラスが存在しているか、あるいは他のオブジェクトと互換性があるかのチェックは行ないません。

オブジェクト・フィールドを指定できる場所

オブジェクト・フィールドは、次の状態のときに使用できます。

自由形式の評価
EVAL 命令を使用して、ある Object 項目 (フィールドまたはプロトタイプされたプロシージャー) を、タイプ Object のフィールドに割り当てることができます。

自由形式の比較
あるオブジェクトを他のオブジェクトと比較することができます。 どんな比較でも指定することができますが、意味を持つのは次の比較のみです。

- 他のオブジェクトと等しいか等しくないか。 2 つのオブジェクトが等しいのは、それらが正確に同一のオブジェクトを表現している場合です。 同じ値を持つ 2 つの異なるオブジェクトは、等しくありません。

  2 つのオブジェクトの値が等しいかどうかをテストする場合は、次のように Java 'equals' メソッドを使用します。

```
D objectEquals    PR              N   EXTPROC(*JAVA
D                                     : 'java.lang.Object'
D                                     : 'equals')
C                   IF        objectEquals (obj1 : obj2)
C                    ...
C                   ENDIF
```

- *NULL と等しいか等しくないか。 オブジェクトは、そのクラスの特定のインスタンスに関連付けられていない場合に、*NULL に等しくなります。

自由形式の呼び出しパラメーター
プロトタイプの中のパラメーターがオブジェクトである場合は、オブジェクトを CALL 命令のパラメーターとしてコーディングできます。

注 :

1. オブジェクトは、入力フィールドまたは出力フィールドとしては無効です。
2. 妥当性検査の割り当てはチェックされません。 たとえば RPG では、クラス「数」のオブジェクトをクラス「ストリング」で定義されているオブジェクト変数に割り当てることができます。 これが正しくなかった場合、ユーザーが「ストリング」変数の使用を試みたときに Java エラーが発生します。

```
D Obj             S               O   CLASS(*JAVA
D                                         :'java.lang.Object')
D Str             S               O   CLASS(*JAVA
D                                         :'java.lang.String')
D Num             S               O   CLASS(*JAVA
D                                         :'java.math.BigDecimal')

 * Since all Java classes are subclasses of class 'java.lang.Object',
 * any object can be assigned to a variable of this class.
 * The following two assignments are valid.
C                   EVAL      Obj = Str
C                   EVAL      Obj = Num
 * However, it would probably not be valid to assign Str to Num.
```

図 *102.* オブジェクト・データ・タイプの例

基底ポインター・データ・タイプ

基底ポインターは、基底付き変数の記憶域を見付けるために使用します。 フィールド、配列、またはデータ構造を特定の基底ポインター変数を基礎とするように定義 し、基底ポインター変数が必要な記憶位置を指すように設定することによって、 記憶域がアクセスされます。

ポインター項目を定義するには、自由形式定義に POINTER キーワード を指定するか、または、固定形式仕様書のデータ・タイプ記入項目にアスタリスク (*) を 指定します。

たとえば、ポインター PTR1 を基礎とし、長さが 5 の文字フィールド である基底付き変数 MY_FIELD について考えてみます。 基底付き変数には、記憶域内に固定した位置がありません。 ポインターを使用して、この変数の記憶域内での現在位置を指示する必要が あります。

次が記憶域のある区域のレイアウトであるとします。

```
-------------------------------------------------------------
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
-------------------------------------------------------------
```

ポインター PTR1 を G を指すように設定すると、

```
   PTR1-------------------.
                          |
                          V
-------------------------------------------------------------
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
-------------------------------------------------------------
```

MY_FIELD は、'G' で始まる記憶域内の位置に置かれるように なり、その値は 'GHIJK' です。 ポインターが 'J' を指すように移動される と、MY_FIELD は 'JKLMN' になります。

```
               PTR1-------------------.
                                      |
                                      V
-------------------------------------------------------------
| A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
-------------------------------------------------------------
```

MY_FIELD が EVAL ステートメントによって 'HELLO' に変更されると、'J' で始まる 記憶域は変更されます。

```
               PTR1-------------------.
                                      |
                                      V
-------------------------------------------------------------
| A | B | C | D | E | F | G | H | I | H | E | L | L | O | O |
-------------------------------------------------------------
```

フィールドの基底ポインターを定義するには、定義仕様書の BASED キーワード (417 ページの『BASED(基底ポインター名)』 を参照) を使用してください。 基底ポインターは、基礎となっているフィールドと同じ有効範囲を持っています。

基底ポインター・フィールドの長さは 16 バイトでなければならず、16 バイト境 界で位置合わせされていなければなりません。 境界位置合わせについてのこの要件によって、データ構造のポインター・サブフィールドが前のフィールドに直接続かない原因となったり、また、複数オカレンス・データ 構造で発生が連続しない原因となる可能性があります。サブフィールドの位置合わせについて詳しくは、214 ページの『データ構造サブフィールドの位置合わせ』 を参照して ください。

基底ポインターのデフォルトの初期化値は *NULL です。

注 : 基底ポインターをコーディングする場合は、十分な大きさの記憶域を設定し、基礎となっているフィールドに正しいタイプのポインターが設定されたことを確認する必要があります。285 ページの図 107 は、基底ポインターをコーディングしない 例のいくつかを示しています。

注 : EVAL ptr = ptr + offset のように、式の中の ポインターにオフセットを加算したり、減算したりすることができます。 ポインターの算術を実行する場合には、指している項目の記憶域外をポインター が指し示さないようにするのは、ユーザーの責任であることに注意してください。 多くの場合、項目の前または後を指しても、 例外は発行されません。

2 つのポインターに減算を行って、両ポインター間のオフセットを決定する場合、 両ポインターは同じスペースまたは同じタイプの記憶域を指していなければなりません。 たとえば、静的記憶域内の 2 つのポインター、または自動記憶域内の 2 つの ポインター、または同じユーザー・スペース内の 2 つのポインターに対して 減算を行うことができます。

注 : データ構造にポインターが含まれていて、そのデータ構造が文字フィールドにコピーされる場合、 またはポインター・サブフィールドが定義されていない別のデータ構造にコピーされる場合には、 コピーされた値からポインター情報が脱落する可能性があります。 ポインターの実際の 16 バイト値はコピーされますが、システムには、16 バイト領域にポインターが含まれていることを示す追加の情報があります。 コピーされた値では、この追加の情報が設定されない可能性があります。

コピーされた値がオリジナルの値にコピーして戻されると、 オリジナルの値からポインターが脱落する可能性があります。

読み取り専用参照 (CONST キーワード) または値 (VALUE キーワード) を使用して、 ポインターを含むデータ構造をプロトタイプ・パラメーターとして渡した場合、 そのパラメーターが LIKEDS キーワードを使用してプロトタイプ化されるのではなく、 文字値としてプロトタイプ化されると、受け取られたパラメーターからポインター情報が脱落する可能性があります。 ポインターを含むデータ構造を戻した場合にも、似たような問題が発生する可能性があります。

基底ポインターの設定

基底付き変数の位置の設定または変更は、次のいずれかの方法で基底ポインターを設定 するか変更して行います。

- INZ(%ADDR(FLD)) によって初期化する。FLD は非基底付き変数。
- %ADDR(X) の結果を指すポインターを割り当てる。X は任意の変数。
- ポインターを別のポインターの値に割り当てる。
- ALLOC または REALLOC を使用する (例については、712 ページの『ALLOC (記憶域の割り振り)』、860 ページの『REALLOC (新しい長さでの記憶域の再割り振り)』、および「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」を参照してください)。
- ポインター演算を使用して、記憶域内でポインターを順方向または逆方向に 動かす。

```
EVAL        PTR = PTR + offset
```

("offset" は、ポインターが移動されるバイト単位での距離です。)

例

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++++
 *
 *  Define a based data structure, array and field.
 *  If PTR1 is not defined, it will be implicitly defined
 *  by the compiler.
 *
 *  Note that before these based fields or structures can be used,
 *  the basing pointer must be set to point to the correct storage
 *  location.
 *
D DSbased         DS                  BASED(PTR1)
D    Field1                    1  16A
D     Field2                     2
D
D ARRAY           S              20A   DIM(12) BASED(PRT2)
D
D Temp_fld        S                *   BASED(PRT3)
D
D PTR2            S                *   INZ
D PTR3            S                *   INZ(*NULL)
```

図 *103.* 基底付き構造およびフィールドの定義

以下は、ポインターにオフセットを加算したり、ポインターからオフセット を減算する方法を示すとともに、2 つのポインターの間のオフセットの差を 決定する方法を示します。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+...8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D.................................Keywords++++++++++++++++++++++++++++++
 *
D P1              s               *
D P2              s               *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
CL0N01+++++++++++++Opcode(E)+Extended Factor 2++++++++++++++++++++++++++++
 *
 * Allocate 20 bytes of storage for pointer P1.
C                   ALLOC     20              P1
 * Initialize the storage to 'abcdefghij'
C                   EVAL      %STR(P1:20) = 'abcdefghij'
 * Set P2 to point to the 9th byte of this storage.
C                   EVAL      P2 = P1 + 8
 * Show that P2 is pointing at 'i'.  %STR returns the data that
 * the pointer is pointing to up to but not incuding the first
 * null-terminator x'00' that it finds, but it only searches for
 * the given length, which is 1 in this case.
C                   EVAL      Result = %STR(P2:1)
C                   DSPLY                     Result            1
 * Set P2 to point to the previous byte
C                   EVAL      P2 = P2 - 1
 * Show that P2 is pointing at 'h'
C                   EVAL      Result = %STR(P2:1)
C                   DSPLY                     Result
 * Find out how far P1 and P2 are apart.  (7 bytes)
C                   EVAL      Diff = P2 - P1
C                   DSPLY                     Diff              5 0
 * Free P1's storage
C                   DEALLOC                   P1
C                   RETURN
```

図 *104.* ポインターの計算

282 ページの図 105 は、年間通算日が必要である場合に、年間通算日形式での日数を入手する方法を示しています。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7....+....
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H DATFMT(*JUL)
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D.................................Keywords++++++++++++++++++++++++++++++
 *
D JulDate         S               D   INZ(D'95/177')
D                                     DATFMT(*JUL)
D JulDS           DS                  BASED(JulPTR)
D  Jul_yy                        2  0
D  Jul_sep                       1
D  Jul_ddd                       3  0
D JulDay          S              3  0
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
CL0N01+++++++++++++Opcode(E)+Extended Factor 2++++++++++++++++++++++++++++
 *
 * Set the basing pointer for the structure overlaying the
 * Julian date.
C                   EVAL      JulPTR = %ADDR(JulDate)
 * Extract the day portion of the Julian date
C                   EVAL      JulDay = Jul_ddd
```

図 *105.* 年間通算日の入手

283 ページの図 106 は、ポインター、基底付き構造、およびシステム API の使用を示しています。 このプログラムは次のことを実行します。

1. 処理する必要があるライブラリーおよびファイル名を受け取ります。
2. QUSCRTUS API を使用してユーザー空間を作成します。

3. API (QUSLMBR) を呼び出して要求されたファイルのメンバーをリストします。
4. QUSPTRUS API を使用してユーザー空間に対するポインターを入手します。
5. ファイル中のメンバー数および最初および最後のメンバー名と一緒にメ ッセージを表示します。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D.................................Keywords++++++++++++++++++++++++++++++++
 *
D SPACENAME       DS
D                                 10    INZ('LISTSPACE')
D                                 10    INZ('QTEMP')
D ATTRIBUTE       S               10    INZ('LSTMBR')
D INIT_SIZE       S                9B 0 INZ(9999999)
D AUTHORITY       S               10    INZ('*CHANGE')
D TEXT            S               50    INZ('File member space')
D SPACE           DS                    BASED(PTR)
D SP1                          32767
 *
 * ARR is used with OFFSET to access the beginning of the
 * member information in SP1
 *
D ARR                             1    OVERLAY(SP1) DIM(32767)
 *
 * OFFSET is pointing to start of the member information in SP1
 *
D OFFSET                          9B 0 OVERLAY(SP1:125)
 *
 * Size has number of member names retrieved
 *
D SIZE                            9B 0 OVERLAY(SP1:133)
D MBRPTR          S                *
D MBRARR          S               10    BASED(MBRPTR) DIM(32767)
D PTR             S                *
D FILE_LIB        S               20
D  FILE           S               10
D  LIB            S               10
D WHICHMBR        S               10    INZ('*ALL      ')
D OVERRIDE        S                1    INZ('1')
D FIRST_LAST      S               50    INZ('    MEMBERS, +
D                                       FIRST =           , +
D                                       LAST =           ')
D IGNERR          DS
D                                 9B 0 INZ(15)
D                                 9B 0
D                                 7A
```

図 *106. API* によるポインターおよび基底付き構造の使用例

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
CL0N01+++++++++++++++Opcode(E)+Extended Factor 2++++++++++++++++++++++++++++
 *
 * Receive file and library you want to process
 *
C     *ENTRY        PLIST
C     FILE          PARM                    FILEPARM         10
C     LIB           PARM                    LIBPARM          10
 *
 * Delete the user space if it exists
 *
C                   CALL      'QUSDLTUS'                           10
C                   PARM                    SPACENAME
C                   PARM                    IGNERR
 *
 * Create the user space
 *
C                   CALL      'QUSCRTUS'
C                   PARM                    SPACENAME
C                   PARM                    ATTRIBUTE
C                   PARM                    INIT_SIZE
C                   PARM      ' '           INIT_VALUE        1
C                   PARM                    AUTHORITY
C                   PARM                    TEXT
 *
 * Call the API to list the members in the requested file
 *
C                   CALL      'QUSLMBR'
C                   PARM                    SPACENAME
C                   PARM      'MBRL0100'    MBR_LIST          8
C                   PARM                    FILE_LIB
C                   PARM                    WHICHMBR
C                   PARM                    OVERRIDE
 *
 * Get a pointer to the user-space
 *
C                   CALL      'QUSPTRUS'
C                   PARM                    SPACENAME
C                   PARM                    PTR
 *
 * Set the basing pointer for the member array
 * MBRARR now overlays ARR starting at the beginning of
 * the member information.
 *
C                   EVAL      MBRPTR = %ADDR(ARR(OFFSET))
C                   MOVE      SIZE          CHARSIZE          3
C                   EVAL      %SUBST(FIRST_LAST:1:3)  = CHARSIZE
C                   EVAL      %SUBST(FIRST_LAST:23:10) = MBRARR(1)
C                   EVAL      %SUBST(FIRST_LAST:41:10) = MBRARR(SIZE)
C     FIRST_LAST    DSPLY
C                   EVAL      *INLR = '1'
```

基底ポインターをコーディングする場合には、十分な大きさの記憶域を指示し、基礎となっているフィールドに正しいタイプのポインターが設定されていることを確認する必要があります。285 ページの図 107 は、基底ポインターをコーディングしない 方法を示しています。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+...
8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D.....................................Keywords+++++++++++++++++++++++++++++
  *
D chr10           S             10a   based(ptr1)
D char100         S            100a   based(ptr1)
D p1              S              5p 0 based(ptr1)
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
CL0N01+++++++++++++Opcode(E)+Extended Factor 2+++++++++++++++++++++++++++++
  *
  *
  * Set ptr1 to the address of p1, a numeric field
  * Set chr10 (which is based on ptr1) to 'abc'
  * The data written to p1 will be unreliable because of the data
  * type incompatibility.
  *
C                   EVAL      ptr1 = %addr(p1)
C                   EVAL      chr10 = 'abc'
  *
  * Set ptr1 to the address of chr10, a 10-byte field.
  * Set chr100, a 100-byte field, all to 'x'
  * 10 bytes are written to chr10, and 90 bytes are written in other
  * storage, the location being unknown.
  *
C                   EVAL      ptr1 = %addr(chr10)
C                   EVAL      chr100 = *all'x'
```

図 *107.* 基底ポインターをコーディングしない方法

プロシージャー・ポインター・データ・タイプ

プロシージャー・ポインターは、プロシージャーまたは関数を指すために使 用されます。 プロシージャー・ポインターは、プログラムにバインドされた入り口点を指します。 プロシージャー・ポインターは、定義仕様書で定義されます。

プロシージャー・ポインター項目を定義するには、自由形式定義 に POINTER(*PROC) キーワードを指定するか、または、 固定形式仕様書のデータ・タイプ記入項目にアスタリスク (*) を 指定し、PROCPTR キーワードも指定します。

プロシージャー・ポインター・フィールドは 16 バイトの長さでなければなら ず、16 バイト境界で位置合わせされていなければなりません。 境界位置合わせについてのこの要件によって、データ構造のポインター・サブフィ ールドが前のフィールドに直接続かない原因となったり、また、複数オカレンス・データ 構造で発生が連続しない原因となる可能性があります。サブフィールドの位置合わせについて詳しくは、214 ページの『データ構造サブフィールドの位置合わせ』を参照してください。

プロシージャー・ポインターのデフォルトの初期化値は *NULL です。

例

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
  *
  * Define a basing pointer field and initialize to the address of the
  * data structure My_Struct.
  *
 D  My_Struct      DS
 D   My_array                    10    DIM(50)
 D
 D Ptr1            S             16*   INZ(%ADDR(My_Struct))
  *
  * Or equivalently, defaults to length 16 if length not defined
  *
 D Ptr1            S               *   INZ(%ADDR(My_Struct))
  *
  * Define a procedure pointer field and initialize to NULL
  *
 D Ptr1            S             16*   PROCPTR INZ(*NULL)
  *
  * Define a procedure pointer field and initialize to the address
  * of the procedure My_Proc.
  *
 D Ptr1            S             16*   PROCPTR INZ(%PADDR(My_Proc))
  *
  * Define pointers in a multiple occurrence data structure and map out
  * the storage.
  *
 DDataS           DS                   OCCURS(2)
 D ptr1                            *
 D ptr2                            *
 D Switch                         1A
```

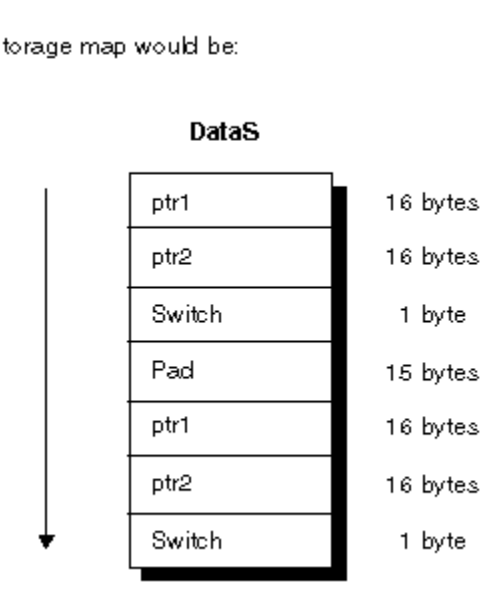


図 *108.* ポインターの定義

データベースのヌル値サポート

ILE RPG プログラムでは、外部記述データベース・ファイルからヌル値可能フィールドを処理する 3 つの異なる方法のいずれかを選択することができます。 これは、制御仕様書で ALWNULL キーワードを 使用する方法によって決まります (ALWNULL もコマンド・パラメーターとして 指定することができます)。

1. ALWNULL(*USRCTL) - ヌル値を含むレコードの読み取り、 書き出し、更新、および削除を行い、ヌル値キーによって検索および位置付けを行います。
2. ALWNULL(*INPUTONLY) - ヌル値を含む レコードを読み取って、ヌル値フィールド内のデータにアクセスします。

3. ALWNULL(*NO) - ヌル値を含むレコードを 処理しません。

注 : プログラム記述ファイルの場合に は、ALWNULL キーワードに指定された値とは無関係に、 レコード中のヌル値は常にデータ・マッピング・エラーの原因になります。

ヌル値可能フィールドおよびキー・フィールドに対するユーザー制御サポート

外部記述ファイルにヌル値可能フィールドが含まれてい て、ALWNULL(*USRCTL) キーワードが制御仕様書に指定されて いる場合には、以下が可能になります。

- 外部記述データベース・ファイルから、ヌル値を含むレコードを 読み取り、書き出し、更新し、削除すること。
- そのフィールドに関連した KFLD の演算項目 2 に標識を指定する ことによって、キー順命令を使用して、ヌル値キーを含むレコードを検索し、位置付けること。
- 式の右側の %NULLIND 組み込み関数を使用して、ヌル値可能フィールドが実際にヌルであるかどうかを判別すること。
- 標識の左側の %NULLIND 組み込み関数を使用して、出力または更新のためにヌル値可能フィールドをヌルに設定すること。

NULLIND キーワードを使用してヌル可能フィールドを定義した場合は、%NULLIND を使用する代わりに、NULLIND キーワードのパラメーターとして指定されたヌル標識を使用することもできます。

ヌル値を含むフィールドをプログラム内で正しく使用することは、 各自の責任で行っていただきます。 たとえば、ヌル値可能フィールドを EVAL 命令の右側で使用すると、MOVE を実行する前に、まず最初にそれがヌルであるかどうかをチェックする必要があります。これを行わないと、結果フィールド値が失われる恐れがあります。 また、ヌル値可能フィールドを、ヌル値可能と定義されていないフィールドを含む ファイル (たとえば、WORKSTN または PRINTER ファイル、あるいは プログラム記述ファイルなど) に出力するときにも、注意が必要です。

注 : ヌル値可能フィールドのヌル標識の値は、入力、出力、ファイル位置付け、および EVAL-CORR の命令の場合にのみ、考慮されます。 以下に、ヌル標識が考慮に入れられない命令の例をいくつか示します。

- ヌル値可能フィールドの DSPLY が、ヌル標識がオンになっている場合でも フィールドの内容を示します。
- ヌル値可能フィールドを別のヌル値可能フィールドに転送し、 演算項目 2 フィールドのヌル標識がオンになっている場合、結果フィールドは その演算項目 2 フィールドからデータを入手します。 結果フィールドの対応するヌル標識はオンに設定されません。
- ヌル値可能フィールドとの比較命令 (SORTA および LOOKUP を含む) は、 ヌル標識を考慮に入れません。

フィールドは、それが外部記述データベース・レコード内で ヌル値可能であり、プログラム内の定数として定義されていない場合 に、ヌル値可能と見なされます。

フィールドが RPG プログラム内でヌル値可能と見なされているときは、ヌル標識がそのフィールドに関連付けられます。 以下の点に注意してください。

- フィールドが複数オカレンス・データ構造またはテーブルである場合、 ヌル標識の配列がそのフィールドと関連付けられます。 各ヌル標識は、テーブルのデータ構造または要素のオカレンスに対応します。
- そのフィールドが配列要素である場合、配列全体がヌル値可能と 見なされます。 ヌル標識の配列はその配列と関連付けられ、各ヌル標識は配列要素に対応します。
- フィールドが、複数オカレンス・データ構造の配列サブフィールドの要素である 場合、ヌル標識の配列が、データ構造の各オカレンスごとに配列と 関連付けられます。

ヌル標識は、プログラムの初期化時にゼロに初期化されるので、 プログラムの実行開始時点ではヌル値可能フィールドにはヌル値は含まれていません。

外部記述データ構造内のヌル可能フィールド

注 : 以下の情報は、EXTNAME または LIKEREC データ構造の抜き出しタイプとして *NULL が指定されなかったときにのみ適用されます。

外部記述データ構造に使用されるファイルにヌル値可能フィールドが定義 されている場合、対応する RPG サブフィールドはヌル可能として定義されます。 同様に、レコード様式にヌル可能フィールドが含まれる場合、LIKEREC で定義されたデータ構造 にはヌル可能サブフィールドが含まれます。 あるデータ構造にヌル可能サブフィールドが含まれる場合には、LIKEDS を使用して そのデータ構造に似せて定義された別のデータ構造にも、ヌル可能サブフィールドが含まれるようになります。 ただし、LIKE キーワードを使用してあるフィールドを別のヌル可能フィールドに似せて定義しても、新規フィールドはヌル可能になりません。

ヌル値可能フィールドの入力

RPG プログラム内でヌル値可能であるフィールド では、DISK、SEQ、WORKSTN および SPECIAL ファイルに関して、以下の 内容が入力時に適用されます。

- ヌル値可能フィールドが外部記述ファイルから読み取られるとき、そのフィールドの ヌル標識は、そのフィールドがレコード内でヌルである場合オンに設定されます。 それ以外の場合は、ヌル標識はオフに設定されます。
- フィールド標識が指定されていて、ヌル値可能フィールドが空である 場合、すべてのヌル標識はオフに設定されます。
- フィールドが 1 つのファイル内でヌル値可能と定義されていて、別の ファイルではヌル値可能でないと定義されている場合、そのフィールド は RPG プログラム内ではヌル値可能と見なされます。 ただし、2 つ目のファイルを読み取るとき、そのフィールドに関連した ヌル標識は常にオフに設定されます。
- 結果フィールド内のデータ構造を使用する、プログラム記述ファイルから の入力命令は、そのデータ構造あるいはそのサブフィールドのいずれに関連した ヌル標識にも影響を与えません。
- プログラム記述ファイルの入力仕様を使用する、ヌル値可能フィールドの 読み取りは、常に、関連のヌル標識をオフに設定します。
- 「フィールド・レコード関係」標識が原因で、ヌル値可能フィールドが 読み取られるように選択されていない場合、関連のヌル標識は変更されません。
- ヌル可能フィールドを含むレコード様式またはファイルが入力命令 (READ、READP、READE、READPE、CHAIN) で使用され、結果フィールドでデータ構造がコーディングされる場合、ヌル可能データ構造のサブフィールドに関して %NULLIND で指定した値は命令によって変更されます。 ファイルの入力フィールドには、 その入力フィールドが入力操作でサブフィールドとして使用される場合を除き、%NULLIND の値は設定されません。

ヌル値可能フィールドは、突き合わせフィールドまたは制御レベル・フィールド としては使用できません。

ヌル値可能フィールドの出力

ヌル値可能フィールドが外部記述ファイルに書き出される (出力 または更新) とき、ヌル値は、そのフィールドのヌル標識がその操作時にオンに 設定されている場合には書き出されます。

ヌル値可能フィールドが外部記述データベース・ファイルに出力される、 あるいはそのファイルで更新されるとき、そのフィールドがヌルであると、 バッファー内に置かれている値はデータ管理によって無視されます。

注 : 出力時にオンになっているヌル標識を持つフィールドには、バッファーに転送 されるデータがあります。 これは、そのフィールドのヌル標識がオンになっている場合 でも、10 進データ・エラーあるいは基底ポインターが設定されていない、 などのエラーが発生することを意味します。

外部記述データベース・ファイルへの出力命令中に、ファイルに、プログラム 内ではヌル値可能と見なされ、ファイル内ではヌル値可能でないと見なされている フィールドが含まれている場合、そのヌル値可能フィールドに関連のヌル標識 は使用されません。

ヌル可能フィールドを含むレコード様式が WRITE または UPDATE 命令で使用され、 結果フィールドでデータ構造がコーディングされる場合、 出力または更新レコードのヌル・バイト・マップを設定するためにそのデータ構造のサブフィールドのヌル属性が使用されます。

ヌル可能フィールドを含むレコード様式が、%FIELDS が指定された UPDATE 命令で使用される場合、 ヌル・バイト・マップ情報は、指定されたフィールドのヌル属性から入手されます。

289 ページの図 109 は、ALWNULL(*USRCTL) オプションが使用されているときに、ヌル値を含むレコードの読み取り、書き出し、更新がどのように行われるかを示したものです。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7....+....
 *
 *
 * Specify the ALWNULL(*USRCTL) keyword on a control
 * specification or compile the ILE RPG program with ALWNULL(*USRCTL)
 * on the command.
 *
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 *H ALWNULL(*USRCTL)
 *
 * DISKFILE contains a record REC which has 2 fields: FLD1 and FLD2
 * Both FLD1 and FLD2 are null-capable.
 *
FDISKFILE  UF A E             DISK
 *
 * Read the first record.
 * Update the record with new values for any fields which are not
 * null.
C                   READ      REC                                    10
C                   IF        NOT %NULLIND(Fld1)
C                   MOVE      'FLD1'        Fld1
C                   ENDIF
C                   IF        NOT %NULLIND(Fld2)
C                   MOVE      'FLD2'        Fld2
C                   ENDIF
C                   UPDATE    REC
 *
 * Read another record.
 * Update the record so that all fields are null.
 * There is no need to set the values of the fields because they
 * would be ignored.
C                   READ      REC                                    10
C                   EVAL      %NULLIND(Fld1) = *ON
C                   EVAL      %NULLIND(Fld2) = *ON
C                   UPDATE    REC
 *
 * Write a new record where Fld 1 is null and Fld 2 is not null.
 *
C                   EVAL      %NULLIND(Fld1) = *ON
C                   EVAL      %NULLIND(Fld2) = *OFF
C                   EVAL      Fld2 = 'New value'
C                   WRITE     REC
```

図 *109.* ヌル値可能フィールドの入力および出力

キー順命令

ヌル値可能キー・フィールドがある場合、 *START または *END を指定した SETLL 命令を使用して、ファイルの 先頭または終わりに位置付けることができます。

ヌル値可能キー・フィールドがある場合には、KFLD 命令の演算項目 2 中の 標識を指定し、キー順入力命令の前に標識をオンに設定することによって、 ヌル値を含むレコードを検索することができます。 ヌル・キーを選択する必要がない場合には、標識をオフに設定します。

KFLD 命令の演算項目 2 が指定されていない場合、 そのキーについてはヌル・キー・バイト・マップ情報はゼロに設定されます。

ヌル値可能キー・フィールドを含むレコード様式を CHAIN、SETLL、READE、または READPE 命令で使用するときに、 キーを指定するために %KDS データ構造を使用する場合、 ヌル・キー・バイト・マップ情報は、%KDS の引数として指定されたデータ構造のサブフィールドのヌル属性から入手されます。

ヌル値可能キー・フィールドを含むレコード様式を CHAIN、SETLL、READE、または READPE 命令で使用するときに、 キー・フィールドのリストを使用する場合、 ヌル・キー・バイト・マップ情報は、指定されたキーのヌル属性から入手されます。

注 : ヌル値可能キー・フィールドのないファイルが キー順入出力命令で処理されるとき、検索引数の %NULLIND 値は ゼロでなければならず、そうでない場合はデータ・マッピング・エラーが発生します。

290 ページの図 110 および 291 ページの図 111 は、ヌル・キーによってレコードを位置付け、検索するためにどのようにキー付き命令が使用されるかを示しています。

```
  // Assume File1 below contains a record Rec1 with a composite key
  // made up of three key fields: Key1, Key2, and Key3.  Key2 and Key3
  // are null-capable.  Key1 is not null-capable.
  // Each key field is two characters long.
*..1....+....2....+....3....+....4....+....5....+....6....+....7....+..
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++
FFile1     IF   E             DISK
  // Define two data structures with the keys for the file
  // Subfields Key2 and Key3 of both data structures will be
  // null-capable.
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++
D Keys            DS                  LIKEREC(Rec1 : *KEY)
D OtherKeys       DS                  LIKEDS(keys)
  // Define a data structure with the input fields of the file
  // Subfields Key2 and Key3 of the data structures will be
  // null-capable.
D File1Flds       DS                  LIKEREC(Rec1 : *INPUT)
 /free
     // The null indicator for Keys.Key2 is ON and the
     // null indicator for Keys.Key3 is OFF, for the
     // SETLL operation below.  File1 will be positioned
     // at the next record that has a key that is equal
     // to or greater than 'AA??CC' (where ?? is used
     // in this example to indicate NULL)

     // Because %NULLIND(Keys.Key2) is ON, the actual content
     // in the search argument Keys.Key2 will be ignored.

     // If a record exists in File1 with 'AA' in Key1, a null
     // Key2, and 'CC' in Key3, %EQUAL(File1) will be true.

     Keys.Key1 = 'AA';
     Keys.Key3 = 'CC';
     %NULLIND(Keys.Key2) = *ON;
     %NULLIND(Keys.Key3) = *OFF;
     SETLL %KDS(Keys) Rec1;
     // The CHAIN operation below will retrieve a record
     // with 'JJ' in Key1, 'KK' in Key2, and a null Key3.
     // Since %NULLIND(OtherKeys.Key3) is ON, the value of
     // 'XX' in OtherKeys.Key3 will not be used.  This means
     // that if File1 actually has a record with a key
     // 'JJKKXX', that record will not be retrieved.

     OtherKeys.Key3 = 'XX';
     %NULLIND(Keys.Key3) = *ON;
     CHAIN ('JJ' : 'KK' : OtherKeys.Key3) Rec1;
     // The CHAIN operation below uses a partial key as the
     // search argument.  It will retrieve a record with 'NN'
     // in Key1, a null key2, and any value including a null
     // value in Key3.  The record is retrieved into the
     // File1Flds data structure, which will cause the
     // null flags for File1Flds.Key2 and File1Flds.Key3
     // to be changed by the operation (if the CHAIN
     // finds a record).

     Keys.Key1 = 'NN';
     %NULLIND(Keys.Key2) = *ON;
     CHAIN %KDS(Keys : 2) Rec1 File1Flds;
```

図 *110.* ヌル値可能キー・フィールドの処理の例

```
 * Using the same file as the previous example, define two
 * key lists, one containing three keys and one containing
 * two keys.
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq.
 C     Full_Kl       KLIST
 C                   KFLD                    Key1
 C                   KFLD      *IN02         Key2
 C                   KFLD      *IN03         Key3
 C     Partial_Kl    KLIST
 C                   KFLD                    Key1
 C                   KFLD      *IN05         Key2
  *
  * *IN02 is ON and *IN03 is OFF for the SETLL operation below.
  * File1 will be positioned at the next record that has a key
  * that is equal to or greater than 'AA??CC' (where ?? is used
  * in this example to indicate NULL)
  *
  * Because *IN02 is ON, the actual content in the search argument
  * for Key2 will be ignored.
  *
  * If a record exists in File1 with 'AA' in Key1, a null Key2, and
  * 'CC' in Key3, indicator 90 (the Eq indicator) will be set ON.
  *
 C                   MOVE      'AA'          Key1
 C                   MOVE      'CC'          Key3
 C                   EVAL      *IN02 = '1'
 C                   EVAL      *IN03 = '0'
 C     Full_Kl       SETLL     Rec1                                  90
```

```
  *
  * The CHAIN operation below will retrieve a record with 'JJ' in Key1,
  * 'KK' in Key2, and a null Key3.  Again, because *IN03 is ON, even
  * if the programmer had moved some value (say 'XX') into the search
  * argument for Key3, 'XX' will not be used.  This means if File1
  * actually has a record with a key 'JJKKXX', that record will not
  * be retrieved.
  *
 C                   MOVE      'JJ'          Key1
 C                   MOVE      'KK'          Key2
 C                   EVAL      *IN02 = '0'
 C                   EVAL      *IN03 = '1'
 C     Full_Kl       CHAIN     Rec1                               80
  *
  * The CHAIN operation below uses a partial key as the search argument.
  * It will retrieve a record with 'NN' in Key1, a null key2, and any
  * value including a null value in Key3.
  *
  * In the database, the NULL value occupies the highest position in
  * the collating sequence.  Assume the keys in File1 are in ascending
  * sequence.  If File1 has a record with 'NN??xx' as key (where ??
  * means NULL and xx means any value other than NULL), that record
  * will be retrieved.  If such a record does not exist in File1, but
  * File1 has a record with 'NN????' as key, the 'NN????' record will
  * be retrieved.  The null flags for Key2 and Key3 will be set ON
  * as a result.
  *
 C                   MOVE      'NN'          Key1
 C                   SETON                                        05
 C     Partial_Kl    CHAIN     Rec1                               70
```

図 *111. KLIST* が指定されたヌル・キー・フィールドの処理の例

ヌル値可能フィールドの入力専用サポート

外部記述入力専用ファイルにヌル値可能フィールドが含まれてい て、ALWNULL(*INPUTONLY) キーワードが制御仕様書に指定されて いる場合には、以下の条件が適用されます。

- データベース・ファイルからレコードを検索しているときに、そのレコード 内にヌル値を含むフィールドがいくつかある場合、ヌル値可能フィールドの データベースのデフォルト値は、ヌル値を含むそれらのフィールド内に置かれます。 デフォルトの値は、ユーザー定義の DDS のデフォルトの値またはシステムのデフォルトの値です。
- レコード内のフィールドにヌル値があるかどうかは判別する ことはできません。

- 入力フィールドが外部記述入力専用ファイルからのヌル値可能フィールドであ る場合には、制御レベル標識、突き合わせフィールド項目、およびフィールド標識を 入力仕様で使用することはできません。
- キー順入力演算命令の演算項目 1 が、 外部記述入力専用ファイル内の ヌル値可能キー・フィールドに対応するときは、キー順命令は使用できません。

注 : ALWNULL コマンド・パラメーターに指定された *INPUTONLY また は *YES についても、同じ条件が適用されます。

ALWNULL(*NO)

外部記述ファイルにヌル値可能フィールドが含まれてい て、ALWNULL(*NO) キーワードが制御仕様書に指定されて いる場合には、以下の条件が適用されます。

- ファイルから検索されたヌル値を含んでいるレコード は、データ・マッピング・エラーの原因となり、エラー・メッセージが出されま す。
- レコード中のデータにアクセスすることができず、レコード中のどのフィー ルドもヌル値を含んでいる入力レコードからの値では、更新することができません。
- このオプションでは、レコードの更新または追加のために、 ヌル値可能フィールド内にヌル値を入れることはできません。 ヌル値可能フィールドにヌル値を入れたい場合には、ALWNULL(*USRCTL) オプションを 使用します。

データベース・データ・マッピング・エラーのエラー処理

すべての入力または出力操作の場合に、データ・マッピング・エラーは重大エラー・メッセージが出される原因となります。 ブロック化された出力の場合は、ブロック中の 1 つまたは複数のレコードにデー タ・マッピング・エラーが含まれていて、ブロックの終わりに達する前にファイル がクローズされた場合には、重大エラー・メッセージが出され、システム・ダン プが作成されます。

数値フィールドの編集

編集は、以下の手段を提供します。

- 数値フィールドに句読点を付ける。これには、通貨記号、コンマ、ピリオド 、マイナス符号、および浮動マイナス符号の印刷が含まれます。
- フィールド符号を右端桁からフィールドの終わりに移動させる。
- ゼロのフィールドをブランクにする。
- 配列中のスペーシングを管理する。
- 日付を含む数値を編集する。
- 通貨記号を浮動させる。
- 印刷フィールドにアスタリスクを埋め込む。

本章では、非浮動数値フィールドにのみ適用します。 外部表示表現で 浮動フィールドを出力するには、出力仕様の 52 桁目にブランクを指定 します。 演算で浮動値の外部表示表現を入手するには、%EDITFLT 組み込み関数を 使用します。

フィールドは編集コードまたは編集語によって編集することができます。 出力仕様を使用して編集された形式でフィールドを印刷することも でき、また、組み込み関数の %EDITC (編集コード) および %EDITW (編集語) を 使用して、演算仕様書でフィールドの編集された値を入手することもできます。

編集されていないフィールドを印刷した場合には、そのフィールドは以下のように印刷されます。

- 浮動フィールドは、外部表示表現で印刷されます。
- 他の数値フィールドはゾーン数値表現で印刷されます。

以下の例は、数値出力フィールドの編集が必要な理由を示しています。

フィールドのタイプ	コンピューター内部のフィールド	未編集フィールドの印刷	編集済みフィールドの印刷
英数字	JOHN T SMITH	JOHN T SMITH	JOHN T SMITH
数値 (正)	0047652	0047652	47652
数値 (負)	004765K	004765K	47652-

未編集の英数字フィールドおよび未編集の正の数値フィールドは、印刷された時 に容易に読み取れますが、未編集の負の数値フィールドは、数値でない K が入っ ているために、判別しにくいことがあります。 この K は、その桁の数字 2 とフィールドの負符号を組み合わせたものです。 これらは、フィールド中に符号のための 1 桁を設定しなくてすむように結合され ています。 このような組み合わせはコンピューター内でフィールドを保管するには便利ですが、 出力を読みにくくしています。 したがって、印刷出力を読みやすくするために、数値フィールドを印刷する前に編 集することが必要になります。

編集コード

編集コードは、事前定義されたパターンに応じて数値フィールドを編集する手段 を提供します。 編集コードは、単純 (X、Y、Z)、組み合わせ (1 から 4、A から D、J から Q)、およびユーザー定義 (5 から 9) の 3 つのカテゴリーに分けられます。 出力仕様で、編集されるフィールドの 44 桁目に編集コードを 入力します。演算仕様書では、%EDITC 組み込み関数の 2 番目のパラメーターとして 編集コードを指定します。

単純編集コード

単純編集コードを使用すれば、句読点を指定する必要なしに数値フィールドを 編集することができます。これらのコードとその機能は次のとおりです。

- X 編集コードは、正のフィールドに 16 進数 F の符号を付け、負のフィールドに 16 進数の D の符号を付けます。 しかし、これはシステムによって行われるので、通常はこのコードを指定する 必要はありません。 先行ゼロは消去されません。 X 編集コードで %EDITC を使用して、数字を先行ゼロの文字に変換できます。 ただし、負の数の場合は予期しない結果になる場合があります。例えば、%EDITC(-00123:'X') の結果は「0012L」になることがあります。
- Y 編集コードは、通常、3 から 9 桁の日付フィールドを編集するために使用されます。 これは、日付フィールドの左端のゼロを最初の区切り記号の前の数字まで (ただ しその数字は含まない) 消去します。 日、月、および年を区切るためにスラッシュが挿入されます。 制御仕様書に 330 ページの『DATEDIT(形式 {区切り記号 })』 キーワードおよび 332 ページの『DECEDIT(*JOBRUN | '値')』 キーワードを指定することによって、編集形式を変更することができます。

  注 : Y 編集コードは *YEAR, *MONTH、および *DAY には有効でありません。
- Z 編集コードは、数値フィールドから符号 (プラスまたはマイナス) を取り 除き、数値フィールドの先行ゼロを消去するものです。 フィールドに小数点は入れられません。

組み合わせ編集コード

組み合わせ編集コード (1 から 4、A から D、J から Q) は、数値フィールドに句読 点を付けるためのものです。

制御仕様書の DECEDIT キーワードによって、小数点に使用される文字および先 行ゼロを消去するかどうかが決まります。 小数点を入れるかどうかとその位置は、ソース・フィールドの小数点の位置に よって決まります。 ソース・フィールドに小数点以下の桁数が指定され、ゼロ・バランスの消去が指定さ れている場合には、そのフィールドがゼロでない場合に のみ 小数点が 組み込まれます。 ゼロ・バランスの消去を指定する場合には、ゼロのフィールドは ブランクとして出力されます。

ゼロ・バランスの消去を指定せず、フィールドがゼロの場合には、 出力は次のいずれかになります。

- 小数点の後に n 個のゼロが続く。ここで n はフィールド中の小数点以下の桁数です。
- 小数点以下の桁数の指定がない場合には、フィールドの 1 の位にゼロ。

12 個の組み合わせ編集コードのいずれであっても、浮動通貨記号またはアスタリ スク保護を使用することができます。 浮動通貨記号は、最初の有効数字の左側に表示されます。 ゼロ・バランスを消去する編集コードが使用されている場合には、ゼロ・バラン スに浮動通貨記号は印刷されません。 ゼロ・バランスを消去する編集コードが使用される場合、ゼロ・バランスに 通貨記号は表示されません。

プログラムの通貨記号は、通貨記号が制御仕様書で CURSYM キーワードに よって指定されていない限り、ドル記号 ($) となります。

出力仕様で浮動通貨記号を指定するためには、出力仕様の 53 から 55 桁目に 通貨記号をコーディングし、編集したいフィールドについて 44 桁目に 編集コードをコーディングします。

組み込み関数 %EDITC の場合、3 番目のパラメーターに浮動通貨記号を指定 します。 プログラムに通貨記号を使用するには、*CURSYM を指定します。 他の通貨記号を 使用するには、長さ 1 の文字定数を指定します。

アスタリスク保護を使用すると、消去される各ゼロがアスタリスク によって置き換えられます。 ゼロ・バランス・ソース・フィールドは、アスタリスクだけからなるフィールドによって 置き換えられます。 出力仕様でアスタリスク保護を指定するには、出力仕様の 53 から 55 桁目に、 編集コードと一緒にアスタリスク定数をコーディングします。 組み込み関数の %EDITC を使用してアスタリスク保護を指定するには、3 番目の パラメーターとして *ASTFILL を指定します。

単純 (X, Y, Z) またはユーザー定義 (5 から 9) 編集コードでは、アスタリスク 充てんおよび浮動通貨記号を使用することはできません。

アスタリスク充てんの前に通貨記号を印刷することができます (固定通貨記号)。 これは、以下のコーディングによって出力仕様で行うことができます。

1. 最初の出力仕様の 53 桁目に通貨記号定数を入れます。 47 から 51 桁目に指定される終了位置には、編集されるフィールドの始めの前に 1 桁 のスペースが必要です。
2. 2 番目の出力仕様では、30 から 43 桁目に編集フィールドを入れ、44 桁目に 編集コードを入れ、47 から 51 桁目に編集フィールドの終了位置を入れて、53 から 55 桁目には '*' を入れます。

これは、通貨記号を %EDITC の結果に連結することに より、%EDITC 組み込み関数を使用して行うことができます。

```
     C           EVAL       X = '$' + %EDITC(N: 'A' : *ASTFILL)
```

出力仕様で、配列全体を印刷するために編集コードを使用する場合は、その配列の各要素 (最初の要素を除く) の前に 2 つのブランクが 入れられます。

注 : %EDITC 組み込み関数を使用して配列を編集することはできません。

294 ページの表 76 は、組み合わせ編集コードの機能を要約したものです。 これらのコードによって編集されたフィールドの形式が左側にリストされています。 負のフィールドには、図の最上部に示されているように、符号なし、CR、マイナ ス符号 (-)、または浮動マイナス符号による句読点を付けることができます。

表 76. 組み合わせ編集コード

		負の バランス標識			
グループ区切り記号付き印刷	**ゼロ・バランス印刷**	**符号なし**	**CR**	**-**	**浮動マイナス符号**
はい	はい	1	A	J	N
はい	いいえ	2	B	K	0
いいえ	はい	3	C	L	P
いいえ	いいえ	4	D	M	Q

ユーザー定義編集コード

編集コード 5 から 9 は、IBM で事前に定義されています。 これらをそのまま使用 したり、あるいはそれを削除して、ユーザー独自のものを作成することができます。 IBM 提供の編集コードの説明については、IBM i Information Center「プログラミング」のカテゴリーを参照してください。

ユーザー定義編集コードによって、編集語の使用が必要になる一般的な編集上 の問題を、編集語を使用せずに処理することができます。 同じ編集語を繰り返しコーディングする代わりに、ユーザー定義編集コードを使 用することができます。 これらのコードは、CRTEDTD (編集記述作成) CL コマンドによってシステムで定 義されます。

小数点以下の桁数を持つように定義されたフィールドを編集する場合には、その フィールドの小数部と整数部の両方について編集マスクを持つ編集語を使用する ようにしてください。 プログラムの中でユーザー定義編集コードを指定した場合 には、そのユーザー定義編集コードに対して加えられたシステムでのすべての変 更は、プログラムが再コンパイルされるまで反映されないことに留意してくださ い。 CRTEDTD について詳しくは、IBM i Information Center「プログラミング」のカテゴリー を参照してください。

編集に関する考慮事項

編集コードのいずれかを指定する場合には、以下の点に留意してください。

- 印刷装置以外のファイルのフィールドの編集には、注意が必要です。 印刷装置ファイル以外のファイルのフィールドを編集する場合には、編集するフィール ドの内容と、それらに対して実行した操作の影響を理解しておいてください。 た とえば、そのようなファイルを入力として使用した場合には、編集によって書き 出されるフィールドは数値フィールドではなく文字フィールドになると考えなけ ればなりません。
- 編集操作によって追加されるデータについて考慮する必要があります。 追加された句読点の数によって、 編集された値の全体の長さが長くなります。 これらの追加される文字を出力仕様で編集するときに考慮しておかないと、 出力フィールドがオーバーラップすることがあります。
- 出力用に指定した終了位置は、 編集後のフィールドの終了位置になります。 たとえば、編集コード J から M のいずれかを指定した場合には、マイナス符号 (ま たは、フィールドが正であればブランク) の位置が終了位置となります。
- 符号なし数値フィールドの場合であっても、符号のための 文字位置がコンパイラーによって割り当てられます。

編集コードの要約

295 ページの表 77 は、編集コードとそれによって提供されるオプションを要約したものです。 この表を簡略化したものが出力仕様の 45 から 70 桁目に印刷されています。297 ページの表 78 は、編集後のフィールドを示しています。

298 ページの表 79 は、出力用に終了位置を指定した同じフィールドに対し て異なる編集コードを使用した場合に、それらの編集コードが与える影響を示したものです。

表 77. 編集コード

				DECEDIT キーワード・パラメーター				
編集コード	**コンマ**	**小数点**	**負のバランスの符号**	**'.'**	**','**	**'0,'**	**'0.'**	**ゼロ抑制**
1	はい	はい	符号なし	.00 または 0	,00 または 0	0,00 または 0	0.00 または 0	はい
2	はい	はい	符号なし	ブランク	ブランク	ブランク	ブランク	はい
3		はい	符号なし	.00 または 0	,00 または 0	0,00 または 0	0.00 または 0	はい
4		はい	符号なし	ブランク	ブランク	ブランク	ブランク	はい
5-9[1]								

表 77. 編集コード (続き)

				DECEDIT キーワード・パラメーター				
編集コード	**コンマ**	**小数点**	**負のバランスの符号**	**'.'**	**','**	**'0,'**	**'0.'**	**ゼロ抑制**
A	はい	はい	CR	.00 または 0	,00 または 0	0,00 または 0	0.00 または 0	はい
B	はい	はい	CR	ブランク	ブランク	ブランク	ブランク	はい
C		はい	CR	.00 または 0	,00 または 0	0,00 または 0	0.00 または 0	はい
D		はい	CR	ブランク	ブランク	ブランク	ブランク	はい
J	はい	はい	- (負符号)	.00 または 0	,00 または 0	0,00 または 0	0.00 または 0	はい
K	はい	はい	- (負符号)	ブランク	ブランク	ブランク	ブランク	はい
L		はい	- (負符号)	.00 または 0	,00 または 0	0,00 または 0	0.00 または 0	はい
M		はい	- (負符号)	ブランク	ブランク	ブランク	ブランク	はい
N	はい	はい	- (浮動マイナス符号)	.00 または 0	,00 または 0	0,00 または 0	0.00 または 0	はい
O	はい	はい	- (浮動マイナス符号)	ブランク	ブランク	ブランク	ブランク	はい
P		はい	- (浮動マイナス符号)	.00 または 0	,00 または 0	0,00 または 0	0.00 または 0	はい
Q		はい	- (浮動マイナス符号)	ブランク	ブランク	ブランク	ブランク	はい
X[2]								
Y[3]								はい
Z[4]								はい

注 :

1. これらはユーザー定義編集コードです。
2. X 編集コードは、正の値に 16 進数 F の符号を付けるものですが、 これはシステムによって行われるので、通常はこのコードを指定する必要は ありません。
3. Y 編集コードは、日付フィールドの左端のゼロを最初の区切り記号の前 の数字 (ただしその数字は含まない) まで消去します。 長さが 7 桁のフィールドの左端 2 桁のゼロを消去します。 また、Y 編集コードは、次のパターンにしたがって、月、日、および年の間に スラッシュ (/) を挿入します。

```
    nn/n
    nn/nn
    nn/nn/n
    nn/nn/nn
  nnn/nn/nn
    nn/nn/nnnn
  nnn/nn/nnnn
 nnnn/nn/nn
nnnnn/nn/nn
```

4. Z 編集コードは、数値フィールドから符号 (プラスまたはマイナス) を 取り除き、先行ゼロを消去するものです。

表 *78.* 編集コードの使用法の例

編集コード	**プラス数字-2 10 進数位置**	**プラス数字-いいえ 10 進数位置**	**マイナス数字-3 10 進数位置**	**マイナス数字-いいえ 10 進数位置**	**ゼロバランス 2 10 進数位置**	**ゼロバランスいいえ 10 進数位置**
編集せず	1234567	1234567	00012ƀ[5]	00012ƀ[5]	000000	000000
1	12,345.67	1,234,567	.120	120	.00	0
2	12,345.67	1,234,567	.120	120		
3	12345.67	1234567	.120	120	.00	0
4	12345.67	1234567	.120	120		
5-9[1]						
A	12,345.67	1,234,567	.120CR	120CR	.00	0
B	12.345.67	1,234,567	.120CR	120CR		
C	12345.67	1234567	.120CR	120CR	.00	0
D	12345.67	1234567	.120CR	120CR		
J	12,345.67	1,234,567	.120-	120-	.00	0
K	12,345,67	1,234,567	.120-	120-		
L	12345.67	1234567	.120-	120-	.00	0
M	12345.67	1234567	.120-	120-		
N	12,345.67	1,234,567	-.120	-120	.00	0
O	12,345,67	1,234,567	-.120	-120		
P	12345.67	1234567	-.120	-120	.00	0
Q	12345.67	1234567	-.120	-120		
X[2]	1234567	1234567	00012ƀ[5]	00012ƀ[5]	000000	000000
Y[3]			0/01/20	0/01/20	0/00/00	0/00/00
Z[4]	1234567	1234567	120	120		

表 78. 編集コードの使用法の例 (続き)

編集コード	プラス 数字-2 10 進数 位置	プラス 数字-いいえ 10 進数 位置	マイナス 数字-3 10 進数 位置	マイナス 数字-いいえ 10 進数 位置	ゼロ バランス 2 10 進数 位置	ゼロ バランス いいえ 10 進数 位置

注 :

1. これらはユーザー定義の編集コードです。
2. X 編集コードは、正の値に 16 進数 F の符号を付けるものですが、 これはシステムによって行われるので、通常はこのコードを指定する必要は ありません。
3. Y 編集コードは、日付フィールドの左端のゼロを最初の区切り記号の前 の数字 (ただしその数字は含まない) まで消去します。 長さが 7 桁のフィールドの左端 2 桁のゼロを消去します。 また、Y 編集コードは、次のパターンにしたがって、月、日、および年の間に スラッシュ (/) を挿入します。

```
   nn/n
   nn/nn
   nn/nn/n
   nn/nn/nn
  nnn/nn/nn
   nn/nn/nnnn  Format used with M, D or blank in position 19
  nnn/nn/nnnn  Format used with M, D or blank in position 19
 nnnn/nn/nn    Format used with Y in position 19
nnnnn/nn/nn    Format used with Y in position 19
```

4. Z 編集コードは、数値フィールドから符号 (プラスまたはマイナス) を 取り除き、数値フィールドの先行ゼロを消去するものです。
5. b はブランクを表します。 これは、負のゼロに対応する 印刷可能文字がない場合に表示されることがあります。

表 79. 終了位置に対する編集コードの影響

	負数、小数部 2 桁、 終了位置の指定 10								
	出力印刷位置								
編集コード	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
編集せず				0	0	4	1	K[1]	
1					4	.	1	2	
2					4	.	1	2	
3					4	.	1	2	
4					4	.	1	2	
5-9[2]									
A			4	.	1	2	C	R	
B			4	.	1	2	C	R	
C			4	.	1	2	C	R	
D			4	.	1	2	C	R	
J				4	.	1	2	-	
K				4	.	1	2	-	

表 79. 終了位置に対する編集コードの影響 (続き)

	負数、小数部 2 桁、 終了位置の指定 10								
	出力印刷位置								
編集コード	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**
L				4	.	1	2	-	
M				4	.	1	2	-	
N				-	4	.	1	2	
O				-	4	.	1	2	
P				-	4	.	1	2	
Q				-	4	.	1	2	
X				0	0	4	1	K[1]	
Y			0	/	4	1	/	2	
Z						4	1	2	

注 :

1. K は負数 2 を表します。
2. これらはユーザー定義の編集コードです。

編集語

上記の編集コードを使用することでは満たすことのできない編集要件があった 場合には、編集語を使用することができます。 編集語は、出力仕様の 53 から 80 桁目に指定される文字リテラルまたは名前付き定数です。 これは数値の編集パターンを記述するもので、以下のものを直接指 定することができます。

- ブランク・スペース
- コンマと小数点、およびそれらの位置
- 不要なゼロの消去
- 先行アスタリスク
- 通貨記号およびその位置
- 定数文字の追加
- 負符号または負の標識としての CR の出力

編集語はテンプレートとして使用され、システムはこれをソース・データに適用し て出力を生成します。

編集語は、出力仕様で直接に指定するか、あるいは出力仕様の編集語フィール ドに名前付き定数名として表すことで、名前付き定数として指定 することができます。 演算仕様書のフィールドの編集された値は、組み込み関数の %EDITW (編集語) を 使用して入手することができます。

編集語の最大長は 115 文字です。

編集語のコーディング方法

編集語を使用して出力するには、次のように出力仕様を コーディングしてください。

位置指定
記入

21 から 29
条件付け標識を入れることができます。

30 から 43
編集するデータを取り出す数値フィールドの名前を入れます。

44
編集コード。 編集語を使用してソース・データを編集する場合には、 ここはブランクでなければなりません。

45
この桁の「B」は、ソース・データが編集および出力された後にゼロまたはブランクに設定されることを指示します。 B を指定しなければ、ソース・データは変更されずにそのまま残ります。

47 から 51
出力レコードのフィールドの終了 (右端) 位置を識別します。

53 から 80
編集語。 26 文字までの長さとすることができますが、 名前付き定数の場合を除き、アポストロフィで囲まなければなりません。 53 桁目に先行アポストロフィを記入するか、あるいはそこから名前付き定数の名前を始めてください。 名前付き定数以外の編集語は、54 桁目から始めなければなりません。

演算仕様書で編集語を使用して編集するには、編集する値を最初の パラメーターとして指定し、さらに編集語を 2 番目のパラメーターとして指定 して、組み込み関数 %EDITW を使用します。

編集語の各部分

編集語は、本体、状況、および拡張部分の 3 つの部分から構成されます。 次の図は、編集語の 3 つの部分を示しています。

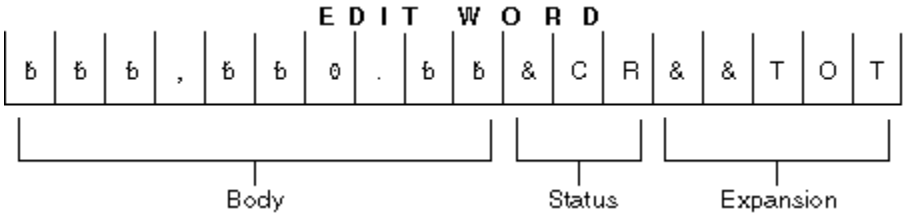


図 *112.* 編集語の各部分

本体 は、ソース・データ・フィールドから編集された結果に転送される 数字のためのスペースです。 本体は、編集語の左端桁から始まります。 編集語本 体のブランク (に 1 つのゼロまたはアスタリスクを加えた) 桁数は、編集される ソース・データ・フィールドの桁数より大きいまたは等しくなければなりません。 本体は、数字で置き換えることができる、右端の文字で終わります。

状況 は、2 文字の英字 CR かマイナス符号 (-) のいずれかの負標識のために使用することができるスペースを定義します。指定された負標識は、ソース・データが負の場合にのみ出力されます。 編集語の中の 最後の置き換え可能文字 (ブランク、ゼロ抑制文字) と負標識の間のすべての文字 も、ソース・データが負の場合にのみ、負標識と一緒に出力されます。ソース・デー タが正の場合には、これらの状況桁はブランクに置き換えられます。 CR または - 標識のない編集語には、状況桁はありません。

状況は、編集語の中の最後のブランクの後に記入しなければなりません。 最後のブランクの後に複数の CR が続いている場合には、 最初の CR だけが状況として取り扱われ、残りの CR は定数として取り扱われます。 マイナス符号が状況と見なされるためには、 それが編集語の中の最後の文字でなければなりません。

拡張部分 は、状況の後に記入される一連のアンパーサンドおよび定数文字です。 アンパーサンドは、出力上ではブランク・スペースによって置き換えられ、 定数はそのまま出力されます。 状況が指定されない場合には、拡張部分が本体に続きます。

編集語の本体の形成

次の文字は、編集語の本体で使用された時に特殊な意味を持ちます。

ブランク

ブランクは、編集される値の対応する桁からの文字によって置き換えられます。 ブランクの桁は、 数字の桁として参照されます。

小数点およびコンマ

小数点およびコンマは、編集語の最初の有効数字の左側にある場合を除いて、 編集語中の位置と同じ編集済み出力フィールド中の相対位置に入れられます。 編集語の最初の有効数字の左側にあった場合には、ブランクとされるか、または アスタリスクで置き換えられます。

以下の各例では、先行ゼロはすべて消去され (デフォルトの値)、小数点は、その 左側に有効数字がなければ表示されません。

編集語	ソース・データ	編集結果には次のように現れます。
'bbbbbbb'	0000072	bbbbb72
'bbbbbbb.bb'	000000012	bbbbbbbb12
'bbbbbbb.bb'	000000123	bbbbbb1.23

ゼロ

編集語の本体の最初のゼロは、ゼロ抑制終了文字として解釈されます。 このゼロは、 ゼロ抑制を終わらせるために入れます。編集語に入れられた後続のゼロは、定数として扱われます (以下の「定数」を参照してください)。

ソース・データ中のすべての先行ゼロは、ゼロ抑制終了文字の桁 (この桁を含めて) まで消去されます。 ゼロ抑制終了文字の桁またはそれより左側に現れた有効数字は、出力されます。

編集語	ソース・データ	編集結果には次のように現れます。
'bbb0bbbbbb'	00000004	bbbb000004
'bbb0bbbbbb'	012345	bbbb012345
'bbb0bbbbbb'	012345678	bb12345678

先行ゼロがゼロ抑制終了文字の桁またはそれより右側の桁まで拡張している場合には、 ゼロ抑制終了文字の桁がブランクで置き換えられます。 このことは、ソース・データに存在しているのと同じ数の先行ゼロが出力に 現れるようにしたいとき、 編集語の本体をソース・データより広くしなければならないことを意味します。

編集語	ソース・データ	編集結果には次のように現れます。
'0bbb'	0156	b156
'0bbbb'	0156	b0156

ゼロ抑制終了文字の右側に入れられた定数 (コンマおよび小数点を含む) は、ソース・データがなくても出力されます。 ゼロ抑制終了文字の左側にある定数は、 その定数より左に位置することになる有効数字がソース・データに 含まれている場合にだけ出力されます。

編集語	ソース・データ	編集結果には次のように現れます。
'bbbbbb0.bb'	000000001	bbbbbbb.01
'bbbbbb0.bb'	000000000	bbbbbbb.00
'bbb,b0b.bb'	00000012	bbbbbb0.12
'bbb,b0b.bb'	00000123	bbbbbb1.23
'b0b,bbb.bb'	00000123	bb0,001.23

アスタリスク

編集語の本体の最初のアスタリスクも、ゼロ抑制を終らせます。編集語に入れられた後続のアスタリスクは、定数として扱われます (以下の「定数」を参照してください)。 編集語中のこのアスタリスクより後に

あるゼロも、すべて定数として取り扱われます。 1 つの編集語にはゼロ抑制終了文字は 1 つだけ可能であり、 その文字は編集語の中の最初のアスタリスクまたは 最初のゼロです。

ゼロ抑制終了文字としてアスタリスクを使用した場合には、消去される先行ゼ ロが出力上ではすべてアスタリスクで置き換えられます。そうでない場合には、上記の「ゼロ」で説明されたのと同じ方法で、アスタリスクにより先行ゼロが消去されます。

編集語	ソース・データ	編集結果には次のように現れます。
'*bbbbbb.bb'	000000123	*000001.23
'bbbbb*b.bb'	000000000	******0.00
'bbbbb*b.bb**'	000056342	****563.42**

アスタリスクの位置より後に現れている先行ゼロは、先行ゼロとして出力さ れる点に注意してください。 アスタリスクの位置にあるものも含めて、消去され た先行ゼロだけがアスタリスクによって置き換えられます。

通貨記号

編集語中の最初のゼロ (ゼロ抑制終了文字) の直前にある通貨記号は、浮動通 貨記号と呼ばれます。 出力においては、すべての先行ゼロが消去され、最高位の 有効数字の直前に通貨記号が現れます。

編集語	ソース・データ	編集結果には次のように現れます。
'bb,bbb,b$0.bb'	000000012	bbbbbbbbb$.12
'bb,bbb,b$0.bb'	000123456	bbbb$1,234.56

編集語の最初の桁に通貨記号を入れた場合には、出力上では常に同じ位置に通 貨記号が現れます。 これは固定通貨記号と呼ばれます。

編集語	ソース・データ	編集結果には次のように現れます。
'$b,bbb,bb0.bb'	000123456	$bbbb1,234.56
'$bb,bbb,0b0.bb'	000000000	$bbbbbbbb00.00
'$b,bbb,*bb.bb'	000123456	$****1,234.56

編集語中でこれ以外の位置にあって、その直後にゼロ抑制終了文字が続いていない通過記号は、定数として取り扱われます (以下の「定数」を参照)。

アンパーサンド

編集されたフィールド中にブランクを生じさせます。 次の例は、電話番号の編 集などに使用することができます。 定数 AREA を印刷するためには、最初の 桁にゼロが必要である点に注意してください。

編集語	ソース・データ	編集結果には次のように現れます。
'0AREA&bbb&NO.&bbb-bbbb'	4165551212	bAREAb416bNO.b555-1212

定数

編集語の本体に記入された上記以外の文字は、すべて定数として取り扱われます。 出力時にすべての定数の左側に有効数字または先行ゼロが 入れられるようなソース・データの場合には、その定数が出力上に現れます。 そうでない場合には、定数の出力は抑制されます。 コンマおよび小数点も、定数の場合と同じ規則に従っています。 以下の例では、ソース・データ内の有効数字の桁数だけでなく ゼロ抑制終了文字の存在が定数の出力に影響している点に注意してください。

以下の編集語は、小切手の印刷などに使用することができます。 2 番目のアス タリスクが定数として取り扱われ、また、3 番目の例では、最初の有効数字 の前の定数は出力されない点に注意してください。

編集語	ソース・データ	編集結果には次のように現れます。
'$bbbbbb**DOLLARS&bb&CTS'	000012345	$****123*DOLLARSb45bCTS
'$bbbbbb**DOLLARS&bb&CTS'	000000006	$********DOLLARSb06bCTS
'$bbbbbbb&DOLLARS&bb&CTS'	000000006	$bbbbbbbbbbbbbbbbb6bCTS

日付は、いずれかの編集語を使用して編集される場合があります。

編集語	ソース・データ	編集結果には次のように現れます。
'bb/bb/bb'	010388	b1/03/88
'0bb/bb/bb'	010389	b01/03/89

編集語の最初のゼロまたはアスタリスクの後にあるゼロまたはアスタリスクは 、定数として取り扱われる点に注意してください。 これは - および CR でも 同じです。

編集語	ソース・データ	編集結果には次のように現れます。
'bb0.bb000'	01234	b12.34000
'bb*.bb000'	01234	*12.34000

編集語の状況の形成

次の文字が編集語の状況桁で使用された場合には、特殊な意味を持ちます。

アンパーサンド

編集された出力フィールド中にブランクを生じさせます。 編集後の出力フィールドにアンパーサンドを入れることはできません。

CR またはマイナス符号

編集される出力の符号がプラス (+) の場合には、それらの桁はブラン クとされます。 編集される出力フィールドの符号がマイナス (-) の場合には、それらの桁 はそのまま残ります。

次の例では、負の値の指示が追加されています。 マイナス符号は、フィールド 内の数値が負の場合にのみ印刷されます。 CR 記号はマイナス符号と同じ機能を果たします。

編集語	ソース・データ	編集結果には次のように現れます。
'bbbbbbb.bb-'	000000123-	bbbbbb1.23-
'bbbbbbb.bb-'	000000123	bbbbbb1.23b

最後の置き換え可能文字と - または CR 記号の間の定数が印刷されるの は、フィールドが負の場合だけで、そうでない場合には、これらの位置には ブランクが表示されます。 ブランクを表すアンパーサンドの使用に注意してください。

編集語	ソース・データ	編集結果には次のように現れます。
'b,bbb,bb0.bb&30&DAY&CR'	000000123-	bbbbbbbbb1.23b30bDAYbCR
'b,bbb,bb0.bb&30&DAY&CR'	000000123	bbbbbbbb1.23bbbbbbbbbb

編集語の拡張部分の形式設定

編集語の拡張部分の文字は、常に使用されます。 拡張部分にブランクを含め ることはできません。 編集された結果にブランクが必要であった場合に は、編集語の本体にアンパーサンドを指定してください。

その番号の値と一緒に表示するために、定数が追加される場合が あります。

編集語	ソース・データ	編集結果には次のように現れます。
'b,bb0.bb&CR&NET'	000123-	bbbb1.23bCRbNET
'b,bb0.bb&CR&NET'	000123	bbbb1.23bbbbNET

語の中間に CR があった場合には、負のフィールド値の指示として検出される ことがある点に注意してください。 SECRET のような語が必要な場合には、次の例 にあるコーディングを使用してください。

編集語	ソース・データ	編集結果には次のように現れます。
'bb0.bb&SECRET'	12345-	123.45bSECRET
'bb0.bb&SECRET'	12345	123.45bbbbbET
'bb0.bb&CR&&SECRET'	12345	123.45bbbbbSECRET

編集語のコーディング規則の要約

出力仕様の編集語に適用される規則は、次のとおりです。

- 44 桁目 (編集コード) は、ブランクとしなければなりません。
- 30 から 43 桁目 (フィールド名) には、数値フィールドの名前を入れなければな りません。
- 編集語は、名前付き定数でない限り、アポストロフィで囲まなければな りません。 53 桁目に先行アポストロフィを入力するか、あるいはそこから名前付 き定数の名前を始めてください。 編集語自体は 54 桁目から始めてください。

一般的に編集語に適用される規則は、次のとおりです。

- 編集語には、編集されるフィールドより多くの数字桁 (ブランクに加えて最 初のゼロまたはアスタリスクの分) を入れることができますが、編集されるフィ ールドより少なくすることはできません。 編集語の数字桁の数が編集されるフィ ールドの数字桁の数より多い場合には、編集の前に先行ゼロがそのフィールドに 追加されます。
- ソース・データからの先行ゼロが必要な場合には、編集語を編集されるフィール ドより 1 桁多くし、編集語の最高位の桁にゼロを入れなければなりません。
- 編集語の本体では、ブランクとゼロ抑制終了文字 (ゼロおよびアスタリスク) のみが 数字桁としてカウントされます。 浮動通貨記号は数字桁としてカウントされません。
- 浮動通貨記号を使用する場合には、編集語に含まれるブランクの数とゼロ抑制終了文字の数の合計 (数字桁) が編集されるフィールドの桁数より大きいかま たは等しくなければなりません。
- 左端のゼロまたはアスタリスクの後に続くすべてのゼロまたはアスタリスク は、定数として取り扱われます。これらは置き換え可能文字ではありません。
- 符号なし整数フィールドを編集している場合には、DB および CR を使用する ことができ、それらは常にブランクとして印刷されます。

外部記述ファイルの編集

外部記述ファイルの出力を編集するには、RPG IV 仕様書の代わりにデータ記述仕様 (DDS) に編集コードを入れてください。 データ記述仕様書への編集コードの指定方法について詳しくは、IBM i Information Center「データベースおよびファイル・システム」のカテゴリーを参照してください。 しかし、編集コードを指定した外部記述ファイルをプログラム記述出力ファイル として書き出す場合には、出力仕様に編集を指定しなければなりません。 この 場合には、データ記述仕様書のすべての編集コードが無視されます。

仕様書

RPG IV でのステートメントのコーディング

この部では、RPG IV の仕様書について説明します。 最初に、キーワードの構文および継続の規則のようにいくつかの仕様書に共通す る情報について説明します。 次に、プログラムに入力しなければならない順序に従って各仕様書が説明されます。 各仕様書の説明では、仕様書のすべてのフィールドがリストされ、記入可能なす べての項目を説明しています。

仕様書

RPG IV のソースは、各種の仕様書にコーディングされます。 各仕様書には、特定の機能のセットがあります。

本書には、個別の RPG IV 仕様書に関する詳細な説明が含まれています。 各フィールドとその記入可能な項目について説明しています。541 ページの『操作』では、507 ページの『演算仕様書』で説明される演算仕様書にコーディングされる命令コードについて説明しています。

RPG IV の仕様タイプ

RPG IV プログラムにコーディングすることができるソース・レコードには、メイン・ソース・セクション、サブプロシージャー・セクション、およびプログラム・データ・セクションの 3 つのグループがあります。 **メイン・ソース・セクション**には、 モジュール内の H、F、D、I、C、および O 仕様書からなる最初のセット、または、自由形式でそれらに相当するものが 含まれます。 制御仕様書で MAIN または NOMAIN が指定されている場合、このセクションにはサイクル・メイン・プロシージャーは含まれません。 したがって、いかなる実行可能演算も含むことができません。 キーワード MAIN または NOMAIN が指定されていない場合、ここには独立型プログラムまたはサイクル・メイン・プロシージャーに相当するものが入ります。 すべてのモジュールに、サブプロシージャーのコーディングの有無にかかわらず 、メイン・ソース・セクションが必要です。

サブプロシージャー・セクションには、 モジュール内でコーディングされるすべてのサブプロシージャーを定義する仕様書が含まれます。 **プログラム・データ・セクション**には、コンパイル時に指定されるデータが入っているレコードが含まれます。

RPG IV 言語は、位置に依存するコードと自由形式のコードの混合で構成されます。 キーワード (制御、ファイル記述、定義、およびプロシージャー) をサポートす る仕様書によって、キーワード・フィールドに自由形式を使用することができます。 制御ステートメント、ファイル記述ステートメント、定義ステートメント、および プロシージャー・ステートメントでは、完全に自由形式の仕様書が許可されます。完全に自由形式の仕様書は、拡張演算項目 2 をサポートする演算コードが指定された演算ステートメントにも許可されます。515 ページの『自由形式の演算ステートメント』を参照してください。 それ以外の場合、RPG IV 記入項目は位置が固定されています。 これを表すために、RPG IV コードのそれぞれの例は、最上部に横方向のスケールを付けたリスト形式になっています。

以下の図は、各グループに記入することができるソース・レコードのタイプとその順 序を示しています。

ノート

RPG IV ソースは、306 ページの表 80 に示されている順序でシステムに入力する必要があります。 どの仕様タイプが欠けていてもかまいませんが、メイン・ソース・セクションからの少な くとも 1 つの仕様タイプが存在していなければなりません。

ファイル記述仕様と定義仕様を混在させることができます。

表 80. RPG IV ソース・プログラム中のソース・レコードとその順序	
ソース・セクション	**仕様書の順序**
メイン・ソース・セクション	H　制御 F、D　ファイル記述および定義 I　入力 C　演算 O　出力
サブプロシージャー・セクション	(プロシージャーごとに繰り返す) P　プロシージャー F、D　ファイル記述および定義 C　演算 P　プロシージャー
** 形式を使用する場合のプログラム・データ	** ファイル変換レコード ** 代替照合順序レコード ** コンパイル時配列およびテーブル・データ
**TYPE 形式を使用する場合のプログラム・データ	(任意の順序で指定) **CTDATA ARRAY1 コンパイル時配列データ **FTRANS ファイル変換レコード **CTDATA TABLE2 コンパイル時テーブル・データ **ALTSEQ 代替照合順序レコード **CTDATA ARRAY3 コンパイル時配列データ

メイン・ソース・セクション仕様書

H

制御 (見出し) 仕様書には、プログラム生成およびコンパイル済みプログラ ムの実行に関する情報を指定します。 この仕様書の記入項目の説明については、318 ページの『制御仕様書』を参照してください。

F

ファイル記述仕様書は、プログラムのグローバル・ファイルを定義します。 この仕様書の記入項目の説明については、349 ページの『ファイル仕様書』を参照してください。

D

定義仕様書には、プログラムで使用される項目を定義します。 この仕様書には、配列、テーブル、データ構造、サブフィールド、定数、独立フィールド、プロトタイプとそれらのパラメーター、およびプロシージャー・インターフェースとそれらのパラメーターが定義されます。 この仕様書の記入項目の説明については、390 ページの『定義仕様書』を参照してください。

I

入力仕様には、入力ファイルのレコードおよびフィールドを記述し、プロ グラムによるそのレコードおよびフィールドの使用方法を指示します。 この仕様書の記入項目の説明については、493 ページの『入力仕様』を参照してください。

C

演算仕様書には、プログラムによって実行される演算を記述し、その実行順 序を指示します。 演算仕様書では、特定の入力および出力操作を制御することができます。 この仕様書の記入項目の説明については、507 ページの『演算仕様書』を参照してください。

O

出力仕様には、レコードおよびフィールドを記述し、それらがプログラム によって書き出される時期を指示します。 この仕様書の記入項目の説明については、517 ページの『出力仕様』を参照してください。

サブプロシージャー仕様書

P

プロシージャー仕様書には、プロトタイプ・プログラムまたはプロシージャ ーのプロシージャー・インターフェース定義を記述します。 この仕様書の記入項目の説明については、532 ページの『プロシージャー仕様書』を参照してください。

F

ファイル記述仕様書は、サブプロシージャーでローカルに使用されるファイルを定義します。 この仕様書の記入項目の説明については、349 ページの『ファイル仕様書』を参照してください。

D

定義仕様書には、プロトタイプ・プロシージャーで使用される項目を定義し ます。 この仕様書には、プロシージャー・インターフェース定義、入り口パラメーター、 およびその他のローカル項目が定義されます。 この仕様書の記入項目の説明については、390 ページの『定義仕様書』を参照してください。

C

演算仕様書によって、プロトタイプ・プロシージャーの論理が実行されます。 この仕様書の記入項目の説明については、507 ページの『演算仕様書』を参照してください。

プログラム・データ

すべてのソース仕様書の後に、プログラム・データが入っているソース・レコードが続 きます。 データ・セクションの最初の行は ** で始まっていなければなりません。

必要に応じて、420 ページの『CTDATA』、337 ページの『FTRANS {(*NONE | *SRC) }』、または 322 ページの『ALTSEQ {(*NONE | *SRC | *EXT) }』 のキーワードを任意の数だけ指定することによって、** の後に続くプログラム・データのタイプを指示することができます。 プログラム・データを適切なキーワードと関連付けることによって、プログラム ・データのグループをソース・レコードの後に任意の順序で入れることができます。

各入力レコードの最初の記入項目は 1 桁目から始める必要があります。 レコード全体に記入項目を埋め込む必要はありません。 未使用の 記入項目に関連した配列要素はデフォルト値で初期化されます。

コンパイル時配列レコードの記入について詳しくは、234 ページの『配列ソース・レコードに関する規則』を参照してください。ファイル変換について詳しくは、195 ページの『ファイル変換』を参照してください。代替照合順序について詳しくは、263 ページの『代替照合順序』を参照してください。

自由形式ステートメント

ソースの先頭行に **FREE が含まれている場合、ソースは完全自由形式であり、1 桁目から行末までの任意の桁に自由形式ステートメントを指定することができます。 行の長さに制限はありません。

そうでない場合は、ソースは桁制限付きであり、自由形式ステートメントは 8 桁目から 80 桁目までにコーディングされます。 6-7 桁目はブランクでなければなりません。

ほとんどの場合、自由形式ステートメントは、 CTL-OPT、DCL-F、DCL-DS、READE、または DCL-PROC などの命令コードで始まります。

場合によっては、命令コードの指定は不要です。

- 演算ステートメントでは、ステートメントの最初にある名前 が自由形式演算内でサポートされている命令コードのいずれかと同じでない限り、 EVAL 命令コードおよび CALLP 命令コードは省略されます。 例えば、代入のターゲットの名前が READ である場合は、EVAL 命令コード が必要です。
- サブフィールドの定義では、サブフィールドの名前が自由形式演算内でサポートされている 命令コードのいずれかと同じでない限り、DCL-SUBF 命令コードは省略可能です。
- パラメーターの定義では、パラメーターの名前が自由形式演算内でサポートされている 命令コードのいずれかと同じでない限り、DCL-PARM 命令コードは省略可能です。

自由形式ステートメントはセミコロンで終わります。

通常、自由形式の行で // の後にあるすべてのテキスト は注記と見なされます。 ただし、// が文字リテラルの中にある場合は、リテラルの一部であると見なされます。 次の例では、1,2,4、および6は注記ですが、3さらに5は、// がリテラルの中にあるため注記ではありません。

3の後に + が続いていますが、これは、リテラルが次の行に続くことを示しています。5の後に、リテラルを終了する引用符が続き、+ はリテラルが次の行のリテラルと連結されることを示します。

注記ではないコードの後に続いて同じ行にある // は、行末コメントと呼ばれることもあります。 2さらに4の注記は、行末の注記です。

```
// comment 1
DCL-S string // comment 2
      CHAR(10);
string = 'abc // not-comment 3 +
         def'; // comment 4
string = 'ghi // not-comment 5' +
         'jkl'; // comment 6
```

注 : 桁制限付き自由形式ソースでは、80 桁以降に表示されるすべてのテキストの前に // が必要です。 ただし、ソース行に継続文字リテラルが含まれる場合、80 桁以降のテキストは許可されていません。

次の例の 80 桁以降の注釈付きテキストは、1、2、3、5 行目については有効です。これらは、1で示されています。しかし、4 行目については有効ではありません。これは、0で示されています。4 行目は継続文字リテラルで終了しているためです。

```
 ...+... 1 ...+... 2 ...+... 3 ... // 7 ...+... 8 ...+... 9 ...+
1           a = 'abc';                          // comment 1
2           b = 'abc' +                         // comment 1
3               'def';                          // comment 1
4           c = 'abc +                          // comment 0
5              def';                            // comment 1
```

自由形式演算でサポートされている命令コードのリストについては、541 ページの『命令コード』を参照してください。

各タイプの自由形式ステートメントについては、以下を参照してください。

- 319 ページの『自由形式の制御ステートメント』
- 350 ページの『自由形式のファイル定義ステートメント』
- 391 ページの『自由形式の定義ステートメント』
- 515 ページの『自由形式の演算ステートメント』
- 532 ページの『自由形式のプロシージャー・ステートメント』

完全自由形式ステートメント

特殊指示 **FREE は、ソース・メンバー全体が完全自由形式コードであることを示します。 完全自由形式コードは、1 桁目から行の末尾までの任意の桁を使用できます。 完全自由形式ソースではソース行の長さに事実上制限はありません。

**FREE は、ソースの先頭行の 1 桁目にのみ指定できます。 行の残りの部分はブランクでなければなりません。

ソースの先頭行の 1 桁目に **FREE が指定されないと、ソース・メンバー全体が桁制限付きになります。311 ページの『共通記入項目』を参照してください。

完全自由形式ソースでは、6 桁目と 7 桁目は特殊状況を含みません。 ソース・ファイルのすべての桁は自由形式 RPG コードでなければなりません。ただし、ソースの終わりに置かれるコンパイル時データ、ファイル変換レコード、および代替照合順序レコードは除きます。307 ページの『プログラム・データ』を参照してください。

ソース・モードは単一のソース・ファイルにのみ適用されます。 コピー・ファイルは、ファイルの先頭行に **FREE が指定されない限り、桁制限付きソース・モードになっているものと見なされます。 コピー・ファイルが終了すると、ソース・モードは /COPY または /INCLUDE 指示を含むソース・ファイルのモードに戻ります。

TAG 命令や入力仕様および出力仕様などの固定形式ステートメントを使用する必要がある場合は、それらのステートメントをコピー・ファイルに入れる必要があります。

/FREE 指示と /END-FREE 指示は、完全自由形式ソース内では使用できません。

メイン・ソース・ファイルとコピー・ファイルをマージして 1 つの新しいソース・メンバーを作る RPG プリプロセッサーを提供する場合は、/COPY 指示または /INCLUDE 指示を含むファイルとは異なるソース・モードを持つコピー・ファイルの処理について、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」の付録 E を参照してください。

自由形式ステートメント内の条件付き指示

自由形式演算ステートメントを除くすべての自由形式ステートメントの内部で、/IF、/ELSEIF、/ELSE、および /ENDIF 指示 を使用できます。

ただし、以下の規則が適用されます。

- /IF、/ELSEIF、または /ELSE 指示の後でステートメントが始まっている場合、 次の指示の前に、そのステートメントの最後のセミコロンが指定されなければなりません。

  次のコードは無効です。 DSPLY ステートメントは /IF 指示の 後で始まっているので、この DSPLY ステートメントに対するセミコロンが /ELSE 指示の 前に指定される必要があります。

```
/IF DEFINED(TRUE)
    DSPLY
/ELSE
    print
/ENDIF
      ('start');
```

  以下のコードは有効です。 DSPLY ステートメントは /IF 指示の後で始まっており、この DSPLY ステートメントに対するセミコロンが /ELSE 指示の前に指定されているので、DSPLY ステートメント全体が /IF 指示と /ELSE 指示の間に指定されています。 同様に、print の呼び出し全体が /ELSE 指示と /ENDIF 指示の間に指定されています。

```
/IF DEFINED(TRUE)
    DSPLY ('start');
/ELSE
    print ('start');
/ENDIF
```

- 条件付きグループの /IF がステートメント内で開始される場合、そのステートメントの最後のセミコロンの前に /ENDIF を指定する必要があります。

次のコードでは、DCL-S ステートメントが始まった後に /IF 指示が 指定されていて、この DCL-S ステートメントに対する最後のセミコロンの後に /ENDIF が 指定されているため、このコードは無効です。

```
DCL-S name
  /IF DEFINED(TRUE)
    CHAR(10);
  /ELSE
    VARCHAR(10);
  /ENDIF
```

条件付きグループ全体がステートメント内にあるため、以下は有効です。 ステートメントのセミコロンは、/ENDIF の後に置きます。

```
DCL-S name
  /IF DEFINED(TRUE)
    CHAR(10)
  /ELSE
    VARCHAR(10)
  /ENDIF
 ;
```

認識しておく必要のある固定形式と自由形式の相違点

更新に使用可能なファイルは自動的に削除に使用可能にはならない

固定形式ファイル定義の「ファイル・タイプ」記入項目に U を指定する と、ファイルは更新と削除の両方の操作を許可するようにオープンされます。 自由形式ファイル定義では、削除操作を許可するようにファイルがオープンされるようにするには、USAGE(*DELETE) を 明示的に指定する必要があります。

EXTNAME および EXTFLD キーワードに対する引用符で囲まれていない名前

引用符で囲まれていない名前の解釈は、固定形式と自由形式で異なります。

- 固定形式では、引用符で囲まれていない名前は、外部名であると解釈されます。
- 自由形式では、引用符で囲まれていない名前は、名前付き定数であると解釈されます。

ヒント : 固定形式のコードと 自由形式のコードが混在している場合、名前ではなくリテラルを使用するように、固定形式の EXTNAME および EXTFLD キーワードを 変更することを検討してください。 例えば、EXTNAME(myfile) を EXTNAME('MYFILE') に、EXTFLD(myextfld) を EXTFLD('MYEXTFLD') に変更します。

DTAARA キーワードに対する引用符で囲まれていない名前

最初のオペランドの引用符で囲まれていない名前の解釈は、 固定形式と自由形式で異なります。

- 固定形式では、DTAARA(myDtaaara) のように DTAARA の最初のオペランドが名前である場合、 このオペランドは、システム上のデータ域の名前 *LIBL/MYDTAARA であると解釈されます。
- 自由形式では、DTAARA(myDtaaara) のように DTAARA の最初のオペランドが名前である場合、 このオペランドは、データ域の名前を含んでいる文字フィールドまたは名前付き定数の名前であると 解釈

されます。 データ域の名前を直接指定するには、 名前を引用符で囲んで DTAARA('MYDTAARA') のように指定します。

ヒント : 固定形式のコードと 自由形式のコードが混在している場合、名前ではなくリテラルを使用するように、固定形式の DTAARA キーワードを 変更することを検討してください。 DTAARA(myDtaara) を DTAARA('MYDTAARA') に変更します。

OVERLAY キーワードは、他のサブフィールドをオーバーレイするためにのみ使用できます。

- 自由形式では、POS キーワードを使用してデータ構造内のサブフィールドの位置を指定する必要があります。 OVERLAY キーワードは、あるサブフィールドを別のサブフィールドにオーバーレイするためにのみ使用できます。

END-DS、END-PI、および END-PR をコーディングするための要件

サブフィールドをコーディングすることが許可されている データ構造 (LIKEDS も LIKEREC キーワードも指定されていないデータ構造) の 場合、別のステートメントとして、または、サブフィールドをコーディング しない場合は DCL-DS ステートメントの末尾に、END-DS を忘れずに指定する必要があります。

プロシージャー・インターフェース の場合、別のステートメントとして、 または、パラメーターをコーディングしない場合は DCL-PI ステートメントの 末尾に、END-PI を忘れずに指定する必要があります。

プロトタイプの場合、 別のステートメントとして、または、パラメーターをコーディングしない場合は DCL-PR ステートメントの 末尾に、END-PR を忘れずに指定する必要があります。

定義名の末尾にある省略符号

定義仕様またはプロシージャー仕様において、名前の末尾に省略符号をコーディング し、ステートメントの残りの部分を次の仕様に指定することに慣れている場合、 この方法は自由形式では可能ではありません。 名前が省略符号で終わっている場合、 次の行にある最初のキーワードが名前の一部であると解釈されます。

例	有効かどうか	フィールド名	説明
`dcl-s name... char(10);`	いいえ	NAMECHAR	プログラマーは名前が 2 番目の行に継続されることを意図していませんが、 コンパイラーは CHAR が名前の一部であると 想定します。 フィールド名は NAMECHAR であり、(10) は構文エラーであると見なされます。
`dcl-s name char(10);`	はい	NAME	名前は省略符号を付けずにコーディングされています。 したがって、コンパイラーは次行で名前の終わりを検索しません。
`dcl-s continued... name char(10);`	はい	CONTINUEDNAME	名前は次行に継続されているため、「continued」の末尾にある省略符号は有効です。

共通記入項目

以下の記入項目は、桁制限付きソースのプログラム・データに先行するすべての RPG 仕様書に共通したものです (309 ページの『完全自由形式ステートメント』を参照してください)。

- 1 から 5 桁目は注記に使用できます。
- 自由形式ステートメントの場合、6 から 7 桁目はブランクでなければなりません。
- 固定形式ステートメントの場合、6 桁目に仕様書タイプが指定されます。 次の英字コードを使用することができます。

記入
仕様書タイプ

H
制御

F
ファイル記述

D
定義

I
入力

C
演算

O
出力

P
プロシー ジャー・

- 注記ステートメント
  - 7 桁目にはアスタリスク (*) が入ります。これによって、仕様の記入内容とは関係なく、その行を注記行として指示します。 自由形式演算仕様書では、注記には // が使用できます。 コンパイラーは、固定形式仕様書上の // で始まるすべての行を、注記と見なします。 // 文字までの 6 桁さえブランクであれば、// はどの桁から始まっても構いません。
  - 6 から 80 桁目はブランクです。
  - 自由形式仕様での注記について詳しくは、 自由形式の注記を 参照してください。
- 7 から 80 桁目はブランクで、6 桁目には有効な指定が入ります。 これは注記でない有効な行で、順序の規則が適用されます。

キーワードの構文

キーワードは、パラメーターを持っていなかったり、 任意指定パラメーターを持ったり、または必須パラメーターを持ったりします。 キーワードの構文は次のとおりです。

```
    Keyword(parameter1 : parameter2)
```

各パラメーターの説明は以下のとおりです。

- 1 つまたは複数のパラメーターは括弧 () で囲みます。

  注 : パラメーターがない場合、括弧 を指定してはなりません。
- コロン (:) を使用して複数のパラメーターを区切ります。

任意指定パラメーターと必須パラメーターを示すために、 以下の国別の規則を使用します。

- 中括弧 { } は任意指定パラメーターまたはパラメーターの任意指定要素を示します。
- 省略記号 (...) は パラメーターが反復可能であることを示します。
- コロン (:) はパラメーターを区切り、複数のパラメーターを指定できることを示します。 コロンで区切られたすべてのパラメーターは、中括弧で囲まれていない限り、必須パラメーターです。
- 縦線 (|) は、キーワードに 1 つのパラメーターしか指定でき ないことを示します。
- キーワード・パラメーターを区切るブランクは、1 つまたは複数のパラメーター を指定できることを示します。

注 : 中括弧、省略記号、および縦線は、キーワード構文の一部 ではないため、ソースに入れてはなりません。

表 *81.* キーワード表記の例

表記法	表記の使用例	記述	入力ソースの例
中括弧 {}	PRTCTL (データ構造 {:*COMPAT })	「データ構造」パラメーターは必須であり、*COMPAT パラメーターは 任意指定です。	PRTCTL (data_struct1)
中括弧 {}	TIME(形式 {区切り記号 })	「形式 {区切り記号 }」パラメーターは必須ですが、このパラメーターの {区切り記号 } の部分は任意指定です。	TIME(*HMS&)
コロン (:)	RENAME(外部形式:内部形式)	「外部形式」パラメーターと「内部形式」パラメーターは 必須です。	RENAME (nameE: nameI)
省略記号 (...)	IGNORE(レコード様式 {:レコード様式... })	「レコード様式」パラメーターは必須で、複数を指定することができます。	IGNORE (recformat1: recformat2: recformat3)
縦線 (|)	FLTDIV {(*NO | *YES)}	*NO または *YES を指定するか、あるいはパラメーターを指定しません。	FLTDIV
ブランク	OPTIONS(*OMIT *NOPASS *VARSIZE *STRING *TRIM *RIGHTADJ)	*OMIT、*NOPASS、*VARSIZE、*STRING、*TRIM、または *RIGHTADJ のいずれかの パラメーターを指定する必要があり、また、複数のパラメーターを指定することもできます。	OPTIONS(*OMIT : *NOPASS : *VARSIZE : *TRIM : *RIGHTADJ)

継続の規則

継続させることができるフィールドは次のとおりです。

- 任意の自由形式ステートメント
- 制御仕様書のキーワード・フィールド
- ファイル記述仕様のキーワード・フィールド
- 定義仕様書のキーワード・フィールド
- 演算仕様書の拡張演算項目 2 フィールド
- 出力仕様の定数/編集語フィールド
- 定義またはプロシージャー仕様書の名前フィールド

継続に関する一般的な規則は次のとおりです。

- 継続記入行は継続される仕様 (6 桁目の H、F、D、C、または O) に関して有 効な行でなければなりません。
- 仕様を複数の行に渡って継続するとき、リテラルまたは名前を分割 する場合を除いて、特殊文字を使用してはなりません。 例えば、 以下のペアは等価です。 最初のペアで、正符号 (+) は行の末尾に現れていても、 演算子です。 2 番目のペアでは、 正符号は継続文字です。

```
C                   eval      x = a + b
C                   eval      x = a +
C                                 b

C                   eval      x = 'abc'
C                   eval      x = 'ab+
C                                  c'
```

- 継続された行の間にはブランク行、空の仕様行、または注記行だけを使用する ことができます。
- 完全なトークンの後で継続を行うことができます。 トークンとは次のものを指します。
  - 名前 (例えば、キーワード、ファイル名、フィールド名)
  - 括弧

  - 区切り文字 (:)
  - 式の演算子
  - 組み込み関数
  - 特殊語
  - リテラル
- リテラルの途中で継続させることもできます。
  - 文字、日付、時刻、およびタイム・スタンプ・リテラルの場合
    - ハイフン (-) は、継続されたフィールドの最初に使用可能な位置に継続があ ることを指示します。
    - プラス (+) は、継続されたフィールドの最初の位置あるいはそれ以降の最初 のブランクでない文字に継続があることを指示します。
  - 図形リテラルの場合
    - ハイフン (-) またはプラス (+) のいずれかを使用して継続を指示すること ができます。
    - リテラルの各セグメントはシフトアウトおよびシフトイン文字によって 囲まなければなりません。
    - 図形リテラルをアセンブルした場合には、最初のシフトアウトおよび最後 のシフトイン文字だけが含まれます。
    - 図形リテラルに使用される継続文字とは無関係に、リテラルは継続記入行の シフトアウト文字の後の最初の文字から継続されます。 シフトアウト文字の前にあるスペースは無視されます。
  - 数値リテラルの場合
    - 継続文字は使用されません。
    - 数値リテラルは、継続されたフィールドの継続記入行の数字または小数点か ら継続されます。
    - 数値リテラルの継続は自由形式ステートメント内では許可されません。
  - 16 進数リテラルおよび UCS-2 リテラルの場合
    - ハイフン (-) またはプラス (+) のいずれかを使用して継続を指示することがで きます。
    - リテラルは次の行の最初のブランクでない文字から継続されます。
- 継続は、自由形式の記入項目の名前の中でも行うことができます。
  - 定義およびプロシージャー仕様書の名前記入項目の中で。名前記入項目内の名前の継続についての詳細は、317 ページの『定義およびプロシージャー仕様書の名前フィールド』を参照してください。
  - ファイルおよび定義仕様書のキーワード記入項目の中で。
  - 演算の拡張演算項目 2 の中で。

  修飾名は、以下のようにピリオドで分割することができます。

```
C                   EVAL      dataStructureWithALongName.
C                               subfieldWithAnotherLongName = 5
```

  名前をピリオドで分割しない場合は、部分名の終わりでブランクを挿入せずに、 省略記号 (...) をコーディングします。

例

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D                                     Keywords-cont++++++++++++++++++++++++++
* Define a 10 character field with a long name.
* The second definition is a pointer initialized to the address
* of the variable with the long name.
D QuiteLongFieldNameThatCannotAlwaysFitInOneLine...
D                 S             10A
D Ptr             S               *   inz(%addr(QuiteLongFieldName...
D                                             ThatCannotAlways...
D                                             FitInOneLine))
D ShorterName     S              5A

*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
CL0N01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++++
C                           Extended-factor2-++++++++++++++++++++++++++++++
* Use the long name in an expression
* Note that you can split the name wherever it is convenient.
C                   EVAL          QuiteLongFieldName...
C                                 ThatCannotAlwaysFitInOneLine = 'abc'

* You can split any name this way
C                   EVAL          P...
C                                 tr = %addr(Shorter...
C                                 Name)
```

制御仕様書のキーワード・フィールド

制御仕様書での継続に関する規則は次のとおりです。

- 次の制御仕様書の 7 桁目以降に指定を 継続します。

例

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H DATFMT(
H       *MDY&
H       )
```

ファイル仕様書のキーワード・フィールド

ファイル仕様書での継続に関する規則は次のとおりです。

- 次のファイル仕様書の 44 桁目以降に指定を継続します。
- 継続記入行の 7 から 43 桁目はブランクでなければなりません。

例

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++++++
F.....................................Keywords++++++++++++++++++++++++++++++
F                                     EXTIND
F                                     (
F                                     *INU1
F                                     )
```

定義仕様書のキーワード・フィールド

定義仕様書でのキーワードの継続に関する規則は次のとおりです。

- 指定された継続文字に応じて、次の定義仕様書の 44 桁目以降に指定を 継続します。
- 継続記入行の 7 から 43 桁目はブランクでなければなりません。

例

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D                                   Keywords-cont++++++++++++++++++++++++
DMARY             C                 CONST(
D                                   'Mary had a little lamb, its -
 * Only a comment or a completely blank line is allowed in here
D                                   fleece was white as snow.'
D                                   )
 * Numeric literal, continues with the first non blank in/past position 44
 *
DNUMERIC          C                 12345
D                                     67
 * Graphic named constant, must have shift-out in/past position 44
DGRAF             C                 G'oAABBCCDDi+
D                                    oEEFFGGi'
```

演算仕様書の拡張演算項目 *2*

演算仕様書での継続に関する規則は次のとおりです。

- 次の演算仕様書の 36 桁目以降に指定を継続します。
- 継続記入行の 7 から 35 桁目はブランクでなければなりません。

例

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
CL0N01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++++
C                             Extended-factor2-++++++++++++++++++++++++++++++
C                   EVAL      MARY='Mary had a little lamb, its +
 * Only a comment or a completely blank line is allowed in here
C                               fleece was white as snow.'
 *
 *  Arithmetic expressions do not have continuation characters.
 *  The '+' sign below is the addition operator, not a continuation
 *  character.
C
C                   EVAL      A = (B*D)/ C +
C                             24
 *  The first use of '+' in this example is the concatenation
 *  operator.  The second use is the character literal continuation.
C                   EVAL      ERRMSG = NAME +
C                                       ' was not found +
C                                       in the file.'
```

自由形式仕様書

自由形式仕様書での継続に関する規則は次のとおりです。

- 自由形式行は次の行に続けることができます。 ステートメントは、セミコロンに出会うまで続きます。

例

```
  /FREE
            time = hours * num_employees
                     + overtime_saved;
  /END-FREE
```

- 名前またはリテラルが継続される場合を除いて、 ステートメントを次の行に続けるための継続文字は使用されません。 名前およびリテラルの継続について詳しくは、313 ページの『継続の規則』を参照してください。
- 数値リテラルには継続は許可されません。 自由形式では数値リテラルを 1 行に指定する必要があります。

出力仕様書の定数/編集語フィールド

出力仕様での継続に関する規則は次のとおりです。

- 次の出力仕様の 53 桁目以降に指定を継続します。
- 継続記入行の 7 から 52 桁目はブランクでなければなりません。

例

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
O.............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat+++
O                                          Continue Constant/editword+++
O                                       80 'Mary had a little lamb, its-

 *
 * Only a comment or a completely blank line is allowed in here
O                                           fleece was white as snow.'
```

定義およびプロシージャー仕様書の名前フィールド

定義およびプロシージャー仕様書での名前の継続に関する規則は 次のとおりです。

- 継続規則は、15 文字より長い名前に適用されます。 部分名の終わりに省略符号 (...) をコーディングすると、任意の名前を (15 文字以下の名前であっても) 複数行に継続できます。
- 名前定義は、次の部分から構成されます。

  1. ゼロまたはそれ以上の継続名前行。 継続名前行は、その記入項目中の最後の非ブランク文字として省略記号を持つものとして 識別されます。 名前は、7 から 21 桁目の中で開始する必要があり、77 桁目まで (80 桁目で 終了する省略記号を付けて) の任意の位置で終了することができます。 名前の開始と省略記号 (...) 文字の間にブランクを挿入することはできません。 これらの条件のいずれかが真とならない場合、その行は主要定義行であると 見なされます。
  2. 名前、定義属性、およびキーワードを含む 1 つの主要定義行。 継続名前行がコーディングされた場合、主要定義行の名前記入項目 はブランクのままになる場合があります。
  3. ゼロまたはそれ以上のキーワード継続記入行。

     例

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D                                     Keywords-cont+++++++++++++++++++++++++
* Long name without continued name lines:
D RatherLongName  S             10A
* Long name using 1 continued name line:
D NameThatIsEvenLonger...
D                 C                   'This is the constant -
D                                     that the name represents.'
* Long name using 1 continued name line:
D NameThatIsSoLongItMustBe...
D   Continued     S             10A
* Compile-time arrays may have long names:
D CompileTimeArrayContainingDataRepresentingTheNamesOfTheMonthsOf...
D  TheYearInGermanLanguage...
D                 S             20A   DIM(12) CTDATA PERRCD(1)
* Long name using 3 continued name lines:
D ThisNameIsSoMuchLongerThanThe...
D    PreviousNamesThatItMustBe...
D    ContinuedOnSeveralSpecs...
D                 PR            10A
D parm_1                        10A   VALUE
*
CL0N01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++++
C                           Extended-factor2-++++++++++++++++++++++++++++++
* Long names defined on calc spec:
C     LongTagName   TAG
C     *LIKE         DEFINE    RatherLongNameQuiteLongName    +5
*
PName++++++++++++..B..................Keywords++++++++++++++++++++++++++++++
PContinuedName++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
* Long name specified on Procedure spec:
P ThisNameIsSoMuchLongerThanThe...
P    PreviousNamesThatItMustBe...
P    ContinuedOnSeveralSpecs...
P                 B
D ThisNameIsSoMuchLongerThanThe...
D    PreviousNamesThatItMustBe...
D    ContinuedOnSeveralSpecs...
D                 PI            10A
D parm_1                        10A   VALUE
```

制御仕様書

制御仕様ステートメント (自由形式仕様書の場合は CTL-OPT 命令コードによって、固定形式仕様書の場合は 6 桁目の H によって識別される) は、プログラムの生成と実行に関する情報を提供します。 ただし、コンパイラーにこの情報を指定するには次の 3 つの異なる方法があり、 コンパイラーはこの情報を次の順序で検索します。

1. ソースに組み込まれている制御ステートメント
2. *LIBL で RPGLEHSPEC という名前が付けられたデータ域名
3. QRPGLE で DFTLEHSPEC という名前が付けられたデータ域名

これらのソースのいずれかが検出されると、値が割り当てられ、指定されていない キーワードにはデフォルト値が割り当てられます。

デフォルトの値については個別の記入項目の説明を参照してください。

注 : コンパイル・オプションのキーワ ードにはデフォルト値がありません。 キーワードの値は、CRTBNDRPG または CRTRPGMOD コマンドで指定した 値を使用して初期化されます。

ヒント :

制御仕様書のキーワードは、モジュール・レベルで適用されます。 これは、モジュール内にコーディングされた複数のプロシージャーがある場合 に、制御仕様書に指定された値がすべてのプロシージャーに適用されることを意味 します。

ソース・ファイル内の制御ステートメント

制御ステートメントを ソース・ファイル内に指定する場合、以下のルールが適用されます。

- 制御ステートメントは複数行に指定できます。 自由形式 の制御ステートメントは、CTL-OPT で始まり、セミコロンで終わります。 固定形式の制御ステートメントは、6 桁目に H がある 1 つ以上の固定形式仕様を含みます。
- 自由形式の制御ステートメントと固定形式の制御ステートメントを混用することができます。
- 複数の制御ステートメントをコーディングし、特定のキーワード (例えば、ALWNULL キーワード) を繰り返すことができない 場合、そのキーワードを含むことができるのは最大 1 つの制御ステートメントのみです。

データ域の制御仕様書としての使用

ソース内に制御仕様書がない場合、 RPG コンパイラーは、制御仕様キーワードが含まれているデータ域がないか検索します。 そういったデータ域が見つからない場合は、デフォルトであるブランクの制御仕様書が 使用されます。

タイプ *CHAR として定義されるデータ域を作成するためには、CL コマンド CRTDTAARA (データ域作成) を使用してください (データ域作成コマンドについて詳しくは、IBM i Information Center を参照してください。) コマンドの初期値パラメーターにキーワードを入力します。

たとえば、デフォルトの日付の形式である *YMD およびデフォルトの日付区切り記号 / を指定する RPGLEHSPEC データ域を作成するためには、次のように記入します。

```
CRTDTAARA DTAARA(MYLIB/RPGLEHSPEC)
          TYPE(*CHAR)
          LEN(80)
          VALUE('datfmt(*ymd) datedit(*ymd/)')
```

データ域は、指定されたキーワードを収容するために必要ないかなるサイズにも することができます。 データ域の全体の長さには、キーワードだけを含めることができます。

ヒント: データ域を使用してキーワードを保持する代わりに、 コピー・ファイルの使用を検討してください。 コピー・ファイルを使用すると、追加の制御ステートメントがソース内にある場合でも コピー・ファイル内のキーワードは使用されます。

例えば、次のソース・ファイル内にはコンパイラーがデータ域を検索するのを阻む制御ステートメントがあるため、このソースではデータ域にあるキーワードは 使用されません。

```
H NOMAIN
D fld             S             10A
```

次のソース・ファイルでは DFTCTLKW コピー・ファイル内のキーワードが使用されます。

```
 /COPY QRPGLESRC,DFTCTLKW
H NOMAIN
D fld             S             10A
```

自由形式の制御ステートメント

自由形式の制御ステートメントは、CTL-OPT で始まり、ゼロ個またはそれ以上のキーワード が続き、セミコロンで終わります。

制御ステートメントの規則

ソース・ファイル内にゼロ個またはそれ以上の制御ステートメントを指定できます。

繰り返すことができない制御仕様キーワードを制御ステートメント内に複数回指定することはできません。

自由形式の制御ステートメントと固定形式の制御ステートメントを混用することができます。 連続する固定形式仕様からなる各グループが、1 つの制御ステートメントを形成します。

プログラムのコンパイルに自由形式制御ステートメントが 含まれている場合、ACTGRP、BNDDIR、または STGMDL キーワード があると DFTACTGRP キーワードはデフォルトで *NO になります。

自由形式の制御ステートメントの内部で使用を許可されている指示は、 /IF、/ELSEIF、/ELSE、および /ENDIF のみです。

```
CTL-OPT
    /IF DEFINED(*CRTBNDRPG)
            ACTGRP(*CALLER)
    /ENDIF
    OPTION(*SRCSTMT);
```

制御ステートメントの例

- 自由形式の 2 つの制御ステートメントがあり、各ステートメントにいくつかのキーワードが指定されています。

```
  ctl-opt datfmt(*iso) timfmt(*iso)
          alwnull(*usrctl);

  ctl-opt option(*srcstmt)ccsid(*char:*jobrun);
```

- キーワードがない制御ステートメントが 1 つあります。 このステートメントの唯一の効果は、RPG コンパイラーが制御仕様書データ域を検索するのを防止することです。

```
  ctl-opt;
```

- 自由形式の制御ステートメントと固定形式の制御ステートメントが混用されています。
  1. 3 つの仕様が 1 つの固定形式ステートメントを形成しています。
  2. 1 つの自由形式ステートメントが CTL-OPT で始まり、セミコロンで終わっています。
  3. 1 つの仕様が 1 つの固定形式ステートメントを形成しています。

```
H OPTION(*SRCSTMT :              1
H        *NODEBUGIO)             1
H ACTGRP(*NEW)                   1
  CTL-OPT ALWNULL(*USRCTL)       2
          CCSID(*UCS2            2
                    :1200)       2
          CCSID(*CHAR:*JOBRUN);  2
H DATFMT(*YMD) TIMFMT(*USA)      3
```

従来型の制御仕様ステートメント

制御仕様書はキーワードだけで構成されます。 キーワードは 7 から 80 桁目のどこにでも入れることができます。 81 から 100 桁目は注記に使用できます。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+...
9 ...+... 10
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++Comments++++++++++
++
```

図 *113.* 制御仕様書の レイアウト

以下は、制御仕様書の例です。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H ALTSEQ(*EXT) CURSYM('$') DATEDIT(*MDY) DATFMT(*MDY/) DEBUG(*YES)
H DECEDIT('.') FORMSALIGN(*YES) FTRANS(*SRC) DFTNAME(name)
H TIMFMT(*ISO)
H COPYRIGHT('(C) Copyright ABC Programming - 1995')
```

6 桁目 (仕様書コード)

自由形式構文	CTL-OPT 命令コード。319 ページの『自由形式の制御ステートメント』を参照してください。

この行を制御仕様書として識別する H が 6 桁目に現れていなければなりません。

7 から 80 桁目 (キーワード)

自由形式構文	自由形式ステートメントで使用可能な桁については、307 ページの『自由形式ステートメント』を参照してください。

制御仕様書キーワードを使用して、プログラムが装置を取り扱う方法および表示す る情報の特定のタイプを決定します。

制御仕様書キーワードには、デフォルト値または CRTBNDRPG および CRTRPGMOD コマンドで 指定されたオプションを指定変更するコンパイル・オプション・キーワードも含まれ ます。 これらのキーワードは、プログラムのコンパイルごとに使用される コンパイル・オプションを決定します。

制御仕様書のキーワード

制御仕様書のキーワードは、パラメーターを持っていなかったり、任意指定パラメーター を持ったり、または必須パラメーターを持ったりします。 キーワードの構文は次のとおりです。

```
    Keyword(parameter1 : parameter2)
```

各パラメーターの説明は以下のとおりです。

- 1 つまたは複数のパラメーターは括弧 () で囲みます。

  注 : パラメーターがない場合、括弧 を指定してはなりません。
- コロン (:) を使用して複数のパラメーターを区切ります。

任意指定パラメーターと必須パラメーターを示すために、 以下の国別の規則を使用します。

- 中括弧 { } は任意指定パラメーターまたはパラメーターの任意指定要素を示します。
- 省略記号 (...) はパラメーターが反復可能であることを示します。
- コロン (:) はパラメーターを区切り、複数のパラメーターを指定できることを示します。 コロンで区切られたすべてのパラメーターは、中括弧で囲まれていない限り、必須パラメーターです。
- 縦線 (|) は、キーワードに 1 つのパラメーターしか指定でき ないことを示します。

- キーワード・パラメーターを区切るブランクは、1 つまたは複数のパラメーター を指定できることを示します。

注 : 中括弧、省略記号、および縦線は、キーワード構文の一部 ではないため、ソースに入れてはなりません。

制御仕様書のキーワードに追加のスペースが必要な場合には、キーワード・フィールド を後続の行に継続させることができます。 320 ページの『従来型の制御仕様ステートメント』および 315 ページの『制御仕様書のキーワード・フィールド』を参照してください。

ACTGRP(*STGMDL | *NEW | *CALLER | '活動化グループ名')

ACTGRP キーワードによって、プログラムが呼び出された時に関連付ける活動化グループを指定することができます。 ACTGRP(*STGMDL) が指定されていて、STGMDL(*SNGLVL) または STGMDL(*INHERIT) が有効の場合、プログラムは、呼び出されると、QILE 活動化グループ内で活動化されます。 ACTGRP(*STGMDL) が指定されていて、STGMDL(*TERASPACE) が有効の場合、 プログラムは、呼び出されると、QILETS 活動化グループ内で活動化されます。 ACTGRP(*NEW) を指定すると、プログラム は新しい活動化グループ内で活動化されます。 ACTGRP(*CALLER) を指定すると、 プログラムは呼び出し元の活動化グループ内で活動化されます。 活動化グループ名を 指定した場合、その名前が、このプログラムの呼び出し時に使用されます。

ACTGRP キーワードを指定しない場合、コマンドに指定した値が 使用されます。

ACTGRP キーワードは CRTBNDRPG コマンドを使用した場合のみ有効です。

DFTACTGRP(*YES) を指定してプログラムを作成する場合、ACTGRP、BNDDIR、または STGMDL のキーワードは使用できません。

ACTGRP キーワードが DFTACTGRP キーワードの設定に与える影響については、333 ページの『DFTACTGRP(*YES | *NO)』 を参照してください。

注 : プログラムの呼び出し時に作成される活動化グループの名前は、 活動化グループ名として入力されたテキストと大文字小文字が完全に一致する名前になります。 RCLACTGRP コマンドでは、ACTGRP パラメーターに小文字テキストを指定することは許されません。 RCLACTGRP コマンドを使用して活動化グループを個別に再利用する必要がある場合には、 活動化グループ名として小文字テキストを入力しないでください。

ALLOC(*STGMDL | *TERASPACE | *SNGLVL)

ALLOC キーワードは、モジュールの記憶域管理命令の記憶域モデルを指定します。

ALLOC キーワードが指定されない場合には、ALLOC(*STGMDL) が想定されます。

- *STGMDL は、メモリー管理命令の記憶域モデルが、モジュールの記憶域モデルと同じになるように指定する場合に使用します。 制御仕様書の STGMDL キーワードを使用して、モジュールの記憶域モデルを制御します。 モジュールの記憶域モデルが *INHERIT の場合、メモリー管理命令に使用される記憶域モデルは、実行時に決定されます。
- *SNGLVL は、メモリー管理命令に単一レベル記憶域モデルが使用されるように指定する場合に使用します。
- *TERASPACE は、メモリー管理命令にテラスペース記憶域モデルが使用されるように指定する場合に使用します。

テラスペースおよび単一レベルのメモリー管理操作について詳しくは、581 ページの『メモリー管理命令』を参照してください。

ALTSEQ {(*NONE | *SRC | *EXT) }

ALTSEQ キーワードは、代替照合順序が使用されるかどうかと、それが使用され る場合にソース仕様に対して内部と外部のいずれのものであるかを指示します。 次のリストは、使用できる別のキーワードとパラメーターの組み合わせの場合にどう なるかを示しています。

キーワード/パラメーター
使用される照合順序

ALTSEQ の指定なし
通常照合順序

ALTSEQ(*NONE)
通常照合順序

ALTSEQ、パラメーターなし
ソースに指定された代替照合順序

ALTSEQ(*SRC)
ソースに指定された代替照合順序

ALTSEQ(*EXT)
SRTSEQ および LANGID コマンドのパラメーターまたはキーワード によって指定された代替照合順序

ALTSEQ を指定しない場合、あるいは *NONE または *EXT を使用して指定した 場合には、代替照合順序テーブルをプログラムの中に指定してはなりません。

ALWNULL(*NO | *INPUTONLY | *USRCTL)

ALWNULL キーワードは、外部記述データベース・ファイルからの、ヌル可能フィールドを含んでいるレコードを使用する方法を指定します。

また、NULLIND キーワードを使用して独自のヌル可能フィールドを定義できるかどうかの制御も行います。 NULLIND キーワードは、ALWNULL(*USRCTL) が指定されたときに使用できます。

ALWNULL(*NO) を指定した場合、ヌル値フィールドを持つレコードを外部記述ファイル から処理することはできません。 ヌル値を含んでいるレコードの取り出しを試みると、 レコード内のデータにアクセスできず、 データ・マッピング・エラーが発生します。

ALWNULL(*INPUTONLY) を指定した場合、外部記述入力専用データベース・ファイルから、 ヌル値を含むヌル値可能フィールドがあるレコードを正常に読み取ることができます。 ヌル値を含むレコードが取り出された場合、データ・マッピング・エラーは発生しない で、ヌル値を含むすべてのフィールドに、データベースのデフォルト値が入ります。 ただし、 以下のいずれも実行することはできません。

- ヌル値可能キー・フィールドの使用
- ヌル値可能フィールドを含むレコードの作成または更新
- プログラムの実行中に、ヌル値可能フィールドが実際にヌルであるかどうかを 判別すること
- ヌル値可能フィールドをヌルに設定すること

ALWNULL(*USRCTL) を指定した場合、外部記述データベース・ファイルから、ヌル値を 含むレコードの読み取り、書き出し、および更新を行うことができます。 ヌル・キーのある レコードはキー順操作を使用して検索することができます。 ヌル値可能フィールドが実際にヌルであるかどうかを判別し、ヌル値可能フィールドを 出力または更新のためにヌルに設定することができます。 ヌル値を含むフィールドを正しく使用することは、 各自の責任で行っていただきます。

ALWNULL キーワードを指定しない場合、コマンドに指定した値が 使用されます。

詳しくは、286 ページの『データベースのヌル値サポート』を参照してください。

AUT(*LIBRCRTAUT | *ALL | *CHANGE | *USE | *EXCLUDE | '権限リスト名')

AUT キーワードは、オブジェクトに対する特定の権限を持っていないユーザー、権限認可リストに載っていないユーザー、およびそのユーザー・グループがオブジェクトに対する特定の権限を持っていないユーザーに与える権限を指定します。 オブジェクト権限認可 (GRTOBJAUT) または オブジェクト権限 取り消し (RVKOBJAUT) の各 CL コマンドを使用すれば、オブジェクトの作成後に、 すべてのユーザーまたは指定したユーザーの権限を 変更することができます。

AUT(*LIBRCRTAUT) を指定すると、オブジェクトの共通認可は、ターゲット・ライブラリー (そのオブジェクトを含んでいる ライブラリー) に関する CRTAUT キーワードから取られます。 値は，オブジェクトの作成時 に決定されます。 作成後にライブラリーの CRTAUT 値が変わった場合には, 新しい値は 既存のオブジェクトに反映されません。

AUT(*ALL) を指定した場合、所有者に限定されているオブジェクト、または 権限リスト管理権限によって制御されているオブジェクトを除くオブジェクトに、 すべての命令に関する権限が与えられます。 ユーザ

ーはオブジェクトの存在を制御し、 その機密保護を指定し、変更し、基本機能を実行できますが、 その所有権を移すことはできません。

AUT(*CHANGE) を指定した場合、所有者に限定されているオブジェクト、または オブジェクト権限およびオブジェクト管理権限によって制御されているオブジェクトを 除くオブジェクトに、すべてのデータ権限、およびすべての命令を実行するための権限が与えられます。 ユーザーはオブジェクトを変更し，オブジェクトに対して基本機能を実行することができます。

AUT(*USE) を指定した場合、オブジェクトに対する基本操作の権限である 操作権限と読み取り権限がオブジェクトに与えられます。 ユーザーはオブジェクトを変更することができません。

AUT(*EXCLUDE) を指定した場合、ユーザーは そのオブジェクトにアクセスできません。

権限認可リスト名はオブジェクトの追加先の、ユーザーと権限に関する権限認可リスト の名前です。 オブジェクトは、この権限認可リストによって機密が保護されます。 オブジェクトに関する共通認可は *AUTL に設定されます。 権限認可リストは、コンパイル時にシステム上に 存在しなければなりません。

AUT キーワードを指定しない場合、コマンドに指定した値が 使用されます。

BNDDIR('バインディング・ディレクトリー名' {:'バインディング・ディレクトリー名'...})

BNDDIR キーワードは、記号の解決で使用するバインディング・ディレクトリーのリストを指定します。

バインディング・ディレクトリー名は、ライブラリー名とその後に続く斜線区切り 記号によって修飾することができます ('ライブラリー名/バインディング・ディレクトリー名')。 ライブラリー名は検索するライブラリーの名前です。 ライブラリー名を指定しない場合、バインディング・ディレクトリー名を探すために *LIBL が使用されます。 CRTBNDRPG を使用してプログラムを作成した場合、 コンパイル時にライブラリー・リストが検索されます。 CRTRPGMOD を使用してモジュールを作成した場合、プログラムまたはサービス・プログラム を作成するためにモジュールが使用される時に、ライブラリー・リストが検索されます。

BNDDIR が制御仕様書とコマンドの両方に指定されている場合は、すべてのバインディング・ディレクトリーがシンボルの解決に使用されます。 制御仕様書の BNDDIR は、コマンド上の BNDDIR を指定変更するものではありません。

BNDDIR キーワードを指定しない場合、コマンドに指定した値が 使用されます。

DFTACTGRP(*YES) を指定してプログラムを作成する場合、BNDDIR、ACTGRP、または STGMDL のコマンド・パラメーターあるいはキーワードは使用できません。

BNDDIR キーワードが DFTACTGRP キーワードの設定に及ぼす影響については、333 ページの『DFTACTGRP(*YES | *NO)』 を参照してください。

CCSID 制御キーワード

CCSID キーワードは何度か指定することができますが、 最初のパラメーターの値は毎回異なっていなければなりません。

CCSID(*EXACT) は、 モジュール内の CCSID の一般的な処理を制御します。

/SET および /RESTORE 指示を使用して、 定義のデフォルト CCSID を一時的に変更することができます。 81 ページの『/SET』 を参照してください。

CCSID(*CHAR)、 CCSID(*GRAPH)、および CCSID(*UCS2) は、モジュールのデフォルト CCSID を設定します。 これらのデフォルト値は、CCSID キーワードがコーディングされていない、 プログラム記述入力フィールドおよび出力フィールド、およびデータ定義に使用されます。

CCSID キーワードは、リテラルおよびコンパイル時データの CCSID にも影響します。 詳しくは、205 ページの『リテラルおよびコンパイル時データの CCSID』 を参照してください。

*CCSID(*EXACT)*

CCSID(*EXACT) キーワードは、モジュール内の CCSID の処理を制御します。

注 : CCSID(*EXACT) が指定されていない場合、 RPG コンパイラーは、リテラル、変数、または、データベース・ファイルの入力バッファーおよび出力バッファー に入っているデータの CCSID に関して誤った想定を行う可能性があります。

- CCSID(*EXACT) が英数字リテラル、グラフィック・リテラル、およびコンパイル時データの CCSID に与える影響については、205 ページの『リテラルおよびコンパイル時データの CCSID』を参照してください。
- CCSID(*EXACT) が、DISK ファイルおよび SEQ ファイルの入力バッファーおよび出力バッファー内の英数字データとグラフィック・データ の CCSID の処理に与える影響については、340 ページの『OPENOPT (* {NO }INZOFL * {NO }CVTDATA)』 を参照してください。
- CCSID(*EXACT) がモジュール内の英数字項目のデフォルト CCSID に与える影響については、325 ページの『CCSID(*CHAR : *JOBRUN | *JOBRUNMIX | *UTF8 | *HEX | 番号)』 を参照してください。
- CCSID(*EXACT) がモジュール内のグラフィック項目のデフォルト CCSID に与える影響については、326 ページの『CCSID(*GRAPH : *JOBRUN | *SRC | *HEX | *IGNORE | 番号)』 を参照してください。
- CCSID(*EXACT) が、外部記述データ構造および LIKEREC キーワードを使用して定義されたデータ構造内の英数字サブフィールドの CCSID に与える影響については、419 ページの『CCSID(*EXACT | *NOEXACT)』 を参照してください。

*CCSID(*CHAR : *JOBRUN | *JOBRUNMIX | *UTF8 | *HEX | 番号)*

CCSID(*CHAR) は、モジュール内の英数字データ定義に使用されるデフォルト文字 CCSID を 設定します。

/SET および /RESTORE 指示を使用して、定義ステートメントに指定された英数字定義に対するデフォルト CCSID を 一時的に変更することができます。81 ページの『/SET』を参照してください。

CCSID(*CHAR : *JOBRUN)
実行時のジョブ CCSID。 ジョブ CCSID が 65535 の場合、デフォルトのジョブ CCSID が使用されます。

文字 X'0E' は、ランタイム・ジョブ CCSID が混合バイト CCSID である場合にのみ、 シフトアウト文字であると想定されます。

詳しくは、250 ページの『文字形式』を参照してください。

CCSID(*CHAR:*JOBRUNMIX)
実行時のジョブ CCSID に関連する混合バイト CCSID。 ジョブ CCSID が 65535 である場合は、デフォルトのジョブ CCSID に関連する混合バイト CCSID が使用されます。

文字 X'0E' は常にシフトアウト文字であると見なされます。

CCSID(*CHAR:*UTF8)
UTF-8; この CCSID の数値は 1208 です。

CCSID(*CHAR:*HEX)
デフォルトでは、文字データには CCSID はありません。 CCSID キーワードなしで定義された文字変数を CCSID 変換で使用することはできません。

CCSID(*CHAR:番号)
番号 は、英数字 CCSID でなければなりません。 任意の 1 バイトまたは混合バイト EBCDIC CCSID、任意の 1 バイトまたは混合バイト ASCII CCSID、または UTF-8 CCSID 1208 を指定できます。

CCSID(*CHAR) が指定されていない場合には、次のようになります。

- CCSID(*EXACT) が指定されている場合、CCSID(*CHAR:*JOBRUN) がデフォルトです。
- CCSID(*EXACT) が指定されていない場合、文字データの CCSID は、ジョブ CCSID に関連した混合バイト CCSID であると 想定されます。 文字データに文字 X'0E' が含まれる場合、この文字はシフトアウト文字として解釈されます。 このために、文字データが UCS-2 データに変換される場合、または、ジョブ CCSID 以外の CCSID の文字データに 変換される場合に、誤った結果が生じることがあります。

外部記述データ構造および LIKEREC キーワードを指定して定義されたデータ構造のサブフィールドの CCSID については、419 ページの『CCSID(*EXACT | *NOEXACT)』 を参照してください。

文字リテラルの CCSID については、205 ページの『リテラルおよびコンパイル時データの CCSID』 を参照してください。

*CCSID(*GRAPH : *JOBRUN | *SRC | *HEX | *IGNORE | 番号)*

CCSID(*GRAPH) は、CCSID キーワードがコーディングされていないデータ定義 に使用されるデフォルトのグラフィック CCSID を設定します。

/SET および /RESTORE 指示を使用して、 定義のデフォルト CCSID を一時的に変更することができます。81 ページの『/SET』 を参照してください。

CCSID(*GRAPH:*JOBRUN)
実行時のジョブ CCSID に関連する DBCS CCSID。

CCSID(*GRAPH:*SRC)
ソース・ファイルを読み取るために使用される CCSID に関連付けられているグラフィック CCSID。 CRTBNDRPG コマンドまたは CRTRPGMOD コマンドの TGTCCSID パラメーターにより指定されています。 TGTCCSID パラメーターは TGTCCSID(*SRC) にデフォルトで設定されています。 これは、1 次ソース・ファイルの CCSID に関連付けられた EBCDIC CCSID です。 TGTCCSID パラメーターについて詳しくは、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」の CRTBNDRPG コマンドの説明を参照してください。

CCSID(*GRAPH:*HEX)
デフォルトでは、グラフィック・データには CCSID はありません。 CCSID キーワードなしで定義されたグラフィック変数を CCSID 変換で使用することはできません。

CCSID(*GRAPH:*IGNORE)
モジュール内のグラフィック定義に CCSID キーワードを 指定することはできません。 モジュール内のグラフィック・データと英数字または UCS-2 データの間の CCSID 変換 は許可されません。 %GRAPH 組み込み関数は使用できません。

CCSID(*GRAPH:番号)
番号は 図形 CCSID でなければなりません。 有効な図形 CCSID は、65535 か、EBCDIC 2 バイト・コード化体系 (X'1200') の CCSID で す。

CCSID(*GRAPH) が指定されていない場合は、次のようになります。

- CCSID(*EXACT) が指定されている場合、CCSID(*GRAPH:*JOBRUN) がデフォルトです。
- CCSID(*EXACT) が指定されていない場合、CCSID(*GRAPH:*IGNORE) が想定されます。

CCSID(*GRAPH:*IGNORE) が有効になっていない場合、 外部記述データ構造内のグラフィック・サブフィールドは 外部ファイル内の CCSID を使用します。

グラフィック・リテラルの CCSID については、205 ページの『リテラルおよびコンパイル時データの CCSID』 を参照してください。

*CCSID(*UCS2 : *UTF16 | 番号)*

CCSID(*UCS2) は、CCSID キーワードがコーディングされていないデータ定義 に使用されるデフォルトの UCS-2 CCSID を設定します。

/SET および /RESTORE 指示を使用して、 定義のデフォルト CCSID を一時的に変更することができます。81 ページの『/SET』 を参照してください。

CCSID(*UCS2:*UTF16)
UTF-16; この CCSID の数値は 1200 です。

CCSID(*UCS2:番号)
番号は UCS-2 CCSID でなければなりません。 有効 な UCS-2 CCSID は UCS-2 コード化体系 (x'7200') になっています。 例えば、UTF-16 CCSID 1200 は、コード化体系 x'7200' になります。

外部記述データ構造内の UCS-2 サブフィールドの CCSID は、常に外部ファイル内の フィールドと同じものになります。

CCSID(*UCS2) が指定されていない場合のデフォルトの UCS-2 CCSID は 13488 です。

UCS-2 リテラルの CCSID については、205 ページの『リテラルおよびコンパイル時データの CCSID』 を参照してください。

CCSIDCVT(*EXCP | *LIST)

CCSIDCVT キーワードを使用して、異なる CCSID のデータ間の変換をコンパイラーが どのように扱うのかを制御できます。

2 つの CCSID が別々の文字セットをサポートしている場合、一方の CCSID にはある文字だが、 他方の CCSID には一致する文字がないといったことがあり得ます。 例えば、日本語文字セットおよびタイ語文字セットは、それぞれ、他方の文字セットの一部ではない多くの文字 を含んでいます。 UCS-2 変数に日本語文字とタイ語文字の両方が含まれていて、その UCS-2 変数 が日本語グラフィック変数に変換される場合、その DBCS 変数にはタイ語文字に一致する文字が含まれないという 結果になります。 変換では、ターゲット文字セット内に一致する文字が見つからない場合、 ターゲット変数に置換文字 が入れられます。

英数字データの置換文字は x'3F' です。 グラフィック・データの置換文字は x'FEFE' です。

CCSIDCVT キーワードには片方または両方のパラメーターを指定できます。 両方のパラメーターを指定する場合はコロンで区切ります。 指定順序は任意です。

- CCSIDCVT(*EXCP : *LIST)
- CCSIDCVT(*LIST : *EXCP)
- CCSIDCVT(*LIST)
- CCSIDCVT(*EXCP)

***EXCP**
*EXCP は、変換の結果として実行時に置換文字になった場合は状況コード 452 の RPG 例外が起こることを 指示します。

***LIST**
*LIST は、コンパイラーがリストにセクションを追加して、 すべての暗黙的および明示的な CCSID 変換を示し、さらに、 ターゲット文字セット内に一致する文字がないために置換文字が結果に入れられることによって生じる データ逸失がどの変換で起こる可能性があるのかも示すことを指示します。

注 : リストに「CCSID 変換の要約」セクションができるのは、プログラムまたはモジュールが作成される場合のみです。 コンパイル時のエラーがあるか、コンパイルで OPTION(*NOGEN) が 指定されている場合には作成されません。

この情報は次の 2 つの目的に使用できます。

- 一部の変数のデータ・タイプを変更して変換の数を減らすことによって、パフォーマンスを向上させることができます。
- 置換文字が使用される結果になる可能性がある変換の一部を除去することによって、プログラムの信頼性を向上させることができます。 例えば、UCS-2 から英数字変数への変換があり、その英数字データは後で UCS-2 に戻される 場合、英数字変数のタイプを UCS-2 に変更すれば、発生する可能性のあったデータ逸失を回避できます。

以下のタイプの変換の結果として、置換文字が発生する可能性があります。

- UCS-2 から英数字または DBCS への変換
- 異なる 2 つの DBCS CCSID 間の変換
- 英数字から DBCS への変換
- DBCS から英数字への変換

モジュール内で使用される CCSID 変換ごとに、その変換のステートメント番号のリストがあります。 詳しくは、261 ページの『変換』を参照してください。

変換の結果として置換文字が生じる可能性がある場合、 変換項目の左側にメッセージ番号が表示されます。 そのメッセージはコンパイルの最後に「メッセージの要約」内に示されます。

実行時まで値が不明のいくつかの CCSID のために、以下の特殊値が使用されます。

***JOBRUN**
ジョブ CCSID、または、ジョブ CCSID が 65535 の場合はデフォルトのジョブ CCSID。
CCSID(*CHAR : *JOBRUN) が制御仕様書に指定された場合は、これが RPG モジュール内の英数字データのデフォルト CCSID です。

***JOBRUN_MIXED**
ジョブ CCSID に関連するか、または、ジョブ CCSID が 65535 の 場合はデフォルトのジョブ CCSID に関連する、混合バイト CCSID。 この CCSID は、ジョブ CCSID が混合バイト CCSID である場合は、 実際のジョブ CCSID と同じであることもあります。 CCSID(*CHAR : *JOBRUN) も CCSID(*EXACT) も制御仕様書に指定されなかった場合は、これが RPG モジュール内の英数字データのデフォルト CCSID です。 詳しくは、324 ページの『CCSID 制御キーワード』を参照してください。

***JOBRUN_DBCS**
ジョブ CCSID に関連するか、または、ジョブ CCSID が 65535 の場合はデフォルトのジョブ CCSID に関連する、2 バイト CCSID。 この CCSID は、英数字データとグラフィック・データの間の変換で、非シフト英数字データの CCSID として使用されることがあります。

***JOBRUN_JAVA**
RPG パラメーターまたは戻り値が英数字として定義されている場合に Java メソッドのパラメーターおよび戻り値に使用される CCSID。 RPG コンパイラーは、この CCSID はジョブ CCSID に関連していると想定します。したがって、RPG コンパイラーは、置換文字が結果として発生するような変換の可能性はないと想定します。

次の例は、CCSID 変換の要約を示します。 最初の項目は、6 個のステートメントに、ジョブ CCSID の英数字データ から CCSID 1200 の UCS-2 データへの変換があることを示しています。 2 番目の項目は、3 つのステートメントに、 CCSID 1200 の UCS-2 データからジョブ CCSID の英数字データへの変換があることを示しています。 2 番目の項目の前にあるメッセージ *RNF7357* は、これらの 変換の結果として英数字データ内に置換文字が置かれる可能性があることを示します。

```
                        C C S I D  C o n v e r s i o n s
            From CCSID       To CCSID     References
            *JOBRUN          1200              27      12     321     426
                                              552     631
RNF7357     1200             *JOBRUN           28     921    1073
   * * * * *  E N D  O F  C C S I D  C O N V E R S I O N S  * * * * *
```

図 *114. CCSID* 変換の要約のサンプル

COPYNEST(番号)

COPYNEST キーワードは、/COPY 指示のネストに関して行うことが できる最大の深さを指定します。 この深さの値は、1 またはそれ以上で、 かつ、2048 またはそれ以下にする必要があります。 デフォルトの深さは 32 です。

COPYRIGHT('版権ストリング')

COPYRIGHT キーワードは、DSPMOD、DSPPGM、または DSPSRVPGM コマンドを使用し て参照することができる版権情報を指定するものです。 版権ストリングは、最大長が 256 の文字リテラルです。 このリテラルは継続の指定で継続させることができます。 (継続記入行の使用に関する規則については、313 ページの『継続の規則』を参照してく ださい。) COPYRIGHT キーワードを指定しない場合には、作成されたモジュールまたはプロ グラムに版権情報は追加されません。

ヒント:

モジュールの版権情報を参照するためには、次のコマンドを使用してください。

```
DSPMOD mylib/mymod DETAIL(*COPYRIGHT)
```

プログラムの場合には、DETAIL(*COPYRIGHT) を指定した DSPPGM コマンドを使用 してください。 この情報には、プログラムにバインドされたすべてのモジュール からの版権情報が含まれています。

同様に、DSPSRVPGM DETAIL(*COPYRIGHT) ではサービス・プログラム中のすべての モジュールの版権情報が得られます。

CURSYM('記号')

CURSYM キーワードは、編集で通貨記号として使用される文字を 指定します。 記号は引用符で囲まれた単一文字でなければなりません。 RPG 文字セット (73 ページの『記号名および予約語』を参照) の任意の文字を使用すること ができますが、以下の文字は除きます。

- 0 (ゼロ)
- * (アスタリスク)
- , (コンマ)
- & (アンパーサンド)
- . (ピリオド)
- - (マイナス符号)
- C (英字 C)
- R (英字 R)
- ブランク

このキーワードが指定されない場合には、\ (円記号) が通貨記号として使用 されます。

CVTOPT(*{NO}DATETIME *{NO}GRAPHIC *{NO}VARCHAR *{NO}VARGRAPHIC)

CVTOPT キーワードは、ILE RPG コンパイラーが、外部記述データベース・ファイルから取り出した日付、時刻、タイム・スタンプ、図形データ・タイプ、および可変長データ・タイプを処理する方法を指定するために使用します。

任意のまたはすべてのデータ・タイプを任意の順序で指定することができます。 しかし、データ・タイプを指定する場合、同じデータ・タイプの *NOxxxx パラメーター を同時に指定してはなりません。その逆も同様です。 例えば，*GRAPHIC を指定した場合に，*NOGRAPHIC も指定することはできず，その逆も指定できません。 パラメーターは、コロンで区切ってください。 1 つのパラメーターを複数回指定してはなりません。

注 : キーワード CVTOPT にペアのパラメーターからのメンバーを指定しない場合、 その特定のデータ・タイプについてコマンドに指定した値が使用されます。 たとえば、制御仕様書にキーワード CVTOPT(*DATETIME : *NOVARCHAR : *NOVARGRAPHIC) を 指定した場合、ペア (*GRAPHIC、*NOGRAPHIC) については、コマンドに暗黙的にまたは 明示的に指定したパラメーターが使用されます。

*DATETIME を指定した場合、日付、時刻、およびタイム・スタンプのデータ・タイプが 固定長文字フィールドとして宣言されます。

*NODATETIME を指定した場合、日付、時刻、およびタイム・スタンプのデータ・タイプは 変換されません。

*GRAPHIC を指定した場合、2 バイト文字セット (DBCS) 図形データ・タイプが 固定長文字フィールドとして宣言されます。

*NOGRAPHIC を指定した場合、2 バイト文字セット (DBCS) 図形データ・タイプは 変換されません。

*VARCHAR を指定した場合、可変長文字データ・タイプが 固定長文字フィールドとして宣言されます。

*NOVARCHAR を指定した場合、可変長文字データ・タイプは 変換されません。

*VARGRAPHIC を指定した場合、可変長 2 バイト文字セット (DBCS) 図形 データ・タイプが固定長文字フィールドとして宣言されます。

*NOVARGRAPHIC を指定した場合、可変長 2 バイト文字セット (DBCS) 図形 データ・タイプは変換されません。

CVTOPT キーワードを指定しない場合、コマンドに指定した値が 使用されます。

DATEDIT(形式 {区切り記号 })

DATEDIT キーワードは、Y 編集コードを使用するときの数値フィールドの 形式を指定します。 区切り記号は任意指定です。 値 (形式) は *DMY、*MDY、または、*YMD とすることができます。 デフォルトの区切り記号は/です。& (アンパーサンド) の区切り記号を使用して、ブランクの区切り記号を指定することができます。

DATFMT(形式 {区切り記号 })

DATFMT キーワードは、プログラム内の日付リテラルの内部日付形式および日付フ ィールドのデフォルトの内部形式を指定します。 定義仕様書の DATFMT キーワードでフィールドの形式を指定することによって、そ の特定のフィールドについて異なる内部日付形式を指定することができます。

/SET および /RESTORE 指示を使用して、 定義のデフォルト日付形式を一時的に変更することができます。81 ページの『/SET』 を参照してください。

DATFMT キーワードが指定されない場合には、*ISO 形式と見なされます。 内部形式について詳しくは、247 ページの『内部形式および外部形式』を参照してください。274 ページの表 71 は、さまざまな時刻形式およびその区切り記号をリストしています。

DCLOPT(*NOCHGDSLEN)

DCLOPT キーワードは宣言に関連するオプションを指定します。

パラメーターは *NOCHGDSLEN でなければなりません。

DCLOPT(*NOCHGDSLEN) が指定されると、以下の条件が満たされれば、自由形式のファイルまたは定義ステートメント内で %SIZE をデータ構造パラメーターと一緒に使用することができます。

- すべてのサブフィールドが定義される。
- データ構造が DIM または OCCURS キーワードで定義された場合は、%SIZE が指定されたとき、次元を定義しなければならない。
- %SIZE とデータ構造定義は、その両方をグローバル宣言内で使用するか、または両方を同じサブプロシージャー内で使用しなければならない。

以下の追加規則が適用されます。

- 入力仕様にデータ構造を指定することはできない。
- 演算仕様の結果フィールドにデータ構造がある場合は、「長さ」記入項目を指定できない。
- 外部記述ファイル内に生成された入力仕様または出力仕様がある場合は、そのファイル内のフィールド名をデータ構造の名前にすることはできない。 (QUALIFIED キーワードも LIKEFILE キーワードもないグローバル・ファイルに対しては入力仕様と出力仕様が生成されます。)

次の例では、

1. 制御ステートメントにキーワード DCLOPT(*NOCHGDSLEN) が指定されています。
2. データ構造 MYDS は明示的な長さなしで定義されています。
3. サブフィールド *DATA* は *LINE* と同様に定義されています。
4. 最初のファイル宣言は、%SIZE(MYDS) の値がまだ知られていないので無効です。
5. フィールド *LINE* が定義され、*MYDS* のすべてのサブフィールドが定義されて、制御ステートメント内に DCLOPT(*NOCHGDSLEN) があるときは、データ構造 *MYDS* が定義され、このデータ構造のサイズが知られます。
6. 2 番目のファイル宣言は、%SIZE(MYDS) の値が知られているため、有効です。

DCLOPT(*NOCHGDSLEN) がないと、コンパイラーがこのプロシージャーのすべてのステートメントを処理し終わるまで *MYDS* のサイズが未知であるため、どちらのファイル宣言も無効になります。

```
ctl-opt dclopt(*nochgdslen); // 1

dcl-ds myDs; // 2
  seq zoned(6:2);
  dat zoned(6);
  data like(line); // 3
end-ds;

dcl-f myfile1 disk(%size(myDS)); // 4

dcl-s line char(100); // 5

dcl-f myfile2 disk(%size(myDS)); // 6
```

DEBUG{(*DUMP | *INPUT | *RETVAL | *XMLSAX | *NO | *YES)}

DEBUG キーワードは、どのデバッグ・エイドがモジュールに生成されるかを制御します。

DEBUG キーワードが 1 つ以上の *CONSTANTS、*DUMP、*INPUT、*RETVAL、または *XMLSAX パラメーターとともに指定されている場合は、どのデバッグ・エイドをモジュールに生成するかを正確に選択できます。

DEBUG キーワードが *YES または *NO とともに指定されている場合は、これ以外のパラメーターを指定することはできません。

- DEBUG(*YES) が指定されている場合、*DUMP オプションおよび *INPUT オプションが使用可能になります。 DEBUG(*YES) の指定は、DEBUG(*DUMP:*INPUT) を指定した場合と同じです。
- DEBUG(*NO) が指定されている場合、デバッグ・オプションを使用できません。

以下のオプションは個別に指定できます。

***CONSTANTS**
名前付き定数の値は、デバッガーで表示できます。

***DUMP**
DUMP 命令が実行されます。

注: DEBUG 命令コードで命令拡張 A を指定することによって、DUMP 命令を強制的に実行できます。この命令拡張は、DEBUG キーワードの値にかかわらず必ずダンプを実行することを意味します。

***INPUT**
すべての外部記述入力フィールドが、プログラムで使用されていない場合でも、入力操作中に読み取られます。 通常、外部記述入力フィールドは、プログラム内で別の方法で使用される 場合のみ、入力操作時に読み取られます。

***RETVAL**
プロシージャーが値を戻す場合、プロシージャーの最後のステートメントにブレークポイントを設定し、特殊変数 *_ QRNU_RETVAL* を評価することによって、戻り値を表示または変更することができます。

- 自由形式のコードでは、END-PROC ステートメントにブレークポイントを設定します。
- 固定形式のコードでは、Procedure-End 仕様にブレークポイントを設定します。

_QRNU_RETVAL は、プロシージャーからの戻り値と同じもので定義されます。

- 戻り値がデータ構造である場合は、*_QRNU_RETVAL* もデータ構造になります。

  例えば、戻り値がキーワード LIKEDS(myDs) で定義されており、*myDs* にサブフィールド *id* および *addr* が指定される場合、次のようになります。

  – データ構造全体を表示できます

  ```
  EVAL _QRNU_RETVAL
  ```

  – 個々のサブフィールドを表示または変更できます

```
EVAL _QRNU_RETVAL.addr
EVAL _QRNU_RETVAL.id = 12345
```

- 戻り値が配列の場合、*_QRNU_RETVAL* は配列になります。 例えば、戻り値がキーワード DIM(5) で定義されている場合、次のようになります。
  - 配列全体を表示できます

```
EVAL _QRNU_RETVAL
```

  - 個々の要素を表示または変更できます

```
EVAL _QRNU_RETVAL(1)
EVAL _QRNU_RETVAL(2) = 25.3
```

***XMLSAX**
名前 _QRNU_XMLSAX を持つ配列は、デバッグ・ビューがある場合 (*NONE 以外の DBGVIEW パラメーターに値を付けてコンパイルした場合)、モジュールに生成されます。 配列の値は接頭部「*XML_」なしの *XML 特殊語の名前です。 例えば、*XML_START_DOCUMENT の値が 1 の場合、_QRNU_XMLSAX(1) の値は「START_DOCUMENT」になります。

サンプル・デバッグ・セッション:

```
> EVAL event
  EVENT = 2
> EVAL _QRNU_XMLSAX(event)
  _QRNU_XMLSAX(EVENT) = 'END_DOCUMENT   '
```

DEBUG キーワードが *NO を付けて指定されている場合は、モジュールにはデバッグ・エイドは生成されないことを示しています。 これは、DEBUG キーワードを完全に省略した場合と同じです。 *NO が指定されている場合は、これ以 外のパラメーターは指定できません。

DEBUG キーワードが *YES を付けて指定されているか、パラメーターなしで指定されている場合は、DEBUG(*INPUT : *DUMP) を指定した場合と同じです。 *YES が指定されている場合は、これ以外のパラメーターは指定できません。 値 *YES は、互換性のために残されています。*INPUT、*DUMP、および *XMLSAX など、 値をより細かく指定することが推奨されています。

例:

```
   * 1. All of the debugging aids are available
  H DEBUG(*CONSTANTS : *INPUT : *DUMP : *RETVAL : *XMLSAX)
 * 2. None of the debugging aids are available
  H DEBUG(*NO)
 * 3. Only the debugging aid related to input fields is available
  H DEBUG(*INPUT)
 * 4. The debugging aids related to the DUMP operation and
 *    to XML-SAX parsing are available
  H DEBUG(*XMLSAX : *DUMP)
```

注 : DEBUG キーワードは、デバッグ可能なモジュールを作成するかどうかは制御しません。 これは、CRTBNDRPG または CRTRPGMOD コマンドの DBGVIEW パラメーターによって制御されます。 DEBUG キーワードは、追加のデバッグ・エイドを制御します。

DECEDIT(*JOBRUN | '値')

DECEDIT キーワードは、編集済み 10 進数の小数点および桁区切り文字 (桁区切り記号) として使用される文字、および先行ゼロを印刷するかどうかを指定します。

注 : 小数点の右側のゼロは常に印刷されます。

キーワード EXPROPTS(*USEDECEDIT) が指定される場合、DECEDIT キーワードは、文字を数値に変換する %DEC などの組み込み関数の小数点および桁区切り文字も指定します。 569 ページの『組み込み関数を使用して文字値を数値に変換するための規則』 を参照してください。

DECEDIT が指定されていない場合、デフォルトの小数点はピリオド (.) であり、デフォルトの桁区切り文字はコンマ (,) です。

*JOBRUN を指定した場合、実行時にジョブに関連している DECFMT 値が使用されます。 使用可能なジョブ 10 進数形式を以下のテーブルに示します。

表 *82. *JOBRUN を指定した DECEDIT*

ジョブ 10 進数形式	小数点	桁区切り文字	先行ゼロの印刷	編集済み 10 進数
ブランク	ピリオド (.)	コンマ (,)	いいえ	.123
I	コンマ (,)	ピリオド (.)	いいえ	,123
J	コンマ (,)	ピリオド (.)	はい	0,123

値を指定した場合、編集済みの 10 進数が、 以下の使用可能な値に従って印刷されます。

表 *83. '値'を指定した DECEDIT*

'値'	小数点	桁区切り文字	先行ゼロの印刷	編集済み 10 進数
'.'	ピリオド (.)	コンマ (,)	いいえ	.123
','	コンマ (,)	ピリオド (.)	いいえ	,123
'0.'	ピリオド (.)	コンマ (,)	はい	0.123
'0,'	コンマ (,)	ピリオド (.)	はい	0,123

DECPREC(30|31|63)

キーワード DECPREC は、式の中の算術演算の 10 進 (パック、ゾーン、または 2 進) 中間値の 10 進精度を指定する場合に使用します。 10 進中間値は常に正しい精度で保持されますが、このキーワードは、%EDITC、%EDITW、%CHAR、%LEN、 および %DECPOS で使用された 10 進数式が表現される方法に影響を与えます。

DECPREC(30)
デフォルトの 10 進精度です。 これは、影響を受ける命令で使用された場合、10 進数値の最大精度が 30 桁になることを示します。 ただし、式の中の少なくとも 1 つのオペランドが 31 桁の 10 進変数である場合、その式では DECPREC(31) が想定されます。 式の中の少なくとも 1 つのオペランドが 32 桁以上の 10 進変数である場合、 その式では DECPREC(63) が想定されます。

DECPREC(31)
影響を受ける命令で使用された場合、10 進数値の最大精度は 31 桁になります。 ただし、式の中の少なくとも 1 つのオペランドが 32 桁以上の 10 進変数である場合、 その式では DECPREC(63) が想定されます。

DECPREC(63)
影響を受ける命令で使用される桁数は、常に 10 進数の精度に関する通常の規則に従って計算され、その最大桁数は 63 桁になります。

DFTACTGRP(*YES | *NO)

DFTACTGRP キーワードは、作成済みのプログラムが、呼び出されたときに実行する場所である活動化グループを指定します。

*YES を指定した場合、このプログラムは常にデフォルトの活動化グループ内で実行されます。 この活動化グループは、すべてのオリジナル・プログラム・モデル (OPM) プログラム が実行される活動化グループです。 これにより、ILE RPG プログラムは、ファイル共用、 ファイルの範囲、RCLRSC、および監視外の例外の処理の分野で OPM RPG プログラムと 同様に動作します。 プログラムが DFTACTGRP(*YES)で作成された時には，ILE 静的バインドは使用可 能でありません。 これは、このプログラムを作成する 場合、BNDDIR、ACTGRP、または STGMDL のコマンド・パラメーターあるいはキーワードは使用できないことを意味します。 さらに，ソース中の呼び出し命令はプロシージャ ーでなくプログラムを呼び出すものでなけ

ればなりません。 DFTACTGRP(*YES)は, プログラム単位でアプリケーションを ILE RPG に移動する時に便利で す。

*NO を指定した場合、プログラムは ACTGRP コマンド・パラメーターまたはキーワード によって指定された活動化グループに関連付けられ、静的バインドが許可されます。 DFTACTGRP(*NO) は、例えば、名前付きの活動化グループで実行したり、 サービス・プログラムにバインドするなど、ILE 概念を利用したい時に 便利です。

DFTACTGRP キーワードは CRTBNDRPG コマンドを使用した場合のみ有効です。

DFTACTGRP キーワードが指定されていない場合のデフォルト値

コンパイル単位内に何らかの自由形式制御ステートメントがあり、 ACTGRP、BNDDIR、または STGMDL キーワードのうちの 1 つ以上が使用されている場合、 DFTACTGRP(*NO) が想定されます。

それ以外の場合、コマンドに指定された値が使用されます。

次の例では、CTL-OPT ステートメントは自由形式制御ステートメントです。 ACTGRP キーワードが指定されているため、DFTACTGRP(*NO) が想定されます。

```
   CTL-OPT OPTION(*SRCSTMT) ACTGRP(*NEW);
```

次の例では、STGMDL キーワードは固定形式制御ステートメント中に指定されていますが、 自由形式制御ステートメントもあるため、DFTACTGRP(*NO) が想定されます。

```
   CTL-OPT;
H OPTION(*SRCSTMT) STGMDL(*INHERIT)
```

次の例では、自由形式制御ステートメントがありますが、 ACTGRP、BNDDIR、または STGMDL キーワードのどれも指定されていません。 CRTBNDRPG コマンドの DFTACTGRP パラメーターに指定された値 が使用されます。

```
   CTL-OPT OPTION(*SRCSTMT);
```

DFTNAME(RPG 名)

DFTNAME キーワードは、デフォルトのプログラムまたはモジュール名を指定します。 *CTLSPEC が作成コマンドに指定された場合には、この「RPG 名」がプログラムま たはモジュール名として使用されます。 RPG 名が指定されていない場合には、プログラムまたはモジュールのデフォルトの名 前はそれぞれ RPGPGM または RPGMOD となります。 名前に関する RPG の規則 (73 ページの『記号名』を参照) が適用されます。

ENBPFRCOL(*PEP | *ENTRYEXIT | *FULL)

ENBPFRCOL キーワードはパフォーマンス・コレクションを使用可能にするかどうかを指定します。

*PEP を指定した場合、パフォーマンス統計が、プログラム入り口プロシージャーの入り口 と出口のみで収集されます。 これは、オブジェクト内のオブジェクトの メイン・プロシージャーではなく、オブジェクトの実際のプログラム入り口 プロシージャーに適用されます。

*ENTRYEXIT を指定した場合、パフォーマンス統計が、オブジェクトのすべての プロシージャーの入り口と出口で収集されます

*FULL を指定した場合、パフォーマンス統計が、すべてのプロシージャーの入り口と 出口で収集されます。また，統計は，外部プロシージャーに対する各呼び出しの前後に収集されます。

ENBPFRCOL キーワードを指定しない場合、コマンドに指定した値が 使用されます。

EXPROPTS(*MAXDIGITS | *RESDECPOS | *ALWBLANKNUM | *USEDECEDIT)

EXPROPTS キーワードは、式に関連したオプションを制御します。

***MAXDIGITS および *RESDECPOS**
*MAXDIGITS または *RESDECPOS の値を指定して、プログラム全体に使用される精度規則のタイプを制御します。

- 指定できるのは *MAXDIGITS および *RESDECPOS のうちの 1 つのみです。 どちらも指定されない場合は、*MAXDIGITS が想定されます。
- *MAXDIGITS が指定または想定される場合は、デフォルトの精度規則が適用されます。
- *RESDECPOS が指定される場合、「結果の小数点以下の桁数」精度規則が適用され、式の中間結果の小数点以下の桁数が結果よりも大きくなるようにします。

注 : 命令コード拡張機能 R および M は、 それぞれ EXPROPTS(*RESDECPOS) および EXPROPTS(*MAXDIGITS) と同じですが、 それは単一自由形式の式の場合です。

***ALWBLANKNUM**
*ALWBLANKNUM が指定され、%DEC、%DECH、%DECH、%INT、%INTH、%UNS、または %UNSH の文字オペランドがブランクまたは空の場合、結果はゼロになります。 XML-INTO および DATA-INTO では、数値フィールドに指定された値がブランクまたは空の場合、フィールドにゼロが入ります。

*ALWBLANKNUM が指定されない場合、ブランク・オペランドはエラーと見なされます。

***USEDECEDIT**
*USEDECEDIT が指定される場合、%DEC、%DECH、%FLOAT、%INTH、%UNS、または %UNSH の文字オペランドを解釈するときや、XML-INTO および DATA-INTO の数値データを解釈するときに、DECEDIT キーワードで指定された小数点および桁区切り文字が使用されます。 569 ページの『組み込み関数を使用して文字値を数値に変換するための規則』 を参照してください。

*USEDECEDIT が指定されない場合、コンマおよびピリオドはどちらも小数点を表すと見なされ、桁区切り文字は使用できません。

***STRICTKEYS**
*STRICTKEYS は、キーのリストまたはキー付きファイル操作のための %KDS で指定された検索引数の規則に影響します。

- *STRICTKEYS が指定されない場合、検索引数はファイル内のキーと同じデータ・タイプである必要がありますが、検索引数には任意の長さまたは CCSID を指定することができます。
- *STRICTKEYS が指定される場合、検索引数の規則はより厳密に適用されます。 検索引数が規則に従っていない場合、コンパイラーによって発行されるメッセージには理由コードが含まれます。
  - UCS-2 およびグラフィック・キー・フィールドの場合、検索引数の CCSID は、キーの CCSID と同じである必要があります。 理由コード: CCSID。
  - 文字キー・フィールドの場合:
    - DATA (*NOCVT) がファイルに対して有効な場合、検索引数の CCSID はキーの CCSID と同じである必要があります。 理由コード: CCSID。
    - DATA (*NOCVT) がファイルに対して有効でない場合、検索引数の CCSID はジョブの CCSID である必要があります。 理由コード: CCSID-NOT-JOB。
    - 検索引数の定義された長さは、キーの定義された長さ以下である必要があります。 理由コード: LEN。
  - 数値キー・フィールドの場合:
    - 検索引数の小数点以下の桁数は、キーの小数点以下の桁数以下である必要があります。 理由コード: DECIMALS。
    - 検索引数の整数位置の数は、キーの整数位置の数以下である必要があります。 理由コード: INTEGERS。
    - 浮動小数点検索引数の長さは、キーの長さ以下である必要があります。 理由コード: LEN。

- キーまたは検索引数のいずれかが浮動小数点である場合、両方を浮動小数点にする必要があります。 理由コード: FLOAT。
- タイム・スタンプ検索引数の長さは、キーの長さ以下である必要があります。 理由コード: LEN。
- 日付検索引数の年の範囲は、キーの日付形式の年の範囲内にある必要があります。 理由コード: DATFMT-YEARS。
- *USA 形式の時刻検索引数は、キーの形式も *USA の場合にのみ有効です。 理由コード: TIMFMT-USA。

例えば、ファイル MYFILE のキーのタイプは PACKED(5:2) です。 次のプログラムについて考えてみます。

```
DCL-S price INT(5);

price = 1234;
CHAIN (price) MYFILE;
```

- EXPROPTS(*STRICTKEYS) が指定されない場合、CHAIN 命令は許可されますが、PACKED 5:2) 値に値 1234 を割り当てることができないため、実行時に数値オーバーフロー・エラーが発生します。
- EXPROPTS(*STRICTKEYS) が指定される場合、コンパイルは理由コード INTEGERS で失敗します。変数 *price* は最大 5 つの整数桁を持つことができますが、PACKED(5:2) キーは最大 3 つの整数桁しか持つことができないためです。

579 ページの『ファイル命令のキー』を参照してください。

EXTBININT{(*NO | *YES)}

EXTBININT キーワードは、外部 2 進形式で小数点以下の桁数がない外部記述フィールドを外部整数形式であるかのように処理するために使用されます。 これを指定しないか、または *NO と指定した場合、外部記述 2 進フィールドは外部 2 進-10 進形式で処理されます。 任意指定の *YES とともに EXTBININT を指定した場合には、外部記述フィールド は次のように処理されます。

DDS 定義
RPG 外部形式

B(n,0) ここで 1 ≤ n ≤ 4
INT(5)

B(n,0) ここで 5 ≤ n ≤ 9
INT(10)

EXTBININT キーワードを指定することによって、ユーザー・プログラムでは、使 用可能なすべての範囲の DDS 2 進数値を使用することができます。 (DDS 2 進数値の範囲は符号付き整数と同じで、5 桁のフィールドの場合は -32768 から 32767、 あるいは 10 桁のフィールドの場合は -2147483648 から 2147483647 です。)

注 : キーワード EXTBININT を指定した場合に、小数点以下の桁数がない 2 進数の外部 記述サブフィールドは、すべて内部 整数形式を持つものとして定義さ れます。

FIXNBR(*{NO}ZONED *{NO}INPUTPACKED)

FIXNBR キーワードは、無効な 10 進数データをコンパイラーが修正するかどうかを指定します。

任意のまたはすべてのデータ・タイプを任意の順序で指定することができます。 しかし、10 進数データ・タイプを指定する場合、同じデータ・タイプ の *NOxxxx パラメーターを同時に指定してはなりません。その逆も同様です。 たとえば、*ZONED を指定した場合、*NOZONED も指定してはなりません。その逆も同様です。 パラメーターは、 コロンで区切ってください。 1 つのパラメーターを複数回指定してはなりません。

注 : キーワード FIXNBR にペアのデータ・タイプからのメンバーを指定しない場合、 その特定のデータ・タイプについてコマンドに指定した値が使用されます。 たとえば、 制御仕様書にキーワード

FIXNBR(*NOINPUTPACKED) を指定した場合、 ペア (*ZONED、*NOZONED) については、コマンドに暗黙的にまたは明示的に指定したデータ・タイプが使用されます。

*ZONED を指定した場合、無効なゾーン 10 進数は、パック・データへの変換時に コンパイラーによって修正されます。 数字フィールドのブランクはゼロとして扱われます。 各桁は妥 当性検査されます。 桁が有効でない場合には, その桁はゼロで置き換えられます。 符号 が有効でない場合には, その符号は 16 進数'F' の正符号に強制的に変更されます。 符号が有効な場合には、符号はそれぞれ該当する正符号の 16 進数 'F' または 負符号の 16 進数 'D' に変更されます。 結果のパック・データが正しくない場合には, そのデータは訂 正されません。

*NOZONED を指定した場合、無効なゾーン 10 進数データは、パック・データへの変換 時にコンパイラーによって修正されないので、実行時に使用された場合には、10 進数 エラーになります。

*INPUTPACKED を指定した場合、入力仕様の処理時に、 無効なパック 10 進数データが 検出されると、内部変数はゼロに設定されます。

*NOINPUTPACKED を指定した場合、入力仕様の処理時に、無効なパック 10 進数データ が検出されると、10 進数エラーが発生します。

FIXNBR キーワードを指定しない場合、コマンドに指定した値が 使用されます。

FLTDIV {(*NO | *YES)}

FLTDIV キーワードは、式の中のすべての除算命令が浮動小数点で 計算され、浮動タイプの値を戻すということを示します。 これを指定しない場合、あるいは *NO とともに指定した場合、除算命令は パック 10 進数形式で実行されます (ただし、2 つのオペランドのいずれか がすでに浮動形式になっている場合はその限りではありません)。

任意指定で *YES とともに FLTDIV を指定すると、すべての 除算命令は浮動形式で実行されます (結果は常に 15 桁の精度になるように 保証されます)。

FORMSALIGN {(*NO | *YES) }

FORMSALIGN キーワードは、1P 標識によって条件付けされた出力ファイルの最初 の行を印刷装置で位置合わせし、繰り返して印刷できることを指示します。 これを指定しないか、または *NO で指定した場合には、位置合わせは実行されません。 任意指定の *YES とともに指定した場合には、1 ページ目の用紙位置合わせが行 われます。

用紙位置合わせに関する規則

- 1 ページ目標識 (1P) によって条件付けされる印刷装置タイプの装置につい ての装置記入項目を使用し、ファイルの出力仕様に指定されたレコードを必要 な回数だけ書き出すことができます。 行は一度だけ印刷されます。 次に、操作員には、その行を再び印刷するか、あるいはプログラムの残りの部分 から続行するオプションがあります。
- 指定されたすべてのスペースおよびスキップは行が印刷されるたびに実行さ れます。
- プログラムの残りの部分から続行するオプションを選択した時には、行は印 刷し直されません。
- すべての印刷装置ファイルに対してこの機能を実行することができます。
- ページ・フィールドが指定された場合には、行が印刷される初回にだけ増や されます。
- 続行オプションを選択した時に、行カウントは、行カウンターが指定されて いる時にこの機能が 1 回だけ実行された場合と同じになります。

FTRANS {(*NONE | *SRC) }

FTRANS キーワードは、ファイル変換を行うかどうかを指定します。 これを任意指定 の *SRC とともに指定した場合、ファイル変換が行われるので、プログラムの中に 変換テーブルを指定しなければなりません。これを指定しないか、または *NONE で指定した場合には、ファイル変換は行わ れないので、変換テーブルが存在していてはいけません。

GENLVL(番号)

GENLVL キーワードはオブジェクトの作成を 制御します。 コンパイル時に検出されたすべてのエラーの重大度レベルが、指定 した生成重大度レベル以下であれば、オブジェクトが作成されます。 値は 0-20 でなければなりません。 重大度 20 より大きいエラーが検出された場合、 オブジェクトは作成されません。

GENLVL キーワードを指定しない場合、コマンドに指定した値が 使用されます。

INDENT(*NONE | '文字値')

INDENT キーワードは、構造化命令を読みやすくするために、ソース・リスト内で字下げするかどうかを指定します。 また、構造化命令の文節にマークを 付けるために使用する文字も指定します。

注 : ここで要求する字下げは、DBGVIEW(*LIST) を指定した時に作成される リスト・デバッグ・ビューには反映されません。

*NONE を指定した場合、構造化命令はソース・リストで 字下げされません。

文字値を指定した場合、ソース・リストは、構造化命令に関して字下げされます。 ステートメントおよび文節の位置合わせは，選択した文字を使用してマークされます。 長さが 2 文字までの任意の文字リテラルを 指定することができます。

注 : ソースにエラーがある場合、字下げは期待したとおりに 表示されない可能性があります。

INDENT キーワードを指定しない場合、コマンドに指定した値が 使用されます。

INTPREC(10 | 20)

INTPREC キーワードは、式の中の 2 進算術演算の整数および符号なし中間値 の 10 進精度を指定する場合に使用します。 整数および符号なし中間値は、常に 8 バイト形式に 保たれます。 このキーワードは、整数と符号なし中間値 が 2 進算術演算 (+、-、*、/) で使用されるときに、10 進形式に変換される方法に だけ作用します。

デフォルト値の INTPREC(10) は、整数および符号なし演算に関して 10 桁の 10 進精度 を示します。 ただし、式の中の少なくとも 1 つのオペランドが 8 バイト整数 または符号なしフィールドである場合、式の結果では、INTPREC 値に 関係なく、20 桁の 10 進精度があることになります。

INTPREC(20) は、整数および符号なし演算の精度が 20 桁であることを示します。

LANGID(*JOBRUN | *JOB | '言語識別子')

LANGID キーワードは、分類順序が *LANGIDUNQ または *LANGIDSHR である場合に使用する言語識別子を指定します。 LANGID キーワードは SRTSEQ コマンド・ パラメーターまたはキーワードと共に使用して、分類順序テーブルを 選択します。

*JOBRUN を指定した場合、RPG オブジェクトが実行される時にその ジョブに関連している LANGID 値が使用されます。

*JOB を指定した場合、RPG オブジェクトが作成される時にその ジョブに関連している LANGID 値が使用されます。

フランス語を示す 'FRA' やドイツ語を示す 'DEU' などの 言語識別子を指定することができます。

LANGID キーワードを指定しない場合、コマンドに指定した値が 使用されます。

MAIN(main_procedure_name)

MAIN キーワードは、このソース・プログラムがリニア・メイン・モジュールのためのものであり、リニア・メイン・プロシージャーを含んでいることを示します。 リニア・メイン・プロシージャーは、*main_procedure_name* パラメーターで指定され、モジュールのプログラム入り口プロシージャーとなります。

main_procedure_name は、ソース・プログラムで定義されているプロシージャーの名前でなければなりません。 リニア・メイン・プロシージャーは、プログラム呼び出しインターフェースを介してのみ呼び出さ

れるものであり、結合プロシージャー呼び出しでは呼び出せません。 リニア・メイン・プロシージャーに対して再帰呼び出しを行うと、その呼び出しは動的プログラム呼び出しになります。

したがって、以下の規則が適用されます。

- リニア・メイン・プロシージャーのプロトタイプが指定される場合、 EXTPGM キーワードを指定する必要があります。
- リニア・メイン・プロシージャーのプロトタイプが指定されず、 プロシージャー・インターフェースが指定される場合、 プロシージャー・インターフェースで EXTPGM キーワードを指定する必要があります。
- プログラムにパラメーターがなく、プログラムが RPG プログラムから呼び出されない場合には、 プロトタイプもプロシージャー・インターフェースも不要です。
- プロシージャーをエクスポートすることはできません。 そのため、main_procedure_name のプロシージャーの始めの指定で、EXPORT キーワードを指定することはできません。

リニア・メイン・モジュールには、RPG プログラム・サイクルのロジックは組み込まれません。 したがって、サイクルに依存する言語機能は指定できません。

注 : NOMAIN キーワードを使用すると、RPG プログラム・サイクルを含まないモジュールを作成することもできます。

詳しくは、100 ページの『リニア・モジュール』を参照してください。

以下の 2 つの例では、リニア・メイン・プログラムとその /COPY ファイルを示します。

```
      *  The prototype for the linear-main procedure must have
      *  the EXTPGM keyword with the name of the actual program.
     D DisplayCurTime  PR                  EXTPGM('DSPCURTIME')
```

図 *115.* 以下のサンプルのリニア・メイン・プログラムで使用されている */COPY* ファイルの *DSPCURTIME*

```
* The program is named DSPCURTIME, and the module has
      * a linear-main procedure called DisplayCurTime.

      * The Control specification MAIN keyword signifies that this is
      * a linear-main module, and identifies which procedure is the
      * special subprocedure which serves as the linear-main procedure,
      * which will act as the program-entry procedure.

     H MAIN(DisplayCurTime)

      * Copy in the prototype for the program
      /COPY DSPCURTIME

      *---------------------------------------------------
      * Procedure name: DisplayCurTime
      *---------------------------------------------------
     P DisplayCurTime  B
     D DisplayCurTime  PI

      /FREE
       dsply ('It is now ' + %char(%time()));
      /END-FREE
     P DisplayCurTime  E
```

図 *116.* プログラムで使用されるリニア・メイン・プロシージャーのサンプル

以下の例では、プロトタイプを必要としないリニア・メイン・プログラムを示します。 プログラムの名前は PRTCUSTRPT で、モジュールには、リニア・メイン・プロシージャーの PrintCustomerReport が含まれています。 このプログラムは、*CMD オブジェクトのコマンド処理プログラムになるものであるため、RPG プロトタイプは必要ありません。 制御仕様書の MAIN キーワードでは、これがリニア・メイン・モジュールであることを示し、 プログラム入り口プロシージャーとして機能するリニア・メイン・プロシージャーとなる、特殊なサブプロシージャーであるプロシージャーを識別します。

```
     H MAIN(PrintCustomerReport)

      *------------------------------------------------
      * Program name: PrintCustomerReport (PRTCUSTRPT)
      *------------------------------------------------
     P PrintCustomerReport...
     P                 B
     F ... file specifications
     D                 PI             EXTPGM('PRTCUSTRPT')
     D  custName                 25A  CONST

       ... calculations, using the custName parameter

     P PrintCustomerReport...
     P                 E
```

図 *117. RPG* プログラムまたはプロシージャー内から呼び出されないリニア・メイン・プログラム

NOMAIN

NOMAIN キーワードは、このモジュール内にメイン・プロシージャーがないことを指示し ます。 これは、メイン・プロシージャーがコーディングされているモジュールをプログラム入口モジュールとすることはできない、という意味でもあります。 したがって、NOMAIN を指定した場合には、CRTBNDRPG コマンドを使用してプログ ラムを作成することはできません。 代わりに、CRTPGM コマンドを使用して、NOMAIN が指定されたモジュールを、プログラム・エントリー・プロシージャーを持つ別のモジュールにバインドするか、CRTSRVPGM コマンドを使用する必要があります。

メイン・モジュール以外のモジュールに、RPG プログラム・サイクルの論理を組み込むことはできません。したがって、サイクルに依存する言語機能を指定することはできません。

注 : NOMAIN キーワードに加えて、MAIN キーワードを使用しても、RPG プログラム・サイクルを含まないモジュールを作成できます。

詳しくは、100 ページの『リニア・モジュール』を参照してください。

OPENOPT (* {NO }INZOFL * {NO }CVTDATA)

一部または すべてのオプションをどのような順序でも指定することができます。 ただし、オプションを指定する場合は、それと同時に *NOxxxx オプションを 指定することはできず、その逆も同様です。 例えば、*INZOFL と *NOINZOFL はどちらか 1 つしか指定できません。 オプションは、 コロンで区切ってください。

***CVTDATA**
*{NO }CVTDATA オプションは、データベース・ファイル (DISK または SEQ として 定義されます) に対する DATA キーワードのデフォルトを設定します。 *CVTDATA を指定した場合、データベース・ファイルのデフォルトは DATA(*CVT) です。 *NOCVTDATA を指定した場合、データベース・ファイルのデフォルトは DATA(*NOCVT) です。

*CVTDATA と *NOCVTDATA のどちらも指定せず、 制御キーワード CCSID(*EXACT) を指定した場合、OPENOPT(*CVTDATA) が想定されます。

*CVTDATA と *NOCVTDATA のどちらも指定せず、CCSID(*EXACT) を指定しない場合で、 データベース・ファイルに明示的に指定された DATA キーワードがない場合は、 ファイルには有効な DATA キーワードがないものと見なされます。

詳しくは、366 ページの『DATA(*CVT | *NOCVT)』を参照してください。

***INZOFL**
1 つまたは複数の印刷装置ファイルがオーバーフロー標識 (OA-OG または OV) を 指定して定義されているプログラムの場合に、キーワード は、そのオーバーフロー標識を、ファイルが開かれた時点で *OFF にリセットする かどうかを指定します。 OPENOPT キーワードが *NOINZOFL と一緒に指定されている 場合、オーバーフロー標識は、関連の印刷装置ファイルが開かれた時点でも 変更されないままです。

このキーワードが指定されない場合、または *INZOFL と一緒に 指定されている場合、オーバーフロー標識は、関連の印刷装置ファイルが 開かれた時点で *OFF に設定されます。

OPTIMIZE(*NONE | *BASIC | *FULL)

OPTIMIZE キーワードは、もしあれば、オブジェクトの最適化のレベルを指定します。

*NONE を指定した場合、生成されるコードは最適化されません。 これは，変換時間の面で一番早いものです。 これによって、デバッグ・モードの間に変数を表示して変更することができます。

*BASIC を指定した場合、生成されるコードにいくらかの最適化が実行されます。 これによって、 プログラムがデバッグ・モードの間にユーザー変数を表示することができますが、変更することはできません。

*FULL を指定した場合、最も効率的なコードが生成されます。 変換時間は最も長くなります。 デバッグ・モードで、ユーザー変数を修正できませんが、表示することはできます。 ただし、表示される値は現行値ではない可能性があります。

OPTIMIZE キーワードを指定しない場合、 コマンドに指定した値が 使用されます。

OPTION(*{NO}XREF *{NO}GEN *{NO}SECLVL *{NO}SHOWCPY *{NO}EXPDDS *{NO}EXT *{NO}SHOWSKP) *{NO}SRCSTMT) *{NO}DEBUGIO) *{NO}UNREF

OPTION キーワードは、ソース・メンバーのコンパイル時に使用するオプションを指定します。

一部または すべてのオプションをどのような順序でも指定することができます。 しかし、コンパイル・オプションを指定する場合、同じコンパイル・オプション の *NOxxxx パラメーターを同時に指定してはいけません。その逆も同様です。 例えば，*XREF を指定した場合に，*NOXREF も指定することはできず，その逆も指 定できません。 オプションは、コロンで区切ってください。 1 つのオプションを複数回指定することはできません。

注 : キーワード OPTION にペアのオプションからのメンバーを指定しない場合、 その特定のオプションについてはコマンドに指定した値が使用されます。 たとえ ば、制御仕様書にキーワード OPTION(*XREF : *NOGEN : *NOSECLVL : *SHOWCPY) を指定した場合、ペア (*EXT、*NOEXT)、(*EXPDDS、 *NOEXPDDS) および (*SHOWSKP、*NOSHOWSKP) については、コマンドに暗黙的にまたは 明示的に指定したオプションが使用されます。

*XREF を指定した場合、(該当する場合は) ソース・メンバーに関する相互参照リストが生成されます。 *NOXREF は相互参照リストが生成されないことを示します。

*GEN を指定した場合、コンパイラーによって戻された最高の重大度レベルが、GENLVL オプションで指定した重大度レベルを超えない場合に、プログラム・オブジェクトが作成されます。*NOGEN の場合、オブジェクトは作成されません。

*SECLVL を指定した場合、メッセージ要約セクションの第 1 レベル・メッセージ・テキストに続く行に第 2 レベルのメッセージ・テキストが印刷されます。*NOSECLVL の場合、第 1 レベル・メッセージ・テキストに続く行に第 2 レベルのメッセージ・テキストは印刷されません。

*SHOWCPY を指定した場合、コンパイラー・リストは、/COPY コンパイラー指示によって組み込まれたメンバーのソース・レコードを示します。*NOSHOWCPY の場合、/COPY コンパイラー指示によって組み込まれたメンバーのソース・レコードは示されません。

*EXPDDS を指定した場合、リスト内の外部記述ファイルの拡張およびキー・フィールド情報が表示されます。*NOEXPDDS を指定した場合、リスト内の外部記述ファイルの拡張およびキー・フィールド情報は表示されません。

*EXT を指定した場合、コンパイル時に参照された外部プロシージャーおよびフィールドがリストに組み込まれます。*NOEXT を指定した場合､コンパイル時に参照された外部プロシージャーおよびフィールドはリストに組み込まれません。

*SHOWSKP を指定した場合、コンパイラーがスキップしたかどうかに関係なく、リストのソース部分にあるすべてのステートメントが表示されます。*NOSHOWSKP の場合､リストのソース部分のスキップされたステートメントは表示されません。 コンパイラーは、/IF、/ELSEIF、または /ELSE 指示ステートメントの結果としてステートメントをスキップします。

*SRCSTMT が指定されている場合、リストのステートメント番号 は、次のようにソース ID と SEU 順序番号から生成されます。

```
stmt_num = source_ID * 1000000 + source_SEU_sequence_number
```

例えば、メイン・ソース・メンバーのソース ID は 0 です。 ソース・ファイルの最初の行に順序番号 000100 がある場合、この仕様のステートメント番号は 100 になります。 ソース ID が 27 で、ソース順序番号 000100 の /COPY ファイル・メンバーからの行のステートメント番号は 27000100 になります。
*NOSRCSTMT は、行番号が順次に割り当てられることを示します。

*DEBUGIO が指定されている場合、すべての入出力仕様に停止点が生成されます。*NODEBUGIO は、これらの仕様で停止点が生成されないことを示します。

*UNREF が指定された場合、すべての変数がモジュールに生成されます。 *NOUNREF が指定された場合、他の何らかのモジュールが必要としない限り、参照されない変数は生成されません。 OPTION(*NOUNREF) に適用される規則は、以下のとおりです。

- EXPORT を指定して定義された変数は、参照されるかどうかに関わらず、モジュールに生成されます。
- 参照されない変数が IMPORT を指定して定義された場合、入力仕様に記載されていればモジュールに生成されます。
- プログラムにおいて、*IN 標識が *INxx 参照によって明示的に使用されない場合、あるいは条件付け標識または結果標識によって暗黙的に使用されない場合、*IN 標識配列および *INxx 標識はモジュールに生成されません。
- EXPORT または IMPORT を指定して変数が定義されていない場合:
  - ファイルに関連付けられる変数や、演算仕様書または出力仕様で使用される変数は、常に生成されます。
  - 定義仕様書でのみ指定されている変数については、参照されない場合にはモジュールに生成されません。
  - 入力仕様でのみ参照される変数は、DEBUG、DEBUG(*YES)、DEBUG(*INPUT) のいずれかが制御仕様書で指定されている場合にのみ、モジュールに生成されます。

OPTION キーワードを指定しない場合、 コマンドに指定した値が使用されます。

PGMINFO(*PCML | *NO | *DCLCASE { : *MODULE | *V6 | *V7 ... })

PGMINFO キーワードは、モジュールまたはプログラムのプログラム・インターフェース情報の生成方法を指定します。

***NO**
*NO を指定すると、プログラム・インターフェース情報は生成されません。

***PCML**
*PCML を指定すると、プログラム・インターフェース情報がプログラム呼び出しマークアップ言語の形式で生成されます。

***MODULE**
*MODULE を指定すると、プログラム・インターフェース情報がモジュールに直接生成されます。 そのモジュールを後で使用して、プログラムまたはサービス・プログラムを作成すると、そのプログラムまたはサービス・プログラムにもプログラム・インターフェース情報が組み込まれます。 これによって、API QBNRPII を使用して、その情報を取り出すことができるようになります。

***DCLCASE**
*DCLCASE を指定すると、PCML 内の名前は、名前の宣言の大/小文字の区別を使用して生成されます。343 ページの『名前の宣言の大/小文字の区別』を参照してください。

***V6 と *V7**
*V6 を指定すると PCML バージョンは 6.0 になることが指定されます。 *V7 を指定すると PCML バージョンは 7.0 になることが指定されます。

バージョン 7.0 の可変長フィールドの処理はバージョン 6.0 とは異なります。 バージョン 7.0 の PCML が生成される場合、可変長配列と可変長サブフィールドに関する制約事項は適用されません。

PCML の可変長フィールドの処理方法に外部依存関係がある場合は *V6 を指定してください。 例えば、ProgramCallDocument クラスを使用する Java コードを使用している場合、Java コードは、構造として生成されている可変長フィールドの PCML に依存します。 外部依存関係を変更できない場合、モジュールの PCML がバージョン 6.0 を使用して生成されるように *V6 を指定してください。

注 : *V6 や *V7 のパラメーターをコーディングする代わりに、 QIBM_RPG_PCML_VERSION 環境変数を使用して PCML バージョンを制御することもできます。

ユーザー・ジョブでバージョン 7.0 で PCML が生成されるように要求するには、以下のコマンドを使用します。

```
ADDENVVAR QIBM_RPG_PCML_VERSION VALUE('7.0')
```

すべてのジョブでバージョン 7.0 で PCML が生成されるように要求するには、以下のコマンドを使用します。

```
ADDENVVAR QIBM_RPG_PCML_VERSION VALUE('7.0') LEVEL(*JOB)
```

ターゲット・リリース V7R2M0 または V7R3M0 でコンパイルされるプログラムのデフォルトの PCML バージョンは 6.0 です。 以降のターゲット・リリースでは、デフォルトが 7.0 に変更されます。

PGMINFO キーワードで *V7 パラメーターをコーディングするよりも、環境変数を使用してデフォルト・バージョンを 7.0 にすることを推奨します。 *V7 パラメーターをコーディングすると、PCML の今後のバージョンで可能性がある PCML の機能拡張を利用できなくなります。

環境変数を使用してデフォルトの PCML バージョンを変更した場合、 バージョン 6.0 で生成されている PCML に外部依存関係があるすべてのモジュールに対して *P6 パラメーターを PGMINFO キーワードに追加できます。

最初のパラメーターは *NO、*PCML、または *DCLCASE のいずれかでなければなりません。 最初のパラメーターが *NO または *DCLCASE の場合は、追加のパラメーターを使用することはできません。

最初のパラメーターが *PCML の場合は、少なくとも 1 つの追加パラメーターが必要となります。 追加のパラメーターは *DCLCASE、*MODULE、あるいは *V6 または *V7 のどちらかを任意に組み合わせて指定できます。

PGMINFO のデフォルトの設定値は、コマンド CRTRPGMOD または CRTBNDRPG のパラメーター PGMINFO および INFOSTMF で指定されている値になります。 キーワード PGMINFO がコマンド・パラメーター PGMINFO および INFOSTMF と競合する場合には、制御仕様書のキーワードの値が、コマンドで指定されている値よりも優先されます。 ただし、コマンド・パラメーターからの要求と PGMINFO キーワードからの要求が異なるが競合しない場合には、コマンド・パラメーターの値と PGMINFO キーワードの値がコンパイラーによってマージされます。

名前の宣言の大/小文字の区別

- 定義ステートメントによってデータ構造、フィールド、またはサブフィールドが定義された場合は、その定義ステートメントに使用された大/小文字の区別が、名前の宣言の大/小文字の区別と見なされます。
- データ構造が LIKEREC データ構造として定義された場合は、サブフィールドの宣言では大文字が使用されます。
- データ構造が外部記述データ構造として定義された場合は、サブフィールドが外部サブフィールド・ステートメント内に出現した場合を除き、サブフィールドの宣言には大文字が使用されます。
- 定義ステートメントによってフィールドが定義されない場合、ソース内で名前が次のいずれかの用途で使用された最初のオカレンスでの大/小文字の区別が、宣言の大/小文字の区別となります。
  - 入力仕様
  -「長さ」記入項目が指定されている固定形式演算仕様
  -「結果フィールド」記入項目に名前が現れる *LIKE DEFINE ステートメント
- LIKEDS を使用してデータ構造が定義されると、新しいデータ構造のサブフィールドの宣言の大/小文字の区別は、親データ構造のサブフィールドと同じになります。

- プロシージャーの宣言の大/小文字の区別として、そのプロシージャーの DCL-PROC または「プロシージャーの始め」ステートメントで指定された名前の大/小文字の区別が使用されます。

例

- PGMINFO(*PCML) や INFOSTMF('mypgm.pcml') などのコマンド・パラメーターによって、ストリーム・ファイルに情報を入れるように指定されており、さらに PGMINFO(*PCML:*MODULE) キーワードによって、モジュールに情報を入れるように指定されている場合には、両方の要求がマージされて、PGMINFO の最終的な値は PGMINFO(*PCML:*ALL) INFOSTMF('mypgm.pcml') になります。
- コマンド・パラメーター PGMINFO(*PCML *ALL) INFOSTMF('/home/mypcml/mypgm.pcml') によって、モジュールとストリーム・ファイルに情報を入れるように指定されており、さらに PGMINFO(*NO) キーワードによって、一切の情報を保存しないように指定されている場合には、PGMINFO キーワードがコマンド値よりも優先され、PGMINFO の最終的な値は PGMINFO(*NO) になります。

PRFDTA(*NOCOL | *COL)

PRFDTA キーワードは、プロファイル作成データのコレクションを 使用可能にするかどうかを指定します。

*NOCOL を指定した場合、このオブジェクトに関する プロファイル作成データのコレクションが使用可能になりません。

*COL を指定した場合、このオブジェクトに関するプロファイル作成データのコレクションが使用可能になります。*COL を指定できるのは、オブジェクトの最適化レベルが *FULL の場合のみです。

PRFDTA キーワードを指定しない場合、コマンドに指定した値が 使用されます。

REQPREXP(*NO | *WARN | *REQUIRE)

REQPREXP キーワードは、メイン・プロシージャーおよびエクスポートされたプロシージャーにプロトタイプが必要かどうかを指定します。

***NO**
プロトタイプは，メイン・プロシージャーにもエクスポートされるプロシージャーにも必要ありません。 これはデフォルト値。

***警告**
メイン・プロシージャーまたはエクスポートされるプロシージャーにプロトタイプが指定されていない場合は，重大度 10 の警告が出されます。

***REQUIRE**
プロトタイプは，メイン・プロシージャーおよびエクスポートされるプロシージャーに必要です。 メイン・プロシージャーまたはエクスポートされるプロシージャーにプロトタイプが指定されていない場合は，重大度 30 のエラーが出されます。

REQPREXP キーワードは、CRTBNDRPG コマンドまたは CRTRPGMOD コマンドの REQPREXP パラメーターによって設定された値をオーバーライドします。

ヒント : 特定のエクスポートされたプロシージャーにプロトタイプが必要ではないことが自明な場合、プロシージャーを開始するステートメント、またはプロシージャー・インターフェース (それがサイクル・メイン・プロシージャーの場合) に REQPROTO(*NO) を指定することができます。

例

- コマンド処理プログラムとしての使用に特化したプログラムでは、他の RPG プログラムから呼び出せるようにするためのプロトタイプは必要ありません。
- Java ネイティブ・メソッドは通常、Java メソッドからのみ呼び出されるため、ほとんどの Java ネイティブ・メソッドは、RPG 呼び出し元のプロトタイプを必要としません。

エクスポートされたプロシージャーについては 538 ページの『REQPROTO(*NO)』、サイクル・メイン・プロシージャーのプロシージャー・インターフェースについては 476 ページの『REQPROTO(*NO)』 を参照してください。

SRTSEQ(*HEX | *JOB | *JOBRUN | *LANGIDUNQ | *LANGIDSHR | 'ソート・テーブル名')

SRTSEQ キーワードは、ILE RPG ソース・プログラムで使用する 分類順序テーブルを指定します。

*HEX を指定した場合、分類順序テーブルは使用されません。

*JOB を指定した場合、*PGM が作成される時のその ジョブに関する SRTSEQ が使用されます。

*JOBRUN を指定した場合、*PGM が実行される時のその ジョブに関する SRTSEQ が使用されます。

*LANGIDUNQ を指定した場合、固有加重テーブルが使用されます。 この特殊な値 は LANGID コマンド・パラメーターまたはキーワードと共に使用され、正しい 分類順序テーブルが決定されます。

*LANGIDSHR を指定した場合、共用加重テーブルが使用されます。 この特殊な値 は LANGID コマンド・パラメーターまたはキーワードと共に使用され、正しい 分類順序テーブルが決定されます。

オブジェクトに使用する分類順序テーブルの名前を指定するために、分類テーブル名 を指定することができます。 この名前は、ライブラリー名とその後に続く斜線区切り 記号によって修飾することができます ('ライブラリー名/分類テーブル名')。 ライブラリー名は検索するライブラリーの名前です。 ライブラリー名を指定しない場合、 分類テーブル名を探すために *LIBL が使用されます。

SRTSEQ パラメーターおよび LANGID パラメーターを使用して代替照合順序を決定したい場合は、制御仕様書に ALTSEQ(*EXT) も指定することが必要です。

SRTSEQ キーワードを指定しない場合、コマンドに指定した値が 使用されます。

STGMDL(*INHERIT | *SNGLVL | *TERASPACE)

STGMDL キーワードは、プログラムまたはモジュールの記憶域モデルを指定します。

- *SNGLVL は、単一レベル記憶域モデルを指定する場合に使用します。
- *INHERIT は、継承記憶域モデルを指定する場合に使用します。
- *TERASPACE は、テラスペース記憶域モデルを指定する場合に使用します。

単一レベル記憶域モデルのプログラムまたはサービス・プログラムが活動化されて実行されると、 プログラムには自動記憶域および静的記憶域用に単一レベル記憶域が提供されます。 単一レベル記憶域のプログラムまたはサービス・プログラムは、単一レベル記憶域活動化グループ内でのみ実行されます。 DFTACTGRP(*YES) を指定してコンパイルされたプログラムは、単一レベル記憶域モデルのプログラムでなければなりません。

STGMDL キーワードが DFTACTGRP キーワードの設定に与える影響については、333 ページの『DFTACTGRP(*YES | *NO)』 を参照してください。

テラスペース記憶域モデルのプログラムまたはサービス・プログラムが活動化されて実行されると、 プログラムには自動記憶域および静的記憶域用に単一レベル記憶域が提供されます。 テラスペース記憶域のプログラムまたはサービス・プログラムは、テラスペース記憶域活動化グループ内でのみ実行されます。

継承記憶域モデルのプログラムまたはサービス・プログラムが活動化されると、 活動化されて入る活動化グループの記憶域モデルを採用します。 言い換えれば，プログラムは呼び出し元のストレージ・モデルを継承します。 *INHERIT 記憶域モデルが選択された場合、 ACTGRP パラメーターまたはキーワードを使用して、活動化グループに対して *CALLER が指定されなければなりません。

継承ストレージ・モデルのモジュールは，単一レベルのストレージ・モデル，テラ・スペースのストレージ・モデル または継承ストレージ・モデルの ストレージ・モデルを使用するプログラムやサービス・プログラムにバインドできます。

単一レベル記憶域モデルのモジュールは、単一レベル記憶域を使用するプログラムおよびサービス・プログラムにしかバインドすることができません。

テラ・スペース・ストレージ・モデルのモジュールは，テラ・スペース・ストレージを 使用するプログラムやサービス・プログラムにのみバインドできます。

STGMDL キーワードを指定しない場合、コマンドに指定した値が 使用されます。

TEXT(*SRCMBRTXT | *BLANK | '記述')

TEXT キーワードによって、オブジェクトおよびその機能について簡単に記述するテキストを入力することができます。 このテキストは、オブジェクト の作成時に使用され、オブジェクト情報の表示時に表示されます。

*SRCMBRTXT を指定した場合、ソース・メンバーのテキストが使用されます。

*BLANK を指定した場合、テキストは表示されません。

リテラルを指定する場合、最大 50 文字まで指定することができ、アポストロフィで 囲む必要があります。(アポストロフィは 50 文字のストリングに 含まれません。)

TEXT キーワードを指定しない場合、コマンドに指定した値が 使用されます。

THREAD(*CONCURRENT | *SERIALIZE)

THREAD キーワードは、作成される ILE RPG モジュールが、マルチスレッド化された環境内で安全に実行されるよう意図されたものであることを示します。 スレッド・セーフティーに関する主な課題の 1 つとして、静的記憶域の処理があります。 複数のスレッドが同じ記憶位置に同時にアクセスすると、予期せぬ結果が生じることがあります。

THREAD キーワードを指定すると、モジュールの静的記憶域に関して、そのモジュールのスレッド・セーフを確保するのに役立ちます。 スレッドごとに別々の静的記憶域を設けるか、一度に 1 つのスレッドのみがモジュールにアクセスできるように制限するか、選択することができます。 同じプログラムまたはサービス・プログラム内で、これら 2 つのタイプのモジュールを組み合わせて使用してもかまいません。 ただし、マルチスレッド環境で実行するすべてのモジュールで、必ず、THREAD キーワードを指定する必要があります。

自動変数については、特に考慮する必要はありません。 自動変数は、プロシージャーの呼び出しごとに作成されるので、本質的にスレッド・セーフです。 プロシージャーの自動記憶域は、それぞれのスレッドに固有の記憶域内に割り振られます。

THREAD(*CONCURRENT) がコーディングされた場合は、条件 *THREAD_CONCURRENT が定義されます。 THREAD(*SERIALIZE) がコーディングされた場合は、条件 *THREAD_SERIALIZE が定義されます。 詳しくは、88 ページの『制御仕様書キーワードに関連する条件』を参照してください。

*THREAD(*CONCURRENT)*

THREAD(*CONCURRENT) を指定すると、複数のスレッドをモジュールで同時に実行できるようになります。 デフォルトでは、モジュールの静的記憶域は、すべてスレッド・ローカル記憶域内にあります。 つまり、静的変数 (コンパイラー内部変数を含む) のコピーを、各スレッドがモジュール内に個別に持つことになります。 これによって、モジュール内で複数のスレッドがプロシージャーを同時に実行できるようになり、さらに各スレッドをそれぞれ完全に独立したものにすることができます。 例えば、プロシージャー PROCA において、あるスレッドがファイルを読み取るループの最中であるときに、同じプロシージャーの前半で別のスレッドを同時に実行して、そのスレッドで独自に使用するファイルをオープンする準備を行う、といったこともできます。 また、グローバル変数 NAME がモジュールに指定されている場合に、NAME の値を、あるスレッドでは「Jack」として、別のスレッドでは「Jill」とすることも可能です。 スレッド・ローカル静的変数を使用すると、複数のスレッドを独立して実行させることができます。

STATIC(*ALLTHREAD) キーワードを使用すると、静的変数の一部をすべてのスレッドで共用することができます。 このキーワードを使用する場合には、プロシージャーで記憶域がスレッド・セーフな方法で使用されるように対策を立てる必要があります。346 ページの『THREAD(*CONCURRENT | *SERIALIZE)』を参照してください。

また、「プロシージャーの始め」の指定で SERIALIZE キーワードを指定することで、個々のプロシージャーへのアクセスを逐次化するという方法をとることもできます。 コードのセクションの特定部分で、一度に 1 つのスレッドのみがアクティブになるようにしたい場合には、そのコードを逐次化されたプロシージャーへ移動することができます。

*THREAD(*SERIALIZE)*

THREAD(*SERIALIZE) を指定すると、モジュール内のプロシージャーへのアクセスが逐次化されます。 モジュール内のコードはすべて、マルチスレッド化された環境内で 呼び出されると、そのコードを 1 度に使用できるスレッドは 1 つだけです。

スレッドに関する一般的な考慮事項

RPG における 2 つのタイプのスレッド・セーフティーの利点と欠点を確認するには、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」のマルチスレッド・アプリケーションに関するセクションを参照してください。 システム機能のうち、マルチスレッド化された環境内で使用できないもの、またはサポートされないものについてのリストを確認するには、以下の URL のトピック「プログラミング」にある資料「マルチスレッド化されたアプリケーション」を参照してください。

```
http://www.ibm.com/systems/i/infocenter/
```

スレッド・セーフ・プログラム内では次の標識は使用できません。

- *INUx 標識
- 外部標識 (*INU1 から *INU8)
- CALL または CALLB 命令の LR 標識

THREAD キーワードを使用するときは、次の点に注意してください。

- モジュール間またはスレッド間で共用される記憶域を、スレッド・セーフ方式で使用できるかどうかは、プログラマー次第です。 これには、以下が含まれます。
  - エクスポートおよびインポートされることにより、明示的に 共用される記憶域
  - プロシージャーが、パラメーターまたはポインター・パラメーター のアドレス、または割り振り済み記憶域を保管し、それをそれ以降の呼び出し で使用する、という理由によって共用される記憶域。
  - 変数の定義で STATIC(*ALLTHREAD) が指定されたために共用されている記憶域
- 共用ファイルが複数の言語 (RPG と C、または RPG と COBOL など) によって使用 される場合、そのファイルに 1 度にアクセスする言語は必ず 1 つだけに 限定してください。

TIMFMT(形式 {区切り記号 })

TIMFMT キーワードは、プログラム内の時刻リテラルの内部時刻形式および時刻フ ィールドのデフォルトの内部形式を指定します。 定義仕様書の TIMFMT キーワードでフィールドの形式を指定することによって、そ の特定のフィールドについて異なる内部時刻形式を指定することができます。

/SET および /RESTORE 指示を使用して、 定義のデフォルト時刻形式を一時的に変更することができます。81 ページの『/SET』を参照してください。

TIMFMT キーワードが指定されない場合には、*ISO 形式と見なされます。内部形式について詳しくは、247 ページの『内部形式および外部形式』を参照してください。

276 ページの表 74 は、サポートされている時刻形式およびその区切り記号をリストしています。

TRUNCNBR(*YES | *NO)

TRUNCNBR キーワードは、オブジェクトの実行時に数値オーバーフローが発生した場合に、切り捨てた値を結果フィールドに転送するか、あるいは、エラーを生成するかどうかを指定します。

注 : TRUNCNBR オプションは式の中で行なわれる演算には適用されません。（式は拡張演算項 目 2 フィールドにあります。） これらの演算でオーバーフローが起こった場合には，常 にエラーが発生します。

*YES を指定した場合、数値オーバーフローが無視されて、切り捨てられた値が 結果フィールドに転送されます。

*NO を指定した場合、数値オーバーフローが検出されると、 実行時エラーが生成されます。

TRUNCNBR キーワードを指定しない場合、コマンドに指定した値が 使用されます。

USRPRF (*USER | *OWNER)

USRPRF キーワードは、作成済みのプログラム・オブジェクトを実行するユーザー・プロファイルを指定します。 プログラム所有者またはプログラム・ユーザーのプロファイルは、そのプログラムを実行したり、そのプログラムでどのオブジェクトが使えるか (各オブジェクトに対してそのプログラムがもつ権限を 含む) を制御したりするために使用されます。 このキーワードは、プログラムがすでに存在する場合には、更新されません。

*USER を指定した場合、プログラムのユーザーのユーザー・プロファイルが、 作成済みのプログラム・オブジェクトを実行します。

*OWNER を指定した場合、プログラムのユーザーと所有者の両方のユーザー・ プロファイルが、作成済みのプログラム・オブジェクトを実行します。 両方の ユーザー・プロファイル内のオブジェクト権限を集めたセットが、プログラムの実行 中にオブジェクトを検出しアクセスするために使用されます。 プログラム中で作成されたオブジェクトは、 プログラムのユーザーによって所有されます。

USRPRF キーワードを指定しない場合、コマンドに指定した値が 使用されます。

USRPRF キーワードは CRTBNDRPG コマンドを使用した場合のみ有効です。

VALIDATE(*NODATETIME)

VALIDATE キーワードは、日付データ、時刻データ、およびタイム・スタンプ・データ が、使用される前に検証される必要があるかどうかを指定します。

このキーワードが指定されていない場合、日付データ、時刻データ、およびタイム・スタンプ・データは、使用される前に検証されます。

*NODATETIME が指定されている場合、コンパイラーは、 日付データ、時刻データ、およびタイム・スタンプ・データの検証の実行を省略します。

このキーワードを指定することによって RPG プログラムのパフォーマンスが向上する可能性があります。 場合によっては、コンパイラーは、日付データ、時刻データ、およびタイム・スタンプ・データ を英数字データであるかのように扱って、実際の日付データ、時刻データ、およびタイム・スタンプ・データを処理する高コストの操作を 回避できます。

VALIDATE(*NODATETIME) で検証が省略されるいくつかの例を以下に示します。

- タイム・スタンプ・フィールドが、比較、ソート、または検索操作で使用される場合、 比較中にはタイム・スタンプ・フィールドの検証は行われません。
- 日付形式 *ISO または *JIS の日付フィールド が、比較、ソート、または検索操作で使用される場合、比較中には日付フィールドの検証は行われません。
- 時刻形式が *USA 以外である時刻フィールドが、比較、ソート、 または検索操作で使用される場合、比較中には時刻フィールドの検証は行われません。
- 日付データ、時刻データ、またはタイム・スタンプ・データの代入が行われ、 ソースとターゲットの形式および区切り記号が同じである場合、 代入の前にソースの検証は行われません。 これは、代入演算、定数参照で渡されるパラメーターおよび値参照で 渡されるパラメーターの一時的な値への代入、RETURN 命令、 入力仕様のためのフィールド移動、および出力仕様のためのフィールド移動にも当てはまります。
- 日付データ、時刻データ、またはタイム・スタンプ・データがフィールド突き合わせ処理または制御レベル 処理のために比較される場合、データが *ISO 形式の場合はデータの検証は行われません。

注意 :

検証が行われないと、不正データは検出されません。

このキーワードは、すべての日付フィールド、時刻フィールド、およびタイム・スタンプ・フィールドに含まれているデータが 常に有効であると確信できる場合にのみ使用してください。 例えば、デフォルトの初期設定がブランクであるデータ構造の場合、 日付サブフィールド、時刻サブフィールド、およびタイム・スタンプ・サブフィールドは、 無効な値であるブランクで初期化されます。 VALIDATE(*NODATETIME) を指定して、これらのサブフィールドのいずれかを 使用した場合、無効なデータが操作で使用されることになります。さらに、代入を介してプログラム内の他のフィールドに 無効データが伝搬されたり、比較操作で無意味な結果になったりする可能性があります。

この注意事項は、省略される検証のリストに入っていない、日付、時刻、およびタイム・スタンプの操作 にも当てはまります。 将来、VALIDATE(*NODATETIME) が指定された場合に省略される検証が追加 される可能性があります。

推奨事項:

- 使用する日付データ、時刻データ、およびタイム・スタンプ・データが常に有効であると確信がある場合は、 次のようにします。
  - 可能な場所では、日付フィールドに *ISO または *JIS 形式を使用し、 時刻フィールドに *USA 以外の形式を使用します。 これによって、比較および代入を伴う操作が、データが英数字であるかのように実行されることが可能になります。
  - それ以外の場合、可能な場所ではすべての日付フィールドおよび時刻フィールドに同じ形式を使用します。 これによって、代入を伴う操作が、データが英数字であるかのように実行されることが可能になります。
- 使用する日付データ、時刻データ、およびタイム・スタンプ・データが常に有効であるとの確信がない場合は、 VALIDATE(*NODATETIME) キーワードを指定しないでください。 このキーワードは、不必要な検証を除去することのみを目的としています。 正しくない日付データ、時刻データ、またはタイム・スタンプ・データをエラーなしで使用できるようにするという目的ではありません。

ファイル仕様書

ファイル仕様書は、モジュールまたはプロシージャーによって使用される各ファイルを識別するものです。プログラム内の各ファイルごとに、対応するファイル仕様書ステートメントが 必要です。

ファイルは、プログラム記述または外部記述のいずれかが可能です。 プログラム記述ファイルの場合には、レコードおよびフィールド記述が (入力お よび出力仕様を使用して) RPG プログラムに組み込まれます。外部記述ファイルには、DDS、SQL コマンド、またはスクリーン・デザイナーやプリント・デザイナー (Rational Developer for i で使用可能なものなど) を使用して外部的に定義された、レコード記述およびフィールド記述があります。

外部記述ファイルは、自由形式では、 ファイル装置キーワードに *EXT を指定するか、 または、キーワードをパラメーターなしで指定することによって定義されます。 固定形式でプログラム記述ファイルを定義するには、 ファイル仕様書の 22 桁目に E を指定します。

自由形式でプログラム記述ファイルを 定義するには、ファイル装置キーワードにレコード長を指定します。固定形式でプログラム記述ファイルを定義するには、 ファイル仕様書の 22 桁目に F を指定します。

ファイル仕様書は、以下の 2 つの異なる形式で指定できます。

- 350 ページの『自由形式のファイル定義ステートメント』
- 352 ページの『従来型のファイル記述仕様書ステートメント』

次の制約事項が適用されます。

- 1 次ファイルは 1 つしか指定できません。 グローバル・ファイルとして指定される必要があります。 1 次ファイルの存在が必ずしも必要というわけではありません。
- モジュールで許可されるレコード・アドレス・ファイルは、1 つだけです。 このファイルは、グローバル・ファイルとして定義される必要があります。
- メイン・ソース・セクションで定義されるグローバル・ファイルとして、最大 8 個の PRINTER ファイルが使用できます。 各プロシージャーでは、最大 8 個のローカル PRINTER ファイルが使用できます。
- 使用できるファイルの数に制限はありません。
- サブプロシージャーで定義されるローカル・ファイルは、全手順ファイルでなければなりません。
- サブプロシージャーで定義されるファイルには、入出力仕様がありません。 そのため、データ構造を使用して、すべての入出力を行う必要があります。

自由形式のファイル定義ステートメント

自由形式のファイル定義ステートメントは、DCL-F で始まり、 その後にファイル名が続き、さらにその後にキーワードが続き、最後はセミコロンで終わります。

ファイル名が 10 文字より長い場合は、EXTDESC キーワードを使用する必要があります。 コンパイル時および実行時のファイル名の使用方法について詳しくは、174 ページの『ファイル名に関する規則』 を参照してください。

デフォルトでは、ファイルの装置は DISK 装置であり、ファイルが外部記述ファイルであることを *EXT パラメーター が示します。 異なる装置を指定する場合、または明示的に DISK 装置を指定する場合は、 最初のキーワードとしてファイル装置キーワードを 指定する必要があります。

別のファイルと類似したファイルを定義する場合は、 最初のキーワードとして LIKEFILE キーワード を指定する必要があります。

ファイルは、USAGE キーワードに従って、 入力、更新、出力、または削除用にオープンされます。

KEYED キーワードを使用して、ファイルをキー付きファイルとして指定します。

自由形式のファイル定義ステートメントの内部で使用を許可されている指示は、 /IF、/ELSEIF、/ELSE、および /ENDIF のみです。

いくつかのファイルの定義例を以下に示します。

1. ファイル *file1a* および *file1b* は、外部記述 DISK ファイルであり、入力用にオープンされます。 装置タイプおよび使用法は、ファイル *file1a* ではデフォルトによって決定され、 *file1b* では明示的に指定されています。
2. ファイル *file2* は、外部記述 PRINTER ファイル であり、出力用にオープンされます。 使用法はデフォルトで決定されます。
3. ファイル *file3* は、外部記述 SEQ ファイル であり、入力用にオープンされます。 使用法はデフォルトで決定されます。
4. ファイル *file4* は、外部記述 WORKSTN ファイル であり、入力および出力用にオープンされます。 使用法はデフォルトで決定されます。
5. ファイル *file5* は、外部記述のキー付き DISK ファイル であり、入力および更新用にオープンされます。

```
DCL-F file1a; 1
DCL-F file1b DISK(*EXT) USAGE(*INPUT);

DCL-F file2 PRINTER; 2

DCL-F file3 SEQ; 3

DCL-F file4 WORKSTN; 4

DCL-F file5 USAGE(*UPDATE) KEYED; 5
```

固定形式のファイル記入項目に対応する自由形式でのコーディング

表 *84.* 固定形式のファイル記入項目に対応する自由形式での指定方法

固定形式の記入項目	**自由形式での指定方法**
ファイル・タイプ	USAGE キーワード
ファイルの指定	ファイルの指定に関連する自由形式構文はありません。 自由形式ファイルは、全手順または出力です。
ファイルの終わり	自由形式ではサポートされていません
ファイルの追加	USAGE(*OUTPUT)
順序	自由形式ではサポートされていません

表 *84.* 固定形式のファイル記入項目に対応する自由形式での指定方法 *(続き)*	
固定形式の記入項目	**自由形式での指定方法**
レコード長	装置キーワード DISK、PRINTER、SEQ、SPECIAL、WORKSTN のパラメーター
限界内処理	自由形式ではサポートされていません
キー・フィールドの長さ	KEYED キーワードの長さパラメーター
レコード・アドレス・タイプ A および K	KEYED キーワード
A および K 以外のレコード・アドレス・タイプ	自由形式ではサポートされていません
ファイル編成	**I** KEYED キーワード **T** 自由形式ではサポートされていません
装置	装置キーワード DISK、PRINTER、SEQ、SPECIAL、WORKSTN

装置タイプ・キーワード

- 369 ページの『DISK{(*EXT | レコード長)}』
- 386 ページの『SEQ{(*EXT | レコード長)}』
- 383 ページの『PRINTER{(*EXT | レコード長)}』
- 387 ページの『SPECIAL{(*EXT | レコード長)}』
- 390 ページの『WORKSTN{(*EXT | レコード長)}』

自由形式のファイル使用法の定義

表 *85.* 固定形式の「タイプ」、「指定」、および「追加」記入項目で使用されるキーワードに対応する自由形式での *USAGE* キーワード					
固定形式の記入項目			**説明**	**対応する USAGE キーワード**	**自由形式構文に関する注**
ファイル・タイプ	**ファイルの指定**	**ファイルの追加**			
I	F		入力全手順	USAGE(*INPUT)	USAGE(*INPUT) は、DISK、SEQ、SPECIAL の場合のデフォルトです
I	F	A	ファイル追加を伴う入力全手順	USAGE(*INPUT : *OUTPUT)	

表 *85.* 固定形式の「タイプ」、「指定」、および「追加」記入項目で使用されるキーワードに対応する自由形式での *USAGE* キーワード *(*続き*)*

固定形式の記入項目			説明	対応する USAGE キーワード	自由形式構文に関する注
ファイル・タイプ	ファイルの指定	ファイルの追加			
U	F		更新全手順	USAGE(*UPDATE : *DELETE)	固定形式では、更新は、ファイルが削除にも使用可能であることを 暗黙に示します。 自由形式では、ファイルが削除に使用可能であることを示すには、*DELETE を明示的に 指定する必要があります。 ファイルが削除に使用可能であることを望まない場合は *DELETE を省略します。
U	F	A	ファイル追加を伴う更新全手順	USAGE(*UPDATE : *DELETE : *OUTPUT)	
O			出力	USAGE(*OUTPUT)	USAGE(*OUTPUT) は、PRINTER の場合のデフォルトです
O		A	ファイル追加を伴う出力	自由形式ではサポートされていません	
C	F		結合全手順	USAGE(*INPUT : *OUTPUT)	USAGE(*INPUT : *OUTPUT) は、WORKSTN の場合のデフォルトです
I または U	P		入力または更新 1 次ファイル	自由形式ではサポートされていません	
I または U	S		入力または更新 2 次ファイル	自由形式ではサポートされていません	
I または C	T		入力または結合テーブル・ファイル	自由形式ではサポートされていません	

従来型のファイル記述仕様書ステートメント

ファイル仕様書の一般的なレイアウトは次のとおりです。

- 6 桁目にファイル仕様書コード (F) が入れられます。
- 仕様書の注記以外の部分は 7 から 80 桁目です。

  - 固定形式の記入項目は 7 から 42 桁目です。 LIKEFILE キーワードで定義されているファイルの場合、17 から 43 桁目の記入項目はブランクにする必要があります。 これらの固定形式の記入項目の値は、LIKEFILE キーワードによって指定される親ファイルから取得されます。
  - キーワードの記入項目は 44 から 80 桁目です。
- 仕様書の注記部分は 81 から 100 桁目です。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+...
9 ...+... 10
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++++++Comments++++++++++
++
```

図 *118.* ファイル仕様書のレイアウト

ファイル記述キーワードの継続記入行

自由形式構文	自由形式ステートメントで使用可能な桁については、307 ページの『自由形式ステートメント』を参照してください。

キーワードに追加のスペースが必要な場合には、次のようにキーワード・フィー ルドを後続の行に継続させることができます。

- 継続記入行の 6 桁目には F が入っていなければなりません。
- 継続記入行の 7 から 43 桁目はブランクでなければなりません。
- 指定は 44 桁目以降から継続されます。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+...
9 ...+... 10
F.....................................Keywords++++++++++++++++++++++++++++++Comments++++++++++
++
```

図 *119.* ファイル記述キーワードの継続記入行のレイアウト

6 桁目 (仕様書コード)

自由形式構文	DCL-F 命令コード

ファイル仕様書にはこの桁に F を入れなければなりません。

7 から 16 桁目 (ファイル名)

自由形式構文	DCL-F 命令コードに続けて名前が指定されます。350 ページの『自由形式のファイル定義ステートメント』を参照

記入
説明

有効なファイル名
プログラムまたはプロシージャーで使用されるすべてのファイルには 固有の名前が付いている必要があります。 ファイル名は 1 から 10 文字の長さにすることができ、7 桁目から始めなけ ればなりません。

コンパイル時および実行時のファイル名の使用方法について詳しくは、174 ページの『ファイル名に関する規則』を参照してください。

プログラム記述ファイル

プログラム記述ファイルの場合、ファイル定義に指定された ファイル名は以下の場所にも記入される必要があります。

- ファイルがグローバル 1 次ファイル、2 次ファイル、または全手順ファイルの場合は、入力仕様
- ファイルが出力ファイル、更新ファイル、または入出力共用ファイルである か、あるいはファイルに対してファイルの追加が指定された場合には、出力仕様または出力演算命令行
- ファイルがテーブルまたは配列ファイルの場合には定義仕様書
- 指定した命令コードにファイル名が必要な場合には演算仕様書

外部記述ファイル

外部記述ファイルで EXTDESC キーワード が指定されていない場合、ファイル定義に指定されたファイル名 が、ファイルのレコード記述を見つけるために使用される名前となります。 外部記述ファイルには次の規則が適用されます。

- 外部記述ファイルの入力および出力仕様はオプションです。 それらは、制御フィールドまたはレコード識別標識のような RPG IV 機能を検索される外部記述に追加している場合にのみ必要です。
- 外部記述が検索される場合には、レコード定義を入力、出力、または演算仕 様書上のそのレコード様式名によって参照することができます。 キーワード QUALIFIED または LIKEFILE によってファイルが修飾されている場合、修飾レコード様式は、ファイルとレコード様式の両方によって参照されます (例: MYFILE.MYFMT)。
- レコード様式名は固有の記号名でなければなりません。 キーワード QUALIFIED または LIKEFILE によってファイルが修飾されている場合は、レコード様式の名前は、ファイルの他の様式に対して固有でなければなりません。 ファイルが修飾されていない場合、レコード様式の名前は、モジュール内の他の名前に対して固有でなければなりません。
- RPG IV は、同じ名前の 2 つのレコード様式を持つ外部記述論理ファイルをサポートしていません。 しかし、このようなファイルであっても、それがプログラム記述ファイルであれ ばアクセスすることができます。

17 桁目 (ファイル・タイプ)

自由形式構文	USAGE キーワード

LIKEFILE キーワードが指定されている場合、この項目はブランクでなければなりません。 親ファイルのファイル・タイプが使用されます。

記入
説明

I
入力ファイル

O
出力ファイル

U
更新ファイル

C
入出力共用 (入出力) ファイル

入力ファイル

入力ファイルとは、プログラムが読み取る情報が入っているファイルのことです。 入力ファイルにはデータ・レコード、配列、またはテーブルを含め、また、レコ ード・アドレス・ファイルとすることもできます。

出力ファイル

出力ファイルとは、情報が書き出されるファイルのことです。

更新ファイル

更新ファイルとは、そのレコードを読み取って更新することができる入力ファイ ルのことです。 更新によって、ファイルに含まれているレコードの 1 つまたは複数のフィールド のデータが変更され、そのレコードが読み取られた同じファイルに書き戻されます。 レコードを削除する場合には、ファイルを更新ファイルとして指定しなければな りません。

入出力共用ファイル

入出力共用ファイルは、入力ファイルでもあり出力ファイルでもあります。 入出 力共用ファイルが処理された場合に、出力レコードには、出力レコードのフィー ルドによって表されるデータだけが含まれます。 この点が、出力レコードのフ ィールドによって変更された入力レコードがその出力レコードに含まれる更新フ ァイルと異なります。

入出力共用ファイルは、SPECIAL または ワークステーション・ファイルに有効です。 入出力共用ファイルは、18 桁目に T (配列またはテーブル置き換えファイル) が入 っている場合には、DISK ファイルまたは SEQ ファイルにも有効です。

18 桁目 (ファイルの指定)

自由形式構文	不要です。 すべての自由形式ファイルは、全手順ファイルまたは 出力ファイルです。

LIKEFILE キーワードが指定されている場合、この項目はブランクでなければなりません。 親ファイルのファイル指定が使用されます。

記入
説明

ブランク
出力ファイル

P
1 次ファイル

S
2 次ファイル

R
レコード・アドレス・ファイル

T
配列またはテーブル・ファイル

F
全手順ファイル

キーワード MAIN または NOMAIN を制御仕様書で指定した場合には、P、S、または R を指定することはできません。

1 次ファイル

複数のファイルをサイクル処理方式で処理する場合には、1 つのファイルを 1 次ファイルとして指定しておかなければなりません。 複数ファイル処理では、1 次ファイルの処理が優先されます。 1 つのプログラムにつき 1 つの 1 次ファイルだけを使用することがで きます。

2 次ファイル

RPG サイクルによって複数のファイルを処理する場合には、追加のファイルは 2 次ファイルとして指定されます。 2 次ファイルは入力可能 (入力、更新、または入出力共用ファイル・ タイプ) でなければなりませ

ん。 2 次ファイルの処理は、ファイル仕様書に 指定された順序および複数ファイル論理の規則によって決まります。

レコード・アドレス・ファイル *(RAF)*

レコード・アドレス・ファイルは、順次編成のファイルで、別のファイルからレ コードを選択するために使用されます。 1 つのプログラムでレコード・アドレス ・ファイルとして指定できるファイルは 1 つだけです。 このファイルはファイル仕様書 に記述され、入力仕様には記述されません。 レコード・アドレス・ファイルはプログラム記述ファイルでなければなりません が、このレコード・アドレス・ファイルを使用して、プログラム記述ファイルま たは外部記述ファイルを処理することができます。

レコード・アドレス・ファイルによって処理されるファイルは 1 次、 2 次、または全手順ファイルでなければならず、 レコード・アドレス・ファイルのファイル仕様書で RAFDATA キーワードに対する パラメーターとしても指定しなければなりません。

レコード・アドレス・ファイルを装置 SPECIAL に指定することはできません。

UCS-2 フィールドは、レコード・アドレス・ファイルの レコード・アドレス・タイプとしては使用できません。

相対レコード番号を含むレコード・アドレス・ファイルの場合には、 35 桁目に T および 22 桁目に F も指定しなければなりません。

配列またはテーブル・ファイル

18 桁目の T によって指定した配列およびテーブル・ファイルは、プログラムの初期化時にロードされます。 配列またはテーブル・ファイルは、入力または入出力共用ファイルとすることが できます。 配列またはテーブル出力ファイルの場合には、この記入項目はブランクのままに しておいてください。 配列およびテーブル入力ファイルの装置として SPECIAL を指定することはできま せん。 外部記述ファイルを配列またはテーブル・ファイルとして指定することはできま せん。

18 桁目に T を指定した場合には、DISK または SEQ ファイルに入出力共用 (17 桁目に C) のファイル・タイプを指定することができます。 入出力共用のファイル・タイプによって、配列またはテーブル・ファイルをファ イルから読み取り、同じファイル (配列またはテーブル置き換えファイル) に書き 出すことができます。 17 桁目の C に加えて、定義仕様書の TOFILE キーワードに対するパラメーターとし て 7 から 16 桁目にファイル名も指定する必要があります。

全手順ファイル

全手順ファイルは、入力は演算命令によって制御されるので、RPG サイクルでは 処理されません。 入力機能を実行するためには、CHAIN または READ のようなファイル命令コードが使用されます。

19 桁目 (ファイルの終わり)

自由形式構文	(自由形式ファイル定義にはサポートされません)

記入
説明

E
このファイルのすべてのレコードは、プログラムの終了前に処理されていな ければなりません。 レコード・アドレス・ファイルによって処理されるファイルには、この記入項目 は有効ではありません。

このオプションを使用するすべてのファイルからのすべてのレコードは、プログ ラムを終了するために RPG サイクルによって LR 標識がオンに設定される前に 処理されていなければなりません。

ブランク
すべてのファイルについて 19 桁目がブランクである場合には、プログラム の終わり (LR) になる前に、すべてのファイルからのすべてのレコードが処理され ていなければなりません。 すべてのファイルについて 19 桁目がブランクではない場合には、このファイル のすべてのレコードは、複数ファイル処理でプログラムの終わりになる前に処 理されていても、また、処理されていなくてもかまいません。

19 桁目を使用して、ファイルからのすべてのレコードが処理される前に、プログ ラムを終了できるかどうかを指示してください。 19 桁目の E は、1 次ファイル、2 次ファイル、またはレコ ード・アドレス・ファイルとして指定した入力ファイル、更新ファイル、または 入出力共用ファイルにのみ適用されます。

すべての 1 次および 2 次ファイルからのレコードを処理しな ければならない場合には、19 桁目がすべてのファイルについてブランクであるか 、あるいは E でなければなりません。 複数の入力ファイルの場合には、 19 桁目に E が指定されたすべての入力ファイルが処理された時に プログラムの終わり (LR) 条件が起こります。 すべてのファイルについて 19 桁目がブランクである場合に は、すべての入力ファイルが処理された時にプログラムの終わり条件が起こります。

突き合わせフィールドを 2 つ以上のファイルに指定して、1 つまたは複数のファイル について 19 桁目に E を指定した場合には、LR 標識は次のことが行われた後でオ ンに設定されます。

- 19 桁目に E の指定がある最後のファイルでファイルの終わり条件が発生した場合。
- 1 次ファイルから処理された最後のレコードと一致する 他のファイルのレコードすべてが、プログラムで処理された場合
- 突き合わせフィールドが一致しない次のレコードになるまで、突き合わせフィールド がないファイルのレコードがプログラムで処理された後。

突き合わせフィールドの指定を含むファイルがないかまたは 1 つしかない場合には、19 桁目に E を指定したすべてのファイルでファイルの終わりが起こった後では、他 のファイルのレコードは処理されません。

20 桁目 (ファイルの追加)

自由形式構文	USAGE キーワード に *OUTPUT を指定します

20 桁目は、入力ファイルまたは更新ファイルにレコードを追加するかどうかを指 示します。 出力ファイルの場合には、この記入項目は無視されます。 LIKEFILE キーワードが指定されている場合、この項目はブランクでなければなりません。

記入
説明

ブランク
入力ファイルまたは更新ファイル (自由形式ファイル定義 の USAGE キーワード、 または固定形式ファイル定義の 17 桁目を参照してください) にレコードを追加することはできません。

A
ファイルの出力レコード仕様の 18 から 20 桁目に "ADD" が含まれている時、 あるいは演算仕様書で WRITE 命令コードを使用した時に、入力ファイルまたは更新 ファイルにレコードが追加されます。

ファイル仕様書の 17 桁目と 20 桁目および出力仕様の 18 から 20 桁目の関係については、520 ページの『18 から 20 桁目 (レコードの追加/削除)』 を参照してください。

21 桁目 (順序)

自由形式構文	(自由形式ファイル定義にはサポートされません)

記入
説明

A またはブランク
突き合わせフィールドは昇順です。

D
突き合わせフィールドは降順です。

21 桁目は、突き合わせフィールドの指定 (入力仕様の 65 から 66 桁目) に使用される 入力フィールドの順序を指定します。 21 桁目は、1 次または 2 次ファイルとして使用される入力フ ァイル、更新ファイル、または入出力共用ファイルにのみ適用されます。 順序情報が含まれているフィールドを識別するためには、入力仕様の 65 から 66 桁目を使用してください。

突き合わせフィールドがある複数の入力ファイルをプログラムで指定した場合 には、21 桁目の順序の指定を使用して、突き合わせフィールドの順序を検査し、レコードの 突き合わせ手法を使用してファイルを処理することができます。 順序の指定が必要な のは、突き合わせフィールドを指定した最初のファイルについてだけです。 他のファ イルについて順序を指定する場合には、指定された順序が同じでなければなりません。そうでない場合には、最初のファイルに指定された順序と見なされます。

突き合わせフィールドがある 1 つの入力ファイルだけをプログラムで指定した場合に は、21 桁目の順序の指定を使用して、そのファイルのフィールドを検査し、ファ イルの順序が正しいことを確認することができます。 これらのフィールドの順序検査は、入力仕様の 65 から 66 桁目に コード M1 から M9 の 1 つを入れ、21 桁目に A、ブランク、または D を記入することによって指定 します。

順序検査は、ファイルからのレコードの中で突き合わせフィールドを使用する場合に 必要です。 突き合わせ入力ファイルから順序が違っているレコードが見つかった場合は、RPG IV 例外/エラー処理ルーチンに制御が与えられます。

22 桁目 (ファイル形式)

自由形式構文	装置タイプ・キーワードのパラメーター

LIKEFILE キーワードが指定されている場合、この項目はブランクでなければなりません。 親ファイルのファイル形式が使用されます。

記入
説明

F
プログラム記述ファイル

E
外部記述ファイル

22 桁目の F は、ファイルのレコードがプログラム内で入力/出力仕様に記述さ れる (配列/テーブル・ファイルおよびレコード・アドレス・ファイルは除く) ことを指示します。

22 桁目の E は、ファイルのレコード記述が RPG IV ソース・プログラムの外部にあることを示します。 これらの記述はコンパイル時にコンパイラーによって入手され、ソース・プログラム に組み込まれます。

23 から 27 桁目 (レコード長)

自由形式構文	装置タイプ・キーワードのレコード長パラメーター

LIKEFILE キーワードが指定されている場合、この項目はブランクでなければなりません。 親ファイルのレコード長が使用されます。

23 から 27 桁目は、プログラム記述ファイルに含まれる論理レコードの長さを指示す るために使用してください。 指定することができる最大レコード・サイズは 32766 ですが、この値は、装置の レコード・サイズに制約があった場合には一時変更されることがあります。 外部記述ファイルの場合には、この記入項目はブランクでなければなりません。

定義されているファイルがレコード・アドレス・ファイルであり、指定されたレコード長が 3 であった場合には、ファイル中の各レコードはオフセット 0 で始まる相対レコード番号に対して 3 バイトの 2 進フィールドから構成されると見なされます。 レコード長が 4 以上であった場合には、レコード・アドレス・ファイル中の各相対レコード番号はオフセット 1 で始まる 4 バイトのフィールドであると見なされます。 レコード長がブランクのままであった場合には、レコード・アドレス・フ ァイルの処理方法を判別するために、実行時に実際のレコード長が検索されます。

実行時にオープンされるファイルの 1 次レコードの長さが 3 である場 合には、3 バイトの相対レコード番号 (1 レコードに 1 つ) と見なされ、そうで ない場合には、4 バイトの相対レコード番号と見なされます。 このサポートを使用すれば、ILE RPG プログラムでシステム/36 環境の SORT ファイルをレコード・アドレス・ファイルとして使用することができます。

表 86. 相対レコード番号が入っているレコード・アドレス・ファイル (RAFRRN) に有効な組み合わせ		
レコード長 23 から 27 桁目	**RAF の長さ 29 から 33 桁目**	**サポートのタイプ**
ブランク	ブランク	実行時に決まるサポート
3	3	システム/36 サポート
> = 4	4	固有のサポート

28 桁目 (限界内処理)

自由形式構文	(自由形式ファイル定義にはサポートされません)

記入
説明

L
レコード・アドレス・ファイルによる限界内順次処理

ブランク
順次処理またはランダム処理

28 桁目を使用して、限界値レコードを含むレコード・アドレス・ファイルによっ てファイルが処理されるかどうかを指示してください。

限界内処理に使用されるレコード・アドレス・ファイルには、上限および 下限から構成されたレコードが入っています。 各レコードには、処理されるファイルのセグメントからの最低のレコード・キー と最高のレコード・キーから構成された限界のセットが入っています。 限界内処理は、1 次、2 次、または全手順ファイルとし て指定されたキー付きファイルに使用することができます。

28 桁目の L の指定は、限界レコードが入っているレコード・アドレス・ファイル によってファイルが処理される場合にのみ有効です。 ファイルのランダムおよび 順次処理は、ファイル仕様書の 18 桁目および 34 桁目の組み合わせおよび指定された演算 命令によって暗黙のうちに指定されます。

命令コード 876 ページの『SETLL (下限の設定)』 および 873 ページの『SETGT (より大きい設定)』 を使用して、ファイルを位置決めすることができます。しかし、これらの命令コードの使用にこの桁の L が必要なわけではありません。

限界内処理について詳しくは、「*Rational Development Studio for i: ILE RPG プログラマーの手引き*」を参照してください。

29 から 33 桁目 (キーまたはレコード・アドレスの長さ)

自由形式構文	KEYED キーワードの長さパラメーター

LIKEFILE キーワードが指定されている場合、この項目はブランクでなければなりません。 親ファイルのキーの長さが使用されます。

記入
説明

1 から 2000
プログラム記述ファイルのキー・フィールドに必要な桁数、またはレコード ・アドレス・ファイル (プログラム記述ファイルでなければなりません) の項目 の長さ。

定義中のプログラム記述ファイルでレコードの識別用にキーを使用する場合には 、各レコード・キーが占める桁数を記入してください。 索引付きファイルにはこの指定が必要です。

キーがパックされている場合には、キー・フィールド長はパックされた長さでな ければなりません。これは、DDS の桁数を 2 で除算し、小数部は無視して 1 を加 算した値になります。

定義中のファイルがレコード・アドレス・ファイルの場合には、レコード・アド レス・ファイル内の各項目が占める桁数を記入してください。

キーが図形の場合には、キー・フィールド長はバイト数で指定する必要があります (たとえば、3 つの図形文字は 6 バイト必要とします)。

ブランク
外部記述ファイルの場合には、これらの桁はブランクでなければ なりません。 (キーの長さは外部記述で指定されます。) プログラム記述ファイルの 場合には、ブランクの指定はキーが使用されないこと を指示します。 23 から 27 桁目 (レコード長) にブランクを指定したレコード・アドレス・ファイル の場合には、29 から 33 桁目もブランクにすることができます。

34 桁目 (レコード・アドレス・タイプ)

自由形式構文	KEYED キーワード

LIKEFILE キーワードが指定されている場合、この項目はブランクでなければなりません。 親ファイルのレコード・アドレス・タイプが使用されます。

記入
説明

ブランク
相対レコード番号を 使用してファイルを処理します。

レコードは連続して読み取られます。

レコード・アドレス・ファイルには相対レコード番号が入っています。

限界値範囲内処理の場合には、レコード・アドレス・タイプ (34 桁目) は処理し ているファイルのタイプと同じになります。

A
文字キー (索引付きファイルまたはレコード・アドレス 限界ファイルとして指定されたプログラム記述ファイルについてのみ有効)。

P
パック 10 進数キー (索引付きファイルまたはレコード・ アドレス限界ファイルとして指定されたプログラム記述ファイルについてのみ有効)。

G
図形キー (索引付きファイルまたはレコード・アドレス 限界ファイルとして指定されたプログラム記述ファイルについてのみ有効)。

K
キー値を使用してファイルを処理します。 この指定が有効なのは、外部記述ファイルに対してだけです。

D
日付キーを使用してファイルを処理します。 この指定が有効なのは、索引付きファイルまたはレコード・アドレス限界ファイ ルとして指定されたプログラム記述ファイルに対してだけです。

T
時刻キーを使用してファイルを処理します。 この指定が有効なのは、索引付きファイルまたはレコード・アドレス限界ファイ ルとして指定されたプログラム記述ファイルに対してだけです。

Z
タイム・スタンプ・キーを使用してファイルを処理します。 この指定が有効なのは、索引付きファイルまたはレコード・アドレス限界ファイ ルとして指定されたプログラム記述ファイルに対してだけです。

F
浮動キー (索引付きファイルまたはレコード・アドレス 限界ファイルとして指定されたプログラム記述ファイルに対しのみ有効です)。

UCS-2 フィールドは、プログラム記述索引付きファイルまたは レコード・アドレス・ファイルのレコード・アドレス・タイプ としては使用できません。

ブランク = キーを使用しない処理

ブランクは、ファイルがキーを使用しないで処理されること、レコード・アドレ ス・ファイルに相対レコード番号が入っていること (35 桁目に T)、またはレコ ード・アドレス限界ファイル内のキーの形式が処理中のファイルのキーと同じで あることを指示します。

キーを使用しないで処理されるファイルは、連続して処理されるか、あるいは相 対レコード番号によってランダムに処理されます。

相対レコード番号による入力処理は、34 桁目のブランクと、CHAIN、SETLL、また は SETGT 命令コードの使用から決定されます。 相対レコード番号による出力処理 は、34 桁目のブランクと、ファイル仕様書での RECNO キーワードの使用から決定されます。

A = 文字キー

この行で定義された索引付きファイル (35 桁目に I) は、文字レコード・キーに よって処理されます。 (検索引数として使用される数値フィールドは、連鎖の 前にゾーン 10 進数に変換されます。) A の指定は、キー・フィールドとして識別されるフィールドのデータ形式と一致 していなければなりません (29 から 33 桁目の長さおよび KEYLOC キーワードに対す るパラメーターとして指定された開始位置)。

この行で定義されたレコード・アドレス限界ファイル (18 桁目に R) には文字キ ーが入っています。 このレコード・アドレス・ファイルによって処理されているファイルは、34 桁目 を A、P、または K にすることができます。

P = パック *10* 進キー

この行で定義された索引付きファイル (35 桁目に I) は、パック 10 進数の数値 キーによって処理されます。 P の指定は、キー・フィールドとして識別されるフィールドのデータ形式と一致 していなければなりません (29 から 33 桁目の長さおよび KEYLOC キーワードに対す るパラメーターとして指定された開始位置)。

この行で定義されたレコード・アドレス限界ファイルには、パック 10 進数形式 のレコード・キーが入っています。 このレコード・アドレス・ファイルによって処理されているファイルは、34 桁目 を A、P、または K にすることができます。

G = 図形キー

この行で定義された索引付きファイル (35 桁目に I) は、図形キーによって処理 されます。 各図形文字には 2 バイトが必要なので、キーの長さは偶数でなければなりません。 この索引付きファイルを処理するために使用されるレコード・アドレス・ファイ ルは、そのファイル仕様書の 34 桁目に 'G' も指定されていなければならず、そのキーの長さも索引付きファイルのキーの長さ (29 から 33 桁目) と同じでなけれ ばなりません。

K = キー

K の指定は、アクセス・パスがキーの値によって構築されるという前提で外部記 述ファイルが処理されることを指示します。 処理がランダムであった場合には、 キーの値がレコードの識別に使用されます。

キー付きファイルでこの桁がブランクであった場合には、レコードは到着順に検 索されます。

D = 日付キー

この行で定義された索引付きファイル (35 桁目に I) は、日付キーによって処理 されます。 D の指定は、キー・フィールドとして識別されるフィールドのデータ形式と一致 していなければなりません (29 から 33 桁目の長さおよび KEYLOC キーワードに対す るパラメーターとして指定された開始位置)。

日付キーの形式および区切り記号を判別する時に使用される順位は次のとおりです。

1. ファイル仕様書に指定された DATFMT キーワードから
2. 制御仕様書に指定された DATFMT キーワードから
3. *ISO

T = 時刻キー

この行で定義された索引付きファイル (35 桁目に I) は、時刻キーによって 処理されます。 T の指定は、キー・フィールドとして識別されるフィールドのデータ形式と一致 していなければなりません (29 から 33 桁目の長さおよび KEYLOC キーワードに対す るパラメーターとして指定された開始位置)。

時刻キーの形式および区切り記号を判別する時に使用される順位は次のとおりです。

1. ファイル仕様書に指定された TIMFMT キーワードから
2. 制御仕様書に指定された TIMFMT キーワードから
3. *ISO

Z = タイム・スタンプ・キー

この行で定義された索引付きファイル (35 桁目に I) は、タイム・スタンプ・キーに よって処理されます。 Z の指定は、キー・フィールドとして識別されるフィールドのデータ形式と一致 していなければなりません (29 から 33 桁目の長さおよび KEYLOC キーワードに対す るパラメーターとして指定された開始位置)。

F = 浮動キー

この行で定義された索引付きファイル (35 桁目の I) は浮動キーによって 処理されます。 キーの長さの指定 (29 から 33 桁目) には、浮動キーの 4 または 8 のいずれかの 値が含まれていなければなりません。 ファイルに浮動キーが含まれている場合、キー順入出力命令のキーとして、 どのタイプの数値変数またはリテラルでも指定することができます。 非浮動レコード・アドレス・タイプの場合、浮動検索引数を持つことはできません。

レコード・アドレス・タイプについて詳しくは、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」を参照してください。

35 桁目 (ファイル編成)

自由形式構文	KEYED キーワード

LIKEFILE キーワードが指定されている場合、この項目はブランクでなければなりません。 親ファイルのファイル編成が使用されます。

記入
説明

ブランク
プログラム記述ファイルはキーを使用しないで処理されるか、 あるいはファイルが外部記述です。

I
索引付きファイル (プログラム記述ファイル でのみ有効)。

T
相対レコード番号が入っているレコード・アドレス・ ファイル (プログラム記述ファイルでのみ有効)。

注 : レコード・アドレス・ファイルは、自由形式ファイル定義ではサポートされません。

35 桁目を使用してプログラム記述ファイルの編成を識別します。

ブランク = キー付きでないプログラム記述ファイル

キーを使用せずに処理されるプログラム記述ファイルは、以下のように処理する ことができます。

- 相対レコード番号によるランダム処理。28 桁目および 34 桁目はブランクで なければなりません。
- 入力順の処理。28 桁目および 34 桁目はブランクでなければなりません。
- レコード・アドレス・ファイルとしての処理。28 桁目はブランクでなければ なりません。

I = 索引付きファイル

索引付きファイルは、以下のように処理することができます。

- キーによるランダム処理または順次処理
- レコード・アドレス・ファイルによる (限界値範囲内順次) 処理。 28 桁目には L が入っていなければなりません。

T = レコード・アドレス・ファイル

相対レコード番号が含まれるレコード・アドレス・ファイル (18 桁目の R によっ て指示される) は、35 桁目の T によって識別することが 必要です。 (レコード・アドレス・ファイルはプログラム記述でなければなりません。) 処理中のファイ ルからの各レコードは、レコード・アドレス・ファイル内の相対レコード番号に 基づいて検索されます。 (相対レコード番号をレコード・アドレス限界ファイル に使用することはできません。)

レコード・アドレス・ファイルの各相対レコード番号は 4 バイトの 2 進フィールドです。したがって、レコード・アドレス・ファイルの各 4 バイト単位に相 対レコード番号が入れられます。 マイナス 1 (-1 または 16 進 FFFFFFFF) の相対レコード番号値によっ て、レコードはスキップされます。 レコード・アドレス・ファイルのすべてのレコードが処理されると、ファイルの 終わりが起こります。

レコード・アドレス・ファイルの処理方法について詳しくは、「*Rational Development Studio for i: ILE RPG プログラマーの手引き*」を参照してください。

36 から 42 桁目 (装置)

自由形式構文	装置タイプ・キーワード

LIKEFILE キーワードが指定されている場合、この項目はブランクでなければなりません。 親ファイルの装置項目が使用されます。

ファイルと関連付けられる RPG IV 装置名を指定するには、36 から 42 桁目を使用してください。 7 から 16 桁目に指定されたファイル名は、プログラム中で使用される入出力装置を 変更できるように、実行時に一時変更することができます。

RPG IV 装置名はシステム装置名と同じではないことに注意してください。

ファイルの装置タイプ

PRINTER
このファイルは印刷装置ファイル、すなわち、印刷装置へ送ることのできる 制御文字の入ったファイルです。

ディスク
このファイルはディスク・ファイルです。 この装置は、順次およびランダム 処理の読み取り/書き込み機能をサポートします。 これらのファイルは、分散データ管理機能 (DDM) によってリモート・システムでアクセ スすることができます。

WORKSTN
このファイルはワークステーション・ファイルです。 入出力はディスプレイまたは ICF ファイルを通じて処理されます。

SPECIAL
このファイルは特殊ファイルです。 入力または出力は、ユーザー提供のプログラムによってアクセスされる装置上で 行われます。 プログラムの名前は、PGMNAME キーワードに対するパラメーターとして指定しな ければなりません。 このプログラムによって使用される、オプション・コード・パラメーターおよび 状況コード・パラメーターを含めたパラメーター・リストが作成されます。 このファイルは固定長の 非ブロック化形式でなければなりません。 詳しくは、381 ページの『PLIST(PLIST 名)』 および 381 ページの『PGMNAME(プログラム名)』 を参照してください。

SEQ
このファイルは順次編成ファイルです。 実際の装置は CL コマンドまたはファイル記述に指定され、ファイル名によって アクセスされます。

43 桁目 (予約済み)

43 桁目はブランクでなければなりません。

44 から 80 桁目 (キーワード)

自由形式構文	rev="v7r3t">自由形式ステートメントで使用可能な桁については、307 ページの『自由形式ステートメント』を参照してください。

44 から 80 桁目は、ファイル仕様書のキーワードのために用意されています。 キーワードは、定義中のファイルに関する追加の情報を指定するために使用され ます。

ファイル記述のキーワード

ファイル記述のキーワードは、パラメーターを持っていなかったり、任意指 定パラメーターを持ったり、または必須パラメーターを持ったりします。 キーワードの構文は次のとおりです。

```
     Keyword(parameter1 : parameter2)
```

ここで、

- 1 つまたは複数のパラメーターは括弧 () で囲みます。

  注 : パラメーターがない場合、括弧 を指定してはなりません。
- コロン (:) を使用して複数のパラメーターを区切ります。

任意指定パラメーターと必須パラメーターを示すために、 以下の国別の規則を使用します。

- 中括弧 { } は任意指定パラメーターまたはパラメーターの任意指定要素を示します。
- 省略記号 (...) はパラメーターが反復可能であることを示します。
- コロン (:) はパラメーターを区切り、複数のパラメーターを指定できることを示します。 コロンで区切られたすべてのパラメーターは、中括弧で囲まれていない限り、必須パラメーターです。
- 縦線 (|) は、キーワードに 1 つのパラメーターしか指定でき ないことを示します。
- キーワード・パラメーターを区切るブランクは、1 つまたは複数のパラメーター を指定できることを示します。

注 : 中括弧、省略記号、および縦線は、キーワード構文の一部 ではないため、ソースに入れてはなりません。

ファイル記述キーワードに追加のスペースが必要な場合には、 キーワード・フィールドを後続の行に継続させることができます。 353 ページの『ファイル記述キーワードの継続記入行』および 315 ページの『ファイル仕様書のキーワード・フィールド』を参照してください。

ALIAS

外部記述ファイルに ALIAS キーワードが指定された場合、別名 (代替名) があれば、RPG コンパイラーはファイルに関連付けられたフィールドにその別名を使用します。LIKEREC キーワードで定義されたデータ構造のサブフィールド名を判別するためにも、その別名を使用します。 RPG ファイルに ALIAS キーワードが指定されない、または外部フィールドに別名が定義されていない場合、 RPG コンパイラーは標準の外部フィールド名を使用します。

注 : 特定の外部フィールドの代替名が引用符で囲まれている場合、標準の外部フィールド名がそのフィールドに対して使用されます。

ALIAS キーワードはすべての外部記述ファイルに対して使用できます。

ALIAS キーワードと一緒に PREFIX キーワードを指定した場合、置換対象文字数を示す、PREFIX の 2 番目のパラメーターは、別名には適用されません。 以下の説明では、ファイル MYFILE にフィールド XYCUSTNM と XYID_NUM があり、XYCUSTNM フィールドの別名が CUSTOMER_NAME であることを想定しています。

- キーワード PREFIX(NEW_) を指定した場合、2 番目のパラメーターがないため、 どの名前についても文字は置換されません。 フィールドおよび LIKEREC サブフィールドに使用される名前は、NEW_CUSTOMER_NAME および NEW_XYID_NUM になります。
- キーワード PREFIX(NEW_:2) を指定した場合、別名を持たないフィールドの名前の 2 文字が置換されます。 フィールドおよび LIKEREC サブフィールドに使用される名前は、NEW_CUSTOMER_NAME および NEW_ID_NUM になります。 XYID_NUM では先頭の 2 文字の "XY" が置換されますが、CUSTOMER_NAME では文字は置換されません。
- キーワード PREFIX(':2) を指定した場合、別名を持たないフィールドの名前から 2 文字が除去されます。 フィールドおよび LIKEREC サブフィールドに使用される名前は、CUSTOMER_NAME および ID_NUM になります。 XYID_NUM では先頭の 2 文字の "XY" が置換されますが、CUSTOMER_NAME から文字は削除されません。
- PREFIX の最初のパラメーターにデータ構造名が含まれる場合 (例えば、PREFIX('MYDS.') など)、LIKEREC キーワードで定義されたデータ構造ではドットの前の接頭部の部分が無視されます。

外部記述ファイルに対する ALIAS キーワードの使用

最初のフィールドに ALIAS キーワードを使用して別名 CUSTOMER_NAME を CUSTNM フィールドと関連付ける、ファイル MYFILE の DDS 仕様。

```
A          R CUSTREC
A            CUSTNM        25A         ALIAS(CUSTOMER_NAME)
A            ID_NUM        12P 0>
```

ALIAS キーワードで非修飾ファイルを定義する RPG プログラムのソース。 このファイルのフィールドは次のとおりです。

- CUSTOMER_NAME (ALIAS 名を使用)
- ID_NUM (標準名を使用)

```
Fmyfile    if   e             disk    ALIAS
 /free
     read myfile;
     if customer_name <> *blanks
     and id_num > 0;
       ...
```

ALIAS キーワードで修飾ファイルを定義し、かつ LIKEREC データ構造を定義する RPG プログラムのソース。 このデータ構造のサブフィールドは次のとおりです。

- CUSTOMER_NAME (ALIAS 名を使用)
- ID_NUM (標準名を使用)

```
Fmyfile    if   e             disk    ALIAS QUALIFIED
D myDs            ds                   LIKEREC(myfile.custRec)
 /free
     read myfile myDs;
     if myDs.customer_name <> *blanks
     and myDs.id_num > 0;
       ...
```

BLOCK(*YES |*NO)

BLOCK キーワードは、ファイルと関連したレコードのブロック化を制御します。 このキーワードが有効なのは、DISK または SEQ ファイルの場合だけです。

このキーワードを指定しない場合、以下の条件を満たしていれば、RPG コンパイラー は入力レコードを非ブロック化し、出力レコードをブロック化 して、SEQ または DISK ファイルの実行時パフォーマンスを改善します。

1. ファイルがプログラム記述であるか、あるいは外部記述の場合には、1 つだ けのレコード様式をもっている。
2. ファイル仕様書でキーワード RECNO が使用されていない。

   注 : RECNO を使う場合、ILE RPG コンパイラーはレコードのブロック化を許可しません。 ただし、ファイルが入力ファイルで RECNO が使われると、高速順次アクセスが設定される場合、データ管理は依然、レコードをブロック化することがあります。 この場合、更新された レコードは直ちには見られなくなります。
3. 次の 1 つが起こっています。
   a. ファイルが出力ファイルである。
   b. ファイルが入出力共用ファイルの場合には、配列または テーブル・ファイルである。
   c. ファイルが入力専用ファイルである場合には、それがレコード・アドレス・ ファイルでないかまたはレコード・アドレス・ファイルによって処理されるもの ではなく、ファイルで READE、READPE、SETGT、SETLL、および CHAIN 命令が使用されていない。 (READE 命令または READPE 命令が使用されている場合には、入力ファイルのレコードのブロック化は行われません。 SETGT 命令、SETLL 命令、または CHAIN 命令が使用されている場合、BLOCK(*YES) キーワードが指定されていなければ、入力ファイルのレコードのブロック化は行われません。)

BLOCK(*YES) が指定されている場合、レコードのブロック化は上記のように 行われます。ただし、入力ファイルで SETLL、SETGT、および CHAIN 命令を使用 することができ、なおかつ、ブロック化が依然として行われます (上記の 3c を参照)。 レコードのブロック化を防ぐために、BLOCK(*NO) を指定することができます。 そうすれば、レコードのブロック化はコンパイラーによって行われません。

COMMIT { (RPG 名) }

COMMIT キーワードによって、コミットメント制御のもとでファイルの処理が可能です。 任意指定パラメーターの RPG 名を指定することができます。 このパラメーターは、タイプ標識のフィールド (すなわち、長さが 1 の文字フィ ールド) として暗黙に定義され、RPG によって '0' に初期化されます。

任意指定パラメーターを指定することによって、コミットメント制御を使用可能 にするかどうかを実行時に制御することができます。 このパラメーターに '1' が入っていた場合には、ファイルは COMMIT 標識がオン でオープンされ、そうでない場合には、ファイルは COMMIT を使用せずにオープ ンされます。 パラメーターはファイルをオープンする前に設定されていなければなりません。 ファイルがプログラムの初期化時にオープンされる場合には、COMMIT パラメー ターを呼び出しパラメーターとして渡すか、あるいはそれを外部標識として定義す ることができます。 ファイルが演算仕様書の OPEN 命令を使用して明示的にオープンされる場合には、 パラメーターを OPEN 命令に先立って設定することができます。

COMMIT および ROLBK 命令コードを使用して、このファイルおよび現在コミット メント制御のもとにあるその他のファイルに対する変更をグループ化し、すべて の変更をまとめて行うか、あるいはまったく行わないようにします。

注 : ファイルが共用オープン・データ経路を使用してすでにオープンされている 場合には、コミットメント制御の値は、以前の OPEN 命令の値と一致してい なければなりません。

DATA(*CVT | *NOCVT)

DATA キーワードは、データベースがファイル操作中に英数字データおよびグラフィック・データに関してジョブ CCSID との間の CCSID 変換 を実行するようにファイルがオープンされるかどうかを制御します。

***CVT**
ジョブ CCSID が 65535 ではない場合、CCSID 変換が行われます。 ジョブ CCSID が 65535 である場合、CCSID 変換は行われません。

***NOCVT**
CCSID 変換は行われません。

DATA キーワードが指定されていない場合は、次のようになります。

- 制御ステートメントに OPENOPT(*NOCVTDATA) が指定されている場合、DATA(*NOCVT) が暗黙的に指定されます。
- そうでない場合は、制御ステートメントに OPENOPT(*CVTDATA) または CCSID(*EXACT) が指定されると、DATA(*CVT) が暗黙的に指定されます。
- 制御仕様書に CCSID(*EXACT) が指定されておらず、OPENOPT(*CVTDATA) と OPENOPT(*NOCVTDATA) のどちらも制御仕様書に指定されていない場合、DATA(*CVT) が指定されているかのようにファイルがオープンされます。 ただし、DATA キーワードは有効であるとは見なされません。 詳しくは、340 ページの『OPENOPT (* {NO }INZOFL * {NO }CVTDATA)』を参照してください。

DATA キーワードは、物理データとバッファーの間の CCSID 変換 をデータベースがどのように扱うのかに影響します。明示的または暗黙的に DATA キーワードを指定することが、データ構造またはその英数字サブフィールドを使用する入出力命令に与える影響について詳しくは、262 ページの『入出力命令中の CCSID 変換』を参照してください。

DATA キーワードの例

以下の例では、次のフィールドがファイル内にあると想定されています。

- CHAR37 は、EBCDIC CCSID 37 の英数字フィールドです。
- CHAR1208 は、UTF-8 CCSID 1208 の英数字フィールドです。
- GRAPH835 は、DBCS CCSID 835 のグラフィック・フィールドです。
- GRAPH834 は、DBCS CCSID 834 のグラフィック・フィールドです。

次のようなデータ構造を想定します。

- データ構造 *custExactIn* 内の各英数字サブフィールドの CCSID は、 ファイル内のフィールドと同じものになります。
- データ構造 *custNoexactIn* 内の各英数字サブフィールドの CCSID は CCSID(*JOBRUN) になります。
- 両方のデータ構造内の各グラフィック・サブフィールドの CCSID は、ファイル内のフィールドと同じものになります。

```
DCL-DS custExactIn LIKEREC(custRec:*INPUT) CCSID(*EXACT);
DCL-DS custNoexactIn LIKEREC(custRec:*INPUT) CCSID(*NOEXACT);
```

ファイルに対して DATA(*CVT) が指定され、ジョブ CCSID が 937 である

CCSID 937 に関連した DBCS CCSID は 835 です。

```
DCL-F custFile USAGE(*UPDATE) DATA(*CVT);

READ custFile custExactIn;
READ custFile custNoexactIn;
```

- 両方の READ 命令で、データベースはファイル内の英数字データ を入力バッファー内では CCSID 937 に変換し、ファイル内の グラフィック・データを入力バッファー内では CCSID 835 に変換します。

- 最初の READ 命令では、入力バッファー内のデータと *custExactIn* データ構造 内のサブフィールドとの間で、以下の変換が実行されます。
  - サブフィールド CHAR37 については、データは CCSID 937 から CCSID 37 に変換されます。
  - サブフィールド CHAR1208 については、データは CCSID 937 から CCSID 1208 に変換されます。
  - サブフィールド GRAPH835 については、バッファー内のデータと サブフィールド内のデータは両方とも CCSID 835 であるため、CCSID 変換は不要です。
  - サブフィールド GRAPH834 については、データは CCSID 835 から CCSID 834 に変換されます。
- 2 番目の READ 命令では、英数字サブフィールドに関して CCSID 変換は不要です。 これは、入力バッファー内の各フィールドのデータの CCSID は *custNoexactIn* データ構造内の 対応するサブフィールドのものと同じであるためです。 サブフィールド GRAPH834 については、データは CCSID 835 から CCSID 834 に変換されます。

ファイルに対して DATA(*CVT) が指定され、ジョブ CCSID が 65535 である

デフォルトのジョブ CCSID が 937 であると想定します。 これは、CCSID(*JOBRUN) の英数字プログラム・フィールドに想定される CCSID です。

```
DCL-F custFile USAGE(*UPDATE) DATA(*CVT);

READ custFile custExactIn;
READ custFile custNoexactIn;
```

- 両方の READ 命令で、ファイル内のデータと入力バッファーとの間で 変換は行われません。 入力バッファー内のフィールドの CCSID は、ファイル内のフィールドのものと同じです。
- 最初の READ 命令では、入力バッファー内の各フィールド のデータの CCSID と *custExactIn* データ構造内の対応するサブフィールドの CCSID は 同じであるため、CCSID 変換は不要です。
- 2 番目の READ 命令では、入力バッファー内の データと *custNoxactIn* データ構造内のサブフィールドとの 間で、以下の変換が実行されます。
  - サブフィールド CHAR37 については、データは CCSID 37 からジョブ CCSID 937 に変換されます。
  - サブフィールド CHAR1208 については、データは CCSID 1208 からジョブ CCSID 937 に変換されます。
  - サブフィールド GRAPH835 については、バッファー内のデータと サブフィールド内のデータは両方とも CCSID 835 であるため、CCSID 変換は不要です。
  - サブフィールド GRAPH834 については、バッファー内のデータと サブフィールド内のデータは両方とも CCSID 834 であるため、CCSID 変換は不要です。

ファイルに対して DATA(*NOCVT) が指定されている

ジョブ CCSID またはデフォルトのジョブ CCSID が 937 であると想定します。 これは、CCSID(*JOBRUN) の英数字プログラム・フィールドに想定される CCSID です。

READ 命令は、ジョブ CCSID が 65535 である上記のシナリオと同じです。

DATFMT(形式 {区切り記号 })

DATFMT キーワードによって、プログラム記述ファイル の すべての 日付フィールドについてデフォルトの外部日付形式および (オプションの) デフォルトの区切り記号 を指定することができます。 このキーワードを指定したファイルが索引付きで、キー・フィールドが 日付の場合には、これによってキー・フィールドのデフォルトの外部形式も 指定されます。

レコード・アドレス・ファイルの場合には、これによってレコード・ アドレス・ファイルから読み取られる日付限界キーの外部日付形式 が指定されます。

対応する入力仕様 (31 から 35 桁目) また出力仕様 (53 から 57 桁目) にフィールドの 日付の形式/区切り記号を指定することによって、ファイル中の個別の入力また は出力日付フィールドに異なる外部形式を指定することができます。

有効な形式および区切り記号については、274 ページの表 71 を参照してください。外部形式について詳しくは、247 ページの『内部形式および外部形式』 を参照してください。

DEVID(フィールド名)

DEVID キーワードは、ファイル中で処理されるレコードを提供するプログラム 装置の名前を指定します。このフィールドは、レコードがファイルから読み取られるたびに更新されます。 また、プログラム装置名をこのフィールドの中に移して、出力操作または装置に 固有の入力操作 (ファイル名別 READ または暗黙のサイクル読み取りは除く) を別 の装置に向けることができます。

フィールド名は、10 文字の英数字フィールドとして暗黙に定義されます。 このフィールドに指定する装置名は左寄せし、ブランクを埋め込まなければなり ません。 当初、このフィールドはブランクです。 ブランク・フィールドは要求元端末装置 を示します。 ファイル用に要求元端末装置を入手しない場合には、ブランク・フィールドを使 用してはなりません。

DEVID フィールドには、プログラムに対する呼び出しのたびに保守が行われます。 プログラム A の中からプログラム B を呼び出しても、プログラム A の DEVID フィ ールドは影響を受けません。 プログラム B には別個の DEVID フィールドが使用されます。 プログラム A に戻ったときには、その DEVID フィールドの値はプログラム B を呼び出す前の値と同じです。 プログラム B がプログラム A のために入手された装置を知っている必要がある場合には、プログラム A はプログラム B を呼び出すときにこの情報を (パラメーター・リストとして) 渡す必要があります。

DEVID キーワードが指定され、MAXDEV キーワードは指定されていない場合には、 プログラムは複数装置ファイル (MAXDEV が *FILE のパラメーターを持つ) と見なします。

要求元端末装置の名前を判別するには、ファイル情報データ構造 (146 ページの『ファイル情報データ構造』を参照) の該当する領域を参照することができます。 あるいは、フィールド名にブランクが含まれる入力または出力命令を処理するこ とができます。 操作の後に、このフィールド名には要求元端末装置の名前が入ります。

DISK{(*EXT | レコード長)}

DISK キーワードは、装置タイプ・キーワードの 1 つです。 これは自由形式ファイル定義で使用されて、ファイル装置が DISK で あることを示します。 これは最初のキーワードでなければなりません。

パラメーターは任意指定です。 デフォルトは *EXT です。

***EXT**
ファイルが外部記述であることを示すには、*EXT を指定します。 これはデフォルト値。

レコード長
ファイルがプログラム記述であることを示すには、レコード長 を 指定します。 レコード長は 1 から 32766 の間でなければなりません。 リテラルまたは 名前付き定数を指定できます。 名前付き定数である場合、ファイル定義ステートメントの前にその定数が定義されている必要があります。

EXTDESC(外部ファイル名)

EXTDESC キーワードを指定すると、ファイルの外部記述を取得するためにコンパイラーがコンパイル時に使用するファイルの指示を行えます。

EXTDESC キーワードで指定されたファイルは、コンパイル時にのみ使用されます。 実行時に、EXTDESC キーワードが指定されなかった場合に適用される規則と同じ規則を使用して、ファイルが検出されます。 EXTDESC キーワードが指定したファイルを実行時に使用したい場合には、追加キーワード EXTFILE(*EXTDESC) を使用できます。

EXTDESC キーワードを指定する位置は、パラメーターとしてレコード様式名を持つキーワード (IGNORE、INCLUDE、RENAME、SFILE など) の前、および実ファイルに妥当性を依存するキーワード (INDDS、SLN など) の前でなければなりません。

EXTDESC のパラメーターは、有効なファイル名を指定する名前付き定数またはリテラルで なければなりません。 名前付き定数である場合、ファイル定義の前にその定数が定義されている必要があります。 値は次のいずれの形式でも指定できます。

```
filename
libname/filename
*LIBL/filename
```

注 :

1. ライブラリー名として *CURLIB は指定できません。
2. ライブラリー名の指定なしにファイル名を指定した場合は、*LIBL が使用されます。
3. 名前の大文字小文字は正確でなければなりません。 例えば、EXTDESC('qtemp/myfile') を指定した場合、ファイルは検出されません。 代わりに、EXTDESC('QTEMP/MYFILE') を指定してください。
4. RPG が外部記述で使用するファイルに対して一時変更を指定した場合、その一時変更は有効になります。 EXTDESC('MYLIB/MYFILE') を指定した場合、RPG は外部記述にファイル MYLIB/MYFILE を使用します。 コマンド OVRDBF MYFILE OTHERLIB/OTHERFILE がコンパイル前に使用されると、実際に使用されるファイルは OTHERLIB/OTHERFILE になります。 7 から 15 桁目に指定されている名前は、RPG ソース・メンバーの内部でのみ使用されるため、その名前に対する一時変更はすべて無視されますので、注意してください。

```
 * At compile time, file MYLIB/MYFILE1 will be used to
 * get the definition for file "FILE1", as specified by
 * the EXTDESC keyword.
 * At runtime, file *LIBL/FILE1 will be opened.  Since
 * the EXTFILE keyword is not specified, the file name
 * defaults to the RPG name for the file.
Ffile1     if   e             disk
F                             extdesc('MYLIB/MYFILE1')
 * At compile time, file MYLIB/MYFILE2 will be used to
 * get the definition for file "FILE2", as specified by
 * the EXTDESC keyword.
 * At runtime, file MYLIB/MYFILE2 will be opened, as
 * specified by the EXTFILE(*EXTDESC) keyword.
Ffile2     if   e             disk
F                             extdesc('MYLIB/MYFILE2')
F                             extfile(*extdesc)
```

図 *120. EXTDESC* キーワードの例

EXTFILE(ファイル名 | *EXTDESC)

EXTFILE キーワードは、どのライブラリーのどのファイルがオープンされるかを指定します。

ファイル名 には、リテラルまたは変数を指定できます。 値は次のいずれの形式でも指定できます。

```
filename
libname/filename
*LIBL/filename
```

特殊値 *EXTDESC を使用すると、EXTDESC キーワードのパラメーターが EXTFILE キーワードに対しても使用されることを指定できます。

注 :

1. ライブラリー名として *CURLIB は指定できません。
2. ライブラリー名の指定なしにファイル名を指定した場合は、*LIBL が使用されます。
3. 名前の大文字小文字は正確でなければなりません。 たとえば、EXTFILE(ファイル名) を指定しその変数であるファイル名の値が 'qtemp/myfile' だったとすると、そのファイルは見付からないことになります。 ファイル名の値は 'QTEMP/MYFILE' でなければなりません。
4. このキーワードは、コンパイル時に外部記述ファイルを見付けるのには使用されることはありません。コンパイル時には、EXTDESC キーワードを使用してファイルを見つけます。

5. EXTFILE(*EXTDESC) を指定した場合、ファイルに対しても、 あるいはファイルが LIKEFILE キーワードを指定して定義されている場合には親ファイルに対しても、EXTDESC キーワードを指定する必要があります。
6. 変数名が使用される場合、その変数はファイルがオープンされる前に設定されていることが必要です。 RPG サイクルの初期化部分で自動的にオープンされるファイルの場合、変数を以下のいずれかの方法によって設定しておく必要があります。
   - D 仕様書で INZ キーワードを使用する。
   - 値を入力パラメーターとして渡す。
   - 別のモジュールによって設定される、プログラムのグローバル変数を使用する。

RPG がオープンするファイルに対して一時変更を指定してある場合、その一時変更は効力を持ちます。 次のコーディングにおいて、RPG プログラム内で **INPUT** という名前を持つファイルの場合、実行時にオープンされるファイルは **ファイル名** フィールドの値によって決まります。

```
  Finput     if   f   10        disk    extfile(filename)
```

ファイル名 フィールドの値が実行時に MYLIB/MYFILE であれば、RPG はファイル MYLIB/MYFILE をオープンすることになります。 コマンド OVRDBF MYFILE OTHERLIB/OTHERFILE が使用された場合は、オープンされる実際のファイルは OTHERLIB/OTHERFILE になります。 INPUT はこの RPG ソース・メンバーの中で使用されている唯一の名前であるため、名前 INPUT に対してはいかなる一時変更も無視されることに注意してください。

```
 * The name of the file is known at compile time
Ffile1     IF   F   10        DISK    EXTFILE('MYLIB/FILE1')
Ffile2     IF   F   10        DISK    EXTFILE('FILE2')
 * The name of the file is in a variable which is
 * in the correct form when the program starts.
 * Variable "filename3" must have a value such as
 * 'MYLIB/MYFILE' or 'MYFILE' when the file is
 * opened during the initialization phase of the
 * RPG program.
Ffile3     IF   F   10        DISK    EXTFILE(filename3)

 * The library and file names are in two separate variables
 * The USROPN keyword must be used, so that the "filename4"
 * variable can be set correctly before the file is opened.
Ffile4     IF   F   10        DISK    EXTFILE(filename4)
F                                     USROPN
D filename4       S             21A
 * EXTFILE variable "filename4" is set to the concatenated
 * values of the "libnam" and "filnam" variables, to form
 * a value in the form "LIBRARY/FILE".
C                   EVAL      filename4 = %trim(libnam) + '/' + filnam
C                   OPEN      file4
 * At compile time, file MYLIB/MYFILE5 will be used to
 * get the external definition for the file "file5",
 * due to the EXTDESC keyword.
 * At runtime, the file MYLIB/MYFILE5 will be opened,
 * due to the EXTFILE(*EXTDESC) keyword.
Ffile5     if   e             DISK
F                             EXTFILE(*EXTDESC)
F                             EXTDESC('MYLIB/MYFILE5')
```

図 *121. EXTFILE* キーワードの例

EXTIND(*INUx)

EXTIND キーワードは、外部標識の値に応じて、ファイルがプログラムの中で使用 されるかどうかを指示します。

EXTIND によってプログラマーは、入力、出力、更新、または入出力共用ファイル の操作を実行時に制御することができます。 指定された標識がプログラムの初期化時にオンであれば、ファイルはオープン されます。 その標識がオンでない場合には、ファイルはオープンされず、処理の間は無視さ れます。 *INU1 から *INU8 標識を次のように設定することができます。

- IBM i 制御言語によって設定します。

- 演算命令の結果の標識としてか、または入力仕様のフィールド標識として 使用される時に設定します。 *INU1 から *INU8 標識をこの方法で設定しても、ファイルの条件付けに影響するこ とはありません。

389 ページの『USROPN』 も参照してください。

EXTMBR(メンバー名)

EXTMBR キーワードは、どのファイルのどのメンバーをオープンするかを指定します。 メンバー名は、'*ALL'、または '*FIRST' が指定できます。 '*ALL' と '*FIRST' は RPG の特殊語ではなくメンバーの「名前」として使われているため、括弧で囲んでしか使用できないことに注意します。 値はリテラルまたは変数を指定できます。 デフォルト値は '*FIRST' です。

名前の大文字小文字は正確でなければなりません。 たとえば、EXTMBR(メンバー名) を指定しその変数であるメンバー名の値が 'mbr1' だったとすると、そのメンバーは見付からないことになります。 メンバー名の値は 'MBR1' でなければなりません。

変数名が使用される場合、その変数はファイルがオープンされる前に設定されていることが必要です。 RPG サイクルの初期化部分で自動的にオープンされるファイルの場合、 変数を以下のいずれかの方法によって設定しておく必要があります。

- D 仕様書で INZ キーワードを使用する。
- 値を入力パラメーターとして渡す。
- 別のモジュールによって設定される、プログラムのグローバル変数を使用する。

FORMLEN(行数)

FORMLEN キーワードは、PRINTER ファイルの用紙の長さを指定します。 用紙の長さは、1 より大きいか等しく、255 より小さいか等しくなければなりま せん。 このパラメーターは、使用する用紙またはページ上で使用可能な正確な行数を指 定します。

用紙の長さを変更するためにプログラムを再コンパイルする必要はありません。 印刷装置ファイル一時変更 (OVRPRTF) コマンドの PAGSIZE パラメーターに新し い値を指定することによって、FORMLEN の行数パラメーターを一時変更することが できます。

FORMLEN キーワードを指定した場合には、FORMOFL キーワードも指定しなければ なりません。

FORMOFL(行番号)

FORMOFL キーワードは、オーバーフロー標識をオンに設定することになるオー バーフロー行番号を指定します。 オーバーフロー行番号は、用紙の長さより小さいかまたは等しくなければなりま せん。 オーバーフロー行として指定された行が印刷される時に、オーバーフロー標識が オンに設定されます。

オーバーフロー行を変更するためにプログラムを再コンパイルする必要はありま せん。 印刷装置ファイル一時変更 (OVRPRTF) コマンドの OVRFLW パラメーターに新し い値を指定することによって、FORMOFL の行数パラメーターを一時変更することが できます。

FORMOFL キーワードを指定した場合には、FORMLEN キーワードも指定しなければ なりません。

HANDLER(プログラムまたはプロシージャー { : 通信域)})

HANDLER キーワードは、ファイルがオープン・アクセス・ファイルであることを示します。 装置が DISK、PRINTER、SEQ、または WORKSTN のファイルに対して指定できます。

最初のパラメーターは、ファイルに対する入力命令および出力命令を処理する プログラムまたはプロシージャーを指定します。 次のように指定できます。

- プログラムまたはライブラリー修飾プログラムの名前を含んでいる 文字リテラル。 名前は大文字小文字が区別されます。

```
'MYPGM'
'*LIBL/MYPGM'
'MYLIB/MYPGM'
```

例えば、次のファイルのハンドラーはプログラム *LIBL/MYPGM です。

```
DCL-F myfile HANDLER('MYPGM');
```

- サービス・プログラム内のプロシージャーの名前を含んでいる 文字リテラル。 最初に、サービス・プログラムが、単にサービス・プログラム名として、 または、ライブラリーで修飾されたサービス・プログラム名として指定されます。 サービス・プログラムの後に、括弧で囲ったプロシージャー名が続きます。 名前は大文字小文字が区別されます。

```
'MYSRVPGM(myProcecedure)'
'*LIBL/MYSRVPGM(myProcedure)'
'MYLIB/MYSRVPGM(myProcedure)'
```

例えば、次のファイルのハンドラーは、サービス・プログラム MYLIB/MYSRVPGM 内のプロシージャー MyProc です。

```
DCL-F myfile HANDLER('MYLIB/MYSRVPGM(MyProc)');
```

- プログラムの名前またはサービス・プログラム内のプロシージャーの名前を含んでいる 文字変数。

例えば、次のファイルのハンドラーは、 文字変数 *handlerName* です。 このファイル は異なるハンドラーで 2 回オープンされます。

```
DCL-F myfile HANDLER(handlerName) USROPN;
DCL-S handlerName CHAR(50);

handlerName = 'MYLIB/MYPGM';
OPEN myfile;
READ myfile;
CLOSE myfile;

handlerName = 'MYLIB/MYSRVPGM(myHandler)';
OPEN myfile;
READ myfile;
CLOSE myfile;
```

- プロシージャー・ポインター。

例えば、次のファイルの ハンドラーは、プロシージャー・ポインター *handlerPointer* です。 このファイル は異なるハンドラーで 2 回オープンされます。

```
DCL-F myfile HANDLER(handlerPointer) USROPN;
DCL-S handlerPointer POINTER(*PROC);

handlerPointer = %PADDR('proc_a');
OPEN myfile;
READ myfile;
CLOSE myfile;

handlerPointer = %PADDR('proc_b');
OPEN myfile;
READ myfile;
CLOSE myfile;
```

- プロトタイプ名。

例えば、次のファイルのハンドラーは、 プロトタイプ *myHandler* です。

```
DCL-F myfile HANDLER(myproc);

/COPY QOAR/QRPGLESRC,QRNOPENACC
DCL-PR myproc;
   parm LIKEDS(QrnOpenAccess_T);
END-PR;
```

注 :

- 最初のパラメーターが変数の場合、その変数はファイルがオープンされる前に 設定されている必要があります。
- ハンドラー・プロシージャーがオープン・アクセス・ファイルと同じモジュール内に ある場合、ファイルは USROPN キーワードを指定して定義されている必要があります。

2 番目のパラメーターは任意指定です。 これは、 RPG プログラムが追加情報をハンドラーと直接共有できるように するためにハンドラーに渡される変数を指定します。

次の例では、ハンドラーは、通信域が、10 文字の *option* サブフィールドのあるデータ構造 であると予期します。

注 : ハンドラーが予期しているように通信域変数を定義するのは、 RPG プログラマーの責任です。 通常、コピー・ファイルを使用して、通信域用のデータ構造のテンプレート を定義します。 RPG プログラムとハンドラーが通信域に同じ定義を確実に使用するようにするためのコピー・ファイルの使用方法を示すオープン・アクセス・ファイルの完全な例については、177 ページの『オープン・アクセス・ハンドラーの例』を参照してください。

```
DCL-F myfile HANDLER('MYPGM' : options) USROPN;

DCL-DS options QUALIFIED;
   detail CHAR(10);
END-DS;

options.detail = 'FULL';
OPEN myfile;
. . .
CLOSE myfile;

options.detail = 'NONE';
OPEN myfile;
. . .
```

IGNORE(レコード様式 {:レコード様式... })

IGNORE キーワードによって、外部記述ファイルからのレコード様式を無視するこ とができます。 無視するレコード様式の外部名は、レコード様式パラメーターとして指定されま す。 1 つまたは複数のレコード様式をコロン (:) で区切って指定することができます。プログラムは、指定されたレコード様式は存在していないものとして実行されま す。 ファイルに含まれているその他のレコード様式はすべて組み込まれます。

ファイルについて IGNORE キーワードを指定した場合には、INCLUDE キーワード を指定することはできません。

修飾ファイルの場合、IGNORE キーワードで使用するレコード様式名は、必ず非修飾形式にしてください。

INCLUDE(レコード様式{:レコード様式...})

INCLUDE キーワードは、組み込むレコード様式名を指定します。ファイルに含ま れているその他のレコード様式はすべて無視されます。 ワークステーション・ファイルの場合には、SFILE キーワードを使用して指定されたレコード 様式もプログラムに組み込まれるので、それらを 2 回指定する必要はありません。 複数のレコード様式をコロン (:) で区切って指定することができます。

ファイルについて INCLUDE キーワードを指定した場合には、IGNORE キーワード を指定することはできません。

修飾ファイルの場合、INCLUDE キーワードで使用するレコード様式名は、必ず非修飾形式にしてください。

INDDS (データ構造名)

INDDS キーワードによって、データ構造名をワークステーションまたはプリンター・ファイルの INDARA 標識に関連付けることができます。 このデータ構造には、 ファイルのデータ管理機能との間で渡される条件付け標識および応答標識が 入り、標識データ構造と呼ばれます。

規則は次のとおりです。

- このキーワードを使用することができるのは、外部記述 PRINTER ファイル、 および外部記述とプログラム記述 ワークステーション・ファイルの場合だけです。
- プログラム記述ファイルの場合、PASS(*NOIND) キーワード を INDDS キーワードと一緒に指定することはできません。
- 同じデータ構造名を複数のファイルに関連付けることができます。
- データ構造名は、定義仕様書でデータ構造として定義する必要があります。 データ構造名は複数オカレンス・データ構造にすることができます。
- 標識データ構造の長さは 常に 99 です。
- 標識データ構造は、デフォルトで、すべてゼロ (複数の '0') に初期化されます。
- SAVEIND キーワードはこのキーワードと一緒に指定することはできません。

このキーワードを指定しない場合、定義済みのすべてのファイルの標識値 を DDS キーワード INDARA と関連付けるために *IN 配列が使用されます。

標識データ構造について詳しくは、215 ページの『特殊なデータ構造』を参照してください。

INFDS(DS 名)

INFDS キーワードによって、ファイルに関連したフィードバック情報 を入れるためのデータ構造を定義して、名前を付けることができます。 データ構造名は、INFDS のパラメーターとして指定されます。 INFDS を複数のファイルについて指定する場合には、関連した各データ構造が固 有の名前を持っていなければなりません。

INFDS は、ファイルと同じ有効範囲にコーディングされる必要があります。 つまり、グローバル・ファイルの場合はメイン・ソース・セクションに、ローカル・ファイルの場合はファイルと同じサブプロシージャーに、それぞれコーディングされなければなりません。 また、記憶域タイプ (静的または自動) がファイルと一致している必要があります。

ファイル情報データ構造について詳しくは、146 ページの『ファイル情報データ構造』を参照してください。

INFSR(SUBR 名)

INFSR キーワードは、ファイル例外/エラーの後に制御を受け取ることができる ファイル例外/エラー処理サブルーチンを識別します。 サブルーチン名は、このファイルのエラーについてはユーザー定義プログラムの 例外/エラー処理サブルーチンに制御が与えられることを指示する *PSSR とするこ とができます。

サブプロシージャーがアクセスするグローバル・ファイルに対しては、INFSR キーワードは指定できません。 INFSR サブルーチンは、ファイルと同じ有効範囲にコーディングされる必要があります。 つまり、ローカル・ファイルの場合はファイルと同じサブプロシージャーに、サイクル・モジュール内のグローバル・ファイルの場合はメイン・ソース・セクションに、それぞれコーディングされなければなりません。

KEYED{(*CHAR : キー長)}

KEYED キーワードは、自由形式ファイル定義で 使用されて、ファイルがキー・シーケンスでオープンされること、および、 ファイルに対してキー付きファイル命令が許可されることを示します。

外部記述ファイルの場合、KEYED キーワードにはパラメーターはありません。

```
DCL-F file1 DISK(*EXT) KEYED;
```

プログラム記述ファイルの場合、英数字キーのみがサポートされます。 KEYED キーワードには、*CHAR とキー長の 2 つの パラメーターがなければなりません。

```
DCL-F file2 DISK(100) KEYED(*CHAR : 10);
```

プログラム記述ファイルを異なるタイプのキーと共に定義したい場合、KEYED(*CHAR:キー長) を指定してキーを英数字として定義し、 必要なデータ・タイプのサブフィールドが含まれているデータ構造をキー付き命令で検索引数として 使用することができます。

KEYLOC(位置)

KEYLOC キーワードは、プログラム記述索引付きファイルのキー・フィールドが 始まるレコード位置を指定します。 このパラメーターは 1 から 32766 でなければなりません。

レコードのキー・フィールドにはレコードを識別する情報が入っています。 キー・フィールドは、ファイル内のすべてのレコードで同じ位置になければなり ません。

LIKEFILE(親ファイル名)

LIKEFILE キーワードは、あるファイルに類似した別のファイルを定義するために使用します。

注 : 以下の説明では、LIKEFILE キーワードを使用して定義されるファイルを、「新規ファイル」という用語で呼びます。 新規ファイルの定義を派生させるために使用される定義を持つ LIKEFILE キーワードのパラメーターを、「親ファイル」という用語で呼びます。

LIKEFILE キーワードに関する規則*:*

- LIKEFILE キーワードを使用してファイルを定義した場合には、QUALIFIED キーワードとみなされます。 LIKEFILE キーワードで定義されたファイルでは、自動的にレコード様式が修飾されます。 親ファイル FILE1 のレコード様式が RECA および RECB である場合、新規ファイル FILE2 のレコード様式は、RPG プログラム内では FILE2.RECA および FILE2.RECB によって参照される必要があります。
- QUALIFIED キーワードは、LIKEFILE キーワードと一緒に指定することはできません。
- 無視されなかった親ファイルのレコード様式は、すべて新規ファイルで使用できます。
- LIKEFILE キーワードを指定する場合、パラメーターとして指定されたファイルがあらかじめソース・ファイルに定義されている必要があります。
- LIKEFILE キーワードがサブプロシージャーで指定され、パラメーターとして指定したファイルがグローバル定義で定義されている場合、コンパイラーは LIKEFILE 定義のスキャン時にグローバル定義を見つけます。
- LIKEFILE によって定義されたファイルでは、入力仕様および出力仕様は生成されないか、または使用できません。 すべての入出力命令を実行するには、結果データ構造を使用する必要があります。
- LIKEFILE を使用してファイルを定義する場合は、親ファイルのファイル仕様書で、ファイルをブロック化するかどうかが明確にされている必要があります。 親ファイルに対して、BLOCK キーワードの指定が必須になる場合があります。 例えば、入力 DISK ファイルの場合、そのファイルで使用される演算命令に応じてファイルがブロック化されるため、LIKEFILE キーワードでファイルが使用される際には BLOCK キーワードが必須となります。 Input-Add DISK ファイルの場合、ファイルをブロック化することができないため、BLOCK キーワードは必要ありません。
- BLOCK(*YES) が指定されたファイルが、LIKEFILE キーワードで指定されたファイルの親ファイルとして使用された場合、親ファイル、または LIKEFILE キーワードを通じて親ファイルに関連するすべてのファイルにおいて、READE、READPE、および READP の各命令を使用することはできません。
- 親ファイルのプロパティーの中には、新規ファイルに継承されるものとそうでないものとがあります。 継承されるプロパティーのうち、ファイル仕様書のキーワードによって一時変更できるものがあります。 継承されないプロパティーは、ファイル仕様書のキーワードを使って、新規ファイルに対して指定できます。詳しくは、377 ページの表 87 を参照してください。

表 87. 継承されるファイル・プロパティーのうち、一時変更できるもの		
プロパティーまたはキーワード	**親ファイルから継承**	**新規ファイルに指定可能**
ファイル・タイプ (入力、更新、出力、結合)	はい	いいえ
ファイルの追加	はい	いいえ
レコード・アドレス・タイプ (RRN、keyed)	はい	いいえ
レコード長 (プログラム記述ファイル)	はい	いいえ
キーの長さ (プログラム記述ファイル)	はい	いいえ
ファイル編成 (プログラム記述ファイル)	はい	いいえ
装置	はい	いいえ
ALIAS	はい	いいえ
BLOCK	はい	いいえ
COMMIT	いいえ	はい
DATFMT	N/A、注 1 を参照	
DEVID	いいえ	はい
ディスク	はい	いいえ
EXTDESC	はい	いいえ
EXTFILE	はい、注 2 を参照	はい
EXTIND	いいえ	はい
EXTMBR	はい、注 2 を参照	はい
FORMLEN	はい	はい
FORMOFL	はい	はい
HANDLER	N/A。HANDLER キーワードは新規ファイルにも親ファイルにもサポートされていません。	
IGNORE	はい	いいえ
INCLUDE	はい	いいえ
INDDS	いいえ	はい
INFDS	いいえ	はい
INFSR	いいえ	はい
KEYED	はい	いいえ
KEYLOC	はい	いいえ
LIKEFILE	はい	N/A
MAXDEV	はい	はい
OFLIND	いいえ	はい

表 87. 継承されるファイル・プロパティーのうち、一時変更できるもの (続き)		
プロパティーまたはキーワード	**親ファイルから継承**	**新規ファイルに指定可能**
PASS	はい	いいえ
PGMNAME	はい	はい
PLIST	いいえ	はい
PREFIX	はい	いいえ
PRINTER	はい	いいえ
PRTCTL	いいえ	はい
QUALIFIED	N/A、QUALIFIED は新規ファイルに常に暗黙指定される	
RAFDATA	N/A、注 3 を参照	
RECNO	いいえ	はい
RENAME	はい	いいえ
SAVEDS	いいえ	はい
SAVEIND	いいえ	はい
SEQ	はい	いいえ
SFILE	はい、注 4 を参照	はい、注 4 を参照
SLN	いいえ	はい
SPECIAL	はい	いいえ
STATIC	いいえ	はい
TEMPLATE	いいえ	はい
TIMFMT	N/A、注 1 を参照	
USAGE	はい	いいえ
USROPN	いいえ	はい
WORKSTN	はい	いいえ

注 :

1. キーワード DATFMT および TIMFMT は、ファイルのプログラム記述の入力仕様にコーディングされる「日付および時刻」フィールドに関連しています。 ただし、LIKEFILE キーワードで定義されるファイルと入力仕様とは無関係です。
2. RPG ファイルに関連付けられる外部ファイルは、親ファイルと新規ファイルの両方に指定されるキーワード EXTFILE および EXTMBR に応じて異なります。 デフォルトでは、ファイルの名前項目で指定される名前が、各ファイルに関連付けられる外部ファイルを指します。 パラメーターが定数の場合、新規ファイルは親ファイルからキーワード EXTFILE または EXTMBR を継承します。 ただし、これらのキーワードが、新規ファイルに指定されることもあります。 EXTFILE または EXTMBR のパラメーターが定数でない場合、キーワード EXTFILE または EXTMBR は継承されません。 親ファイルおよび新規ファイルの EXTFILE および EXTMBR の値のうち、親ファイルの LIKEFILE で定義されたものに関する例として、実行時に使用される可能性のある外部ファイルについて、以下の表で説明します。

表 88. ファイル仕様書の例: *EXTFILE* および *EXTMBR*	
ファイル仕様書	**実行時に使用される外部ファイル (太字部分は、継承される値)**
EXTFILE および EXTMBR の値が両方とも定数である場合の例	

表 88. ファイル仕様書の例: *EXTFILE* および *EXTMBR* (続き)

ファイル仕様書	実行時に使用される外部ファイル (太字部分は、継承される値)
FFILE1 IF E DISK FFILE2 LIKEFILE(FILE1)	*LIBL/FILE1(*FIRST) *LIBL/FILE2(*FIRST)
FFILE1 IF E DISK **EXTFILE('MYLIB/MYFILE')** FFILE2 LIKEFILE(FILE1)	MYLIB/MYFILE(*FIRST) **MYLIB/MYFILE**(*FIRST)
FFILE1 IF E DISK FFILE2 LIKEFILE(FILE1) **EXTFILE('MYLIB/MYFILE')**	*LIBL/FILE1(*FIRST) MYLIB/MYFILE(*FIRST)
FFILE1 IF E DISK **EXTFILE('MYLIB/MYFILE1')** FFILE2 LIKEFILE(FILE1) **EXTFILE('MYLIB/MYFILE2')**	MYLIB/MYFILE1(*FIRST) MYLIB/MYFILE2(*FIRST)
FFILE1 IF E DISK **EXTMBR('MBR1')** FFILE2 LIKEFILE(FILE1)	*LIBL/FILE1(MBR1) *LIBL/FILE2**(MBR1)**
FFILE1 IF E DISK FFILE2 LIKEFILE(FILE1) **EXTMBR('MBR1')**	*LIBL/FILE1(*FIRST) *LIBL/FILE2(MBR1)
FFILE1 IF E DISK EXTMBR('MBR1') FFILE2 LIKEFILE(FILE1) **EXTFILE('MYLIB/MYFILE2')**	*LIBL/FILE1(MBR1) MYLIB/MYFILE2**(MBR1)**
EXTFILE および EXTMBR の値が両方とも変数である場合の例	
FFILE1 IF E DISK **EXTFILE(extfileVariable)** FFILE2 LIKEFILE(FILE1) Value of extfileVariable: 'MYLIB/MYFILE'	MYLIB/MYFILE(*FIRST) *LIBL/FILE2(*FIRST)
FFILE1 IF E DISK FFILE2 LIKEFILE(FILE1) **EXTFILE(extfileVariable)** Value of extfileVariable: 'MYLIB/MYFILE'	*LIBL/FILE1(*FIRST) MYLIB/MYFILE(*FIRST)
FFILE1 IF E DISK **EXTFILE(extfileVariable1)** FFILE2 LIKEFILE(FILE1) **EXTFILE(extfileVariable2)** Value of extfileVariable1: 'MYLIB/MYFILE1' Value of extfileVariable2: 'MYLIB/MYFILE2'	MYLIB/MYFILE1(*FIRST) MYLIB/MYFILE2(*FIRST)
FFILE1 IF E DISK **EXTMBR(extmbrVariable)** FFILE2 LIKEFILE(FILE1) Value of extmbrVariable: 'MBR1'	*LIBL/FILE1(MBR1) *LIBL/FILE2(*FIRST)
FFILE1 IF E DISK FFILE2 LIKEFILE(FILE1) **EXTMBR(extmbrVariable)** Value of extmbrVariable: 'MBR1'	*LIBL/FILE1(*FIRST) *LIBL/FILE2(MBR1)
FFILE1 IF E DISK **EXTMBR(extmbrVariable)** FFILE2 LIKEFILE(FILE1) **EXTFILE(extfileVariable)** Value of extmbrVariable: 'MBR1' Value of extfileVariable: 'MYLIB/MYFILE2'	*LIBL/FILE1(MBR1) MYLIB/MYFILE2(*FIRST)
EXTFILE および EXTMBR の値に変数と定数が混用されている場合の例	

表 88. ファイル仕様書の例: *EXTFILE* および *EXTMBR* (続き)

ファイル仕様書	実行時に使用される外部ファイル (太字部分は、継承される値)
FFILE1 IF E DISK **EXTFILE(extfileVariable1) EXTMBR('MBR1')** FFILE2 LIKEFILE(FILE1) Value of extfileVariable1: 'MYLIB/MYFILE1'	MYLIB/MYFILE1(MBR1) *LIBL/FILE2**(MBR1)**
FFILE1 IF E DISK **EXTMBR(extmbrVariable)** FFILE2 LIKEFILE(FILE1) Value of extmbrVariable: 'MBR1'	*LIBL/FILE1(MBR1) *LIBL/FILE2(*FIRST)
FFILE1 IF E DISK **EXTFILE('MYLIB/MYFILE1')** **EXTMBR(extmbrVariable)** FFILE2 LIKEFILE(FILE1) Value of extmbrVariable: 'MBR1'	MYLIB/MYFILE1(MBR1) **MYLIB/MYFILE1**(*FIRST)

3. RAFDATA キーワードは、1 次ファイルおよび 2 次ファイルにのみ関係します。 ただし、親ファイルは全手順ファイルでなければなりません。
4. SFILE キーワードは、レコード様式をサブファイル・レコード様式にするよう指定し、サブファイルに相対レコード番号を指定する際に使用する変数の名前も指定します。 特定のレコード様式がサブファイル・レコード様式であることを、新規ファイルは自動的に継承します。 ただし、RRN の指定に使用される変数の名前は継承しません。 サブファイルに相対レコード番号を指定するのに使用する変数を指定するために、新規ファイルに SFILE キーワードを指定する必要があります。

MAXDEV(*ONLY | *FILE)

MAXDEV キーワードは、ワークステーション・ファイルについて定義される装置の最大数 を指定します。 デフォルトの値の *ONLY は単一装置ファイルを指示します。 *FILE を指定した場合には、ファイルのオープン時に (ファイル作成コマンドで ワークステーション・ファイルについて定義された) 装置の最大数が検索され、実行時には SAVEIND および SAVEDS のスペース割り振りが行われます。

共用ファイルでは、MAXDEV 値は入手する装置の数を制限するためには使用されま せん。

DEVID、SAVEIND、または SAVEDS を指定し、MAXDEV を指定しなかった場合には、 プログラムは複数装置ファイルのデフォルトの値 (MAXDEV が *FILE のパラメーター を持つ) と見なします。

OFLIND(標識)

OFLIND キーワードは、オーバーフローが起こった時に印刷する PRINTER ファイルの行を条件付けするためのオーバーフロー標識を指定します。 この指定が有効なのは、PRINTER 装置の場合だけです。 OFLIND キーワードが指定されていない場合には、デフォルトのオーバーフロー処理 (すなわち、オーバーフロー時の自動ページ・スキップ) が行われます。

有効なパラメーターは次のとおりです。

***INOA から *INOG, *INOV:**
指定されたオーバーフロー標識は、プログラム記述印刷装置ファイルでオー バーフローが起こった時に印刷する行を条件付けします。

***IN01-*IN99:**
オーバーフロー行に行が印刷された場合、またはスペースやスキップ命令の 実行時にオーバーフロー行に達するかオーバーフロー行を超えた場合にオンに設定されます。

名前:
タイプ標識とともに定義される変数の名前であり、配列ではありません。 この標識は、オーバーフロー行に達してプログラムがこのオーバーフロー条件を処理する必要がある場合に、オンに設定されます。

この標識の動作は、標識 *IN01 から *IN99 と同じになります。

注 : *INOA から *INOG、および *INOV の標識は外部記述ファイルには有効でありません。

1 つのファイルに割り当てていることができるのは 1 つのオーバーフロー標識だけです。 1 つの モジュール内で複数の PRINTER ファイルにオーバーフロー標識を割り当てる場合には、その標識が各ファイルごとに固有のものでなければなりません。 複数のファイルでは、その 1 つが別のプロシージャーで定義されていても、グローバル標識は使用できません。

PASS(*NOIND)

PASS キーワードは、標識がプログラマーの制御のもとで渡されるか、あるいは DDS のキーワード INDARA に基づいて渡されるかを決定します。 このキーワードを指定することができるのは、プログラム記述ファイルの場合だ けです。 入力および出力での標識の受け渡しにユーザーが責任を持つことを指示するために は、対応するプログラム記述 ワークステーション・ファイルのファイル仕様書に PASS(*NOIND) を指定してください。

PASS(*NOIND) が指定された場合に、ILE RPG コンパイラーは出力時にデータ管理機能に標識を渡すことも、入力時にデータ管理機能から標識を受け取ることもしません。 その代わりに標識は、入力または出力レコードにフィールド (*INxx、*IN(xx)、 または *IN 形式) として記述することによって渡されます。 これらはデータ記述仕様書 (DDS) で必要とされる 順序で指定しなければなりません。 DDS リスト出力を使用してこの順序を判別することが できます。

このキーワードが指定されていない場合には、コンパイラーは DDS に INDARA が 指定されたものと見なします。

注 : ファイルの DDS に INDARA キーワードが指定されている 場合、PASS(*NOIND) を指定してはなりません。 DDS に INDARA キーワードが 指定されていない場合には、PASS(*NOIND) を指定する必要があります。

PGMNAME(プログラム名)

PGMNAME キーワードは、(SPECIAL の装置記入項目によって指示された) 特殊な入 出力装置に対するサポートを処理するプログラムを識別します。

注 : このパラメーターは有効なプログラム名でなければならず、バインドされたプロ シージャー名であってはなりません。

詳しくは、363 ページの『36 から 42 桁目 (装置)』 および 381 ページの『PLIST(PLIST 名)』 を参照してください。

PLIST(PLIST 名)

PLIST キーワードは、SPECIAL ファイル用のプログラムに渡されるパラメーター ・リストの名前を識別します。 この記入項目によって識別されたパラメーターは、プログラムによって渡される パラメーター・リストの終わりに追加されます。(プログラムは PGMNAME キーワードを使用して指定されます。381 ページの『PGMNAME(プログラム名)』 を参照してください。) このキーワードを指定することが できるのは、ファイル記述行の装置記入項目 (36 から 42 桁目) が SPECIAL に なっている場合だけです。

PREFIX(接頭部 {:置き換えられる文字数 })

PREFIX キーワードは、外部記述ファイル中のフィールドの名前を部分的に変更する ために使用されます。最初のパラメーターで指定される文字が、 外部記述ファイルのすべてのレコード内に定義されたすべてのフィールドの名前に 接頭部として付加されます。 文字は、名前 (例えば PREFIX(F1_))、または文字リテラル (例えば PREFIX('F1_')) として指定できます。接頭部にピリオドが含まれる場合 (例えば PREFIX('F1DS.') または PREFIX('F1DS.A')) には、文字リテラルを使用する必要があります。 それぞれの名前の先頭から文字を削除するには、 最初のパラメーターとして空のストリングを PREFIX(':削除する数) のように指 定します。 さらに、既存の名前の中で置き換えられる文字 (それがあった場合) の数を指示する 数値をオプションで指定することができます。 「置き換えられる文字数」を指定しない場合には、名前の先頭にストリングが付加さ れます。

「置き換えられる文字数」を指定する場合には、0 から 9 の値を含み、小数点以下の桁数 のない数値定数としなければなりません。 たとえば、PREFIX(YE:3) の指定によって、フィールド名 'YTDTOTAL' が 'YETOTAL' に変更されます。 ゼロの値の指定は、「置き換えられる文字数」をまったく指定しないことと同じです。

別名に接頭部を適用する場合、「置き換えられる文字数」のパラメーターは使用されません。 PREFIX キーワードと ALIAS キーワードの相互の影響については、ALIAS キーワードを参照してください。

規則は次のとおりです。

- あるファイルについて PREFIX キーワードが指定されている場合に入力仕様のフィールドを明示的に名前変更するには、 入力仕様の「外部フィールド名 (External Field Name)」(桁 21 - 30) に指定する、 正しいフィールド名を選択する必要があります。 指定する名前は、名前変更を指定する前に接頭部付きの名前が使用されていたかどうかによって異なります。
  - 以前に接頭部付きの名前に対する参照が行われている場合には、接頭部付きの名前を指定する必要があります。
  - 以前に接頭部付きの名前に対する参照が行われていない場合には、 入力フィールドの外部名を指定する必要があります。

  名前変更命令をコーディングした後は、その入力フィールドは新しい名前で参照する必要があります。 詳細については、入力仕様の「外部フィールド名 (External Field Name)」を参照してください。
- 接頭部を適用した後の名前の合計長が RPG フィールド名の最大長を超えては なりません。 ファイルが LIKEFILE キーワードも QUALIFIED キーワードも指定せずに定義された グローバル・ファイルである場合、名前全体の長さは、 入力仕様および出力仕様の「名前」記入項目の長さである 14 文字を超えてはなりません。
- 接頭部が付けられる名前の文字数が、「置き換えられる文字数」パラメーター によって表された値以下であってはなりません。 つまり、接頭部を適用した後の名前が、接頭部ストリングと同じ 長さになってはならない、ということです。
- 接頭部が文字リテラルの場合は、ピリオドを含めるか、またはピリオドで終わることができます。 この場合、フィールド名はすべて、同一の修飾されたデータ構造のサブフィールドであることが必要です。 データ構造は、修飾されたデータ構造として定義する必要があります。 例えば PREFIX('F1DS.') の場合、データ構造 F1DS は修飾されたデータ構造として定義する必要があり、ファイルにフィールド FLD1 および FLD2 があるときは、データ構造にはサブフィールド F1DS.FLD1 および F1DS.FLD2 が必要です。 同様に、PREFIX('F2DS.A') の場合、データ構造 F2DS は修飾されたデータ構造である必要があり、ファイルにフィールド FLD1 および FLD2 があるときは、データ構造にはサブフィールド F2DS.AFLD1 および F2DS.AFLD2 が必要です。
- 接頭部が文字リテラルの場合は英大文字でなければなりません。
- 外部記述データ構造がファイル内のフィールドの定義に使用されている場合は、ファイル内のフィールド名がデータ構造のサブフィールド名と同一であることを確認する必要があります。 以下の表には、名前「XYNAME」に複数の接頭部が付けられる場合の、外部記述ファイルおよび外部記述データ構造に必要な接頭部が示されています。 「内部名」の列にドットが含まれる場合 (例えば D1.NAME)、外部記述データ構造は QUALIFIED として定義され、ファイル指定の PREFIX にドットを含める必要があります。

ファイルの PREFIX	外部記述データ構造の PREFIX	内部名
PREFIX(A)	PREFIX(A)	AXYNAME
PREFIX(A:2)	PREFIX(A:2)	ANAME
PREFIX('D.')	なし	D.XYNAME
PREFIX('D.' : 2)	PREFIX('' : 2)	D.NAME
PREFIX('D.A')	PREFIX(A)	D.AXYNAME
PREFIX('D.A' : 2)	PREFIX(A : 2)	D.ANAME
PREFIX(':2)	PREFIX('' : 2)	NAME

例:

以下の例では、ファイル NEWFILE のフィールド名の先頭に接頭部「NEW_」を追加し、ファイル OLDFILE のフィールド名の先頭に「OLD_」を追加しています。

```
Fnewfile   o    e             disk    prefix(NEW_)
Foldfile   if   e             disk    prefix(OLD_)
```

```
C                   READ      OLDREC
C                   EVAL      NEWIDNO = OLD_IDNO
C                   EVAL      NEWABAL = OLD_ABAL
C                   WRITE     NEWREC
```

以下の例では、ファイル FILE1 および外部記述データ構造 DS1 の両方で PREFIX(N:2) を使用しています。 ファイル指定の接頭部により、FILE1 のフィールド XYIDNUM および XYCUSTNAME はプログラム内で NIDNUM および NCUSTNAME として認識される ようになります。また、データ指定の接頭部により、データ構造にはサブフィールド NIDNUM および NCUSTNAME が存在するようになります。 READ 命令時にレコードからのデータは DS1 のサブフィールドに移動され、サブプロシージャー processRec に受け渡されてレコード内のデータが処理されます。

```
Ffile1     if   e             disk    prefix(N:2)
D ds1           e ds                  extname(file1) prefix(N:2)
C                   READ      file1
C                   CALLP     processRec (ds1)
```

以下の例では、MYFILE 内のフィールドを修飾されたデータ構造 MYDS のサブフィールドに関連付けるために、接頭部 'MYDS.' を使用します。

```
Fmyfile    if   e             disk    prefix('MYDS.')
D myds          e ds                  qualified extname(myfile)
```

次の例では、MYFILE 内のフィールドを修飾されたデータ構造 MYDS2 のサブフィールドに関連付けるために、接頭部 'MYDS2.F2':3 を使用します。 サブフィールドそのものには、先頭の 3 文字を 'F2' で置き換えることにより、さらに接頭部が付きます。 このファイルが使用するフィールドは、MYDS2.F2FLD1 と MYDS2.F2FLD2 になります。 (データ構造 MYDS2 は同じ接頭部を使用して定義する必要があります。 ただし、これはデータ構造名は含まないため、まったく同じ名前にはなりません。)

```
A          R REC
A            ACRFLD1       10A
A            ACRFLD2        5S 0
Fmyfile2   if   e               disk    prefix('MYDS2.F2':3)
D myds2         e ds                    qualified extname(myfile)
D                                       prefix('F2':3)
```

PRINTER{(*EXT | レコード長)}

PRINTER キーワードは、装置タイプ・キーワードの 1 つです。 これは自由形式ファイル定義で使用されて、ファイル装置が PRINTER で あることを示します。 これは最初のキーワードでなければなりません。

パラメーターは任意指定です。 デフォルトは *EXT です。

***EXT**
ファイルが外部記述であることを示すには、*EXT を指定します。 これはデフォルト値。

レコード長
ファイルがプログラム記述であることを示すには、レコード長 を 指定します。 レコード長は 1 から 32766 の間でなければなりません。 リテラルまたは 名前付き定数を指定できます。 名前付き定数である場合、ファイル定義ステートメントの前にその定数が定義されている必要があります。

PRTCTL (データ構造 {:*COMPAT })

PRTCTL キーワードは、動的印刷装置制御の使用を指定します。 「データ構造」パラメーターとして指定されたデータ構造によって、用紙制御情 報および行カウント値が参照されます。 PRTCTL キーワードが有効なのは、プログラム記述ファイルの場合だけです。

任意指定パラメーター *COMPAT は、データ構造のレイアウトに RPG III との互 換性があることを指示します。 *COMPAT が指定されないデフォルトの場合には、拡張された長さのデータ構造を使用 することが必要になります。

拡張された長さの *PRTCTL* データ構造

このデータ構造には最小でも 15 バイトが必要です。 PRTCTL データ構造のレイアウトは次のとおりです。

データ構造の桁
サブフィールドの内容

1 〜 3
印刷前スペースの値が入る 3 桁の文字フィールド (有効な指定: ブランクまたは 0 から 255)

4 〜 6
印刷後スペースの値が入る 3 桁の文字フィールド (有効な指定: ブランクまたは 0 から 255)

7 〜 9
印刷前スキップの値が入る 3 桁の文字フィールド (有効な指定: ブランクまたは 1 から 255)

10 〜 12
印刷後スキップの値が入る 3 桁の文字フィールド (有効な指定: ブランクまたは 1 から 255)

13 〜 15
現在の行カウント値を含む小数点以下の桁数がない 3 桁の数字 (ゾーン 10 進数) フィールド

**COMPAT PRTCTL* データ構造

データ構造の桁
サブフィールドの内容

1
印刷前スペースの値が入る 1 桁の文字フィールド (有効な指定: ブランクまたは 0 から 3)

2
印刷後スペースの値が入る 1 桁の文字フィールド (有効な指定: ブランクまたは 0 から 3)

3 〜 4
印刷前スキップの値が入る 2 桁の文字フィールド (有効な指定: ブランク、1 から 99、100 から 109 の場合は A0 から A9、110 から 112 の場合は B0 から B2)

5 〜 6
印刷後スキップの値が入る 2 桁の文字フィールド (有効な指定: ブランク、1 から 99、100 から 109 の場合は A0 から A9、110 から 112 の場合は B0 から B2)

7 〜 9
現在の行カウント値を含む小数点以下の桁数がない 3 桁の数字 (ゾーン 10 進数) フィールド

拡張された長さのデータ構造の最初の 4 つのサブフィールドに含まれる値は、出 力仕様の 40 から 51 桁目 (スペースおよびスキップ記入項目) に使用できるもの と同じ値です。 出力仕様のスペースおよびスキップ記入項目 (40 から 51 桁目) がブランクでサ ブフィールド 1 から 4 もブランクであった場合には、デフォルトの値は印刷後 1 行スペ ースになります。 PRTCTL オプションが指定されている場合には、40 から 51 桁目がブランクの出力レ コードにのみ使用されます。 PRINTER ファイルのスペースとスキップの値 (サブフィールド 1 から 4) は、プログ ラムの実行中にこれらのサブフィールドの値を変更することによって制御するこ とができます。

サブフィールド 5 には、現在の行カウント値が入れられます。 ILE RPG コンパイラーは最初の出力行が印刷された後までサブフィールド 5 を初期化しません。 その後、コンパイラーは、ファイルに対する各出力操作の後にサブフィールド 5 を変更します。

QUALIFIED

QUALIFIED キーワードは、ご使用の RPG ソースにおけるファイルのレコード様式の指定方法を制御します。

このキーワードを指定する場合、RPG ソースで指定されたレコード様式は、ファイル名で修飾されている必要があります。 例えば、修飾ファイル FILE1 のファイル様式 FMT1 は、FILE1.FMT1 と指定しなければなりません。 レコード様式名は、RPG ソース内で使用されている他の名前と同じものにすることができます。

このキーワードが指定されていない場合は、レコード様式をファイル名で修飾してはいけません。 様式 FMT1 は、FMT1 として指定します。 レコード様式名は、RPG ソース内で固有の名前でなければなりません。

QUALIFIED キーワードに関する規則:

- ファイルが修飾されている場合、ソース内のすべての箇所でレコード名を修飾する必要があります。 ただし、ファイル仕様書のキーワード RENAME、INCLUDE、IGNORE および SFILE のパラメーターとして指定されている場合は除きます。 これらのキーワードのパラメーターとして指定されている場合は、名前を修飾してはいけません。
- ファイルが修飾された場合、入力仕様および出力仕様がファイルで使用できなくなるか、または生成されません。 つまり、ファイルの外部にあるフィールドは、プログラム内のフィールドとして自動的に定義されないということです。 すべての入出力は、結果データ構造を使用して行う必要があります。
- QUALIFIED キーワードが有効なのは、外部記述ファイルの場合のみです。
- QUALIFIED キーワードは、LIKEFILE キーワードと一緒に指定することはできません。 LIKEFILE キーワードで定義されたファイルのレコード様式は、必ず修飾されてしまいます。

```
 * file1 has formats HDR, INFO, ERR.
 * file2 has format INFO.
 * The QUALIFIED keyword is used for both files, making it
 * unnecessary to rename one of the "INFO" formats.

 * Note that the record format names are not qualified when
 * specified in keywords of the File specification.
Ffile1     if   e             disk qualified
F                                  ignore(hdr)
F                                  rename(err:errorRec)
Ffile2     o    e             disk qualified
 * The record formats must be qualified on all specifications other
 * than the File specification for the file.
D ds1              ds                  likerec(file1.info : *input)
D errDs            ds                  likerec(file1.errorRec : *input)
D ds2              ds                  likerec(file2.info : *output)
 /free
       read file1.info ds1;
       eval-corr ds2 = ds1;
       write file2.info ds2;
       read file1.errorRec errDs;
```

図 *122. QUALIFIED* キーワードの例

RAFDATA(ファイル名)

RAFDATA キーワードは、レコード・アドレス・ファイル (RAF) (18 桁目に R) に ついて処理されるデータ・レコードが入れられる入力または更新ファイルの名前 を識別します。 詳しくは、356 ページの『レコード・アドレス・ファイル (RAF)』を参照してください。

RECNO(フィールド名)

RECNO キーワードは、DISK ファイルが相対レコード番号によって処理されること を指定します。 相対レコード番号によって処理される出力ファイル、ランダム WRITE 演算命令に よって参照される出力ファイル、または出力仕様で ADD と一緒に使用され た出力ファイルの場合には、RECNO キーワードを指定しなければなりません。

RECNO キーワードを指定することができるのは、入力/更新ファイルの場合だけです。 検索されたレコードの相対レコード番号は、ファイルを再位置決めするすべての 命令 (READ、SETLL、または OPEN など) の場合にこの「フィールド名」に入れら れます。 これは、小数点以下の桁数のない数値として定義しなければなりません。 フィールドの長さは、ファイルの最長レコード番号が十分に入るものでなければ なりません。

コンパイラーは、ファイルに RECNO キーワードが指定されている場合に、SEQ ま たは DISK ファイルのレコードをブロック化または非ブロック化するようにはオ ープンしません。 キーワードの RECNO と BLOCK(*YES) を同じファイルについて指定することはで きない点に注意してください。

注 : ファイル追加が指定された入力ファイル または更新ファイル (自由形式ファイル定義の場合は USAGE キーワード、 固定形式ファイル定義の場合は 17 桁目と 20 桁目を 参照してください) に対して RECNO キーワードが指定される場合、出力命令が成功するためには、フィールド名パラメーターの 値は、削除レコードの相対レコード番号を参照していなければなりません。

RENAME(外部形式:内部形式)

RENAME キーワードによって、外部記述ファイル中のレコード様式の 名前を変更することができます。 名前を変更するレコード様式の外部名は、「外部形式」パラメーターとして入力 されます。 「内部形式」パラメーターは、プログラムで使用される場合のレコードの名前で す。 外部名がプログラムの中でこの名前に置き換えられます。

接頭部を追加することによってすべてのフィールドの名前を変更するためには、PREFIX キーワードを使用してください。

修飾ファイルの場合、RENAME キーワードの両方パラメーターで使用するレコード様式名は、必ず非修飾形式にしてください。

SAVEDS(DS 名)

SAVEDS キーワードによって、各装置について保管および復元されるデータ構造を 指定することができます。 入力操作に先立って、装置の操作用のデータ構造が保管されます。 入力操作の後 に、この現行入力操作と関連した装置のデータ構造が復元されます。 このデータ 構造は、データ域データ構造、ファイル情報データ構造、またはプログラム状況 データ構造とすることはできません。また、コンパイル時配列または実行時前配 列を含めることもできません。

SAVEDS キーワードが指定されていない場合には、保管および復元は行われませ ん。 共用ファイルについて SAVEDS を指定してはなりません。

SAVEDS を指定し、MAXDEV を指定しなかった場合には、ILE RPG プログラムは複数装置ファイル (MAXDEV が *FILE のパラメーターを持つ) を想定します。

SAVEIND(番号)

SAVEIND キーワードは、混合または複数装置ファイルに接続された各装置ごとに 保管および復元される標識の番号を指定します。 入力操作に先立って、前の入力または出力操作と関連した装置用の標識が保管されます。 入力操作の後に、この現行入力操作と関連した装置の標識が復元されます。

SAVEIND キーワードに対するパラメーターとして 1 から 99 の数値を指定してくださ い。 SAVEIND キーワードが指定されていない場合、あるいは MAXDEV キーワードが指 定されていないかまたは *ONLY パラメーターと一緒に指定されている場合に は、標識は保管および復元されません。

DDS キーワード INDARA を指定した場合には、SAVEIND キーワードに対して指定 する数値はその DDS で使用しているどの応答標識よりも小さくなければなりませ ん。 たとえば、DDS に INDARA および CF01(55) を指定した場合には、SAVEIND キー ワードの最大値は 54 になります。 共用ファイルについて SAVEIND キーワードを使用してはなりません。

INDDS キーワードはこのキーワードと一緒に指定することはできません。

SAVEIND キーワードを指定し、MAXDEV キーワードを指定しなかった場合には、ILE RPG プログラムは複数装置ファイルと見なします。

SEQ{(*EXT | レコード長)}

SEQ キーワードは、装置タイプ・キーワードの 1 つです。 これは自由形式ファイル定義で使用されて、ファイル装置が SEQ で あることを示します。 これは最初のキーワードでなければなりません。

パラメーターは任意指定です。 デフォルトは *EXT です。

***EXT**
ファイルが外部記述であることを示すには、*EXT を指定します。 これはデフォルト値。

レコード長
ファイルがプログラム記述であることを示すには、レコード長 を 指定します。 レコード長は 1 から 32766 の間でなければなりません。 リテラルまたは 名前付き定数を指定できます。 名前付き定数である場合、ファイル定義ステートメントの前にその定数が定義されている必要があります。

SFILE(レコード様式:RRN フィールド)

SFILE キーワードは、外部記述 ワークステーション・ファイルに指定されたサブファイルを内 部的に定義するために使用されます。 この再形式設定パラメーターは、サブファイルとして処理されるレコード様式の RPG IV 名を識別します。 RRN フィールド・パラメーターは、このサブファイルの相対レコード番号フィールド名を識別します。 DDS 中の各サブファイルごとに 1 つの SFILE キーワードを指定しなければなりません。

LIKEFILE キーワードを使用して、あるファイルに類似したディスプレイ・ファイルを定義する際に、親ファイルにサブファイルがある場合、新規ファイル内のサブファイルごとに SFILE キーワードを指定する必要があります。 これにより、サブファイルに相対レコード番号フィールドの名前を指定できるようになります。

TEMPLATE キーワードを指定してファイルが定義されている場合には、SFILE キーワードの rrnfield パラメーターは指定されません。

RRN フィールド・パラメーターによって識別されたフィールドには、READC または CHAIN 命令によって検索されたすべてのレコードの相対レコード番号が入れら れます。 このフィールドは、サブファイルへの WRITE 命令または ADD を使用する出力命令の場合に RPG IV が使用するレコード番号を指定するためにも使用されます。 RRN フィールド・パラメーターとして指定するフィールド名は、小数点以下の桁 数のない数値として定義されていなければなりません。 このフィールドには、ファイルの最大レコード番号が十分に入るだけの桁数がな ければなりません。 (IBM i Information Center の「データベースおよびファイル・システム」のカテゴリーの SFLSIZ キーワード の説明を参照してください。)

相対レコード番号の処理は、SFILE 定義の一部として暗黙に定義されます。 複数のサブファイルを定義する場合には、各サブファイルごとに SFILE キーワー ドの指定が必要です。

SFILE キーワードを SLN キーワードと一緒に使用しないでください。

修飾ファイルの場合、SFILE キーワードの最初のパラメーターで使用するレコード様式名は、必ず非修飾形式にしてください。

SLN(番号)

SLN (開始行番号) キーワードは、レコード様式を表示装置ファイルに書き出す位 置を決定します。 主要ファイル記述行の 36 から 42 桁目に WORKSTN、および 17 桁 目に C または O が入っていなければなりません。 ファイルの DDS では、1 つまたは複数のレコード様式にキーワード SLNO(*VAR) を指定しなければなりません。 SLN キーワードを指定した場合には、パラメーターは長さが 2 で小数点以下の桁 数のない数値フィールドとしてプログラム中で自動的に定義されます。

SLN キーワードを SFILE キーワードと一緒に使用しないでください。

SPECIAL{(*EXT | レコード長)}

SPECIAL キーワードは、装置タイプ・キーワードの 1 つです。 これは自由形式ファイル定義で使用されて、ファイル装置が SPECIAL で あることを示します。 これは最初のキーワードでなければなりません。

パラメーターは任意指定です。 デフォルトは *EXT です。

***EXT**
ファイルが外部記述であることを示すには、*EXT を指定します。 これはデフォルトです。

レコード長
ファイルがプログラム記述であることを示すには、レコード長 を 指定します。 レコード長は 1 から 99999 の間でなければなりません。 リテラルまたは 名前付き定数を指定できます。 名前付き定数である場合、ファイル定義ステートメントの前にその定数が定義されている必要があります。

STATIC

STATIC キーワードは、RPG ファイル制御情報が静的記憶域に保持されるように指示します。 サブプロシージャーへの呼び出しには、すべて同じ RPG ファイル制御情報が使用されます。 RPG ファイル制御情報では、サブプロシージャーへの呼び出し全体にわたって、その状態が保持されます。 サブプロシージャーの終了時にファイルが開いていると、次にサブプロシージャーを呼び出した時にもファイルは開いたままになっています。

STATIC キーワードが指定されない場合、RPG ファイル制御情報は自動記憶域に保持されます。 サブプロシージャーを呼び出すたびに、独自のバージョンの RPG ファイル制御情報が使用されます。 RPG ファイル制御情報は、サブプロシージャーを呼び出すたびに初期化されます。 サブプロシージャーの終了時にファイルが開いていても、サブプロシージャーが終了した時点でファイルはクローズされます。

STATIC キーワードに関する規則*:*

- STATIC キーワードは、サブプロシージャー内のファイル定義にしか指定できません。 グローバル定義で定義されたファイルに対して、STATIC キーワードが暗黙的に指定されます。
- STATIC キーワードで定義されたファイルは、CLOSE 命令によって明示的にクローズされるまで、あるいは活動化グループが終了するまで、開いたままになります。
- ファイル情報データ構造 (INFDS) がファイルに定義されている場合、データ構造に対して指定した STATIC キーワードは、ファイルに対して指定した STATIC キーワードと一致している必要があります。

```
P numInStock      b                   export
 * File "partInfo" is defined as STATIC.  The file will be
 * opened the first time the procedure is called, because
 * the USROPN keyword is not specified.
 * Since there is no CLOSE operation for the file, it
 * will remain open until the activation group ends.
FpartInfo  if   e           k disk    static
 * File "partErrs" is not defined as STATIC, and the USROPN
 * keyword is used.  The file will be opened by the OPEN
 * operation, and it will be closed automatically when the
 * procedure ends.
FpartErrs  o    e             disk    usropn

D numInStock      pi            10i 0
D    id_no                      10i 0 value
D partInfoDs      ds                  likerec(partRec:*input)
D partErrDs       ds                  likerec(errRec:*output)

 /free       // Search for the input value in the file
       chain id_no partRrec partInfoDs;
       if  not %found(partInfo);          // write a record to the partErrs file indicating
          // that the id_no record was not found.  The
          // file must be opened before the record can
          // be written, since the USROPN keyword was
          // specified.
          partErrDs.id_no = id_no;
          open partErrs;
          write errRec partErrDs;
          return -1; // unknown id
       endif;
       return partInfoDs.qty; /end-free
P numInStock      e
```

図 *123.* ファイル仕様書での *STATIC* キーワードの例

TEMPLATE

TEMPLATE キーワードは、このファイル定義をコンパイル時にのみ使用するよう指定します。 TEMPLATE キーワードで定義されたファイルは、プログラムには組み込まれません。 LIKEFILE キーワードを使用して、プログラム内に後で他のファイルを定義する際の基盤としてのみ、テンプレート・ファイルを使用できます。

TEMPLATE キーワードに関する規則*:*

- テンプレート・ファイルの RPG 記号名は、ファイル仕様書での LIKEFILE キーワードまたは定義仕様書での LIKEFILE キーワードの、いずれかのパラメーターとしてのみ使用できます。
- テンプレート・ファイルのレコード様式の RPG 記号名は、LIKEREC 定義キーワードのパラメーターとしてのみ使用できます。
- LIKEFILE 定義によって継承されないキーワードは、テンプレート・ファイルでは使用できません。

  詳しくは、378 ページの表 88 を参照してください。

TIMFMT(形式 {区切り記号 })

TIMFMT キーワードによって、プログラム記述ファイル のすべての 時刻フィールドについてデフォルトの外部時刻形式および (オプションの) デフォルトの区切り記号 を指定することができます。 このキーワードを指定したファイルが索引付きで、キー・フィールドが時刻の場 合には、指定した時刻の形式によってキー・フィールドのデフォルトの外部形式も指 定されます。

レコード・アドレス・ファイルの場合には、これによってレコード・ アドレス・ファイルから読み取られる時刻限界キーの外部時刻形式 が指定されます。

対応する入力仕様 (31 から 35 桁目) また出力仕様 (53 から 57 桁目) にフィールドの 時刻の形式/区切り記号を指定することによって、ファイル中の個別の入力また は出力時刻フィールドに異なる外部形式を指定することができます。

有効な形式および区切り記号については、276 ページの表 74 を参照してください。外部形式について詳しくは、247 ページの『内部形式および外部形式』 を参照してください。

USAGE(*INPUT *OUTPUT *UPDATE *DELETE)

USAGE キーワードは自由形式ファイル定義で 使用されて、ファイルの使用法を指定します。

USAGE キーワードは任意指定ですが、指定される場合は、少なくとも 1 つのパラメーターを指定する必要があります。

***INPUT**
入力

***OUTPUT**
output

***UPDATE**
入力および更新

***DELETE**
入力、更新、および削除

複数の使用法を暗黙に示す値もあります。 例えば、*UPDATE は入力と更新の両方を暗黙に示します。 次の USAGE キーワードは同じ意味です。

```
USAGE(*INPUT : *UPDATE)
USAGE(*UPDATE)
```

ファイルのデフォルトの使用法は、ファイルの装置タイプに基づきます。

- 装置タイプが DISK、SEQ、または SPECIAL のファイルの場合、 デフォルトの使用法は *INPUT です。
- 装置タイプが PRINTER のファイルの場合、デフォルトの使用法は *OUTPUT です。
- 装置タイプが WORKSTN のファイルの場合、デフォルトの使用法は *INPUT および *OUTPUT です。

USROPN

USROPN キーワードにより、ファイルはプログラムの初期化時にはオープンされ ません。 これによってファイルの最初のオープンの制御がプログラマーに与えられます。 ファイルは、演算仕様書の中で OPEN 命令を使用して明示的にオープンしなけれ ばなりません。 このキーワードは、1 次、2 次、テーブル、またはレコード・アドレス・ファイルとして指定された入力ファイル、あるいは 1P (1 ページ目) 標識によって条件付けされた出力ファイルの場合には有効でありません。

USROPN キーワードが必要なのは、プログラマー制御の最初のファイル・オープン の場合だけです。 たとえば、ファイルがオープンされていて、後から CLOSE 命令によってクローズ される場合には、プログラマーはファイル仕様書に USROPN キーワード指定せ ずに (OPEN 命令を使用して) ファイルを再オープンすることができます。

371 ページの『EXTIND(*INUx)』 も参照してください。

WORKSTN{(*EXT | レコード長)}

WORKSTN キーワードは、装置タイプ・キーワードの 1 つです。 これは自由形式ファイル定義で使用されて、ファイル装置が WORKSTN で あることを示します。 これは最初のキーワードでなければなりません。

パラメーターは任意指定です。 デフォルトは *EXT です。

***EXT**
ファイルが外部記述であることを示すには、*EXT を指定します。 これはデフォルト値。

レコード長
ファイルがプログラム記述であることを示すには、レコード長 を 指定します。 レコード長は 1 から 32766 の間でなければなりません。 リテラルまたは 名前付き定数を指定できます。 名前付き定数である場合、ファイル定義ステートメントの前にその定数が定義されている必要があります。

ファイル・タイプと処理方式

390 ページの表 89 には、各種のファイル・タイプおよび処理方式に対するファイル仕様書の 28 桁目、34 桁目、および 35 桁目に指定できる項目を示してあります。 ディスク・ファイルの処理方式には、次のものがあります。

- 相対レコード番号処理
- 連続処理
- キーによる順次処理
- キーによるランダム処理
- 限界内順次処理

表 *89. DISK* ファイルの処理方式

アクセス	メソッド	命令コード	28 桁目	34 桁目	35 桁目	説明
ランダム	RRN	CHAIN	ブランク	ブランク	ブランク	レコードの物理的な順序によるアクセス
			自由形式構文: (キーワードは不要です)			
順次	キー	READ READE READP READPE サイクル	ブランク	ブランク	I	キーによる順次アクセス
			自由形式構文: KEYED キーワード			
順次	限界値範囲内	READ READE READP READPE サイクル	L	A、P、G、D、T、Z、または F	I	レコード・アドレス限界ファイルによって制御されるキーによる順次アクセス
			自由形式構文: (自由形式ファイル定義にはサポートされません)			
順次	RRN	READ サイクル	ブランク	ブランク	T	レコード・アドレス・ファイルの RRN 番号に制限される順次アクセス
			自由形式構文: (自由形式ファイル定義にはサポートされません)			

各種のファイル処理方式の詳細については、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」の 「データベース・ファイルのアクセス」の章の 「ディスク・ファイルの処理方式」の項を参照してください。

定義仕様書

定義仕様書は、次のものを定義するために使用することができます。

- 独立フィールド
- 名前付き定数

- データ構造およびそのサブフィールド
- プロトタイプ
- プロシージャー・インターフェース
- プロトタイプ・パラメーター

データ構造、定数、プロトタイプ、およびプロシージャー・インターフェースについて詳しくは、199 ページの『データおよびプロトタイプの定義』も参照してください。データ・タイプおよびデータ形式について詳しくは、246 ページの『データ・タイプおよびデータ形式』も参照してください。

配列およびテーブルは、データ構造サブフィールドかまたは独立フィールドとし て定義することができます。配列およびテーブルの定義および使用について詳しくは、229 ページの『配列およびテーブルの使用』も参照してください。

定義仕様書は、モジュールまたはプログラム内の 2 つの場所、すなわち、メイン・ソース・ セクションおよびサブプロシージャーに入れることができます。 メイン・ソース・セクションでは、すべてのグローバル定義を作成します。 サブプロシージャーでは、プロトタイプに必要なプロシージャー・インターフェ ースおよびそのパラメーターを定義します。 プロトタイプ・プロシージャーの処理時に必要とされるすべてのローカル・デー タ項目も定義されます。 プロトタイプ・プロシージャー内の定義はすべてローカル定義です。 それらは、(サイクル・メイン・プロシージャーも含め) 他のどのプロシージャーにも認識されません。有効範囲について詳しくは、95 ページの『定義の有効範囲』を参照してください。

キーワード・フィールドで、キーワードに対するパラメーターとして組み込み関数 (BIF) を 使用できます。定義仕様書でこれを 使用できるのは、すべての引数の値がコンパイル時に分かってい る場合だけです。 定義仕様書のキーワード DIM、OCCURS、OVERLAY、および PERRCD のパラメーターと して指定する場合には、BIF のすべての引数がプログラムの早い段階で定義され ていなければなりません。組み込み関数の使用について詳しくは、551 ページの『組み込み関数』を参照してください。

自由形式の定義ステートメント

自由形式のデータ定義ステートメントは、宣言命令コードの 1 つで 始まり、その後に名前が続くか、項目に名前がない場合は *N が続き、 さらにその後にキーワードが続き、最後はセミコロンで終わります。

宣言命令コードは次のとおりです。

DCL-C
名前付き定数を定義します

DCL-DS
データ構造を定義します

END-DS
データ構造を終了します

{DCL-PARM}
パラメーターを定義します

DCL-PI
プロシージャー・インターフェースを定義します

END-PI
プロシージャー・インターフェースを終了します

DCL-PR
プロトタイプを定義します

END-PR
プロトタイプを終了します

DCL-S
独立フィールドを定義します

{DCL-SUBF}
サブフィールドを定義します

必要な場合、定義を複数行に分割してもかまいません。 以下は等価な定義です。

1. 定義は 1 行に指定されています。

2. 定義は複数行に分割されています。
3. 名前は 2 行に分割されていて、キーワードは名前の 2 つ目の部分と同じ行に指定されています。
4. 名前は 2 行に分割されていて、キーワードは名前の 2 つ目の部分の次の行に指定されています。

注 : 名前の最後の部分の後には省略符号は使用されません。 自由形式の仕様書では、省略符号は、名前が 2 つの部分に分かれていて別々の行に指定されている場合に、それら 2 つの部分を 結合するためにのみ使用されます。 名前をステートメントの残りの部分に結合するためには使用されません。

```
DCL-S abcdefghij CHAR(10);  1
```

```
   DCL-S                    2
         abcdefghij
 CHAR
    (
     10
    )
     ;
```

```
DCL-S abcde...              3
         fghij CHAR(10);
```

```
DCL-S abcde...              4
         fghij
               CHAR(10);
```

自由形式のデータ定義ステートメントの内部で使用を許可されている指示は、 /IF、/ELSEIF、/ELSE、および /ENDIF のみです。

```
DCL-S G_Working_Date DATE(*ISO)
        /IF DEFINED(main_module)
          EXPORT INZ(*SYS)
        /ELSE
          IMPORT
        /ENDIF
          ;
```

注 : 項目の名前が、自由形式演算内で使用を許可されている命令コードのいずれかと同じである場合を除いて、 DCL-PARM または DCL-SUBF は任意指定です。

LIKEREC および LIKEDS キーワードなしで定義されるデータ構造 が自由形式ステートメントで始まっているか、または、 サブフィールドのどれかが自由形式ステートメントで指定されている場合、 そのデータ構造は END-DS ステートメントで終わる必要があります。

```
 DCL-DS ds1;
    subf1 CHAR(10);
 END-DS;

D ds2              DS
    subf2 CHAR(10);
 END-DS;

 DCL-DS ds3;
D  subf3                        10a
 END-DS;
```

同様に、プロトタイプまたはプロシージャー・インターフェース が自由形式ステートメントで始まっているか、パラメーターのどれかが自由形式ステートメント で指定されている場合、そのプロトタイプまたはプロシージャー・インターフェース は END-PR または END-PI ステートメントで終わる必要があります。

```
 DCL-PR pr1;
    subf1 CHAR(10);
 END-PR;

D pr2              PR
    parm2 CHAR(10);
 END-PR;

 DCL-PI pi3;
D  parm3                        10a
 END-PI;
```

サブフィールドのないデータ構造の場合、END-DS を DCL-DS ステートメントの 一部として、セミコロンの直前に指定できます。

```
DCL-DS ds1 LEN(100) END-DS;
```

同様に、プロシージャー・インターフェースまたはプロトタイプにパラメーターがない場合、 END-PI または END-PR を DCL-PI または DCL-PR ステートメントの 一部として、セミコロンの直前に指定できます。

```
DCL-PR pr1 INT(10) END-PR;
```

データ・タイプ・キーワード

- 417 ページの『BINDEC(桁数 {: 小数点以下の桁数})』
- 419 ページの『CHAR(長さ)』
- 420 ページの『DATE{(形式 {区切り記号 }) }』
- 440 ページの『FLOAT(バイト数)』
- 440 ページの『GRAPH(長さ)』
- 442 ページの『IND』
- 441 ページの『INT(桁数)』
- 453 ページの『OBJECT{(*JAVA:クラス名)}』
- 473 ページの『PACKED(桁数 {: 小数点以下の桁数})』
- 473 ページの『POINTER{(*PROC)}』
- 484 ページの『TIME{(形式 {区切り記号 }) }』

- 484 ページの『TIMESTAMP { (秒の小数部の桁数) }』
- 485 ページの『UCS2(length)』
- 486 ページの『UNS(桁数)』
- 486 ページの『VARCHAR(長さ {:2 | 4})』
- 487 ページの『VARGRAPH(長さ {:2 | 4})』
- 487 ページの『VARUCS2(length {:2 | 4})』
- 488 ページの『ZONED(桁数 {: 小数点以下の桁数})』

固定形式定義と自由形式定義でのキーワード使用の相違点

表 *90.* 自由形式定義でサポートされないキーワードに対応する自由形式での指定方法

キーワード	自由形式での指定方法
CLASS	クラス情報は OBJECT キーワードのパラメーターとして指定されます。
DATFMT	日付形式は DATE キーワードのパラメーターとして指定されます。
FROMFILE	FROMFILE キーワードを指定したい場合は、固定形式定義を使用する必要があります。
PACKEVEN	PACKEVEN キーワードは、自由形式でのサブフィールド定義にはサポートされていない開始位置および終了位置の使用に関連しています。
PROCPTR	プロシージャー・ポインター・データ・タイプは POINTER(*PROC) として指定されます。
TIMFMT	時刻形式は TIME キーワードのパラメーターとして指定されます。
TOFILE	TOFILE キーワードを指定したい場合は、固定形式定義を使用する必要があります。
VARYING	可変長フィールドは、VARCHAR、VARGRAPH、および VARUCS2 キーワードを使用して定義されます。

表 *91.* 自由形式定義と固定形式定義で使用方法が異なるキーワード

キーワード	差異
DTAARA	427 ページの『データ構造の自由形式 DTAARA キーワード』と 426 ページの『フィールドまたはサブフィールドの自由形式 DTAARA キーワード』を参照してください。
EXPORT	自由形式定義では、*DCLCASE を使用して外部名を指定できます。
EXTFLD	自由形式定義では、 外部フィールド名は文字リテラルとして、または、文字リテラルを表す前もって定義された名前付き定数 として指定される必要があります。
EXTNAME	自由形式定義では、外部ファイルおよび外部レコード形式 は、文字リテラルとして、または、文字リテラルを表す前もって定義された名前付き定数として 指定される必要があります。
EXTPROC	自由形式定義では、*DCLCASE を使用して外部名を指定できます。
IMPORT	自由形式定義では、*DCLCASE を使用して外部名を指定できます。
LIKE	自由形式定義では、長さ調整は LIKE キーワードの 2 番目のパラメーターとして指定されます。

表 *91.* 自由形式定義と固定形式定義で使用方法が異なるキーワード *(*続き*)*	
キーワード	**差異**
OVERLAY	自由形式定義では、OVERLAY キーワードのパラメーターをデータ構造の名前にすることはできません。 データ構造内のサブフィールドの開始位置を明示的に指定するには、POS キーワードを使用します。

自由形式の名前付き定数の定義

自由形式での名前付き定数の定義は、DCL-C で始まります。その後に定数の名前が 続き、さらにその後に、直接指定された値、または CONST キーワードを使用して指定された 値が続き、最後はセミコロンで終わります。

自由形式の名前付き定数の定義例

- 次の例は、*CON_1* および *CON_2* という名前の 2 つの定数を示しています。 一方の例では CONST キーワードが使用されていて、他方の例では定数の値が直接定義されています。 CONST キーワードを指定する場合と指定しない場合とで意味の違いはありません。

```
DCL-C CON_1 CONST(1);
DCL-C CON_2 2;
```

- 次の例の名前付き定数 *array_total_size* は、組み込み関数の値を使用して定義されています。

```
DCL-S array CHAR(25) DIM(100);
DCL-C array_total_size
         %SIZE(array:*ALL);
```

自由形式の独立フィールド定義

自由形式の独立フィールドは DCL-S で始まり、その後にフィールドの 名前が続き、さらにその後にキーワードが続き、最後にセミコロンで終わります。

データ・タイプ・キーワードを指定する場合は、それが最初のキーワードである必要があります。

自由形式の独立フィールド定義の例

- *limit* という名前の 5 桁のパック 10 進数フィールドが定義されています。 小数点以下の桁数はゼロであり、100 に初期化されます。 PACKED キーワードは、データ・タイプ・キーワードであるため、最初に指定される必要があります。

```
DCL-S limit PACKED(5) INZ(100);
```

- *num* という名前のフィールドが LIKE キーワードを使用して定義されており、0 に初期化されます。 データ・タイプ・キーワードはありません。 LIKE キーワードは どこにあってもかまいません。

```
DCL-S num INZ(0)
          LIKE(limit);
```

固定形式の独立フィールド記入項目に対応する自由形式でのコーディング

表 92. 固定形式の独立フィールド記入項目に対応する自由形式での指定方法

固定形式の記入項目	自由形式での指定方法
長さ	データ・タイプ・キーワードの長さパラメーター または LIKE キーワードの長さ調整パラメーター
内部データ・タイプ	データ・タイプ・キーワード
小数点以下の桁数	データ・タイプ・キーワードの小数点以下の桁数パラメーター

自由形式のデータ構造定義

データ構造は DCL-DS ステートメントで始まります。

DCL-DS ステートメントに LIKEDS キーワードも LIKEREC キーワードも 指定されていない場合、DCL-DS ステートメント の後にはゼロ個またはそれ以上のサブフィールドが続き、 その後に END-DS ステートメントが続きます。

DCL-DS ステートメント

最初のステートメントは DCL-DS で始まり、 その後にデータ構造の名前が続くか、データ構造に名前がない場合は *N が続き、 さらにその後にキーワードが続き、最後はセミコロンで終わります。

サブフィールド

注 : LIKEDS または LIKEREC キーワードを使用して定義された データ構造には、サブフィールドは指定されません。

399 ページの『自由形式のサブフィールド定義』を参照してください。

END-DS ステートメント

- LIKEDS または LIKEREC キーワードを使用して定義された データ構造には、END-DS は指定されません。
- END-DS の後にデータ構造の名前を続けることができます。

```
DCL-DS custInfo QUALIFIED;
   id INT(10);
   name VARCHAR(50);
   city VARCHAR(50);
   orders LIKEDS(order_t) DIM(100);
   numOrders INT(10);
END-DS custInfo;
```

- データ構造に名前がない場合、 END-DS はオペランドなしで指定する必要があります。
- サブフィールドがない場合、END-DS を DCL-DS ステートメント の一部として、キーワードに続けて、セミコロンの前に指定できます。 この場合、END-DS の後にデータ構造の名前を指定することはできません。

```
DCL-DS prtDs LEN(132) END-DS;
```

外部記述データ構造

EXT キーワード または EXTNAME キーワード のいずれかを最初のキーワードとして指定します。

```
DCL-DS myfile EXT END-DS;
DCL-DS extds1 EXTNAME('MYFILE') END-DS;
```

EXT キーワードを指定する場合、 その後にあるキーワードの 1 つとして EXTNAME キーワードも指定できます。

```
DCL-DS extds2 EXT INZ(*EXTDFT) EXTNAME('MYFILE') END-DS;
```

プログラム状況データ構造 (PSDS)

PSDS を定義するには、PSDS キーワードを指定します。 *STATUS などの事前定義サブフィールドには、データ・タイプ・キーワードの代わりに予約語を指定します。

```
DCL-DS pgm_stat PSDS;
   status *STATUS;
   routine *ROUTINE;
   library CHAR(10) POS(81);
END-DS;
```

PSDS のすべての事前定義サブフィールドのリストについては、163 ページの『プログラム状況データ構造』を参照してください。

ファイル情報データ構造 (INFDS)

INFDS の名前は、 ファイル定義の INFDS キーワード のパラメーターとして指定されます。

*STATUS などの事前定義サブフィールドには、 データ・タイプ・キーワードの代わりに予約語を指定します。

```
DCL-F myfile DISK(*EXT) INFDS(myfileInfo);
DCL-DS myfileInfo;
   status *STATUS;
   opcode *OPCODE;
   msgid CHAR(7) POS(46);
END-DS;
```

INFDS のすべての事前定義サブフィールドのリストについては、147 ページの『ファイル・フィードバック情報』を参照してください。

ファイル情報データ構造

DTAARA キーワードに *AUTO パラメーターを指定します。

```
DCL-DS dtaara_ds DTAARA(*AUTO) LEN(100);
   name CHAR(10);
END-DS;
```

自由形式のデータ構造の例

1. LIKEDS および LIKEREC を使用して定義されたデータ構造が、END-DS のない 1 つのステートメント としてコーディングされています。

```
DCL-DS info LIKEDS(info_T);
DCL-DS inputDs LIKEREC(custFmt : *INPUT);
```

2. 3 つのサブフィールドがあるデータ構造 *cust_info*。 END-DS ステートメントは名前なしで指定されています。

```
DCL-DS cust_info;
   id INT(10);
   name VARCHAR(25);
   startDate DATE(*ISO);
END-DS;
```

3. 一部のサブフィールドの定義に DCL-SUBF が使用されているデータ構造。
   - サブフィールド *select* は、自由形式演算内で使用を許可されている命令コードと同じ名前です。 このサブフィールドには DCL-SUBF が必要です。 542 ページの表 103 を参照してください。
   - サブフィールド *name* は、命令コードと同じ名前ではないため、DCL-SUBF は不要です。
   - サブフィールド *address* は、命令コードと同じ名前ではないため、 DCL-SUBF は必要ありませんが、あっても有効です。

```
DCL-DS *N;
   DCL-SUBF select CHAR(10);
   name CHAR(10);
   DCL-SUBF address CHAR(25);
END-DS;
```

4. データ構造 *order_info*。 END-DS ステートメントが名前とともに指定されています。

```
DCL-DS order_info QUALIFIED;
   part LIKEDS(part_info_T);
   quantity INT(10);
   unit_price PACKED(9 : 2);
   discount PACKED(7 : 2);
END-DS order_info;
```

5. 追加のサブフィールドがある外部記述データ構造。

```
DCL-DS cust_info EXTNAME('CUSTFILE');
   num_orders INT(10);
   earliest_order DATE(*ISO);
   latest_order DATE(*ISO);
END-DS;
```

6. 外部ファイル 'CUSTINFO' の名前と同じ名前の外部記述データ構造。 このデータ構造には、EXTFLD キーワードで示される外部サブフィールドがあります。 EXTFLD キーワードは、サブフィールド名が外部名と同じである場合はパラメーターなしで指定されます。

```
DCL-DS custInfo EXT;
   cust_name EXTFLD('CSTNAM');
   id_no EXTFLD INZ(0);
END-DS;
```

7. ネストされたデータ構造サブフィールドのデータ構造。

```
DCL-DS employeeInfo QUALIFIED;
   id_no int(10);
   DCL-DS name;
      last VARCHAR(25);
      first VARCHAR(25);
      initial CHAR(1);
   END-DS;
   type VARCHAR(10);
END-DS;

employeeInfo.id_no = 12345;
employeeInfo.name.first = 'Robin';
employeeInfo.name.last = 'Hood';
```

8. 名前の付いていないデータ構造。

```
DCL-DS *N;
   string CHAR(100);
   string_array CHAR(1) DIM(100) OVERLAY(string);
END-DS;
```

固定形式のデータ構造記入項目に対応する自由形式でのコーディング

表 *93.* 固定形式のデータ構造記入項目に対応する自由形式での指定方法

固定形式の記入項目	自由形式での指定方法
外部	最初のキーワードとして指定された EXT キーワード または EXTNAME キーワード
データ構造タイプ	PSDS キーワード、 または、DTAARA キーワードの *AUTO パラメーター
長さ	LEN キーワード

自由形式のサブフィールド定義

自由形式サブフィールド定義は、サブフィールド名で始まるか、 または、サブフィールド名が後に指定されている DCL-SUBF または DCL-DS 命令コードで始まります。

サブフィールド名が自由形式演算内で使用を許可されている命令コードのいずれかと同じである場合、DCL-SUBF 命令コードを 指定する必要があります。542 ページの表 103 を参照してください。

DCL-DS 命令コードは、ネストされたデータ構造サブフィールドを定義するために使用します。

サブフィールド名の後にはキーワード、および最後にセミコロンが続きます。

プログラム記述サブフィールド

サブフィールドに名前がない場合は、サブフィールド名に *N を 指定します。

データ・タイプ・キーワードを指定する場合は、それが最初のキーワードである必要があります。

サブフィールドがプログラム状況データ構造 またはファイル情報データ構造内の事前定義サブフィールドである場合、データ・タイプ・キーワードの代わりにサブフィールド予約語を指定します。

自由形式定義では、OVERLAY キーワードは、あるサブフィールドを別のサブフィールドにオーバーレイするためにのみ 使用できます。 サブフィールドがデータ構造内で次に使用可能な場所に置かれるようにしたくない場合は、 POS キーワード を使用して開始位置を指定します。

外部記述サブフィールド

外部記述サブフィールドの最初のキーワードは EXTFLD キーワードでなければなりません。 サブフィールド名がサブフィールドの外部名と同じである場合、 EXTFLD キーワードのパラメーターは省略可能です。

プログラム記述サブフィールド

サブフィールドに名前がない場合は、サブフィールド名に *N を 指定します。

データ・タイプ・キーワードを指定する場合は、それが最初のキーワードである必要があります。

サブフィールドがプログラム状況データ構造 またはファイル情報データ構造内の事前定義サブフィールドである場合、データ・タイプ・キーワードの代わりにサブフィールド予約語を指定します。

自由形式定義では、OVERLAY キーワードは、あるサブフィールドを別のサブフィールドにオーバーレイするためにのみ 使用できます。 サブフィールドがデータ構造内で次に使用可能な場所に置かれるようにしたくない場合は、 POS キーワード を使用して開始位置を指定します。

ネストされたデータ構造サブフィールド

ネストされたデータ構造のサブフィールドとしてデータ構造を定義できるのは、その親データ構造が自由形式構文で定義されており、修飾されている場合です。 ネストされたデータ構造のサブフィールドも、自由形式構文で定義する必要があります。

ネストされたデータ構造のサブフィールドは自動的に修飾されます。 以下の例では、サブフィールド *address* が、ネストされたデータ構造のサブフィールドとして定義されています。 DCL-DS ステートメントで開始して (1)、サブフィールドが続き、その後に END-DS ステートメントが続きます (2)。*address* サブフィールドのサブフィールドは、*person.address* で修飾されています (3).

```
DCL-DS person QUALIFIED;
   name VARCHAR(25);
   DCL-DS address; 1
      num int(5);
      street VARCHAR(25);
      city VARCHAR(25);
      province VARCHAR(25);
      postcode VARCHAR(6);
   END-DS address; 2
   age int(5);
END-DS person;

person.address.street = 'Elm Ave.'; 3
```

ネストされたデータ構造は配列で設定することができます。 以下の例では、サブフィールド *person* (1) が、データ構造 *family* 内にネストされたデータ構造配列サブフィールドとして定義され、サブフィールド *pets* (2) が、サブフィールド *person* 内のネストされた別の配列となっています。*pets* サブフィールドのサブフィールドは、*family.person(index).pets(index)* で修飾されています (3).

```
DCL-DS family QUALIFIED;
   num int(5);
   DCL-DS person DIM(10);  1
      name VARCHAR(25);
      age int(5);
      numPets int(5);
      DCL-DS pets DIM(5);  2
         name VARCHAR(25);
         type VARCHAR(25);
      END-DS pets;
   END-DS persom;
END-DS;

family.person(1).numPets = 1;
family.person(1).pets(1).type = 'fish'; 3
```

サブフィールドの例

自由形式サブフィールド定義の例については、396 ページの『自由形式のデータ構造定義』を参照してください。

固定形式のサブフィールド記入項目に対応する自由形式でのコーディング

表 *94.* 固定形式の外部記述サブフィールド記入項目に対応する自由形式での指定方法	
固定形式の記入項目	**自由形式での指定方法**
外部	最初のキーワードとして指定された EXTFLD キーワード

表 *95.* 固定形式のプログラム記述サブフィールド記入項目に対応する自由形式での指定方法	
固定形式の記入項目	**自由形式での指定方法**
数値から	POS キーワード
*STATUS などの特殊値から	データ・タイプ・キーワードの代わりに特殊値を指定します
終了位置	完全に等価のものはありません。 サブフィールドの終了位置は、POS キーワード と項目の長さを組み合わせたものから決められます。
「長さ」、「内部データ・タイプ」、 および「小数点以下の桁数」記入項目	396 ページの『固定形式の独立フィールド記入項目に対応する自由形式でのコーディング』を参照してください。

自由形式のプロトタイプ定義

プロトタイプは DCL-PR ステートメントで始まります。

DCL-PR ステートメントの後には、ゼロ個またはそれ以上のパラメーター が続き、その後に END-PR ステートメントが続きます。

プロトタイプを開始する DCL-PR ステートメント

最初のステートメントは DCL-PR で始まり、 その後にプロトタイプの名前が続き、 さらにその後にキーワードが続き、最後はセミコロンで終わります。

プロトタイプ・パラメーター

405 ページの『自由形式のパラメーター定義』を参照してください。

プロトタイプを終了する END-PR ステートメント

- END-PR の後にプロトタイプの名前を続けることができます。
- パラメーターがない場合、END-PR を DCL-PR ステートメント の一部として、キーワードに続けて、セミコロンの前に指定できます。 この場合、END-PR の後にプロトタイプの名前を指定することはできません。

自由形式のプロトタイプの例

1. パラメーターが 1 つ指定された、プログラム用のプロトタイプ。 プログラムの外部名は「MYPGM」です。

```
DCL-PR myPgm EXTPGM; 1
   name CHAR(10) CONST;
END-PR;
```

2. 3 個のパラメーターがあるプロトタイプ *addNewOrder*。 END-PR ステートメントは名前なしで指定されています。

```
DCL-PR addNewOrder;
   id INT(10) CONST;
   quantity INT(10) CONST;
   price PACKED(7 : 2) CONST;
END-PR; 2
```

3. END-PR ステートメントに名前が指定されています。

```
DCL-PR addNewOrder;
   id INT(10) CONST;
   quantity INT(10) CONST;
   price PACKED(7 : 2) CONST;
END-PR addNewOrder; 3
```

4. プロトタイプにはパラメーターはないため、END-PR は DCL-PR ステートメントの一部として指定されています。

```
  DCL-PR getCurrentUser CHAR(10) END-PR; 4
```

5. 一部のサブフィールドを DCL-PARM を使用して定義しているプロトタイプ。

   a. パラメーター *select* は、 自由形式演算内で使用を許可されている命令コードと同じ名前です。 このパラメーターには DCL-PARM が必要です。 542 ページの表 103 を参照してください。

   b. パラメーター *name* は、命令コードと同じ名前ではないため、DCL-PARM は不要です。

   c. パラメーター *address* は、命令コードと同じ名前ではないため、 DCL-PARM は必要ありませんが、あっても有効です。

```
DCL-PR myProc;
   DCL-PARM select CHAR(10); 5a
   name CHAR(10); 5b
   DCL-PARM address CHAR(25); 5c
END-PR;
```

6. その他の例については、439 ページの『外部名としての *DCLCASE の指定』を参照してください。

固定形式のプロトタイプ記入項目に対応する自由形式でのコーディング

表 96. 固定形式のプロトタイプ記入項目に対応する自由形式での指定方法

固定形式の記入項目	自由形式での指定方法
「長さ」、「内部データ・タイプ」、 および「小数点以下の桁数」記入項目	396 ページの『固定形式の独立フィールド記入項目に対応する自由形式でのコーディング』を参照してください。

自由形式プロシージャー・インターフェース定義

プロシージャー・インターフェースは DCL-PI ステートメントで始まります。

DCL-PI ステートメントの後には、ゼロ個またはそれ以上のパラメーター が続き、その後に END-PI ステートメントが続きます。

プロシージャー・インターフェースを始めるための DCL-PI ステートメント

最初のステートメントは DCL-PI で始まり、その後にプロシージャーの 名前が続くか、プロシージャーの名前を繰り返したくない場合は *N が続き、さらにその後にキーワードが続き、最後はセミコロンで終わります。

- サイクル・メイン・プロシージャーのプロトタイプを指定しない場合、 プロシージャー・インターフェースの名前として *N を使用します。
- リニア・メイン・プロシージャーのプロトタイプを指定しない場合、 プロシージャー・インターフェースに対して EXTPGM キーワードを パラメーターなしで指定することができます。

プロシージャー・インターフェースのパラメーター

405 ページの『自由形式のパラメーター定義』を参照してください。

プロシージャー・インターフェースを終了するための END-PI ステートメント

- END-PI の後にプロシージャーの名前を続けることができます。
- パラメーターがない場合、END-PI を DCL-PI ステートメント の一部として、キーワードに続けて、セミコロンの前に指定できます。 この場合、END-PI の後にプロシージャー名を指定することはできません。

自由形式のプロシージャー・インターフェースの例

1. プロトタイプのないサイクル・メイン・プロシージャーのプロシージャー・インターフェース。プロシージャー・インターフェース名に *N が指定されています。 このプロシージャー・インターフェース内には 1 つのパラメーター *name* が指定されています。 (この例は、サイクル・メイン・プロシージャーが指定されている完全なプログラムを示しています)

```
CTL-OPT OPTION(*SRCSTMT);

DCL-PI *N; 1
   name CHAR(10) CONST;
END-PI;

DSPLY ('Hello ' + name);
RETURN;
```

2. プロトタイプのないリニア・メイン・プロシージャーのプロシージャー・インターフェース。 EXTPGM キーワードがパラメーターなしで指定されています。 (この例は、リニア・メイン・プロシージャーが指定されている完全なプログラムを示しています)

```
CTL-OPT MAIN(sayHello) OPTION(*SRCSTMT);

DCL-PROC sayHello;
   DCL-PI *N EXTPGM; 2
      name CHAR(10) CONST;
   END-PI;

   DSPLY ('Hello ' + name);
END-PROC;
```

3. 3 つのパラメーターがあるプロシージャー・インターフェース。 END-PI ステートメントは名前なしで指定されています。

```
DCL-PROC addNewOrder;
   DCL-PI *N;
      id INT(10) VALUE;
      quantity INT(10) CONST;
      price PACKED(7 : 2) CONST;
   END-PI; 3
   ...
END-PROC;
```

4. END-PI ステートメントに名前が指定されています。

```
DCL-PROC addNewOrder;
   DCL-PI *N;
      id INT(10) CONST;
      quantity INT(10) CONST;
      price PACKED(7 : 2) CONST;
   END-PI addNewOrder; 4
   ...
END-PROC;
```

5. DCL-PI ステートメントの一部として END-PI が指定されています。 (この例は、完全なプロシージャーを示しています)

```
DCL-PROC getCurrentUser;
   DCL-PI *N CHAR(10) END-PI; 5

   DCL-S currentUser CHAR(10) INZ(*USER);
   RETURN currentUser;
END-PROC;
```

6. DCL-PARM を使用して一部のサブフィールドを定義しているプロシージャー・インターフェース。

   a. パラメーター *select* は、 自由形式演算内で使用を許可されている命令コードと同じ名前です。 このパラメーターには DCL-PARM が必要です。 542 ページの表 103 を参照してください。

   b. パラメーター *name* は、命令コードと同じ名前ではないため、DCL-PARM は不要です。

   c. パラメーター *address* は、命令コードと同じ名前ではないため、 DCL-PARM は必要ありませんが、あっても有効です。

```
DCL-PI *N;
   DCL-PARM select CHAR(10); 6a
   name CHAR(10); 6b
   DCL-PARM address CHAR(25); 6c
END-PI;
```

7. その他の例については、439 ページの『外部名としての *DCLCASE の指定』を参照してください。

固定形式のプロシージャー・インターフェース記入項目に対応する自由形式でのコーディング

表 97. 固定形式のプロシージャー・インターフェース記入項目に対応する自由形式での指定方法	
固定形式の記入項目	**自由形式での指定方法**
「長さ」、「内部データ・タイプ」、 および「小数点以下の桁数」記入項目	396 ページの『固定形式の独立フィールド記入項目に対応する自由形式でのコーディング』を参照してください。

自由形式のパラメーター定義

自由形式パラメーター定義は、パラメーター名で始まるか、 または、パラメーター名が後に指定されている DCL-PARM 命令コードで始まります。

パラメーター名が自由形式演算内で使用を許可されている命令コードのいずれかと同じである場合、DCL-PARM 命令コードを 指定する必要があります。542 ページの表 103 を参照してください。

パラメーター名の後にキーワードが続き、最後にセミコロンがあります。

プロトタイプ定義内でパラメーターに名前を指定したくない場合、 パラメーター名に *N を指定します。

データ・タイプ・キーワードを指定する場合は、それが最初のキーワードである必要があります。

自由形式パラメーター定義の例については、403 ページの『自由形式プロシージャー・インターフェース定義』および 401 ページの『自由形式のプロトタイプ定義』を参照してください。

固定形式のパラメーター記入項目に対応する自由形式でのコーディング

表 98. 固定形式のパラメーター項目に対応する自由形式での指定方法	
固定形式の記入項目	**自由形式での指定方法**
「長さ」、「内部データ・タイプ」、 および「小数点以下の桁数」記入項目	396 ページの『固定形式の独立フィールド記入項目に対応する自由形式でのコーディング』を参照してください。

従来型の定義仕様書ステートメント

定義仕様書の一般的なレイアウトは次のとおりです。

- 定義仕様書コード (D) は 6 桁目に記入されます。
- 仕様書の注記以外の部分は 7 から 80 桁目です。
  - 固定形式の記入項目は 7 から 42 桁目です。
  - キーワードの記入項目は 44 から 80 桁目です。
- 仕様書の注記部分は 81 から 100 桁目です。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+...
9 ...+... 10
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++Comments++++++++++
++
```

図 *124.* 定義仕様書のレイアウト

定義仕様書のキーワード継続記入行

自由形式構文	自由形式ステートメントで使用可能な桁については、307 ページの『自由形式ステートメント』を参照してください。

キーワードに追加のスペースが必要な場合には、 次のようにキーワード・フィールドを後続の行に継続させることができます。

- 継続記入行の 6 桁目には D が入っていなければなりません。
- 継続記入行の 7 から 43 桁目はブランクでなければなりません。
- 指定は 44 桁目以降から継続されます。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+...
9 ...+... 10
D.....................................Keywords++++++++++++++++++++++++++++++Comments++++++++++
++
```

図 *125.* 定義仕様書のキーワード 継続記入行のレイアウト

定義仕様書の継続名前行

自由形式構文	名前の最後の部分に省略符号 (...) をコーディングしないでください。391 ページの『自由形式の定義ステートメント』を参照してください。 自由形式ステートメントで使用可能な桁については、307 ページの『自由形式ステートメント』を参照してください。

15 文字までの名前は、定義仕様書の名前記入項目に 指定することができるので継続は必要ありません。 部分名の終わりに省略符号 (...) をコーディングすると、任意の名前を (15 文字以下の名前であっても) 複数行に継続できます。 名前定義は、次の部分から構成されます。

1. ゼロまたはそれ以上の継続名前行。 継続名前行は、その記入項目 中の最後の非ブランク文字として省略記号を持つものとして識別されます。 名前 は、7 から 21 桁目の中で開始する必要があり、77 桁目まで (80 桁目で 終了する省略記号を付けて) の任意の位置で終了することができます。 名前の開始と 省略記号の間にはブランクを挿入することはできません。 これらの条件のいずれかが真とならない場合、その行は主要定義行で あると解析されます。
2. 名前、定義属性、およびキーワードを含む 1 つの主要定義行。 継続名前行がコーディングされた場合、主要定義行の名前記入項目 はブランクのままになる場合があります。
3. ゼロまたはそれ以上のキーワード継続記入行。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+...
9 ...+... 10
DContinuedName+++++++++++++++++++++++++++++++++++++++++++++++++++++++++Comments++++++++++
++
```

図 *126.* 定義仕様書の 継続名前行の レイアウト

6 桁目 (仕様書コード)

自由形式構文	391 ページの『自由形式の定義ステートメント』を参照してください。

定義仕様書にはこの桁に D を入れなければなりません。

7 から 21 桁目 (名前)

自由形式構文	391 ページの『自由形式の定義ステートメント』を参照してください。

記入
説明

名前
定義している項目の名前。

ブランク
データ構造サブフィールド定義のフィラー・フィールド、またはデータ構造 定義の名前なしデータ構造を指定します。

RPG IV 記号名に関する通常の規則が適用されます。予約語を使用することはできません (73 ページの『記号名』を参照)。 名前は、指定されたスペースのどの桁からでも始めることができます。 したがって、字下げを使用してデータ構造中のデータの形状を指示することができます。

継続名前行の場合、名前は、継続名前行の 7 から 80 桁目、および主要定義行 の 7 から 21 桁目に指定されます。 従来の名前の定義と同様に、大文字小文字の区別は問題になりません。

外部記述サブフィールドの場合には、ここで指定された名前が EXTFLD キーワー ドに指定された外部サブフィールド名に置き換わります。

プロトタイプ・パラメーターの定義の場合には、この名前記入項目は任意指定で す。 名前を指定しても、無視されます。 (プロトタイプ・パラメーターは、24 から 25 桁目にブランクとその後に PR の指定 または別のプロトタイプ・パラメーターの定義が続く定義仕様書です。)

ヒント:

プロトタイプを定義していて、7 から 21 桁目に指定された名前が プロシージャーの外部名として役立たない場合には、EXTPROC キーワード を使用して有効な外部名を指定してください。例えば、ILE C で書かれたプロシージャーのプロトタイプを定義している場合には、小文字の外部名が必要になることがあります。

22 桁目 (外部記述)

自由形式構文	データ構造の場合、 EXT キーワード または EXTNAME キーワードを 最初のキーワードとして指定します。 サブフィールドの場合、EXTFLD キーワード を最初のキーワードとして使用します。

この桁は、データ構造またはデータ構造サブフィールドを外部記述として識別す るために使用されます。データ構造またはサブフィールドをこの仕様で定義しているのではない場合には 、このフィールドをブランクのままにしておかなければなりません。

記入
データ構造の説明

E
データ構造を外部記述として識別します。サブフィールド定義は外部で作成 されます。 EXTNAME キーワードが指定されていない場合には、7 から 21 桁目にはデータ構造定 義が入っている外部記述ファイルの名前を入れなければなりません。

ブランク
プログラム記述の場合です。このデータ構造のサブフィールド定義がこの指 定の後に続きます。

記入
サブフィールドの説明

E
データ構造サブフィールドを外部記述として識別します。 外部記述サブフィールドの指定が必要なのは、EXTFLD および INZ のような キーワードが使用された場合だけです。

ブランク
プログラム記述の場合です。 データ構造サブフィールドはこの仕様行で定義さ れます。

23 桁目 (データ構造のタイプ)

自由形式構文	**S** PSDS キーワード。 **U** DTAARA キーワードの *AUTO パラメーター。

この記入項目は、定義中のデータ構造のタイプを識別するために使用されます。 データ構造を定義していない場合には、この記入項目をブランクのまま にしておかなければなりません。

記入
説明

ブランク
定義しているデータ構造はプログラム状況データ構造またはデータ域データ 構造ではないか、あるいはこの仕様でデータ構造を定義中ではありません。

S
プログラム状況データ構造。 プログラム状況データ構造として指定することができるデータ構造は 1 つだけで す。

U
データ域データ構造。

RPG IV は初期化時にデータ域を検索し、プログラムの終了時にそれを再書き出しします。

- DTAARA キーワードが指定された場合には、DTAARA キーワードに対するパラ メーターが外部データ域の名前として使用されます。 名前が変数の場合は、 値はプログラミングが開始する前に設定する必要があります。 これは、以下の方法で行うことができます。
  - 変数をパラメーターとして渡す。
  - INZ キーワードを使用して変数を明示的に初期化する。
  - IMPORT キーワードおよび EXPORT キーワードを使用して別のモジュールと変数を共用し、 呼び出す前に値が設定されるようにする。
- DTAARA キーワードが指定されていない場合には、7 から 21 桁目の名前が外部デ ータ域の名前として使用されます。
- DTAARA キーワードまたは 7 から 21 桁目のいずれによっても名前が指定されて いない場合には、*LDA (内部データ域) が外部データ域の名前として使用されます。

24 から 25 桁目 (定義タイプ)

自由形式構文	391 ページの『自由形式の定義ステートメント』を参照してください。

記入
説明

ブランク
この仕様は、データ構造サブフィールドか、あるいはプロトタイプまたはプ ロシージャー・インターフェース定義内のパラメーターのいずれかを定義します。

C
この仕様は定数を定義します。 25 桁目はブランクでなければなりません。

DS
この仕様はデータ構造を定義します。

PR
この仕様はプロトタイプおよび戻り値 (それがあった場合) を定義します。

PI
この仕様はプロシージャー・インターフェースおよび戻り値 (それがあった 場合) を定義します。

S
この仕様は独立フィールド、配列、またはテーブルを定義します。 25 桁目はブランクでなければなりません。

データ構造、プロトタイプ、およびプロシージャー・インターフェースの定義は、 24 から 25 桁目がブランクでない最初の定義仕様書、または定義仕様書でない最初の指定で終わります。

定義のタイプによってグループ化された有効なキーワードのリストについては、489 ページの表 100 を参照してください。

26 から 32 桁目 (開始位置)

自由形式構文	POS キーワードを使用します。 *STATUS などの予約語の場合、最初のキーワードとして予約語を指定します。

26 から 32 桁目に記入項目を入れることができるのは、データ構造内のサブフィール ドの位置を定義している場合だけです。

記入
説明

ブランク
ブランクの開始位置は、「終了位置/長さ」フィールドの値によってサブフ ィールドの長さが指定されるか、あるいはこの仕様行ではサブフィールドは定義 されていないことを指示します。

NNNNNNN
データ構造内のサブフィールドの絶対開始位置。 値は 1 から 9999999 の範囲で、桁を右寄せにして指定する必要があります。

予約語
「開始位置」および「終了位置/長さ」フィールド (26 から 39 桁目) には、プ ログラム状況データ構造またはファイル情報データ構造に関する予約語を (左寄 せして) 使用することができます。 これらの特殊な予約語は、データ構造内のサブフィールドの位置を定義します。 プログラム状況データ構造に関する予約語は、*STATUS、*PROC、*PARM、および *ROUTINE です。 ファイル情報データ構造 (INFDS) に関する予約語は、*FILE、*RECORD、*OPCODE、 *STATUS、および *ROUTINE です。

33 から 39 桁目 (終了位置/長さ)

自由形式構文	**終了位置** POS キーワードを使用して開始位置を指定し、データ・タイプ・キーワードの長さパラメーターを使用して、長さを文字数または桁数で指定します。 **長さ** データ・タイプ・キーワードの長さパラメーター。

記入
説明

ブランク
33 から 39 桁目がブランクになるのは次の場合です。

- 名前付き定数をこの仕様行で定義しているか、あるいは
- 独立フィールド、パラメーター、またはサブフィールドが別のフィールドと 類似 (LIKE) のものとして定義されているか、あるいは
- 独立フィールド、パラメーター、またはサブフィールドの長さが暗黙に指定 されるタイプであるか、あるいは
- サブフィールドの属性が他の場所で定義されているか、あるいは
- データ構造が定義されている場合です。 データ構造の長さはサブフィールドの終了位置の最大値です。 データ構造は LIKEDS キーワードを使用して定義することも、LIKEREC キーワードを使用して定義することもできます。

NNNNNNN
33 から 39 桁目には、1 から 9999999 の数値 (右寄せ) が入ります (以下参照)。

- 開始位置フィールド (26 から 32 桁目) に数値が入っている場合には、このフィールドの数値はデータ構造内のサブフィールドの絶対終了位置を指定します。
- 開始位置フィールドがブランクの場合には、このフィールドの数値は次のも のを指定します。
  - データ構造全体の長さ、または
  - 独立フィールドの長さ、または
  - パラメーターの長さ、または
  - サブフィールドの長さ。 データ構造内で、このサブフィールドは、開始位置がデータ構造中で前に定義さ れたすべてのサブフィールドの最大終了位置より大きくなるように位置決めされ ます。 サブフィールドが確実に正しく位置合わせされるように、サブフィールドが基底ポ インターまたはプロシージャー・ポインターのタイプとして定義された場合には 埋め込みが挿入されます。

注 :

1. 図形フィールドまたは UCS-2 フィールドの場合には、ここに指定される数はバイト数ではなく、図形文字または UCS-2 文字の数になります (1 図形文字または UCS-2 文字 = 2 バイト)。 数値フィールドの場合には、 ここに指定される数は桁数です (パック・フィールドおよびゾーン・フィールドの数値フィールドの場合は 1 から 63、2 進形式の数値フィールドの場合は 1 から 9、 整数および符号なしの数値フィールドの場合は 3、5、10、または 20)。
2. 浮動数値フィールドの場合、指定される数字はバイト数 であり、桁数ではありません (4 または 8 バイト)。
3. 長さが 9999999 を超える文字、UCS-2、または図形を定義する場合は、「長さ」記入項目を指定する代わりに LEN キーワードを使用してください。 LEN キーワードによって長さを定義するサブフィールドを、明示的に配置する場合は、OVERLAY キーワードを使用してください。 OVERLAY キーワードの 1 番目のパラメーターにデータ構造名を、2 番目のパラメーターにサブフィールドの開始位置をコーディングします。

+|-nnnnn
この記入項目が有効なのは、独立フィールドまたはサブフィールドが LIKE キーワードを使用して定義される場合です。 この仕様行で定義中の独立フィールドまたはサブフィールドの長さは、これら桁 に記入された値を LIKE キーワードに対するパラメーターとして指定されたフィ ールドの長さに加減算することによって決まります。

注 :

1. 図形フィールドまたは UCS-2 フィールドの場合には、ここに指定される数はバイト数ではなく、図形文字または UCS-2 文字の数になります (1 図形文字または UCS-2 文字 = 2 バイト)。 数値フィールドの場合には、ここに指定される数は桁数です。
2. 浮動フィールドの場合、記入項目はブランクまたは +0 である必要があります。 浮動フィールドのサイズは、他の数値の場合のように変更することはできません。
3. タイム・スタンプ・フィールドの場合、長さを指定するときには、19 を指定するか、または 21 から 32 の長さを指定しなければなりません。

予約語
26 から 32 桁目を使用して特殊な予約語を入力する場合には、このフィールドは 1 つの大きなフィールド (26 から 39 桁目) を作成する 1 つ前のフィールドの拡張部 分となります。 これにより、長さが 7 文字より長い名前を持つ予約語の場合に、このフィールド に拡張させることができます。 409 ページの『26 から 32 桁目 (開始位置)』 の「予約語」を参照してください。

40 桁目 (内部データ・タイプ)

自由形式構文	データ・タイプ・キーワード

この記入項目によって、独立フィールド、パラメーター、 またはデータ構造サブフィールドが内部的に記憶される 方法を指定することができます。 データ項目を外部的に記憶する方法 (すなわち、それが外部で記憶される場合) とは無関係に、この記入項目はデータ構造の内部表現と厳密に関係しています。 可変長の

文字、図形、および UCS-2 形式を定義するには、キーワード VARYING を指定する必要があります。指定しない場合、形式は固定長になります。

記入
説明

ブランク
LIKE キーワードが指定されていない場合には、次のようになります。

- 小数点以下の桁数の指定がブランクであれば、項目は文字として定義されま す。
- 小数点以下の桁数の指定がブランクでなければ、項目は独立フィールド またはパラメーターの場合はパック形式の数値として定義され、サブフィールド の場合はゾーン形式の数値として定義されます。

注 : LIKE、LIKEDS、および LIKEREC キーワードが指定される場合は、記入項目はブランクにする必要があります。

A
文字 (固定長形式または可変長形式)

B
数値 (2 進数形式)

C
UCS-2 (固定長形式または可変長形式)

D
日付

F
数値 (浮動形式)

G
図形 (固定長形式または可変長形式)

I
数値 (整数形式)

N
文字 (標識形式)

O
オブジェクト

P
数値 (パック 10 進数形式)

S
数値 (ゾーン形式)

T
時刻

U
数値 (符号なし形式)

Z
タイム・スタンプ

*****
基底ポインターまたはプロシージャー・ポインター

41 から 42 桁目 (小数点以下の桁数)

自由形式構文	データ・タイプ・キーワード の小数点位パラメーターまたは TIMESTAMP キーワードのパラメーター

41 から 42 桁目は、数値サブフィールドまたは独立フィールドの小数点以下の桁数を 指示するために使用されます。 フィールドが非浮動数値の場合、常にこれらの桁への記入が必要です。 小数点以下の桁数がな

い場合には、42 桁目にゼロ (0) を入れてください。 たとえば、整数または符号なしフィールド (40 桁目のタイプが I または U) の場 合には、この記入項目にゼロが必要です。

記入
説明

ブランク
値が数値でない (その値が浮動フィールドある場合を除いて) か、 LIKE キーワードによって定義されています。

0 から 63
小数点以下の桁数: 数値フィールドで小数点の右側の桁数。

この記入項目を指定できるのは、「終了位置/長さ」フィールドと組み合わせた 場合だけです。「終了位置/長さ」フィールドがブランクの場合には、この記入項目の値はプロ グラム中のどこか別の場所で (たとえば、外部記述データベース・ファイルに よって) 定義されています。

小数点以下の桁数の項目は、タイム・スタンプの秒の小数部の桁数を指定するのにも使用できます。 タイム・スタンプの長さが指定されていない場合、ゼロから 12 までの範囲で秒の小数部の 桁数を指定できます。 秒の小数部の桁数をゼロと指定した場合、タイム・スタンプの長さは 19 に なります。 秒の小数部の桁数を 1 から 12 までの範囲で指定した場合、 タイム・スタンプの長さは、20 に小数部の桁数を加算したものになります。

43 桁目 (予約済み)

43 桁目はブランクでなければなりません。

44 から 80 桁目 (キーワード)

自由形式構文	自由形式ステートメントで使用可能な桁については、307 ページの『自由形式ステートメント』を参照してください。

桁制限付きでは、44 から 80 桁目は定義仕様書キーワードのために用意されています。 キーワードを使用して、データとその属性を記述して定義します。 この区域を使用して、フィールドを完全に定義するために必要なすべてのキーワ ードを指定します。

定義仕様書のキーワード

定義仕様書のキーワードは、パラメーターを持っていなかったり、任意指 定パラメーターを持ったり、または必須パラメーターを持ったりします。 キーワードの構文は次のとおりです。

```
    Keyword(parameter1 : parameter2)
```

ここで、

- 1 つまたは複数のパラメーターは括弧 () で囲みます。

  注 : パラメーターがない場合、括弧 を指定してはなりません。
- コロン (:) を使用して複数のパラメーターを区切ります。

任意指定パラメーターと必須パラメーターを示すために、 以下の国別の規則を使用します。

- 中括弧 { } は任意指定パラメーターまたはパラメーターの任意指定要素を示します。
- 省略記号 (...) はパラメーターが反復可能であることを示します。
- コロン (:) はパラメーターを区切り、複数のパラメーターを指定できることを示します。 コロンで区切られたすべてのパラメーターは、中括弧で囲まれていない限り、必須パラメーターです。
- 縦線 (|) は、キーワードに 1 つのパラメーターしか指定でき ないことを示します。
- キーワード・パラメーターを区切るブランクは、1 つまたは複数のパラメーター を指定できることを示します。

注 : 中括弧、省略記号、および縦線は、キーワード構文の一部 ではないため、ソースに入れてはなりません。

定義仕様書のキーワードに追加のスペースが必要な場合には、キーワード・フィールド を後続の行に継続させることができます。 406 ページの『定義仕様書のキーワード継続記入行』 および 315 ページの『定義仕様書のキーワード・フィールド』 を参照してください。

以下のキーワード・リストには、 組み込み SQL を使用する ILE RPG ソース用に許可されている SQLTYPE などのキーワード は含まれていません。 これらの追加キーワードについては、IBM i Information Center の組み込み SQL プログラミングに関するトピックを参照してください。

ALIAS

外部記述データ構造に ALIAS キーワードが指定されると、 RPG コンパイラーは、サブフィールドに別名 (代替名) があれば、それを使用します。 データ構造に ALIAS キーワードが指定されない、または外部フィールドに別名が定義されていない場合、 RPG コンパイラーは標準の外部フィールド名を使用します。

別名が使用されていて、サブフィールドの名前を変更する場合には、 EXTFLD キーワードのパラメーターとしてその別名を指定します。 EXTFLD キーワードでは継続がサポートされないため、1 つのソース指定で名前全体を指定してください。414 ページの図 127 は、同じファイルについて定義された、2 つのデータ構造を含む例を示しています。 ALIAS キーワードがコーディングされたデータ構造では、EXTFLD キーワードのパラメーターとして、別名 CUSTOMER_ADDRESS を使用します。 ALIAS キーワードがコーディングされていないデータ構造では、EXTFLD キーワードのパラメーターとして、標準名 CUSTAD を使用します。

注 : 特定の外部フィールドの代替名が引用符で囲まれている場合、標準の外部フィールド名がそのフィールドに対して使用されます。

ALIAS キーワードと一緒に PREFIX キーワードを指定した場合、置換対象文字数を示す、PREFIX の 2 番目のパラメーターは、別名には適用されません。 以下の説明では、外部ファイル MYFILE にフィールド XYCUSTNM と XYID_NUM があり、XYCUSTNM フィールドの別名が CUSTOMER_NAME であることを想定しています。

- キーワード PREFIX(NEW_) を指定した場合、2 番目のパラメーターがないため、 どの名前についても文字は置換されません。 RPG サブフィールドに使用される名前は、NEW_CUSTOMER_NAME と NEW_XYID_NUM になります。
- キーワード PREFIX(NEW_:2) を指定した場合、別名を持たないフィールドの名前から 2 文字が除去されます。 RPG サブフィールドに使用される名前は、NEW_CUSTOMER_NAME と NEW_ID_NUM になります。 XYID_NUM では先頭の 2 文字の "XY" が置換されますが、CUSTOMER_NAME では文字は置換されません。
- キーワード PREFIX(':2) を指定した場合、別名を持たないフィールドの名前から 2 文字が除去されます。 RPG サブフィールドに使用される名前は、CUSTOMER_NAME と ID_NUM になります。 XYID_NUM では先頭の 2 文字の "XY" が置換されますが、 CUSTOMER_NAME では文字は置換されません。

```
 * The DDS specifications for file MYFILE, using the ALIAS keyword
 * for the first two fields, to associate alias name CUSTOMER_NAME
 * with the CUSTNM field and alias name CUSTOMER_ADDRESS
 * with the CUSTAD field.
A          R CUSTREC
A            CUSTNM        25A         ALIAS(CUSTOMER_NAME)
A            CUSTAD        25A         ALIAS(CUSTOMER_ADDRESS)
A            ID_NUM        12P 0

 * The RPG source, using the ALIAS keyword.
 * The customer-address field is renamed to CUST_ADDR
 * for both data structures.
D aliasDs       e ds                  ALIAS
D                                     QUALIFIED EXTNAME(myfile)
D   cust_addr   e                     EXTFLD(CUSTOMER_ADDRESS)
D noAliasDs     e ds
D                                     QUALIFIED EXTNAME(myfile)
D   cust_addr   e                     EXTFLD(CUSTAD)
 /free
    // The ALIAS keyword is specified for data structure "aliasDs"
    // so the subfield corresponding to the "CUSTNM" field has
    // the alias name "CUSTOMER_NAME"
    aliasDs.customer_name = 'John Smith';
    aliasDs.cust_addr = '123 Mockingbird Lane';
    aliasDs.id_num = 12345;

    // The ALIAS keyword is not specified for data structure
    // "noAliasDs", so the subfield corresponding to the "CUSTNM"
    // field does not use the alias name
    noAliasDs.custnm = 'John Smith';
    aliasDs.cust_addr = '123 Mockingbird Lane';
    noAliasDs.id_num = 12345;
```

図 *127.* 外部記述データ構造に対する *ALIAS* キーワードの使用

ALIGN{(*FULL)}

ALIGN キーワードは、浮動、整数、および符号なしサブフィールドを 位置合わせするために使用されます。 ALIGN が指定されると、2 バイト・サブフィールドは 2 バイト境界に位置合わせ され、4 バイト・サブフィールドは 4 バイト境界に、さら に 8 バイト・サブフィールドは 8 バイト・サブフィールドは 8 バイト境界に それぞれ位置合わせされます。 位置合わせは、浮動、整数、または符号なしの各サブフィールドへのアクセス時のパフォーマンスを向上させる可能性があります。

ALIGN はデータ構造定義で指定してください。 ただし、ファイル情報データ構造 (INFDS) とプログラム状況 データ構造 (PSDS) のどちらにも ALIGN を指定することはできません。

位置合わせは、キーワード OVERLAY なしに長さ表記とともに定義されたデータ構造 サブフィールドについてのみ行なわれます。 絶対表記法で定義されたかまたは OVERLAY キーワードを使用しているサブフィー ルドの位置合わせが正しく行なわれない場合には、診断メッセージが出されます。

ポインター・サブフィールドは、ALIGN が指定されているかどうかに関係 なく、常に 16 バイト境界に位置合わせされます。

データ構造の長さを必要な位置合わせの倍数にしたい場合は ALIGN(*FULL) を指定します。

- パラメーターをある関数または API に渡している場合、データ構造の長さを、必要な位置合わせの倍数にすることが必要な場合があります。 例えば、_packed キーワードを指定せずに、C で定義されたデータ構造パラメーターを使用して C 関数または API を呼び出す場合、API の RPG データ構造を ALIGN(*FULL) で定義する必要があります。 ALIGN(*FULL) を使用すると、データ構造が確実に十分な大きさになります。例えば、*regex_t* データ構造 (ソース・ファイル QSYSINC/H のメンバー REGEX 内で定義される) の長さが 656 であるとします。 ALIGN キーワードを指定して RPG で定義されている突き合わせ用データ構造は長さが 644 しかありません。 一方 ALIGN(*FULL) キーワードで定義されている突き合わせ用データ構造は正しい長さの 656 となります。 ALIGN が *FULL パラメーターを使用して指定されない場合、*regcomp()* API の呼び出しによりデータ破壊が引き起こされる可能性があります。API が必要としているより RPG データ構造が小さいためです。

- データ構造の配列の要素間、または複数オカレンス・データ構造のオカレンス間の距離を %SIZE を使用して決定するときに、そのデータ構造が ALIGN(*FULL) で定義されている場合には %SIZE(ds_name) を使用できます。 ただし、データ構造が ALIGN だけを使用して定義されている場合、または ALIGN キーワードが指定されておらずデータ構造にポインターが含まれている場合は、 %ELEM(ds_name) で除算した %SIZE(ds_name:*ALL) を使用する必要があります。

警告 : OVERLAY キーワードが配列の要素間の距離に与える影響について詳しくは、687 ページの『%SIZE (サイズ (バイト数) の検索)』 を参照してください。

ALIGN キーワードの影響を示す例

以下に挙げた複数の例のデータ構造はいずれも同じサブフィールドを使用しています。

1. データ構造 *ds1_no_align* は ALIGN キーワードを指定せずに定義されています。 整数サブフィールド sub2 は 2 桁目にあります。 文字サブフィールド sub3 は 6 桁目にあります。 データ構造の終わりに埋め込みはないため、サイズは 6 になります。

   データ構造の配列バージョンに関して、*ALL を指定した %SIZE の結果 (2) は、1 つのパラメーターのみを指定した %SIZE の結果 (1) の倍数になります。1).

```
DCL-DS ds1_no_align QUALIFIED;
   sub1 CHAR(1);
   sub2 INT(10);
   sub3 CHAR(1);
END-DS;

DCL-DS ds1_no_align_arr LIKEDS(ds1_no_align) DIM(2);

size = %SIZE(ds1_no_align);             // 6
size = %SIZE(ds1_no_align_arr);         // 6  1
size = %SIZE(ds1_no_align_arr : *ALL);  // 12 2
```

2. データ構造 *ds2_simple_align* は、パラメーターを指定しない ALIGN キーワードで定義されています。 整数サブフィールド sub2 は 4 バイトの位置合わせが必要なので、5 桁目にあります。 文字サブフィールド sub3 は 9 桁目にあります。 データ構造の終わりに埋め込みはないため、サイズは 9 になります。

   データ構造の配列バージョンに関して、*ALL を指定した %SIZE の結果 (2) は、1 つのパラメーターのみを指定した %SIZE の結果 (1) の倍数にはなりません。1).

```
DCL-DS ds2_simple_align ALIGN QUALIFIED;
   sub1 CHAR(1);
   sub2 INT(10);
   sub3 CHAR(1);
END-DS;

DCL-DS ds2_simple_align_arr LIKEDS(ds2_simple_align) DIM(2);

size = %SIZE(ds2_simple_align);            // 9
size = %SIZE(ds2_simple_align_arr);        // 9  1
size = %SIZE(ds2_simple_align_arr : *ALL); // 24 2
```

3. データ構造 *ds3_align_full* は ALIGN(*FULL) キーワードで定義されています。 整数サブフィールド sub2 は 4 バイトの位置合わせが必要なので、5 桁目にあります。 文字サブフィールド sub3 は 9 桁目にあります。データ構造の調整しないサイズは 9 ですが、データ構造のサイズは、任意のサブフィールドに必要な最も大きな位置合わせの倍数である必要があります。 サイズが強制的に 12 になるように、データ構造の終わりに埋め込みが追加されます。

   データ構造の配列バージョンに関して、*ALL を指定した %SIZE の結果 (2) は、1 つのパラメーターのみを指定した %SIZE の結果 (1) の倍数になります。1).

```
DCL-DS ds3_align_full ALIGN(*FULL) QUALIFIED;
   sub1 CHAR(1);
   sub2 INT(10);
   sub3 CHAR(1);
END-DS;

DCL-DS ds3_align_full_arr LIKEDS(ds3_align_full) DIM(2);

size = %SIZE(ds3_align_full);             // 12
size = %SIZE(ds3_align_full_arr);         // 12  1
size = %SIZE(ds3_align_full_arr : *ALL);  // 24  2
```

詳しくは、214 ページの『データ構造サブフィールドの位置合わせ』 を参照してください。

ALT(配列名)

ALT キーワードは、コンパイル時または実行時前配列またはテーブルが交互形式 になっていることを指示するために使用されます。

ALT キーワードによって定義される配列は交互配列で、パラメーターとして指定 された配列名が主配列になります。 代替配列定義は、主配列定義の前にあっても、その後にあってもかまいません。

主配列のキーワードによって、両方の配列のロードが定義されます。 初期化データは、主/交互/主/交互/ ... のように主配列から始まる交互 の順になっています。

交互配列定義で、PERRCD、FROMFILE、TOFILE、および CTDATA キーワードは有効 でありません。

ALTSEQ(*NONE)

ALTSEQ(*NONE) キーワードを指定すると、制御仕様書に ALTSEQ キーワードが 指定されている場合でも、このフィールドにかかわる比較に代替照合順序は 使用されません。 データ定義仕様書の ALTSEQ(*NONE) は、ALTSEQ、ALTSEQ(*SRC) また は ALTSEQ(*EXT) のいずれかが制御仕様書でコーディングされている 場合に限り、有効になります。 この条件が満たされなければ、このキーワードは無視されます。

ALTSEQ(*NONE) は、次の場合に有効なキーワードとなります。

- 文字独立フィールド
- 文字配列
- 文字テーブル
- 文字サブフィールド
- データ構造
- プロシージャー・インターフェースまたはプロトタイプ定義の文字戻り値
- 文字プロトタイプ・パラメーター

ASCEND

ASCEND キーワードは、データの次のいずれかの順序を記述するために使用されま す。

- 配列
- 実行時前またはコンパイル時にロードされるテーブル
- プロトタイプ・パラメーター

421 ページの『DESCEND』も参照してください。

昇順とは、配列またはテーブルの項目が最低のデータ項目から始まり (照合順序 に従って)、最高のデータまで進むことを意味します。 等しい値の項目も許されています。

配列またはテーブルがデータとともにロードされる時に、実行時前配列またはテ ーブルが指定された順序であるかどうかが検査されます。 配列またはテーブルの順序が違っていた場合には、RPG IV 例外/エラー

処理ルーチンに制御が渡されます。 実行時配列 (入力仕様または演算仕様書、あるいはその両方によってロードされる) の順序は検査されません。

実行時前の配列またはテーブルをロードするにはファイル内のデータが不十分であり、その配列またはテーブルが ASCEND キーワードで定義されている場合は、その配列の末尾にある要素の値がファイルからロードされた要素の値より小さければ、RPG モジュールの初期化時に状況コード 1041 のエラーが出されます。

ALTSEQ(*EXT) が指定された場合には、コンパイル時配列またはテーブルの順序の 検査時に代替照合順序が使用されます。 代替順序が実行時まで不明の場合、順序は実行時に検査され、配列またはテーブルの順序が違っていた場合は、RPG IV 例外/エラー処理ルーチンに制御が渡されます。

LOOKUP 命令、%LOOKUPxx 組み込み関数、または %TLOOKUPxx 組み込み関数を使用して、項目を配列またはテーブルから検索してその項目が検索引数と比較して大きいかまたは小さいかを判別する場合には、順序 (昇順または降順) を指定しなければなりません。

配列を用いて SORTA 命令コードを使用し、順序を指定しない場合には、昇順と見なされます。

BASED(基底ポインター名)

データ構造または独立フィールドについて BASED キーワードを指定した場合には 、キーワード・パラメーターとして指定された名前を使用して**基底ポインター**が作成されます。 この基底ポインターには、定義中の**基底付き**データ構造または独立フィールドのアドレス (記憶位置) が保留されます。 言い換えると、7 から 21 桁目に指定された名前は、基底ポインターに含まれている 位置に記憶されたデータを参照するために使用されます。

注 : 基底付きデータ構造または独立フィールドを使用する前に、基底ポインターには 正しいアドレスが割り当てられていなければなりません。

配列を基底付き独立フィールドとして定義する場合には、 実行時 配列でなければなりません。

基底付きフィールドをサブプロシージャーの中で定義した場合には、フィールド と基底ポインターの両方がローカルとなります。

BINDEC(桁数 {: 小数点以下の桁数})

BINDEC キーワードは、数字データ・タイプのキーワードの 1 つです。 これは自由形式定義で使用されて、項目の 形式が 2 進-10 進形式であることを示します。

これは最初のキーワードでなければなりません。

最初のパラメーターは必須です。 これは総桁数を指定します。 1 から 9 までの値を指定できます。

2 番目のパラメーターは任意指定です。 これは小数点以下の桁数を指定します。 ゼロから桁数までの範囲の 値を指定できます。 デフォルトはゼロです。

各パラメーターにはリテラルまたは名前付き定数を指定できます。 名前付き定数である場合、定義ステートメントの前にその定数が定義されている必要があります。

次に例を示します。

- フィールド *salary* は、桁数が 5 で小数点以下の桁数が 2 の 2 進-10 進フィールド として定義されています。
- フィールド *age* は、桁数が 3 で小数点以下の桁数がデフォルトの 0 である 2 進- 10 進フィールド として定義されています。
- フィールド *price* は、桁数が 7 で小数点以下の桁数が 3 の 2 進-10 進フィールド として定義されています。 小数点以下の桁数は、名前付き定数 *NUM_DEC_POS* を使用して定義されています。

```
DCL-S salary BINDEC(5 : 2);
DCL-S age BINDEC(3);
DCL-C NUM_DEC_POS 3;
DCL-S price BINDEC(7 : NUM_DEC_POS);
```

CCSID 定義キーワード

このキーワードは、英数字定義、グラフィック定義、 および UCS-2 定義の CCSID を設定します。

番号は、0 から 65535 の範囲の整数で、 有効な CCSID 値でなければなりません。

- 有効な英数字 CCSID は、65535 か、 または、エンコード・スキーム x'1100' または x'1301' の EBCDIC CCSID、 または、エンコード・スキーム X'2100'、X'3100'、X'4100'、X'4105'、X'5100'、X'2300'、X'3300' の ASCII CCSID、 または UTF-8 CCSID 1208 です。
- 有効な図形 CCSID は、65535 か、EBCDIC 2 バイト・コード化体系 (X'1200') の CCSID で す。
- 有効 な UCS-2 CCSID は UCS-2 コード化体系 (x'7200') になっています。

プログラム記述フィールドの場合、CCSID キーワードは、 CCSID(*CHAR)、CCSID(*GRAPH)、または CCSID(*UCS2) キーワード が指定された制御仕様書または /SET 指示で設定されたデフォルトをオーバーライドします。

いくつかの特殊値が許可されています。

***DFT**
CCSID(*DFT) は、モジュールの現行デフォルト CCSID が使用されることを示します。 これは、LIKE キーワードを使用するときに便利です。これは、このキーワードが 指定されていない場合、新しいフィールドはソース・フィールドの CCSID を 継承するためです。 現在のデフォルトの CCSID について詳しくは、81 ページの『/SET』を参照してください。

***{NO}EXACT**
CCSID(*EXACT) および CCSID(*NOEXACT) は、EXTNAME または LIKEREC キーワードに *NULL が抜き出しタイプとして指定されないとき、外部記述データ構造および LIKEREC キーワードで定義されたデータ構造に対して有効です。 データ構造に対する CCSID キーワードは、英数字サブフィールドの CCSID を制御します。 詳しくは、419 ページの『CCSID(*EXACT | *NOEXACT)』 を参照してください。

***HEX または 65535**
CCSID(*HEX) は、英数字定義およびグラフィック定義に 対して有効です。 これは、データに CCSID がないと見なされることを示します。 CCSID(*HEX) または CCSID(65535) を指定して定義された 項目を CCSID 変換で使用することはできません。

***JOBRUN**
CCSID(*JOBRUN) は、英数字定義およびグラフィック定義 に対して有効です。 これは、データが実行時にジョブ CCSID であることを指示します。

***JOBRUNMIX**
CCSID(*JOBRUNMIX) は、英数字定義に対して有効です。 これは、CCSID がジョブ CCSID に関連する混合バイト CCSID であることを示します。

***UTF8**
CCSID(*UTF8) は、英数字定義に対して有効です。 これは、CCSID が、UTf-8 Unicode データの CCSID である 1208 であることを示します。

***UTF16**
CCSID(*UTF16) は、UCS-2 定義に対して有効です。 これは、CCSID が、UTF-16 Unicode データの CCSID である 1200 であることを示します。

このキーワードが指定されていない場合は、次のようになります。

- データ・タイプが指定されている場合、モジュールの現行デフォルト CCSID が想定されます。 現在のデフォルトの CCSID について詳しくは、81 ページの『/SET』を参照してください。
- LIKE キーワードが指定されている場合、 新しいフィールドの CCSID は LIKE フィールドと同じものになります。

注 :

- 文字定義に対してこのキーワードが指定されておらず、 制御ステートメントに CCSID(*EXACT) も CCSID(*CHAR) も指定されておらず、 CCSID(*CHAR) が指定された有効な /SET ステートメントもない場合、 文字フィールドの CCSID は、ジョブ CCSID に関連した混合 CCSID です。 混合 CCSID は、1 バイト文字セット (SBCS) データと 2 バイト文字セット (DBCS) データ の両方を処理できます。 例えば、ジョブ CCSID が 1 バイト CCSID 37 の場合、 混合 CCSID は 937 です。 文字 X'0E' および X'0F' はシフト文字で

あると解釈されます。 そのため、このデータが別の CCSID に変換される際、データに偶然に X'0E' または X'0F' が含まれていて、ジョブ CCSID が DBCS 対応 CCSID ではない場合、誤った結果になることがあります。

- CCSID(*GRAPH : *IGNORE) が 指定または想定されているときには、グラフィック定義に対してこのキーワードを使用することは許可されません。

*CCSID(*EXACT | *NOEXACT)*

CCSID キーワードを指定することによって、外部記述データ構造または LIKEREC キーワードを 使用して定義されたデータ構造の英数字外部サブフィールドの CCSID を 制御できます。

CCSID(*EXACT) を指定すると、英数字サブフィールドの CCSID はファイル内の外部フィールドと同じ CCSID になります。

CCSID(*NOEXACT) を指定すると、英数字サブフィールドの CCSID は英数字データの現行デフォルト CCSID になり、 英数字サブフィールドには明示的に指定された CCSID キーワードがあると見なされます。

CCSID キーワードが指定されていない場合は、次のようになります。

- 制御ステートメントに CCSID(*EXACT) が 指定されている場合、CCSID(*NOEXACT) が想定されます。 CCSID キーワードが明示的に指定されていない場合、 英数字サブフィールドには暗黙的に指定された CCSID キーワード があると見なされます。
- 制御ステートメントに CCSID(*EXACT) が指定されていない場合、 サブフィールドの CCSID はデフォルト英数字 CCSID になります。 ただし、デフォルト英数字 CCSID が明示的に 設定されていない場合、英数字サブフィールドには指定された CCSID キーワードがないと 見なされます。

明示的または暗黙的に CCSID キーワードを指定することが、データ構造またはその英数字サブフィールドを使用する入出力命令に与える影響について詳しくは、262 ページの『入出力命令中の CCSID 変換』を参照してください。

CHAR(長さ)

CHAR キーワードは、自由形式定義で使用されて、 項目が固定長文字であることを示します。

これは最初のキーワードでなければなりません。

パラメーターは長さをバイト単位で指定します。 1 から 16,773,104 までの値を指定できます。

パラメーターにはリテラルまたは名前付き定数を指定できます。 名前付き定数である場合、定義ステートメントの前にその定数が定義されている必要があります。

次に例を示します。

- フィールド *cust_name* は、100 文字の 固定長文字フィールドとして定義されています。
- フィールド *message* は、5000 文字 の固定長文字フィールドとして定義されています。 長さは、名前付き定数 *MSG_LEN* を使用して定義されています。

```
DCL-S cust_name CHAR(100);
DCL-C MSG_LEN 5000;
DCL-S message CHAR(MSG_LEN);
```

可変長文字項目の定義については、486 ページの『VARCHAR(長さ {:2 | 4})』を参照してください。

CLASS(*JAVA:クラス名)

このキーワードは、オブジェクト定義のクラスを指定します。

クラス名は、定数の文字値でなければなりません。

注 : CLASS キーワードは、自由形式定義では使用されません。 代わりに、クラス情報は OBJECT キーワードによって指定されます。

CONST{(定数)}

CONST キーワードは、次のために使用されます。

- 名前付き定数の値を指定するため
- 参照によって渡されるパラメーターが読み取り専用であることを指示するため

名前付き定数の値を指定する場合 には、CONST キーワード自体は 任意指定です。 すなわち、定数の値は CONST キーワードの使用の有無にかかわらず指定する ことができます。

このパラメーターは、リテラル、形象定数、または組み込み関数としなければなり ません。 定数は、該当する継続規則に従って後続の行に継続させることができます (詳しくは、313 ページの『継続の規則』を参照)。

キーワード DIM、OCCURS、PERRCD、または OVERLAY のパラメーターとして名前付 き定数を使用する場合には、その使用に先立って名前付き定数を定義し ておかなければなりません。

読み取り専用参照パラメーターを指定する場合 には、プロトタイプとプ ロシージャー・インターフェースの両方のパラメーター定義の定義仕様書にキーワ ード CONST を指定します。 キーワードに対してパラメーターを指定しないことは許されません。

キーワード CONST が指定されている場合、コンパイラーは、パラメーターを、プロトタイプになったパラメーターと同じデータ・タイプおよび長さで定義された一時的 な場所にコピーして、その場所のアドレスを渡すことがあります。 この原因となる一部の条件として、渡されたパラメーターが式であったり、あるいは渡されたパラメーターが異なる形式である場合があります。

注意!

パラメーターが呼び出されたプログラムまたはプロシージャーで変更されない ことに確信がもてない限り、このキーワードをプロトタイプ定義で使用 してはいけません。

呼び出されたプログラムまたはプロシージャーが同じプロトタイプのプロシージ ャー・インターフェースを使用してコンパイルされる場合には、コンパイラーに よってこの点が検査されるので、心配する必要はありません。

CONST パラメーターはプロシージャー内のステートメントによって変更することはできませんが、プロシージャー外のステートメントの結果によって、またはグローバル変数を直接参照することによって、値が変更される場合があります。

定数の値によってパラメーターを渡すことには、値によって渡すのと同じ利 点があります。 とくに、リテラルおよび式を渡すことが可能です。

CTDATA

CTDATA キーワードは、配列またはテーブルがコンパイル時データを使用してロー ドされることを指示します。 データはプログラムの終わりで ** または **CTDATA (配列/テーブル名) の指定に 続けて指定されます。

配列またはテーブルがコンパイル時にロードされた時には、ソース・プログラムと 一緒にコンパイルされ、プログラムに組み込まれます。 このような配列またはテーブルは、プログラムの実行のたびに別個にロードする 必要はありません。

DATE{(形式 {区切り記号 }) }

DATE キーワードは、自由形式定義で使用されて、 項目のタイプが日付であることを示します。

これは最初のキーワードでなければなりません。

パラメーターは任意指定です。 これは日付形式と区切り記号を指定します。 日付項目のデフォルト形式については、273 ページの『日付データ・タイプ』 を参照してください。

次の例では、 フィールド *date_dft* は、モジュールのデフォルト形式の日付フィールドとして 定義されていて、 フィールド *date_mdy* は、形式が *MDY で区切り記号がハイフンであると定義されています。

```
DCL-S date_dft DATE;
DCL-S date_mdy DATE(*MDY-);
```

DATFMT(形式 {区切り記号 })

DATFMT キーワードは、日付タイプの独立フィールド、データ構造サブフィールド、 プロトタイプ・パラメーター、あるいはプロトタイプまたはプロシージャー・イ ンターフェース定義上の戻り値について内部日付データ形式と、オプションで区切 り文字を指定します。 このキーワードは、日付タイプの外部記述データ構造サブフィールドの場合には 自動的に生成され、コンパイル時に判別されます。

DATFMT が指定されない場合に、日付フィールドには、制御仕様書の DATFMT キーワ ードによって指定されたとおりの日付の形式と区切り記号が入れられます (それが あった場合)。 制御仕様書になにも指定されていなければ、*ISO 形式になります。

注 : DATFMT キーワードは、自由形式定義では使用されません。 代わりに、日付形式は DATE キーワードのパラメーターとして指定されます。

有効な形式および区切り記号については、274 ページの表 71 を参照してください。内部形式について詳しくは、247 ページの『内部形式および外部形式』 を参照してください。

DESCEND

DESCEND キーワードは、次のいずれかの中のデータの順序を記述します。

- 配列
- 実行時前またはコンパイル時にロードされるテーブル
- プロトタイプ・パラメーター

416 ページの『ASCEND』 も参照してください。

降順とは、配列またはテーブルの項目が最高のデータ項目から始まり (照合順序 に従って)、最低のデータまで進むことを意味します。 等しい値の項目も許されています。

配列またはテーブルがデータとともにロードされる時に、実行時前配列またはテ ーブルが指定された順序であるかどうかが検査されます。 配列またはテーブルの順序が違っていた場合、RPG IV 例外/エラー処理ルーチンに制御が渡されます。 実行時配列 (入力仕様または演算仕様書、あるいはその両方によってロードされる) の順序は検査されません。

実行時前の配列またはテーブルをロードするにはファイル内のデータが不十分であり、その配列またはテーブルが DESCEND キーワードで定義されている場合は、その配列の末尾にある要素の値がファイルからロードされた要素の値より小さければ、RPG モジュールの初期化時に状況コード 1041 のエラーが出されます。

ALTSEQ(*EXT) が指定された場合には、コンパイル時配列またはテーブルの順序の 検査時に代替照合順序が使用されます。 代替順序が実行時まで不明の場合、順序は実行時に検査され、配列またはテ ーブルの順序が違っていた場合は、RPG IV 例外/エラー処理ルーチンに制御が渡されます。

LOOKUP 命令、%LOOKUPxx 組み込み関数、または %TLOOKUPxx 組み込み関数を使用して、項目を配列またはテーブルから検索してその項目が検索引数と比較して大きいかまたは小さいかを判別する場合には、順序 (昇順または降順) を指定しなければなりません。

配列を用いて SORTA 命令コードを使用し、順序を指定しない場合には、昇順と見なされます。

DIM({*AUTO:|*CTDATA|*VAR:}数値定数)

DIM(数値定数)
DIM(*AUTO : 数値定数)
DIM(*VAR : 数値定数)
DIM(*CTDATA)

DIM キーワードは、配列、テーブル、プロトタイプ・パラメーター、配列データ構造、またはプロトタイプあるいはプロシージャー・イン ターフェース定義の戻り値の中の要素の数を定義します。

最初のパラメーターが *CTDATA でない限り、数値定数が必要です。 小数点以下の桁数はゼロ (0) でなければなりません。 リテラル、名前付き定数、または組み込み関数とすることができます。

定数の値は、キーワードの処理時には既知である必要はありませんが、コンパイル時には既知でなければなりません。

DIM がデータ構造定義に指定される場合、データ構造は修飾されたデータ構造である必要があり、サブフィールドは完全修飾名 (つまり、"dsname(x).subf") として参照される必要があります。 CTDATA、FROMFILE、TOFILE、および PERRCD などの他の配列キーワード は、配列データ構造定義では使用できません。

DIM(*CTDATA)

DIM キーワードの最初のパラメーターが *CTDATA の場合、配列またはテーブルの次元は、配列またはテーブルのコンパイル時データのレコード数によって決まります。 CTDATA キーワードは必要ありません。

以下の例では、配列 ARR が DIM(*CTDATA) で定義されています1配列のコンパイル時データにレコードは 3 つあるため1、配列の次元は 3 になります。

```
DCL-S arr CHAR(10) DIM(*CTDATA); //  1

**CTDATA arr   2
abc
def
ghi
```

DIM(*CTDATA) で定義された配列とテーブルの規則は以下のとおりです。

- DIM(*CTDATA) は、独立型の配列およびテーブルでのみ有効です。
- コンパイル時データにはレコードあたり 1 要素が必要です。 PERRCD キーワードが指定されている場合、パラメーターは 1 でなければなりません。
- 代替配列またはテーブルがある場合も、 DIM(*CTDATA) で定義することが必要です。
- コンパイル時データは **CTDATA で示す必要があります。
- コンパイル時データは、その他すべてのデータの前に存在する必要があります。

可変次元配列

DIM キーワードの最初のパラメーターが *AUTO または *VAR の場合、配列の次元は変数になります。

DIM キーワードの 2 番目のパラメーターは、配列要素の最大数を示します。

配列の要素数はゼロに初期化されます。 配列の次元が増えた場合、配列の初期設定値が使用されます。

変動次元配列について詳しくは、422 ページの『可変次元配列』を参照してください。

可変次元配列

可変次元配列は DIM(*AUTO:maximum_elements) または DIM(*VAR:maximum_elements) を使用して定義されます。 2 番目のパラメーターは、配列の要素の最大数を示します。

配列の要素数はゼロに初期化されます。 配列の要素数は、%ELEM 組み込み関数を使用して変更できます。以下の例では、配列の要素の最大数は 100 (1) です。配列の現在の要素数は、%ELEM 組み込み関数を使用して 25 に設定されています -1).

```
DCL-S var_array1 CHAR(10) DIM(*VAR : 100); // 2

%ELEM(var_array1) = 25; // 2
```

可変次元配列が DIM(*AUTO) を使用して定義される場合は以下のようになります。

- 要素数は、現在の要素数より大きい要素に値を代入しても増やすことができます。
- 配列が代入ステートメントのターゲットである場合、指標 *NEXT を指定できます。424 ページの『例: DIM(*AUTO) で定義された配列に指標として *NEXT を使用』を参照してください。

以下の例では、配列は最大 1000 要素 (1) で定義されています。値が配列の要素 100 (2) に代入された場合、要素の現在の数は 100 に増やされます。 前の要素数がゼロであったので、要素 1 から 99 は配列の初期設定値で初期化されます。

値が配列の要素 50 (3) に代入された場合、要素数は変更されません。50 は現在の要素数より小さいからです。

```
DCL-S auto_array1 CHAR(10) DIM(*AUTO : 1000);  // 1

auto_array1(100) = 'abc'; // 2
auto_array1(50) = 'abc';  // 3
```

現在の要素数は %ELEM を使用して減らすこともできます。 以下の例では、要素 100 (1) への代入のために、現在の要素数は 100 に変更されています。%ELEM への代入により 25 に減らされます -2).

```
DCL-S auto_array2 CHAR(10) DIM(*AUTO : 1000);

auto_array2(100) = 'abc'; // 1
%elem(auto_array2) = 25;  // 2
```

可変次元配列の制約事項

- 可変次元配列には、最上位の定義で、独立型の配列かデータ構造配列のいずれかのみを指定できます。テーブル、複数オカレンス・データ構造、サブフィールド、プロシージャーの戻り値、およびプロシージャーのパラメーターは指定できません。
- 可変次元配列は固定形式の計算で使用できません。
- 可変次元配列はポインターに基づいて作成することはできません。
- 可変次元配列はインポートおよびエクスポートできません。
- 可変次元配列がプロトタイプ・プロシージャーまたはプログラムにパラメーターとして渡される場合、そのパラメーターは OPTIONS(*VARSIZE) を使用して定義する必要があります。
- 可変次元配列には、タイプ Object を設定できません。
- 可変次元配列はヌル値可能にすることができません。
- 可変次元データ構造配列のサブフィールドは、ヌル標識も同じデータ構造内のサブフィールドである場合を除いて、ヌル値可能にすることができません。
- 可変次元配列は、指標が指定される場合を除いて、入力仕様に指定することはできません。
- 可変次元配列は、指標が指定される場合を除いて、出力仕様に指定することはできません。
- 可変次元配列は、先読み入力仕様に指定することはできません。
- 可変次元配列は、コンパイル時配列および実行時前配列にすることはできません。 CTDATA キーワードと FROMFILE キーワードを使用できません。

例: DIM(*AUTO) で定義された配列に指標として *NEXT を使用

以下の例では、配列は DIM(*AUTO) で定義されています。 ファイル CUSTFILE にフィールド NAME があると想定します。 CUSTNAMES(*NEXT) への代入時 (1)、次元は 1 だけ増やされ、NAME の値は新規要素に代入されます。

```
DCL-F custfile;
DCL-S custNames VARCHAR(10) DIM(*AUTO : 1000);

READ custfile;
DOW NOT %EOF;
   custNames(*next) = NAME;  // 1
   READ custfile;
ENDDO;
```

変動次元配列に関する考慮事項

- **警告 :** %ADDR を使用して配列または配列要素のアドレスを取得するときに要素の数が変更された場合は、そのアドレスをもう一度取得してください。
- **警告 :** 現在、デバッガーは、配列の次元が変数であることを認識しません。 この配列で使用できない要素を使用したデバッガーでの作業は回避する必要があります。

  デバッグ中に、*_ QRNU_VARDIM_ELEMS_arrayname* を評価して、要素の現在の数を判別することができます。 デバッガーでこの値を変更しても、配列内の実際の現行要素数には影響ありません。

  例えば、配列が *myArray* という名前の場合、デバッグ・セッションは次のようになります。

```
> EVAL _qrnu_vardim_elems_myarr
  _QRNU_VARDIM_ELEMS_MYARR = 0
> EVAL _qrnu_vardim_elems_myarr
  _QRNU_VARDIM_ELEMS_MYARR = 3
> EVAL myarr(1..3)
  MYARR(1) = 'Jack  '
  MYARR(2) = 'Sally '
  MYARR(3) = 'Tom   '
```

例: 新規要素の値を変更せずに、現在の要素数を増やす

通常は要素数が増えると、新規要素は配列の初期設定値で初期化されます。 新規要素のデータがすでに正しいことを分かっている場合、%ELEM の 2 番目のパラメーターとして *KEEP を指定すると、新規要素が初期化されないようにすることができます。

以下の例では、現在の要素数が 5 に設定されており (1)、新規要素は初期設定値「?」に初期化されます。

現在の要素数が 3 に設定されている場合 (2)、要素 4 と 要素 5 には、直前の代入ステートメントによって値「d」と「e」が入ります。

現在の要素数が 4 に設定されており、%ELEM の 2 番目のパラメーターとして *KEEP が指定された場合 (3)、新規要素 4 の値は変更されず、値は「d」になります。

現在の要素数が 5 に設定されており、%ELEM の 2 番目のパラメーターとして *KEEP が指定されていない場合 (4)、新規要素 5 の値は初期設定値「?」に設定されます。

```
DCL-S var_array2 CHAR(10) INZ('?') DIM(*VAR : 1000);

%elem(var_array2) = 5;       // 1
var_array2(1) = 'a';
var_array2(2) = 'b';
var_array2(3) = 'c';
var_array2(4) = 'd';
var_array2(5) = 'e';
%elem(var_array2) = 3;       // 2
%elem(var_array2:*KEEP) = 4; // 3
%elem(var_array2) = 5;       // 4
```

例: 配列に十分な要素数が割り振られるようにする

%ELEM の 2 番目のパラメーターとして *ALLOC を指定すると、配列の現在の要素数を変更せずに、配列に割り振られる要素数を増やすことができます。

以下の例では、配列を参照によってプロシージャーに渡し、一部の配列要素の値を設定します。 プロシージャーの呼び出し前、配列に割り振られる要素数は 100 に設定されています (1)。ただし、現在の要素数はゼロのままです。

プロシージャーに渡される 2 番目のパラメーターは、プロシージャーが設定できる要素数を指定します。 プロシージャーは、設定した要素数を戻します (2).

*KEEP を指定した %ELEM を使用して新しい現在の要素数を設定します -3).

```
DCL-S var_array3 CHAR(10) INZ('?') DIM(*VAR : 1000);
DCL-S num_elems INT(10);
DCL-PR proc INT(10);
   array CHAR(10) DIM(1000) OPTIONS(*VARSIZE);
   max_elems INT(10) VALUE;
END-PR;
%elem(var_array3:*ALLOC) = 100;      // 1
num_elems = proc(var_array3 : 100);  // 2
%elem(var_array3:*KEEP) = num_elems; // 3
```

DTAARA キーワード

DTAARA キーワードの構文は、自由形式仕様書と固定形式仕様書で異なります。

自由形式の独立フィールドまたはサブフィールド

DTAARA {(名前) }

426 ページの『フィールドまたはサブフィールドの自由形式 DTAARA キーワード』を参照してください。

自由形式のデータ構造

DTAARA { { (名前) } { *AUTO } { *USRCTL }}

427 ページの『データ構造の自由形式 DTAARA キーワード』を参照してください。

固定形式

DTAARA {({*VAR:} データ域名) }

428 ページの『固定形式 DTAARA キーワード』を参照してください。

注 : 引用符で囲まれていない名前がどのように扱われるのかは、自由形式定義と固定形式定義とで異なります。 DTAAARA(name) キーワードを例にとって説明します。 これが自由形式定義の中にある場合、*name* は、 名前付き定数または変数の名前であると想定され、 その名前付き定数または変数は、実行時にデータ域の名前を保持します。 これが固定形式定義の中にある場合、*LIBL/NAME が実行時のデータ域の名前 であると想定されます。

DTAARA キーワードは、独立フィールド、データ構造、データ構造サブフィールド 、またはデータ域データ構造を外部データ域と関連づけるために使用されます。DTAARA キーワードは、*DTAARA DEFINE 命令コードと同じ機能を持っています (756 ページの『*DTAARA DEFINE』 を参照)。

DTAARA キーワードはメイン・ソース・セクションでのみ使用することができます。 サブプロシージャーでは使用できません。

3 種類のデータ域を作成することができます。

- *CHAR 文字
- *DEC 数値
- *LGL 論理

さらに、リモート・システム上の上記の 3 つのタイプのいずれかのデータ域を 指し示す DDM データ域 (タイプ *DDM) も作成することができます。

データ域に関連付けることができるのは、文字タイプ、数値タイプ (浮動数値を除く)、 および標識タイプのみです。 システム上の実際のデータ域は、プログラム内のフィールドと同じタイプ で、長さと小数点以下の桁数も同じでなければなりません。 標識フィールドは論理データ域または文字データ域と 関連付けることができます。 データ域に他のタイプを保管する場合、 そのデータ域のデータ構造を使用することができ、さらにポインター以外の任意のタイプのサブフィールド をコーディングすることができます。 ポインターはデータ域には保管できません。

リテラルまたは変数でデータ域の名前を指定する

値は次のいずれの形式でも指定できます。

```
dtaaraname
libname/dtaaraname
*LIBL/dtaaraname
```

注 :

1. ライブラリー名として *CURLIB は指定できません。
2. ライブラリー名の指定なしにデータ域名を指定した場合は、*LIBL が使用されます。
3. 名前の大文字小文字は正確でなければなりません。 例えば、データ域の名前が変数に入っていて、その変数の 値が 'qtemp/mydta' である場合、データ域は検出されません。 値は 'qtemp/mydta' ではなく、'QTEMP/MYDTA' でなければなりません。

重要 : データ域データ構造の 名前に変数を指定する場合、この変数にはプログラム開始前に値が設定されている必要があります。 この値は、変数を初期化して入り口パラメーターとして渡すか、または IMPORT キーワードおよび EXPORT キーワードを使用して、変数を別のプログラムと共用すること で、設定することができます。

フィールドまたはサブフィールドの自由形式 *DTAARA* キーワード

自由形式のフィールド定義またはサブフィールド定義の DTAARA キーワードの オプション・パラメーターは、データ域の外部名です。 このパラメーターが指定されていない場合、フィールドまたはサブフィールド の名前が外部名として使用されます。

例

次の例では、データ域はパラメーターなしで定義されています。したがって、 フィールド名が外部名として使用されます。 データ域 *LIBL/MYDTAARA が実行時に使用されます。

```
DCL-DS mydtaara CHAR(100) DTAARA;
```

次の例では、DTAARA キーワードがパラメーターなしで指定されています。 したがって、サブフィールド名が外部名として使用されます。 データ域 *LIBL/MYDTAARA が実行時に使用されます。

```
DCL-DS data_struct;
   mydtaara CHAR(100) DTAARA;
END-DS;
```

データ構造の自由形式 *DTAARA* キーワード

自由形式データ構造定義での DTAARA キーワードのパラメーターは、次のとおりです。

***AUTO**
データ構造はデータ域データ構造です。 *AUTO が指定されていて、データ域を IN、OUT、または UNLOCK 命令コードで 使用したい場合は、*USRCTL パラメーターも指定する必要があります。

***USRCTL**
データ構造はユーザー制御データ域です。つまり、 IN、OUT、または UNLOCK 命令コードで使用することができます。 *AUTO パラメーターが指定されていない場合、*USRCTL がデフォルトです。 データ構造に名前がない場合、 データ構造が影響を受ける可能性があるのは、IN、OUT、または UNLOCK 命令のオペランド が *DTAARA である (これは、命令がモジュール内のすべてのユーザー制御データ域を使用して処理を行うことを 意味しています) 場合のみです。

データ域名
名前パラメーターは、リテラル、名前付き定数、または変数とすることができます。 詳しくは、426 ページの『リテラルまたは変数でデータ域の名前を指定する』 を参照してください。 名前パラメーターを指定する場合は、それが最後のパラメーターである必要があります。 名前パラメーターが指定されていない場合、データ構造名が使用されます。 データ構造名も指定されていない場合は、実行時に *LDA データ域が使用されます。

例

次の例では、データ域の定義には *AUTO パラメーターのみが指定されています。したがって、これはデータ域データ構造です。 これを IN、OUT、または UNLOCK 命令コードで使用することはできません。 名前は指定されていないため、データ域 *LIBL/MYDTAARA が実行時に使用されます。

```
DCL-DS info DTAARA(*AUTO);
   company CHAR(50);
   city CHAR(25);
END-DS;
```

次の例では、データ域は *AUTO および *USRCTL の両方のパラメーターを指定して定義されています。 したがって、これはデータ域データ構造であり、IN、OUT、または UNLOCK 命令コードで使用することもできます。 名前パラメーターが指定されているため、データ域 'MYLIB/MYDTAARA' が実行時に使用されます。

```
DCL-DS info DTAARA(*AUTO : *USRCTL : 'MYLIB/MYDTAARA');
   company CHAR(50);
   city CHAR(25);
END-DS;
```

次の例では、データ域は名前を指定せずに定義されています。 DTAARA キーワードには *AUTO パラメーターのみが指定されています。 データ域名はデータ構造名にも DTAARA キーワードにも指定されていないため、 *LDA データ域が実行時に使用されます。

```
DCL-DS *N DTAARA(*AUTO);
   company CHAR(50);
   city CHAR(25);
END-DS;
```

次の例では、データ構造には名前がなく、 DTAARA キーワードに *AUTO パラメーターは使用されていません。 IN 命令は *DTAARA を指定しています。したがって、実行時に IN 命令が使用されると、 名前のないデータ域にデータ域 MYLIB/MYDTAARA が読み込まれます。 IN 命令の実行後、サブフィールド *subf* に、データ域の内容が保持されます。

```
DCL-DS *N DTAARA('MYDTAARA');
   subf CHAR(50);
END-DS;

IN *DTAARA;
DSPLY subf;
```

固定形式 *DTAARA* キーワード

固定形式仕様書内では DTAARA キーワードを以下の方法で指定できます。

DTAARA
データ域の外部名は指定されないため、 7 から 21 桁目に指定された名前が外部データ域の名前にもなります。 パラメーターもデータ構造名も指定されていない場合、デフォルト の値は *LDA になります。

DTAARA(名前)
実行時のデータ域の外部名として、名前 パラメーターが使用されます。 例えば、DTAARA(mydtaara) と指定すると、 データ域 *LIBL/MYDTAARA が実行時に使用されます。 *LDA または *PDA を DTAARA キーワードのパラメーターとして指定することもできます。

DTAARA(文字リテラル)
文字リテラル パラメーターの値によって、実行時の データ域の名前が決まります。 詳しくは、426 ページの『リテラルまたは変数でデータ域の名前を指定する』 を参照してください。

DTAARA(*VAR : 名前)
名前 パラメーターには、名前付き定数または変数を指定できます。 詳しくは、426 ページの『リテラルまたは変数でデータ域の名前を指定する』 を参照してください。

DTAARA キーワードを指定した場合には、データ域に対して IN、OUT、および UNLOCK 命令コードを使用することができます。

データ構造の定義仕様書の 23 桁目に U が記入されている場合、 そのデータ構造はデータ域データ構造です。

EXPORT{(外部名)}

EXPORT キーワードの指定によって、グローバルに定義されたデータ構造またはモ ジュール内で定義済みの独立フィールドをプログラム中の別のモジュールによっ て使用することができます。 データ項目のための記憶域は、EXPORT 定義を含むモジュールの中で割り振られます。 外部名パラメーターを 指定する場合は、文字リテラルまたは定数でなければなりません。

自由形式定義では、 外部名 パラメーターに *DCLCASE を指定して、 項目の外部名が、大/小文字も含めて項目の名前と同じであることを指示できます。 439 ページの『外部名としての *DCLCASE の指定』 を参照してください。

定義仕様書の EXPORT キーワードは、データ項目をエクスポートするために 使用されますが、プロシージャー名のエクスポートには使用できません。 プロシージャー名をエクスポートする には、プロシージャー仕様書の EXPORT キーワードを使用します。

注 : 記憶域の初期化は、プログラム入り口プロシージャー (モジュールが含まれてい るプログラムの一部) が初めて呼び出された時に行われます。 前の呼び出しでプロシージャーが LR のオンによって終了したか、あるいは異常終 了したとしても、RPG IV は、最初の呼び出し時以後はこの記憶域に対するいかなる 初期化も行ないません。

EXPORT を指定する時には、次の制約事項が適用されます。

- データ項目をエクスポート・データとして定義できるのは 1 つのモジュール だけです。
- *ENTRY PLIST の中の PARM の結果のフィールド記入項目に指定された フィールドをエクスポートすることはできません。
- 名前のないデータ構造をエクスポートすることはできません。
- 基底付きデータ項目をエクスポートすることはできません。
- 同じ外部フィールド名をモジュールごとに複数回指定することは できません。また、外部プロシージャー名として使用することもできません。
- IMPORT と EXPORT の両方を同じデータ項目に指定することは できません。

複数オカレンス・データ構造またはテーブルの場合には、各モジュールにオカレンス番号 またはテーブル指標のそれぞれ固有のコピーが入れられます。 オカレンス番号または 指標は各モジュールごとにローカルなものであるため、どのモジュールで の OCCUR または LOOKUP 命令もローカルな影響しか与えません。

441 ページの『IMPORT{(外部名)}』 も参照してください。

ヒント :

キーワードの IMPORT と EXPORT を使用すると、モジュール間 に「隠れた」インターフェースを定義できます。 この結果として、これらのキーワードの使用 を、該当のアプリケーション全体にグローバルであるデータ項目だけに 限定する必要が生じます。 また、このグローバル・データは、1 回設定されると、それ以外の場所で変更されることのない、グローバル属性のようなものに限定するようにお勧めします。

EXT

EXT キーワードは、自由形式のデータ構造定義で使用されて、 データ構造が外部記述であることを示します。 EXTNAME キーワードが一緒に指定されていない 場合、サブフィールド定義が入っている外部ファイルを見つけるためにデータ構造の名前が使用されます。

EXT キーワードを指定する場合は、それが最初のキーワードである必要があります。

EXTFLD{(フィールド名)}

EXTFLD キーワードは、外部記述データ構造の中のサブフィールドの名前を変更す るために使用されます。また、自由形式定義では、サブフィールドが外部サブフィールドであることを 示すためにも使用されます。

EXTFLD キーワードのパラメーターとして、サブフィールドの外部名を入力します。

- 自由形式定義では、EXTFLD キーワードは最初のキーワードでなければなりません。 サブフィールドの名前がフィールドの外部名と同じである場合、 パラメーターは任意指定です。 指定する場合、外部名は文字リテラルまたは名前付き定数として指定する必要があります。 名前付き定数である場合、定義ステートメントの前にその定数が定義されている必要があります。
- 固定形式定義では、EXTFLD キーワードのパラメーターとしてサブフィールドの外部名 を記入し、プログラムで使用される名前を名前フィールド (7 から 21 桁目) に指定します。 外部名には、単純名または文字リテラルが可能です。

文字リテラルを指定する場合は、外部名の大/小文字を正確に指定する必要があります。 例えば、外部名が MYFIELD である場合、固定形式定義では 大/小文字混合の名前 (myField や myfield など) をフィールド名パラメーターに指定できますが、 リテラルとして指定する場合は「MYFIELD」でなければなりません。

次の例では、ファイル内の 3 つの外部フィールド名 は、*NAME*、*ADR*、 および *ID* です。

1. サブフィールド *name* は 'UNKNOWN' に初期設定されます。
   a. 自由形式版のコードでは、サブフィールド名は外部名と同じであるため、EXTFLD のパラメーターは必要ありません。

b. 固定形式版のコードでは、このサブフィールドは名前変更されないため、EXTFLD キーワード は指定されていません。

2. サブフィールド *address* は外部フィールド *ADR* から名前変更されます。
   a. 自由形式版のコードでは、外部名はリテラルとして指定されています。
   b. 固定形式版のコードでは、外部名は単純名として指定されています。
3. サブフィールド *id_number* は 外部フィールド *ID* から名前変更され、-1 に初期設定されます。
   a. 自由形式版のコードでは、 外部名は値が「ID」である名前付き定数 *ID_EXT_NAME* として指定されています。
   b. 固定形式版のコードでは、 外部名はリテラルとして指定されています。

```
 DCL-C ID_EXT_NAME 'ID'; 3a
 DCL-DS custInfo EXTNAME('CUSTMAST');
    name EXTFLD INZ('UNKNOWN'); 1a
    address EXTFLD('ADR'); 2a
    id_number EXTFLD(ID_EXT_NAME) INZ(-1); 3a
 END-DS;

D custInfo      E DS                  EXTNAME(custMast)
D  name         E                     INZ('UNKNOWN') 1b
D  address      E                     EXTFLD(adr) 2b
D  id_number    E                     EXTFLD('ID') INZ(-1) 3b
```

名前が有効な単純 RPG 名でない場合には、リテラルとして指定する必要があります。 例えば、外部フィールド A.B を名前変更する場合には、EXTFLD('A.B') と指定します。

このキーワードは任意指定です。 指定されていない場合には、外部定義から抜き出された名前がデータ構造サブフ ィールド名として使用されます。

データ構造に PREFIX キーワードが指定されている場合、EXTFLD によって名前が変更されるフィールドには接頭部は適用されません。414 ページの図 127 に、ALIAS キーワードと一緒に EXTFLD キーワードを指定した例が示されています。

EXTFMT(コード)

EXTFMT キーワードは、コンパイル時および実行時前数値配列およびテーブルの外 部データ・タイプを指定するために使用されます。 外部データ・タイプは、ファイル中のレコードの中のデータの形式です。 この指定によって、プログラムの中で配列またはテーブルの内部処理に使用され る形式 (内部形式) が影響を受けることはありません。

注 : EXTFMT に指定された値は、TOFILE と FROMFILE の両方のキーワードによっ て指定された名前が異なっていても、それらのキーワードで識別されたファイル に適用されます。

パラメーターに使用できる値は次のとおりです。

B
配列またはテーブルのデータは 2 進数形式です。

C
配列またはテーブルのデータは UCS-2 形式です。

I
配列またはテーブルのデータは整数形式です。

L
数値配列またはテーブル要素のデータには先行 (左側) のプラス符号または マイナス符号があります。

R
数値配列またはテーブル要素のデータには後書き (右側) のプラス符号また はマイナス符号があります。

P
配列またはテーブルのデータはパック 10 進数形式です。

S
配列またはテーブルのデータはゾーン 10 進形式です。

U
配列またはテーブルのデータは符号なし形式です。

F
配列またはテーブルのデータは浮動数値形式です。

注 :

1. EXTFMT キーワードが指定されていない場合、外部形式のデフォルトの値として、非浮動配列およびテーブルの場合は 'S' が使用され、浮動実行前配列 およびテーブルの場合は外部表示浮動表現が使用されます。
2. コンパイル時配列およびテーブルの場合は、 データ・タイプが浮動でない限り、使用できる値は S、L、および R です。 データ・タイプが浮動の場合は、EXTFMT キーワードは使用できません。
3. EXTFMT(I) または EXTFMT(U) を使用すると、1 から 5 桁 となるように定義された配列は、要素ごとに 2 バイトを 占めます。 6 から 10 桁となるように定義された配列は、要素ごとに 4 バイトを 占めます。 11 から 20 桁となるように定義された配列は、要素ごとに 8 バイトを 占めます。
4. UCS-2 配列のデフォルトの外部形式は文字です。 UCS-2 コンパイル時データに使用できる文字数は、UCS-2 配列内 の 2 バイト文字の数です。 データの中に図形データが組み込まれている場合に、そのデータの 中に 2 バイト・データとシフトアウト文字およびシフトイン文字がある と、配列要素内に配置できる実際のデータ量は少なくなり、要素の残りの 部分にはブランクが埋め込まれます。 たとえば、4 文字の UCS-2 配列の場合、コンパイル時データ内 に指定できる 2 バイト文字は 1 つだけです。コンパイル時データ が 'oXXi' (ただし 'XX' は UCS-2 文字の U'yyyy' に変換されます) の 場合、UCS-2 要素には、値 U'yyyy002000200020' が入ります。

EXTNAME(ファイル名{:形式名}{:*ALL| *INPUT|*OUTPUT|*KEY|*NULL})

EXTNAME キーワードは、定義中のデータ構造のサブフィールド記述として使用さ れるフィールド記述が入っているファイルの名前を指定するために使用されます。

ファイル名パラメーターは必須です。 オプションで形式名を指定して、 コンパイラーにファイル内の特定の形式を指示す ることができます。 形式名パラメーターが指定されない場合には、最初のレコード様式が使用されます。

- 自由形式定義では、ファイル名パラメーターおよび形式名パラメーター は、文字リテラルか、または文字リテラルを表す名前付き定数でなければなりません。 パラメーターが名前付き定数である場合、その定数は 定義ステートメントの前に定義されている必要があります。
- 固定形式定義では、ファイル名パラメーターおよび形式名パラメーター には、名前または文字リテラルのいずれかを指定できます。

文字リテラルを指定する場合は、ファイル名または形式名の大/小文字を正確に指定する必要があります。例えば、MYFILE という外部ファイルの場合、そのファイル名パラメーターについては、myFile、myfile などのように大/小文字を組み合わせて指定することもできますが、リテラルとして指定するのであれば「MYFILE」としなければなりません。 ファイル名が文字リテラルの場合は、以下のいずれかの形式になります。

```
'LIBRARY/FILE'
'FILE'
'*LIBL/FILE'
```

残りの抜き出しタイプ・パラメーターは、外部レコード内のどのフィールドを抜き出すか指定します。

- *ALL はすべてのフィールドを抜き出します。
- *INPUT は入力可能フィールドのみを抜き出します。
- *OUTPUT は出力可能フィールドのみを抜き出します。
- *KEY はキー・フィールドのみを抜き出します。

ファイル内のフィールドと同じデータ・タイプを持つサブフィールドを定義するのではなく、それらのサブフィールドがすべて標識であることを示すために、*NULL を指定することもできます。 データベース・ファイルの場合、これらの標識のレイアウトは、レコードのヌル・バイト・マップと同じです。

抜き出しタイプが指定されていない場合、または *NULL のみが指定されている場合、コンパイラーは入力バッファーのフィールドを抜き出します。

注 :

1. 形式名が指定されていない場合、レコードは、デフォルトの値であるファイル内の最初のレコードになります。
2. *INPUT および *OUTPUT の場合、*NULL が指定されなければ、データ構造に含まれるサブフィールドは、外部レコード記述内と同じ開始位置を占めます。 *NULL が指定されると、標識はデータベース・ファイルの外部レコードのヌル・バイト・マップ内のヌル標識と同じ開始位置を占めます。 その他のタイプのファイルの場合は、標識サブフィールドの開始位置は順次に割り当てられます。

外部記述データ構造 (自由形式定義で EXT キーワードがあるか、 自由形式定義で 22 桁目に E がある) であり、EXTNAME キーワードが 指定されていない場合、データ構造名が外部名に使用されます。

*NULL が指定されないと、コンパイラーは外部記述データ構造のすべてのフィールドについて、次の定義仕様書項目を生成します。

- サブフィールド名 (データ構造に ALIAS キーワードが指定されているか、EXTFLD キーワードによってフィールドが名前変更されているか、 あるいは定義仕様書の PREFIX キーワードを使用して接頭部を適用する場合を除いて、名前は 外部名と同じになります。)
- サブフィールドの長さ
- サブフィールドの内部データ・タイプ (タイプに CVTOPT 制御仕様キーワードおよびコマンド・パラメーターが指定されていない限り、外部タイプと同じものになります。 その場合、データ・タイプは文字になります)。

*NULL が指定されると、同じ方法でサブフィールド名が生成されます。 ただし、長さは 1 になり、データ・タイプは標識になります。

LIKEDS および LIKEREC を除くすべてのデータ構造キーワードは、EXTNAME キーワードと一緒に使用すること ができます。

ただし、*NULL が指定された場合は、CCSID(*EXACT) を指定することはできません。

抜き出しタイプ *NULL で定義されたデータ構造を入出力操作と一緒に使用することはできません。

抜き出しタイプなしで定義されたデータ構造を入出力操作と一緒に使用することはできません。

EXTPGM{(名前)}

EXTPGM キーワードは、プロトタイプが動的プログラム呼び出しを表すことを示します。

パラメーターは、プロトタイプが定義されているプログラムの 外部名を指定します。 この名前は文字定数または 文字変数とすることができます。

プロトタイプ名が 10 文字以下の場合、パラメーターは 任意指定です。 パラメーターが指定されていない場合、外部プログラム名は、大文字化した形式のプロトタイプ名と同じです。 次の例では、プログラム「QCMDEXC」のためのプロトタイプを定義しています。

```
DCL-PR qcmdExc EXTPGM;
   ...
```

プロトタイプに対して EXTPGM も EXTPROC も指定されていない場合、 コンパイラーは、プロシージャー用のプロトタイプを定義していると 想定し、大文字化した形式のプロトタイプ名を外部プロシージャー名に 割り当てます。

EXTPGM によりプロトタイプまたはプロシージャー・インターフェースで定義されたすべてのパラメーターは、参照によって 渡すことが必要です。 さらに、戻り値を定義することはできません。

EXTPROC{({*CL|*CWIDEN|*CNOWIDEN|*JAVA:クラス名:}名前|*DCLCASE)}

EXTPROC キーワードは、次の形式のいずれかが可能です。

EXTPROC
EXTPROC キーワードをパラメータなしで指定した場合、呼び出されるプロシージャーの外部名は、プロトタイプの名前の大文字の形式、または EXTPROC キーワードがプロシージャー・インターフェース定義に指定されている場合はプロシージャーの名前になります。

EXTPROC(*CL:名前)
ILE CL で書かれた外部プロシージャー、または ILE CL によって呼び出される RPG プロシージャーを指定します。 ユーザーのプログラムで使用される戻り値のデータ・タイプについて、CL と RPG とで 異なる処理が行われる場合には、*CL を使用します。 たとえば、戻り値が 1A のときに CL プロシージャーによって呼び出される RPG プロシージャーをプロトタイプする場合に、*CL を使用します。

EXTPROC(*CWIDEN:名前|*CNOWIDEN:名前)
ILE C で書かれた外部プロシージャー、または ILE C によって呼び出される RPG プロシージャーを指定します。

ユーザーのプログラムが、C で RPG とは異なる処理がされるデータ・タイプの値によって受け渡される戻り値またはパラメーターを使用する場合には、*CNOWIDEN または *CWIDEN を使用します。 C によって呼び出される RPG プロシージャーを定義するとき、あるいは C プロシージャーのプロトタイプを定義するときに、 戻り値または値によって渡されるパラメーターが 1A、1G または 1C、5U、5I、あるいは 4F である場合に、*CWIDEN または *CNOWIDEN を使用します。

ILE C ソースがそのプロシージャーに対して `#pragma argument(procedure-name,nowiden)` を含んでいる場合は *CNOWIDEN を使用し、そうでない場合は *CWIDEN を使用します。

EXTPROC(*JAVA:クラス名:名前)
Java で書かれたメソッド、または Java によって呼び出される RPG ネイティブ・メソッドを指定します。 最初のパラメーターは *JAVA です。 2 番目のパラメーターは、メソッドのクラスを含む文字定数です。 3 番目のパラメーターは、メソッド名を含む文字定数です。 特別なメソッド名 *CONSTRUCTOR は、そのメソッドがコンストラクターであることを示します。 つまり、このメソッドはクラスをインスタンス化する (新たなクラス・インスタンスを作成する) のに使用できます。

Java プロシージャーの呼び出しについて詳しくは、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」を参照してください。

EXTPROC(名前)
RPG または COBOL で書かれているか、あるいは RPG または COBOL によって呼び出される、外部プロシージャーを指定します。 この形式は、RPG、COBOL、C、または CL のどれから呼び出されることもできるプロシージャーにも使用されます。 この場合、戻り値およびパラメーターが、前述した *CL、*CWIDEN、および *CNOWIDEN のいずれの場合にもいかなる問題も起きないようにしておくことが必要です。

EXTPROC キーワードは、プロトタイプが定義されているプロシージャーの外部名 を指示します。 この名前は文字定数、またはプロシージャー・ポインターとすることができます。

自由形式定義では、*DCLCASE を名前として 指定することによって、外部名は定義している名前から派生し、名前で使用されたのと同じ大/小文字 になることを指示できます。439 ページの『外部名としての *DCLCASE の指定』を参照してください。

EXTPROC を指定した場合には、バインド呼び出しが行われます。

EXTPGM と EXTPROC のどちらも指定しなかった場合には、コンパイラーはプロシージャーが定義されているものと見なし、プロトタイプの名前の大文字の形式、または EXTPROC キーワードがプロシージャー・インターフェース定義に指定されている場合はプロシージャーの名前をそのプロシージャーに割り当てます。

EXTPROC に指定された名前 (または、EXTPGM も EXTPROC も指定されていないか、EXTPROC(*DCLCASE) が指定されている場合は、プロトタイプ名またはプロシージャー名) が「CEE」または下線 ('_') で始まっている場合、コンパイラーは、これをシステム組み込みとして扱います。 システム提供の API との混同を避けるために、ユーザーのプロシージャーに "CEE" で始まる名前を付けないでください。

たとえば、プロシージャー SQLAllocEnv (サービス・プログラム QSQCLI の 中にある) のプロトタイプを定義するには、次の定義仕様書をコーディングする ことができます。

```
      D SQLEnv          PR                  EXTPROC('SQLAllocEnv')
```

プロシージャー・ポインターを指定する場合は、呼び出しで使用する前に、 有効なアドレスを割り当てておかなければ なりません。 これは、その戻り値とパラメーターにプロトタイプ定義との整合性があるプロシ ージャーを指していることが必要です。

プロシージャーのプロトタイプが指定される場合、EXTPROC キーワードはそのプロトタイプに対して指定されます。 それ以外の場合、EXTPROC キーワードはプロシージャー・インターフェースに対して指定されます。 プロシージャーが別の RPG モジュールから呼び出される場合にのみ、プロトタイプの明示的な指定が必要になります。 プロシージャーが同じモジュール内からのみ呼び出される場合、または RPG 以外の呼び出し元のみによって呼び出される場合には、 プロシージャー・インターフェースからプロトタイプを暗黙的に派生させることができます。

434 ページの図 128 はパラメーターとしてプロシージャー・ポインター を持っている EXTPROC キーワードの例を示しています。

```
 * Assume you are calling a procedure that has a procedure
 * pointer as the EXTPROC. Here is how the prototype would
 * be defined:
D DspMsg          PR            10A   EXTPROC(DspMsgPPtr)
D  Msg                         32767A
D  Length                          4B 0 VALUE
 * Here is how you would define the prototype for a procedure
 * that DspMsgPPtr could be assigned to.
D MyDspMsg        PR                  LIKE(DspMsg)
D  Msg                         32767A
D  Length                          4B 0 VALUE
 * Before calling DSPMSG, you would assign DSPMSGPPTR
 * to the actual procedure name of MyDspMsg, that is
 * MYDSPMSG.
C                   EVAL      DspMsgPPtr = %paddr('MYDSPMSG')
C                   EVAL      Reply = DspMsg(Msg, %size(Msg))
 ...
P MyDspMsg        B
```

図 *128.* プロシージャー・ポインターを持つ *EXTPROC* の使用

```
char RPG_PROC (short s, float f);
char C_PROC (short s, float f);
#pragma argument(RPG_PROC, nowiden)
#pragma argument(C_PROC, nowiden)

/* "fn" calls the RPG procedure with unwidened parameters,       */
/* and expects the return value to be passed according to C      */
/* conventions.                                                  */
void fn(void)
{
   char c;

   c = RPG_PROC(5, 15.3);
}

/* Function C_PROC expects its parameters to be passed unwidened.*/
/* It will return its return value using C conventions.          */
char C_PROC (short s, float f);
{
   char c = 'x';

   if (s == 5 || f < 0)
   {
      return 'S';
   }
   else
   {
      return 'F';
   }
}
```

図 *129. *CNOWIDEN を指定した EXTPROC の使用 - C コーディング*

```
D RPG_PROC        PR             1A   EXTPROC(*CNOWIDEN : 'RPG_PROC')
D   short                        5I 0 VALUE
D   float                        4F   VALUE

D C_PROC          PR             1A   EXTPROC(*CNOWIDEN : 'C_PROC')
D   short                        5I 0 VALUE
D   float                        4F   VALUE

P RPG_PROC        B                   EXPORT
D                 PI             1A
D   short                        5I 0 VALUE
D   float                        4F   VALUE

D   char          S              1A

 * Call the C procedure
C                   EVAL      c = C_PROC(4 : 14.7)

 * Return the value depending on the values of the parameters
C                   IF        short < float
C                   RETURN    'L'
C                   ELSE
C                   RETURN    'G'
C                   ENDIF

P                 E
```

図 *130. *CNOWIDEN* を指定した *EXTPROC* の使用 - *RPG* コーディング

```
char RPG_PROC (short s, float f);
char C_PROC (short s, float f);

/* Function "fn" calls the RPG procedure with widened parameters,*/
/* and expects the return value to be passed according to C      */
/* conventions.                                                  */
void fn(void)
{
   char c;

   c = RPG_PROC(5, 15.3);
}

/* Function C_PROC expects its parameters to be passed widened.  */
/* It will return its return value using C conventions.          */
char C_PROC (short s, float f);
{
   char c = 'x';

   if (s == 5 || f < 0)
   {
      return 'S';
   }
   else
   {
      return 'F';
   }
}
```

図 *131. *CWIDEN を指定した EXTPROC の使用 - C コーディング*

```
D RPG_PROC        PR             1A   EXTPROC(*CWIDEN : 'RPG_PROC')
D   short                        5I 0 VALUE
D   float                        4F   VALUE

D C_PROC          PR             1A   EXTPROC(*CWIDEN : 'C_PROC')
D   short                        5I 0 VALUE
D   float                        4F   VALUE

P RPG_PROC        B                   EXPORT
D                 PI             1A
D   short                        5I 0 VALUE
D   float                        4F   VALUE

D   char          S              1A

 * Call the C procedure
C                   EVAL      c = C_PROC(4 : 14.7)

 * Return the value depending on the values of the parameters
C                   IF        short < float
C                   RETURN    'L'
C                   ELSE
C                   RETURN    'G'
C                   ENDIF

P                 E
```

図 *132. *CWIDEN* を指定した *EXTPROC* の使用 - *RPG* コーディング

```
         /* CL procedure CL_PROC */
         DCL &CHAR1 TYPE(*CHAR) LEN(1)

         /* Call the RPG procedure */
         CALLPRC RPG_PROC RTNVAR(&CHAR1)
```

図 *133. *CL* を指定した *EXTPROC* の使用 - *CL* コーディング

```
D RPG_PROC        PR             1A   EXTPROC(*CL : 'RPG_PROC')

P RPG_PROC        B                   EXPORT
D                 PI             1A

C                   RETURN    'X'

P                 E
```

図 *134. *CL* を指定した *EXTPROC* の使用 - *RPG* コーディング

```
P isValidCust     B                   EXPORT
D                 PI               N  EXTPROC(*CL : 'isValidCust')
D   custId                        10A  CONST
D isValid         S                N  INZ(*OFF)
 /free
    ... calculations using the "custId" parameter
    return isValid;
 /end-free
P                 E
```

図 *135. CL* 呼び出し元のみによって呼び出されるプロシージャーのプロシージャー・インターフェースにおける *EXTPROC* の使用

外部名としての **DCLCASE* の指定

自由形式定義では、EXTPROC、EXPORT、および IMPORT キーワードの外部名として *DCLCASE を指定できます。

EXPORT および IMPORT キーワードでの *DCLCASE

項目の外部名は項目の名前と同じであり、大/小文字も名前で使用されているのと同じです。

次に例を示します。

- エクスポートされたフィールド *currentCity* の外部名は「currentCity」です。
- インポートされたデータ構造 *customerOptions* の外部名は「customerOptions」です。

```
DCL-S currentCity LIKE(name_T) EXPORT(*DCLCASE);

DCL-DS customerOptions LIKEDS(custOpt_T) IMPORT(*DCLCASE);
```

EXTPROC キーワードでの *DCLCASE

プロシージャーまたはメソッドの外部名は、EXTPROC キーワード を含んでいるプロトタイプまたはプロシージャー・インターフェースの名前と同じであり、 大/小文字も名前で使用されているのと同じです。EXTPROC キーワードがプロシージャー・インターフェース定義に 指定されていて、そのプロシージャー・インターフェースに名前が付いていない場合、外部名はプロシージャーの名前と同じであり、大/小文字 も DCL-PROC ステートメントに指定されているプロシージャー名と同じです。

次に例を示します。

1. *getCustomerCity* プロトタイプの 外部名は「getCustomerCity」です。
2. *getBytes* プロトタイプ のメソッド名は「getBytes」です。

3. *getNextOrder* プロシージャーの外部名は、プロシージャーの最初にある DCL-PROC ステートメントから、「getNextOrder」になります (3a)。これは、プロシージャー・インターフェースにプロシージャー名が指定されていないためです (3b).
4. *addQuotes* プロシージャー の外部名は、プロシージャー・インターフェース・ステートメントから、「addQuotes」 になります (4b)。これは、このプロシージャー・インターフェースにプロシージャー名が指定されているためです。 DCL-PROC ステートメントに指定されている大/小文字が違う名前 (4a) は無視されます。

```
DCL-PR getCustomerCity EXTPROC(*DCLCASE); 1
   ...
DCL-PR getBytes EXTPROC(*JAVA : 'java.lang.String' : *DCLCASE); 2
   ...
DCL-PROC getNextOrder; 3a
   DCL-PI *N EXTPROC(*DCLCASE); 3b
     ...
DCL-PROC ADDQUOTES 4a
   DCL-PI addQuotes EXTPROC(*DCLCASE); 4b
     ...
```

FLOAT(バイト数)

FLOAT キーワードは、数字データ・タイプのキーワードの 1 つです。 これは自由形式定義で使用されて、項目の形式が浮動形式であることを示します。

これは最初のキーワードでなければなりません。

パラメーターは長さをバイト単位で指定します。 これは 4 または 8 のいずれかで なければなりません。

パラメーターにはリテラルまたは名前付き定数を指定できます。 名前付き定数である場合、定義ステートメントの前にその定数が定義されている必要があります。

次に例を示します。

- フィールド *variance* は、8 バイトの浮動フィールドとして定義されています。

```
DCL-S variance FLOAT(8);
```

FROMFILE(ファイル名)

FROMFILE キーワードは、定義している実行時前配列またはテーブル用の入力データ を持つファイルを指定するために使用されます。 FROMFILE キーワードは、プログラムで使用される実行時前配列またはテーブルの それぞれについて指定しなければなりません。

485 ページの『TOFILE(ファイル名)』 も参照してください。

GRAPH(長さ)

GRAPH キーワードは、自由形式定義で使用されて、 項目が固定長グラフィックであることを示します。

これは最初のキーワードでなければなりません。

パラメーターは、長さを 2 バイト文字単位で指定します。 1 から 8,386,552 までの値を指定できます。

パラメーターにはリテラルまたは名前付き定数を指定できます。 名前付き定数である場合、定義ステートメントの前にその定数が定義されている必要があります。

次に例を示します。

- フィールド *cust_name* は、100 文字の固定長グラフィック・フィールドとして定義されています。
- フィールド *message* は、5000 文字の固定長グラフィック・フィールドとして定義されています。 長さは、名前付き定数 *MSG_LEN* を使用して定義されています。

```
DCL-S cust_name GRAPH(100);
DCL-C MSG_LEN 5000;
DCL-S message GRAPH(MSG_LEN);
```

可変長グラフィック項目の定義について詳しくは、487 ページの『VARGRAPH(長さ {:2 | 4})』を参照してください。

IMPORT{(外部名)}

IMPORT キーワードは、定義しているデータ項目用の記憶域は別のモジュールで割 り振られますが、このモジュールでアクセスできることを指定します。 外部名パラメーターを 指定する場合は、文字リテラルまたは定数でなければなりません。

自由形式定義では、 外部名 パラメーターに *DCLCASE を指定して、 項目の外部名が、大/小文字も含めて項目の名前と同じであることを指示できます。 439 ページの『外部名としての *DCLCASE の指定』を参照してください。

名前はインポートされるものとして定義されていますが、その名前のエクスポート 定義が含まれているモジュールがプログラムの中にない場合には、連係時にエラ ーが起こります。428 ページの『EXPORT{(外部名)}』を参照してください。

定義仕様書の IMPORT キーワードは、データ項目をインポートするために 使用されますが、プロシージャー名のインポート には使用できません。 プロシージャー名は、EXPORT キーワードがプロシージャー仕様書 に指定されているとき、プログラム内のすべてのモジュールに暗黙的に インポートされます。

IMPORT が指定された時には、次の制約事項が適用されます。

- データ項目を初期化することはできません (INZ キーワードは使用でき ません)。 エクスポート・モジュールがデータに関するすべての初期化を管理します。
- インポート・フィールドをコンパイル時または実行時前配列またはテーブル 、あるいはデータ域として定義することはできません。 (キーワード CTDATA、 FROMFILE、TOFILE、EXTFMT、PERRCD、および DTAARA を使 用することはできません。)
- エクスポート・モジュールで初期値が定義されているので、インポート・フ ィールドを RESET 命令コードに対する引数として指定することはできません。
- *ENTRY PLIST の中の PARM の結果のフィールドにインポート・フィールドを 指定することはできません。
- インポート・フィールドを基底付きとして定義することはできません (キー ワード BASED は使用できません)。
- このキーワードは名前のないデータ構造には使用できません。
- 使用できる他のキーワードは DIM、EXTNAME、LIKE、OCCURS、 および PREFIX だけです。
- 同じ外部フィールド名をモジュールごとに複数回指定することは できません。また、外部プロシージャー名として使用することもできません。

複数オカレンス・データ構造またはテーブルの場合には、各モジュールにオカレンス番号 またはテーブル指標のそれぞれ固有のコピーが入れられます。 オカレンス番号または 指標は各モジュールごとにローカルなものであるため、どのモジュールで の OCCUR または LOOKUP 命令もローカルな影響しか与えません。

INT(桁数)

INT キーワードは、数字データ・タイプのキーワードの 1 つです。 これは自由形式定義で使用されて、項目の形式が符号付き整数形式であることを示します。

これは最初のキーワードでなければなりません。

パラメーターは長さを桁数で指定します。 これは、3、5、10、または 20 (それぞれ、1 バイト、2 バイト、4 バイト、8 バイトを占めます) のいずれかでなければなりません。

パラメーターにはリテラルまたは名前付き定数を指定できます。 名前付き定数である場合、定義ステートメントの前にその定数が定義されている必要があります。

次に例を示します。

- フィールド *num_elems* は、10 桁の整数フィールドとして定義されています。 これは記憶域で 4 バイトを占めます。

```
DCL-S num_elems INT(10);
```

IND

IND キーワードは、自由形式定義で使用されて、 項目が標識であることを示します。

これは最初のキーワードでなければなりません。

次に例を示します。

- フィールド *found* は、標識として定義されています。

```
DCL-S found IND;
```

INZ{(初期値)}

INZ キーワードは、独立フィールド、データ構造、データ構造サブフィールド、またはオブジェクトをそのデータ・タイプごとのデフォルトの値、あるいはオプションで、括弧の中に指定された定数に初期化します。

- プログラム記述データ構造の場合、INZ キーワードにパラメーターを指定することはできません。
- 外部記述データ構造の場合は、*EXTDFT パラメーターだけが 指定できます。
- LIKEDS キーワードを使用して定義されているデータ構造の場合、値 *LIKEDS は、サブフィールドが親データ構造と同じ方法で初期化されることを指定します。これは、親サブフィールドにおいて INZ キーワードで指定された初期化についてのみ適用されます。 CTDATA キーワードまたは FROMFILE キーワードで指定された初期化については適用されません。 親データ構造に CTDATA または FROMFILE で初期化されたサブフィールドがいくつか存在する 場合、INZ(*LIKEDS) を使用して初期化されたデータ構造には、CTDATA データまたは FROMFILE データは存在しません。
- オブジェクトの場合は *NULL パラメーターのみが許可されます。 すべてのオブジェクトは、INZ(*NULL) を指定しているかどうかにかかわらず *NULL に初期化されます。

指定される初期値は、初期化されるタイプと整合性がなければなりません。 この初期値は、リテラル、名前付き定数、形象定数、組み込み関数、または 特殊値 *SYS、*JOB、*EXTDFT、*USER、*LIKEDS、または *NULL のいずれかが可能です。 日付または時刻データ・タイプ・フィールドあるいは日付または時刻の値を持つ 名前付き定数を初期化する場合には、日付または時刻フィールドの実際の 形式とは無関係に、リテラルの形式には制御仕様書から引き出されたとおりのデフォルト の形式との整合性がなければなりません。

UCS-2 フィールドは、文字、UCS-2、または図形定数によって初期化されます。 定数が UCS-2 ではない場合、コンパイル時にコンパイラーによって暗黙的に UCS-2 へ変換されます。

数値フィールドは、どのタイプの数値リテラルでも初期化する ことができます。 しかし、浮動リテラルは浮動フィールドでしか使用する ことはできません。 数値フィールドは、16 桁以下 の 16 進数リテラルによって初期化できます。 この場合、16 進数リテラルは符号なし数値と見なされます。

INZ(*EXTDFT) を指定すると、DDS 内の DFT キーワードからの デフォルト値を使用して、外部記述データ構造サブフィールドが初期化されます。 DFT または定数値が指定されない場合、フィールド・タイプの DDS デフォルト値 が使用されます。 サブフィールド仕様のパラメーターを指定して、または指定せずに INZ を コーディングすることにより、DDS 内で指定されている値を一時変更できます。

外部データ構造定義に INZ(*EXTDFT) を指定すると、外部記述サブフィールド がすべて、その DDS デフォルト値に初期化されます。 外部記述データ構造に追加のプログラム記述サブフィールドが ある場合、これらは RPG デフォルト値に初期化されます。

INZ(*EXTDFT) を使用するときは、次の点に留意してください。

- 日付または時刻フィールドの DDS 値が RPG 内部形式で ない場合、その値は事実上プログラムの内部形式に変換されます。
- 外部記述は物理ファイル内になければなりません。
- DDS 内のヌル可能フィールドに *NULL が指定 されている場合、コンパイラーは初期値として、そのフィールドの DDS デフォルト値 を使用します。
- 可変長フィールドに DFT('') が指定されている場合、そのフィールド は長さ 0 のストリングを使用して初期化されます。
- INZ(*EXTDFT) は、CVTOPT オプションが有効である場合には使用 できません。

INZ(*USER) を指定すると、文字フィールドまたはサブフィールド はどれも、現行ユーザー・プロファイルの名前に初期化されます。 文字フィールドの長さは少なくとも 10 文字にする必要があります。 フィールドの長さが 10 文字より長い場合、ユーザー名 はフィールド内で左寄せされ、残りの部分にはブランクが使用されます。

日付フィールドは *SYS または *JOB に初期化 できます。 時刻フィールドとタイム・スタンプ・フィールドは *SYS に 初期化できます。

ネストされたデータ構造の初期化における INZ キーワードの使用についての詳細な説明は、215 ページの『ネストされたデータ構造の初期化』を参照してください。

INZ キーワードによって定義されたデータ構造、データ構造サブフィールド、ま たは独立フィールドを *ENTRY PLIST のパラメーターとして指定することはでき ません。

注 : INZ パラメーターが指定されていない 場合には、次のようになります。

- 初期化されたデータ構造の静的独立フィールドおよびサブフィールドは、それらの RPG でのデフォルトの初期値 (たとえば、文字の場合はブランク、数値の場合は 0) に初期化されます。
- 初期化されていない (データ構造についての定義仕様書に INZ が指定されて いない) データ構造のサブフィールドは (データ・タイプとは無関係 に) ブランクに初期化されます。

このキーワードと BASED または IMPORT との組み合わせは正しくありません。

LEN(length)

LEN キーワードは、データ構造、文字定義、UCS-2 定義、またはグラフィック定義で文字の長さを定義するために使用されます。 このキーワードは、データ構造定義で有効です。また、タイプ記入項目が A (英数字)、C (UCS-2)、または G (グラフィック) であるプロトタイプ定義、プロトタイプ・パラメーター定義、独立フィールド定義、およびサブフィールド定義でも有効です。

注 : 自由形式定義では、 LEN キーワードはデータ構造定義にのみ使用されます。 他の定義タイプの場合、項目の長さはデータ・タイプ・キーワードのパラメーターとして指定されます。

LEN キーワードに関する規則*:*

- 「データ・タイプ」記入項目には、データ・タイプ A、C、または G を指定する必要があります。
- 「長さ」記入項目を指定している場合や、サブフィールドで「開始位置」記入項目と「終了位置」記入項目を指定している場合は、LEN キーワードを指定することはできません。 9,999,999 を超える長さを指定するには、LEN キーワードを使用する必要があります。
- LEN キーワードを使用して、LIKE 定義の長さ調整をすることはできません。
- 長さは文字単位で指定します。UCS-2 定義および図形定義の場合、1 文字は 2 バイトで表されます。

```
 * Use the LEN keyword to define a standalone field of one million
 * characters and a standalone array of 100 characters.
D paragraph       S                 A   LEN(1000000) VARYING(4)
D splitPara       S                 A   LEN(100) DIM(10000)


 * Use the LEN keyword to define a data structure of length 16000000,
 * and to define three subfields.  Since the lengths of the parameters
 * are less than 9999999, they can be defined using from-and-to, or length
 * notation, or the LEN keyword.
D info            DS                    LEN(16000000)
D   name                                G   LEN(100) OVERLAY(info : 14000001)
D   address                       5000G   OVERLAY(info : 14000301)
D   country                1     40G


 * Use the LEN keyword to define a prototype that returns a varying
 * UCS-2 value that is up to 5000 UCS-2 characters long, and to define
 * two alphanumeric parameters.  Since the lengths of the parameters
 * are less than 9999999, they can be defined either using length notation
 * or the LEN keyword.
D getDftDir       PR                C   VARYING LEN(5000)
D   usrprf                          A   LEN(10) CONST
D   type                           5A   CONST
```

図 *136. LEN* キーワードの例

LIKE(名前 {: 長さ調整 })

LIKE キーワードは、項目を既存の項目と類似させて定義するために使用されます。オブジェクトでの LIKE の使用について詳しくは、445 ページの『LIKE(オブジェクト名)』を参照してください。

LIKE キーワードを指定した場合には、定義されている項目はパラメーターとして指定された項目の長さおよびデータ・フォーマットを引き継ぎます。 独立フィールド、プロトタイプ、およびデータ構造サブフィールドはこのキーワ ードを使用して定義することができます。 LIKE のパラメーターは、独立フィールド、データ構造、データ構造サブフィール ド、プロシージャー・インターフェース定義中のパラメーター、またはプロトタ イプ名とすることができます。 データ・タイプ記入項目 (40 桁目) はブランクでなければなりません。

このキーワードは *LIKE DEFINE 命令コードと類似しています (755 ページの『*LIKE DEFINE』 を参照)。ただし、定義されたデータがその長さだけでなくデータ形式および CCSID を引き継ぐ点で *LIKE DEFINE とは異なっています。

注 : ALTSEQ(*NONE)、NOOPT、ASCEND、DESCEND、CONST、次元、ヌル可能性などの属性は、LIKE のパラメーターから、定義された項目に継承されません。 継承されるのは、データ・タイプ、長さ、形式、小数点以下の桁数、および CCSID のみです。

配列のような項目を定義するために LIKE が使用されたときは、配列の次元を定義するため DIM キーワードが必要となります。 ただし、別の配列と同じ次元を持つ配列の定義には DIM(%ELEM(配列)) を使用できます。

LIKE のパラメーターがプロトタイプであった場合には、定義されている項目はそ のプロトタイプの戻り値と同じデータ・タイプになります。 戻り値がない場合には、エラー・メッセージが出されます。

定義している項目の長さを調整できます。 長さ調整は、自由形式定義の LIKE キーワードの 2 番目のパラメーターに指定するか、または、 固定形式定義の長さ記入項目に指定します。 長さ調整は正符号 (+) または負記号 (-) を付けて指定する必要があります。

以下は、LIKE キーワードを異なるデータ・タイプで使用する際のいくつかの考慮 事項です。

- **文字フィールド**の場合、長さ調整は追加する (または減らす) 文字数です。
- **数値フィールド**の場合、長さ調整は追加する (または減らす) 桁数 です。 整数フィールドまたは符号なしフィールドの場合、調整値は、調整後のフィールドの桁数が 3、5、10、または 20 になるようにする必要があります。 浮動フィールドの場合、長さ調整を行うことはできません。
- **グラフィック・フィールドまたは UCS-2 フィールド**の場合、長さ調整は追加する (または減らす) グラフィックまたは UCS-2 の文字数 (1 グラフィックまたは UCS-2 文字 = 2 バイト) です。

- **日付、時刻、タイム・スタンプ、基底ポインター、またはプロシージャー・ポインターの各フィールド**の場合、長さ調整は許可されていません。

LIKEDS を使用して、別のデータ構造と同じサブフィールドを持つようデータ構造を定義することができます。

LIKE キーワードを使用してデータを定義する例

以下の例では、最初に自由形式で、次に固定形式で示されています。

1. フィールド *Long_name* は、 フィールド *Name* と類似であり、長さを 5 文字増やしたものと定義されています。
2. サブフィールド配列 *NameList* は、フィールド *Name* と類似であると定義されています。 配列の各要素は、値 *ALL'X' で初期化されます。
3. プロトタイプ *GetBonus* は、フィールド *Salary* と類似であり、長さを 2 桁減らしたものと定義されています。

```
DCL-S Name CHAR(20);
DCL-S Long_name LIKE(Name : +5); 1

DCL-DS Struct;
   NameList LIKE(Name) DIM(20) INZ(*ALL'X'); 2
END-DS;

DCL-PR GetBonus LIKE(Salary : -2); 3
   Employee_Id INT(10) VALUE;
END-PR;
```

図 *137.* 他のフィールドと類似 *(LIKE)* のフィールドを自由形式で定義する

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++++
D.................................Keywords++++++++++++++++++++++++++++++++
D  Name           S              20
D  Long_name      S              +5    LIKE(Name) 1

D  Struct         DS
D   NameList                           LIKE(Name) DIM(20) INZ(*ALL'X') 2

D  GetBonus       PR          -2    LIKE(Salary) 3
D   Employee_Id                 10I 0 VALUE
```

図 *138.* 他のフィールドと類似 *(LIKE)* のフィールドを固定形式で定義する

*LIKE(*オブジェクト名*)*

LIKE キーワードを使用して、あるオブジェクトが以前に定義済みのオブジェクトと同じクラスを持つように指定することができます。 CLASS キーワードの値だけが継承されます。

```
* Variables MyString and OtherString are both Java String objects.
D MyString        S               O   CLASS(*JAVA
D                                         :'java.lang.String')
D OtherString     S                   LIKE(MyString)
 * Proc is a Java method returning a Java String object
D Proc            PR                  EXTPROC(*JAVA:'MyClass':'meth')
D                                     LIKE(MyString)
```

図 *139. LIKE* による他のオブジェクトと同様なオブジェクトの定義

注 : *LIKE DEFINE 命令は、オブジェクトの定義には使用できません。 LIKE キーワードを使用する必要があります。

LIKEDS(データ構造名)

LIKEDS キーワードは、データ構造、データ構造のサブフィールド、プロトタイプされた戻り値、または プロトタイプ・パラメーターを、別のデータ構造と同様に定義するために使用します。 新たな項目のサブフィールドは、LIKEDS キーワードのパラメーターとして指定されている親データ構造のサブフィールドと同じになります。

LIKEDS を使用して定義されているデータ構造は、親データ構造が修飾されていない場合でも、自動的に修飾されます。 サブフィールドは、修飾表記 DSNAME.SUBFIELDNAME を使用して参照する必要があります。 親データ構造に名前が付いていないサブフィールドがある場合、 子データ構造も同じ名前無しのサブフィールドを持ちます。

LIKEDS は、修飾されたデータ構造のサブフィールドに関してコーディングすることができます。 LIKEDS がデータ構造のサブフィールド定義でコーディングされる場合、 サブフィールドのデータ構造は自動的に QUALIFIED として定義されます。 LIKEDS サブフィールドのデータ構造のサブフィールドは、完全修飾形式 "ds.subf.subfa" で 参照されます。 LIKEDS を使用して定義されたサブフィールドは、それ自体がデータ構造であり、データ構造が必要な任意の場所で使用することができます。

ALIGN キーワードおよび ALTSEQ キーワードの値は、新しいデータ構造によって継承されます。 OCCURS、DIM、NOOPT、NULLIND、および INZ キーワードの値は継承されません。 親データ構造と同じ方法でサブフィールドを初期化するには、INZ(*LIKEDS) を指定します。

ただし、親データ構造の**サブフィールド**に対して指定された DIM、NOOPT、および NULLIND キーワードの値は新しいデータ構造によって継承されます。

```
 * Data structure qualDs is a qualified data structure
 * with two named subfields and one unnamed subfield

D qualDs          DS                  QUALIFIED
D  a1                            10A
D                                 2A
D  a2                             5P 0 DIM(3)
 * Data structure unqualDs is a non-qualified data structure
 * with one named subfield and one unnamed subfield

D unqualDs        DS
D  b1                             5A
D                                 5A
 * Data structure likeQual is defined LIKEDS(qualDs)

D likeQual        DS                  LIKEDS(qualDs)
 * Data structure likeUnqual is defined LIKEDS(unqualDs)

D likeUnqual      DS                  LIKEDS(unqualDs)
 /FREE
        // Set values in the subfields of the
        // parent data structures.

        qualDs.a1 = 'abc';
        qualDs.a2(1) = 25;
        b1 = 'xyz';

        // Set values in the subfields of the
        // child data structures.

        likeQual.a1 = 'def';
        likeQual.a2(2) = -250;
        likeUnqual.b1 = 'rst';

        // Display some of the subfields

        dsply likeQual.a1;  // displays 'def'

        dsply b1;           // displays 'xyz'
```

図 *140. LIKEDS* を使用したデータ構造の定義

```
D sysName         DS                  qualified
D   lib                         10A   inz('*LIBL')
D   obj                         10A
D userSpace       DS                  LIKEDS(sysName) INZ(*LIKEDS)
 // The variable "userSpace" was initialized with *LIKEDS, so the
 // first 'lib' subfield was initialized to '*LIBL'.  The second
 // 'obj' subfield must be set using a calculation.
C                   eval      userSpace.obj = 'TEMPSPACE'
```

図 *141. INZ(*LIKEDS)* の使用

```
P createSpace     B
D createSpace     PI
D  name                           LIKEDS(sysName)
 /free
     if name.lib = *blanks;
          name.lib = '*LIBL';
     endif;
     QUSCRTUS (name : *blanks : 4096 : ' ' : '*USE' : *blanks);
 /end-free
P createSpace     E
```

図 *142.* サブプロシージャー内のデータ構造パラメーターの使用

修飾データ構造名を使用した LIKEDS の指定

1. データ構造 *employees* は、修飾データ構造 *employee_info* のサブフィールドとして定義されています。
2. check_employee プロシージャーが呼び出され、*employee_info.employees* データ構造配列の要素が渡されます。
3. check_employee プロシージャーの *employee* パラメーターは、修飾データ構造 *employee_info.employees* を持つ LIKEDS キーワードを使用しています。
4. *employee* パラメーターのサブフィールドは、修飾データ構造サブフィールド *employee_info.employees* のサブフィールドと同じになります。

```
DCL-DS employee_info QUALIFIED;
   num_employees INT(10);
   DCL-DS employees DIM(20);  // 1
      name VARCHAR(25);
      salary PACKED(7:2);
   END-DS;
END-DS;

...
ok = check_employee (employee_info.employees(i)); // 2
...

DCL-PROC check_employee;
   DCL-PI *N IND;
      employee LIKEDS(employee_info.employees);  // 3
   END-PI;

   if (employee.salary < MIN_SALARY);  // 4
      ...
   endif;

END-PROC;
```

LIKEFILE(ファイル名)

LIKEFILE キーワードは、ファイル名パラメーターと同じ特性を持つファイルとして、プロトタイプ・パラメーターを定義するために使用します。

注 : 以下の説明では、LIKEFILE キーワードを使用して定義されたプロシージャー内のパラメーターを、「ファイル・パラメーター」という用語で呼びます。 パラメーターの定義を導き出すためにその定義が使用される LIKEFILE キーワードのパラメーターを、「親ファイル」という用語で呼びます。 呼び出し元によってプロシージャーに渡されるファイルを、「渡されるファイル」という用語で呼びます。

プロトタイプ・パラメーターの *LIKEFILE* キーワードに関する規則*:*

- LIKEFILE キーワードのファイル名パラメーターには、ファイル仕様書で事前に定義されているファイルを指定しなければなりません。
- 定義仕様書において、ファイル仕様書キーワードを LIKEFILE キーワードと共に指定することはできません。 ファイル・パラメーターでは、LIKEFILE キーワードのパラメーターとして指定されたファイルのファイル仕様書によって指定されている、すべての設定を使用します。
- OPTIONS(*NOPASS) または OPTIONS(*OMIT) 以外の定義キーワードを指定することはできません。
- ファイル・パラメーターは、RPG プログラムおよびプロシージャーの間でのみ渡すことができます。 他のプログラミング言語のファイル・パラメーター (COBOL ファイルなど) や、C fopen() 関数または open() 関数によって戻されるファイルとの互換性はありません。
- ファイルは必ず参照によって渡されます。 呼び出し先プロシージャーは、呼び出し元プロシージャーと同じファイルを直接処理します。 例えば、呼び出し元プロシージャーがレコードを読み取り、呼び出し先プロシージャーがそのレコードを更新して戻した場合、呼び出し元がそのレコードを再度更新することはできません。
- ファイルのブロック化属性がファイル仕様書から判別できない場合は、ファイル名パラメーターに BLOCK キーワードを指定する必要があります。

ファイル・パラメーターの受け渡しと使用に関する規則

- 渡されるファイルは、プロトタイプ・パラメーターと同じ親ファイルを使用して定義する必要があります。
- ファイル・パラメーターは修飾されます。 親ファイル FILE1 のレコード様式が REC1 および REC2 であるとすると、ファイル・パラメーター PARM のレコード様式は、呼び出し先プロシージャーでは PARM.REC1 および PARM.REC2 となります。
- 渡されるファイルのあらゆる設定 (ファイル仕様書キーワードを使用して定義される設定) は、アクセス方法にかかわらず (ファイルに直接アクセス、パラメーターを渡すことによってアクセス)、そのファイルにアクセスするすべてのプロシージャーについて有効になります。 例えば、外部ファイル名を保持する変数に EXTFILE キーワードが指定されている場合に、呼び出し先プロシージャーがそのファイルをオープンすると、呼び出し元の変数の値を使用して、オープンされるファイルの名前が設定されます。 ファイルに関連付けられている変数に対して、呼び出し先プロシージャーがキーワードを介してアクセスまたは変更を実行する必要がある場合、呼び出し元プロシージャーはそれらの変数を別々のパラメーターとして渡さなければなりません。
- ファイル・フィードバック組み込み関数の %EOF(ファイル名)、%EQUAL(ファイル名)、%FOUND(ファイル名)、%OPEN(ファイル名)、および %STATUS(ファイル名) を呼び出し先プロシージャーで使用して、ファイル・パラメーターの現在の状態を判別することができます。 この場合は、ファイル・パラメーターの名前をオペランドとして組み込み関数に指定してください。

  モジュール間でのファイル・パラメーターの引き渡しについて詳しくは、184 ページの『ファイルに関連付けられる変数』および 185 ページの『ファイルを渡す方法および関連付けられている変数を含むデータ構造を渡す方法の例』を参照してください。

```
 * Define a file template to be used for defining actual files
 * and the file parameter
Finfile_t  IF   E             DISK    TEMPLATE BLOCK(*YES)
F                                     EXTDESC('MYLIB/MYFILE')
F                                     RENAME(R01M2:inRec)

 * Define two actual files that can be passed to the file parameter
Ffile1                                LIKEFILE(infile_t)
F                                     EXTFILE('MYLIB/FILE1')
Ffile2                                LIKEFILE(infile_t)
F                                     EXTFILE('MYLIB/FILE2')

 * Define a data structure type for the file data
D inData_t        DS                  LIKEREC(infile_t.inRec:*INPUT)
D                                     TEMPLATE

 * Define the prototype for a procedure to handle the files
D nextValidRec    PR              N
D   infile                            LIKEFILE(infile_t)
D   data                              LIKEDS(inData_t)

 * Define variables to hold the record data
D f1Data          DS                  LIKEDS(inData_t)
D f2Data          DS                  LIKEDS(inData_t)

 /FREE
       // Process valid records from each file until one
       // of the files has no more valid records
       DOW nextValidRec(file1 : f1Data)
       AND nextValidRec(file2 : f2Data);
          // ... process the data from the files
       ENDDO;
       *INLR = '1';
 /END-FREE

 * The procedure that will process the file parameter
P nextValidRec    B
D nextValidRec    PI              N
D   infile                            LIKEFILE(infile_t)
D   data                              LIKEDS(inData_t)
 /FREE
    // Search for a valid record in the file parameter
    READ infile data;
    DOW NOT %EOF(infile);
       IF data.active = 'Y';
          RETURN *ON;             // This is a valid record
       ENDIF;
       READ infile data;
    ENDDO;
    RETURN *OFF;                  // No valid record was found
 /END-FREE
P nextValidRec    E
```

図 *143.* ファイルをパラメーターとしてプロシージャーに渡す

LIKEREC(intrecname{:extract-types})

LIKEREC キーワードは、データ構造、データ構造サブフィールド、プロトタイプされた戻り値、またはプロトタイプ・パラメーターを、レコード と同様に定義するために使用します。 データ構造のサブフィールドは、レコードのフィールドと同じになります。 LIKEREC は、2 番目の任意指定パラメーターを使用できます。このパラメーターはレコードのどのフィールドをデータ構造に入れるかを指定します。 これには, 次のものがあります。

- *ALL 外部レコードのすべてのフィールドが抜き出されます。
- *INPUT すべての入力可能フィールドが抜き出されます。 (これがデフォルトです。)
- *OUTPUT すべての出力可能フィールドが抜き出されます。
- *KEY キー・フィールドが、キーが DDS の K 指定に定義されている順番に抜き出されます。

ファイル内のフィールドと同じデータ・タイプを持つサブフィールドを定義するのではなく、それらのサブフィールドがすべて標識であることを示すために、*NULL を指定することもできます。 データベース・ファイルの場合、これらの標識のレイアウトは、レコードのヌル・バイト・マップと同じです。

LIKEREC キーワードの使用時には、次の点を考慮する必要があります。

- キーワード LIKEREC の最初のパラメーターはプログラム内のレコード名です。 レコード名の名前が変更された場合、これはレコードの内部名になります。
- LIKEREC の残りの抜き出しタイプパラメーターは、システム上のファイルの関連レコードの定義と一致する必要があります。*INPUT は、入力可能レコードおよび更新可能レコードにのみ使用できます。*OUTPUT は、出力可能レコードにのみ使用できます。*ALL はどのレコード・タイプにも使用できます。*KEY はキー付きファイルにのみ使用できます。 指定されない場合は、このパラメーターのデフォルト値 *INPUT が使用されます。 ただし、DISK ファイルからのレコード名に対して抜き出しタイプ・パラメーターが指定されず、出力バッファーのレイアウトが入力バッファーのレイアウトと完全に一致するときは、WRITE 命令でデータ構造を使用できます。
- *NULL が指定されていない *INPUT および *OUTPUT の場合は、データ構造に含まれるサブフィールドは外部レコード記述内と同じ開始位置を占めます。
- ファイルの接頭部が指定された場合、指定された接頭部はサブフィールドの名前に使用されます。
- *NULL が指定されると、標識はデータベース・ファイルの外部レコードのヌル・バイト・マップ内のヌル標識と同じ開始位置を占めます。 その他のタイプのファイルの場合は、標識サブフィールドの開始位置は順次に割り当てられます。
- レコード内のフィールドが入力仕様において明示的に名前変更された場合でも、 内部名ではなく、外部名 (おそらく接頭部) が使用されます。
- ALIAS キーワードを使用してファイルが定義されている場合、別名が、データ構造のサブフィールドに使用されます。365 ページの『外部記述ファイルに対する ALIAS キーワードの使用』に、ファイルが ALIAS キーワードを使用して定義され、LIKEREC キーワードでデータ構造を定義した例が示されています。
- LIKEREC を使用して定義されたデータ構造は、QUALIFIED データ構造です。 サブフィールドの名前は、DS1.SUBF1 という新しいデータ構造名によって修飾されます。
- LIKEREC は、修飾されたデータ構造のサブフィールドに関してコーディングすることができます。 LIKEREC がデータ構造のサブフィールド定義でコーディングされる場合、 サブフィールドのデータ構造は自動的に QUALIFIED として定義されます。 LIKEREC サブフィールドのデータ構造のサブフィールドは、完全修飾形式 "ds.subf.subfa" で 参照されます。 LIKEREC を使用して定義されたサブフィールドは、 それ自体がデータ構造であり、データ構造が必要な任意の場所で使用することができます。
- 抜き出しタイプ *NULL で定義されたデータ構造を入出力操作と一緒に使用することはできません。

NOOPT

NOOPT キーワードは、このキーワードが指定された独立フィールド、パラメーター、またはデータ構造については最適化は実行されない ことを指示します。 NOOPT の指定によって、データ項目の内容は確実に直前に割り当てられた 値となります。 これは、例外処理に使用される値が入っているフィールドに必要となることがあ ります。

注 : 最適化プログラムでは、一部の値をレジスターに保存し、通常のプログラム実行 時に事前に定義された点でのみ記憶域にそれらを復元することができます。 例外処理はこの通常の 実行順序を中断し、したがって、レジスターに 含まれているプログラム変数はそれらに割り当てられた記憶位置には戻されない ことがあります。 結果として、それらの変数が例外処理に使用された時に、直前に割り当てられた 値が入っていないことがあります。 NOOPT キーワードによりこれらの流れ が確実に行われます。

参照によって渡されるデータ項目を NOOPT キーワードとともに指定した場合には 、プロトタイプまたはプロシージャー・インターフェース・パラメーター定義に も NOOPT キーワードも指定しなければなりません。 値によって渡されるパラメーターにはこの要件は適用されません。

ヒント :

OPM RPG/400® プログラムの中で定義されるデータ項目は、すべて暗黙に NOOPT とともに定義されます。したがって、OPM プログラム用のプロトタイプを作成している場合には、そのプ ロトタイプの中に定義されるすべてのパラメーターに NOOPT を指定しなければな りません。 これによって、プロトタイプのユーザーのエラーが防止されます。

独立フィールド定義、パラメーター、またはデータ構造定義に使用可能なすべて のキーワードは、NOOPT と一緒に使用することができます。

NULLIND{(ヌル標識)}

NULLIND キーワードを使用すれば、フィールドまたはデータ構造に対して %NULLIND 値を明示的に定義することができます。

ある項目に対して NULLIND キーワードが指定されると、このキーワードのパラメーターは、その定義しようとしている項目に対してヌル標識、ヌル標識配列、またはヌル標識データ構造として使用されます。

定義しようとしている項目がデータ構造でなければ、NULLIND キーワードのパラメーターを省略することができます。 その場合は、変数や配列はヌル可能ですが、%NULLIND 組み込み関数を使用してヌル標識のアドレスを指定する必要があります。

NULLIND キーワードへのパラメーターとして指定された標識については、その名前、または関連項目の %NULLIND 組み込み関数のいずれかによってアドレス指定することができます。 例えば、フィールド *myField* が *NULLIND(myNullind)* で定義された場合、*%NULLIND(myField)* と *myNullind* はどちらも標識 *myNullind* を表します。 配列データ構造 *myDs* が *NULLIND(myDsNulls)* で定義された場合、*myDsNulls(i).addr* と *%NULLIND(myDs(i).addr)* はどちらも標識 *myDsNulls(i).addr* を表します。

- NULLIND キーワードのパラメーターは、サブフィールドのデータ・タイプ以外のあらゆる方法で定義しようとしている項目と一致しなければなりません。
  - この項目がデータ構造である場合は、外部記述データ構造でなければなりません。 NULLIND キーワードのパラメーターも、EXTNAME または LIKEREC キーワードの抜き出しタイプで、*NULL パラメーターを追加して、項目として外部記述されなければなりません。
  - 個々の外部サブフィールドに対して NULLIND キーワードを指定することはできません。 外部記述データ構造のサブフィールドは、外部フィールドがヌル可能である場合は自動的にヌル可能として定義されます。 特定の標識を外部記述サブフィールドのヌル標識として定義するためには、外部記述ヌル標識データ構造を定義し、それを外部記述データ構造と関連付ける必要があります。
  - プログラム記述データ構造に対して NULLIND キーワードを指定することはできません。 外部記述データ構造の追加のプログラム記述サブフィールドに対して NULLIND キーワードを指定できますが、このデータ構造が入出力操作と一緒に使用されると、プログラム記述サブフィールドのヌル標識は考慮されません。
  - 項目が配列である場合は、NULLIND キーワードのパラメーターも同じ次元を持つ配列でなければなりません。
  - 項目がサブフィールドである場合は、NULLIND キーワードのパラメーターも同じデータ構造内のサブフィールドでなければなりません。 データ構造が修飾されている場合は、修飾データ構造名を付けずに NULLIND キーワードのパラメーターを指定する必要があります。 下記の例を参照してください。
  - LIKE キーワードで定義されたフィールドまたは LIKEDS キーワードで定義されたデータ構造は、LIKE または LIKEDS キーワードのパラメーターの NULLIND キーワードを継承しません。 しかし、LIKEDS キーワードで定義されたデータ構造は、NULLIND キーワードのパラメーターとして指定されたデータ構造のサブフィールドに指定された NULLIND キーワードによって定義された関係を継承します。 下記の例を参照してください。
  - NULLIND キーワードのパラメーターの有効範囲および記憶域タイプは、項目と同じでなければなりません。 例えば、項目がサブプロシージャー内の静的フィールドである場合は、NULLIND キーワードのパラメーターも同じサブプロシージャー内の静的フィールドでなければなりません。
  - プロトタイプ・パラメーターに NULLIND キーワードが指定されたとき
    - NULLIND キーワードのパラメーターは必須です。
    - NULLIND キーワードのパラメーターはパラメーター・リスト内の別のパラメーターでなければなりません。
    - OPTIONS(*NULLIND) はパラメーターには使用できません。
- NULLIND キーワードのパラメーターはヌル可能とすることもヌル可能サブフィールドを持つこともできません。
- NULLIND キーワードのパラメーターは 1 つの項目にしか指定できません。

- NULLIND キーワードは、ALWNULL(*USRCTL) キーワードが有効である場合にのみ使用できます。

例

- 次の例では、フィールド DUEDATE はパラメーターを持たない NULLIND キーワードで定義されています。DUEDATE のヌル標識は、%NULLIND 組み込み関数を使用してアドレス指定されます。

```
dcl-s dueDate date nullind;

if not %nullind(dueDate) and dueDate > %date();
   sendReminder (custId : dueDate);
endif;
```

- 次の例では、配列 *empBonus* はパラメーター *empBonus_null* を持つ NULLIND キーワードで定義されています。*empBonus_null* も同じ次元を持つ配列として定義されています。 *empBonus* のヌル標識については、%NULLIND 組み込み関数または *empBonus_null* 配列を使用してアドレス指定することができます。

  次の 2 つの FOR ループは同じ振る舞いをします。

  1. 最初の FOR ループでは、ヌル標識が直接使用されています。
  2. 2 番目の FOR ループでは、%NULLIND が使用されています。

```
dcl-c MAX_EMPLOYEES 50;
dcl-s empBonus packed(5:2) dim(MAX_EMPLOYEES) nullind(empBonus_null);
dcl-s empBonus_null ind dim(MAX_EMPLOYEES);
dcl-s numEmployees int(10);

for i = 1 to numEmployees;
   if not empBonus_null(i); // 1
      applyBonus (emp(i) : empBonus(i));
   endif;
endfor;

for i = 1 to numEmployees;
   if not %nullind(empBonus(i)); // 2
      applyBonus (emp(i) : empBonus(i));
   endif;
endfor;
```

- 次のトリガー・プログラムの例では、データ構造 *beforeDs* は EXTNAME(*INPUT) で定義されており、データ構造 *beforeNull* は EXTNAME(*INPUT : *NULL) で定義されています。 データ構造 *beforeNull* は、データ構造 *beforeDsl* のヌル標識データ構造として指定されています。 *beforeNull* データ構造のサブフィールドについては、それらのサブフィールドの名前によってアドレス指定することも、%NULLIND 組み込み関数を *beforeDsl* データ構造のサブフィールドと一緒に使用してアドレス指定することもできます。

  次の 2 つの IF ステートメントは同じ振る舞いをします。

  1. 最初の IF ステートメントでは、ヌル標識が直接使用されています。
  2. 2 番目の IF ステートメントでは、ヌル標識のアドレスを指定するために %NULLIND が使用されています。

```
dcl-ds before extname('CUSTFILE':*input) based(pBefore)
             nullind(beforeNull) end-ds;
dcl-ds beforeNull extname('CUSTFILE':*input:*null) based(pBeforeNull)
             end-ds;

pBefore = %addr(trgbuf) + trgbuf.beforeOffset;
pBeforeNull = %addr(trgbuf) + trgbuf.beforeNullMapOffset;

if beforeNull.quantity; // 1
   ...

if %nullind(beforeDs.quantity); // 2
   ...
```

- 次の例では、修飾されたデータ構造 *ds1* 内の 2 つのサブフィールドが NULLIND キーワードによって定義されています。

  1. サブフィールド *sub1_null* および *sub2_null* が NULLIND キーワードのパラメーターとして指定されるとき、これらのサブフィールドはデータ構造名によって修飾されずに指定されます。
  2. データ構造 *DS1* のサブフィールドが演算で使用されるとき、データ構造が修飾されるので、これらのサブフィールドはデータ構造名によって修飾されます。

```
dcl-ds ds1 qualified;
   sub1 char(10) nullind(sub1_null); 1
   sub2 likerec(custRec:*input) nullind(sub2_nulls); 1
   sub1_null ind;
   sub2_nulls likerec(custRec:*input:*null);
end-ds;

if ds1.sub1_null 2
or ds1.sub2_nulls.quantity; 2 ;
   ...
```

- 次の例は、サブフィールドのヌル標識関係が LIKEDS キーワードを使用して定義されたデータ構造によってどのように継承されるかを示しています。ただし、ヌル可能性は LIKE キーワードを使用して定義されたフィールドには継承されません。

  1. データ構造 *DS2* は、データ構造 *DS1* がパラメーターとして指定された LIKEDS キーワードを使用して定義されます。 *DS1* の定義については、上記の例を参照してください。
  2. フィールド *FLD1* は NULLIND キーワードで定義されます。
  3. フィールド *FLD2* は、フィールド *FLD1* がパラメーターとして指定された LIKE キーワードで定義されます。 ヌル可能性は LIKE キーワードによって継承されないので、*FLD2* はヌル可能ではありません。
  4. *DS2* のサブフィールドのヌル標識関係は、*DS1* のサブフィールドに対して定義された NULLIND キーワードから継承されます。 *DS2.SUB1* はヌル可能であり、そのヌル標識は *DS2.SUB1_NULL* です。

```
dcl-ds ds2 likeds(ds1) inz; 1
dcl-s fld1 char(10) nullind(null1); 2
dcl-s fld2 like(fld1); 3

if %nullind(ds2.sub1); 4
  ...
```

OBJECT{(*JAVA:クラス名)}

OBJECT キーワードは、自由形式定義で使用されて、 項目のタイプがオブジェクトであることを示します。

これは最初のキーワードでなければなりません。

OBJECT キーワードが Java コンストラクター・メソッドの戻り値のタイプを定義するために使用される場合、パラメーターは任意指定です。 この場合、戻り値のクラスは Java メソッドのクラスと同じであるため、クラスをもう一度指定する必要はありません。 Java コンストラクター用のプロトタイプの定義については、433 ページの『EXTPROC{({*CL|*CWIDEN|*CNOWIDEN|*JAVA:クラス名:}名前|*DCLCASE)}』を参照してください。

それ以外の場合、両方のパラメーターが必須です。

最初のパラメーターは *JAVA でなければなりません。

2 番目のパラメーターは、オブジェクトの Java クラスを指定します。 Java クラスの指定については、278 ページの『オブジェクト・データ・タイプ』を参照してください。 このパラメーターは、リテラルまたは名前付き定数でなければなりません。

注 : Java コンストラクターの戻り値を定義するときに OBJECT キーワード のパラメーターを指定する場合は、クラスは EXTPROC キーワードで指定されたクラスと同じである必要があります。

次に例を示します。

- フィールド *str* は、クラス java.lang.String のオブジェクト・フィールドとして定義されています。

- Java BigDecimal オブジェクト・コンストラクター用のプロトタイプ *newBigDecimal* の戻り値は、オブジェクトとして定義されています。 OBJECT キーワードには パラメーターが指定されていないため、戻り値 'java.math.BigDecimal' のクラスは、 EXTPROC キーワードに指定されたクラスから派生します。

```
DCL-S str OBJECT(*JAVA : 'java.lang.String');

DCL-PR newBigDecimal OBJECT EXTPROC(*JAVA : 'java.math.BigDecimal'
                                    : *CONSTRUCTOR);
   val VARUCS2(100) CONST;
END-PR;
```

OCCURS(数値定数)

OCCURS キーワードによって、複数オカレンス・データ構造のオカレンス数を指定することがで きます。

この数値定数パラメーターは、小数点以下の桁数のない 0 より大きい値でな ければなりません。 これは、数値リテラル、数値を戻す組み込み関数、または数値定数とすること ができます。

定数の値は、キーワードの処理時には既知である必要はありませんが、コンパイル時には既知でなければなりません。

このキーワードは、プログラム状況データ構造、ファイル情報データ構造、また はデータ域データ構造には有効ではありません。

複数オカレンス・データ構造にポインター・サブフィールドが含まれている場合には、ポ インターに関するシステム記憶域の制約事項のために、繰り返し相互間の距離は正確 に 16 の倍数になっていなければなりません。 これは、繰り返し相互間の距離が各繰り返しの長さより大きくなる場合があることを意味 します。

以下は、ポインター・サブフィールドを持つ複数オカレンス・データ構造の記憶域割り振り を示す例です。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++++
D DS1             DS                    OCCURS(2)
D   POINTER                        16*
D   FLD5                            5
D DS2             DS                    OCCURS(2)
D   CHAR16                         16
D   CHR5                            5
```

記憶域でのフィールドの割り振り。 DS1 のオカレンスはそれぞれ 32 バイトで、DS2 の オカレンスはそれぞれ 21 バイトです。

DS1 OCCURRENCE 1			DS1 OCCURRENCE 2		
POINTER	FLD5	(fill)	POINTER	FLD5	(fill)

DS2 OCCURRENCE 1		DS2 OCCURRENCE 2	
CHAR16	CHR5	CHAR16	CHR5

図 *144.* ポインター・サブフィールドを持つ複数オカレンス・データ構造の記憶域割り振り

OPDESC

OPDESC キーワードは、操作記述子がプロトタイプの中で定義されたパラメーター と一緒に渡されることを指定します。

OPDESC を指定した場合には、参照によって渡されるすべての文字または図形パラ メーターと一緒に操作記述子が渡されます。 値によって渡されたパラメーターの操作記述子を検索しようとしても、結果はエ ラーとなります。

注 : UCS-2 フィールドの場合、操作記述子は渡されません。

プロトタイプに OPDESC が含まれているプロトタイプ・プロシージャーで、CALLP を使用することは、CALLB (D) を使用してプロシージャーを呼び出すことと同じです。 操作記述子は、式の中で呼び出されたプロシージャーの場合も渡されます。

このキーワードは、プロトタイプ定義とプロシージャー・インターフェース定義 の両方に適用されます。EXTPGM キーワードと一緒に使用することはできません。

注 : 独自のプロシージャーに OPDESC キーワードを使用すると、パラメーター情報を取得する CEEDOD などの API の呼び出し方法が、RTNPARM キーワードによって影響を受けることがあります。 詳しくは、477 ページの『RTNPARM』および 674 ページの『%PARMNUM (パラメーター番号を戻す)』を参照してください。

OPDESC キーワードの例については、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」のサービス・プログラムの例を参照してください。

OPTIONS(*NOPASS *OMIT *VARSIZE *EXACT *STRING *TRIM *RIGHTADJ *NULLIND)

OPTIONS キーワードは、次のようなオプションを渡す 1 つまたは複数のパラメー ターを指定するために使用されます。

- パラメーターを渡す必要があるかどうか。 456 ページの『OPTIONS(*NOPASS)』を参照してください。
- 参照によって渡されるパラメーターについて特殊値 *OMIT を渡すことができ るかどうか。 456 ページの『OPTIONS(*OMIT)』を参照してください。
- 参照によって渡されるパラメーターをプロトタイプに指定された長さより短 くすることができるかどうか。 457 ページの『OPTIONS(*VARSIZE)』を参照してください。
- 渡される可能性のあるパラメーターに対して、コンパイラーが厳密な規則を適用するかどうか。 459 ページの『OPTIONS(*EXACT)』を参照してください。
- 呼び出されるプログラムまたはプロシージャーが、ヌル文字で 終了するストリングを指し示すポインターを予期して、文字式を、渡される パラメーターとして指定できるようにするかどうか。 464 ページの『OPTIONS(*STRING)』を参照してください。
- 渡される前にパラメーター内のブランクをトリミングするかどうか。 464 ページの『OPTIONS(*TRIM)』を参照してください。
- 渡されたパラメーター内でパラメーター値を右寄せするかどうか。 464 ページの『OPTIONS(*RIGHTADJ)』を参照してください。
- ヌル・バイト・マップをパラメーター付きで受け渡しするかどうか。 466 ページの『OPTIONS(*NULLIND)』を参照してください。

複数のオプションを指定することができます。 たとえば、任意指定パラメーターをプロトタイプの指示より短くすることができ ることを指定するためには、OPTIONS(*VARSIZE : *NOPASS) とコーディングされます。

パラメーターが正しく処理されるようにするための手順を必要とするオプション

オプション *NOPASS、*OMIT、および *VARSIZE を渡すパラメーターの場合、 そのプロシージャーのプログラマーは、これらのオプションが確実に処理されるようにする必要があります。 たとえば、OPTIONS(*NOPASS) がコーディングされていて、パラメーターを 渡すようにした場合、プロシージャーは、そのパラメーターにアクセスする 前にそれが渡されていることをチェックする必要があります。 コンパイラーは、これについては一切チェックしません。 これらのオプションを使用するパラメーターの情報を取得する CEEDOD または CEETSTA などの API を呼び出す場合、API の呼び出し方法が、RTNPARM キーワードによって影響を受けることがあります。 詳しくは、477 ページの『RTNPARM』および 674 ページの『%PARMNUM (パラメーター番号を戻す)』を参照してください。

OPTIONS(*NOPASS)

OPTIONS(*NOPASS) を定義仕様書に指定した場合には、呼び出し時にパラメーターを渡 す必要はありません。 この指定より後のすべてのパラメーターにも、*NOPASS の指定が必要です。 プログラムまたはプロシージャーにパラメーターが渡されないと、呼び出されたプログラムまたはプロシージャーは、単に、パラメーター・リストにそのパラメーターが含まれていないものとして機能します。

警告 : 呼び出されたプログラムまたはプロシージャー内で渡されなかった パラメーターがアクセスされると、予測できない結果を招きます。

呼び出されたプログラムまたはプロシージャー内で渡されなかったパラメーターが変更されると、プログラムまたはプロシージャーが戻された後に、呼び出しとは無関係と思われる区域で、予測できない結果が生じる可能性があります。

次の例は、パラメーターがオプションであることを示すため に OPTIONS(*NOPASS) を使用するプロトタイプおよびプロシージャー をコーディングする方法を示しています。

```
 * The following prototype describes a procedure that expects
 * either one or two parameters.
D FormatAddress   PR            45A
D   City                        20A   CONST
D   Province                    20A   CONST OPTIONS(*NOPASS)
 * The first call to FormatAddress only passes one parameter.  The
 * second call passes both parameters.
C                   EVAL      A = FormatAddress('North York')
C                   EVAL      A = FormatAddress('Victoria' : 'B.C.')
C                   RETURN
 *---------------------------------------------------------------
 * FormatAddress:
 * This procedure must check the number of parameters since the
 * second was defined with OPTIONS(*NOPASS).
 * It should only use the second parameter if it was passed.
 *---------------------------------------------------------------
P FormatAddress   B
D FormatAddress   PI            45A
D   City                        20A   CONST
D   ProvParm                    20A   CONST OPTIONS(*NOPASS)
D Province        S             20A   INZ('Ontario')
 * Set the local variable Province to the value of the second
 * parameter if it was passed.  Otherwise let it default to
 * 'Ontario' as it was initialized.
C                   IF        %PARMS > 1
C                   EVAL      Province = ProvParm
C                   ENDIF
 * Return the city and province in the form City, Province
 * for example 'North York, Ontario'
C                   RETURN    %TRIMR(City) + ',' + Province
P FormatAddress   E
```

図 *145.* パラメーターがオプションであることを示すための *OPTIONS(*NOPASS)* の使用

OPTIONS(*OMIT)

OPTIONS(*OMIT) を指定した場合は、そのパラメーターに値 *OMIT を使用できます。*OMIT を使用できるのは、CONST パラメーターと、参照によって渡されるパラメーターに対してのみです。 省略されるパラメーターについて詳しくは、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」のプログラムおよびプロシージャーの呼び出しに関する章を参照してください。

警告 : 呼び出されたプログラムまたはプロシージャー内で渡されなかった パラメーターがアクセスされると、予測できない結果を招きます。

次の例は、特殊値の *OMIT がパラメーターとして渡される可能性が あることを示すために、OPTIONS(*OMIT) を使用してプロトタイプおよび プロシージャーをコーディングする方法を示しています。

```
FQSYSPRT   O    F   10        PRINTER USROPN
 * The following prototype describes a procedure that allows
 * the special value *OMIT to be passed as a parameter.
 * If the parameter is passed, it is set to '1' if an error
 * occurred, and '0' otherwise.
D OpenFile        PR
D   Error                       1A   OPTIONS(*OMIT)
C                   SETOFF                                       10
 * The first call to OpenFile assumes that no error will occur,
 * so it does not bother with the error code and passes *OMIT.
C                   CALLP     OpenFile(*OMIT)
 * The second call to OpenFile passes an indicator so that
 * it can check whether an error occurred.
C                   CALLP     OpenFile(*IN10)
C                   IF        *IN10
C                   ... an error occurred
C                   ENDIF
C                   RETURN
 *---------------------------------------------------------------
 * OpenFile
 * This procedure must check the number of parameters since the
 * second was defined with OPTIONS(*OMIT).
 * It should only use the second parameter if it was passed.
 *---------------------------------------------------------------
P OpenFile        B
D OpenFile        PI
D   Error                       1A   OPTIONS(*OMIT)
D SaveIn01        S               1A
 * Save the current value of indicator 01 in case it is being
 * used elsewhere.
C                   EVAL      SaveIn01 = *IN01
 * Open the file.  *IN01 will indicate if an error occurs.
C                   OPEN      QSYSPRT                            01
 * If the Error parameter was passed, update it with the indicator
C                   IF        %ADDR(Error) <> *NULL
C                   EVAL      Error = *IN01
C                   ENDIF
 * Restore *IN01 to its original value.
C                   EVAL      *IN01 = SaveIn01
P OpenFile        E
```

図 *146. OPTIONS(*OMIT)* の使用

OPTIONS(*VARSIZE)

OPTIONS(*VARSIZE) が有効なのは、パラメーターが文字、図形、または UCS-2 データ・タイプの参照、または任意のタイプの配列を表す参照によって渡される場合だけです。

参照によって渡されるパラメーターの場合、渡されるパラメーターのアドレスが呼び出しで渡されます。ただし、読み取り専用参照パラメーター用に一時域が使用される場合は除きます。その場合は、一時域のアドレスが 呼び出しで渡されます。読み取り専用の参照パラメーターでの一時域の使用について詳しくは、420 ページの『CONST{(定数)}』 を参照してください。

OPTIONS(*VARSIZE) を指定した場合には、渡されるパラメーターはプロトタイプ に定義された長さより短いか、または長い可能性があります。 渡された量だけの データにアクセスするのは、呼び出し先プログラムまたはサブプロシージャーの 責任になります。 渡すデータの量を伝えるために、長さを含む別の パラメーターを渡すか、または、サブプロシージャー用の操作記述子を使用 することができます。 可変長フィールドの場合、渡されたパラメーターの現在の 長さを決定するために %LEN 組み込み関数を使用することができます。

固定長フィールドの場合に、OPTIONS(*VARSIZE) を省略するときは、 少なくとも プロトタイプによって要求されている量の データを渡す必要があります。可変長フィールドの場合、パラメーターは、 定義で宣言された最大長を持っていなければなりません。

警告 : 渡されるパラメーターがパラメーターのプロトタイプ・サイズよりも短い場合に、呼び出されたプログラムまたはプロシージャーが渡されたパラメーターより多くのパラメーターを変更すると、呼び出し側プログラムまたはプロシージャー内のデータが破壊されます。 プログラムまたはプ

ロシージャーが戻った後、呼び出しとは無関係と思われる区域で、予測できない結果が生じる可能性があります。

次の例は、OPTIONS(*VARSIZE) を使用して、可変長パラメーターを使用可能にする プロトタイプおよびプロシージャーをコーディングする方法を示しています。

```
 * The following prototype describes a procedure that allows
 * both a variable-length array and a variable-length character
 * field to be passed.  Other parameters indicate the lengths.
D Search          PR             5U 0
D   SearchIn                    50A   OPTIONS(*VARSIZE)
D                                     DIM(100) CONST
D   ArrayLen                     5U 0 VALUE
D   ArrayDim                     5U 0 VALUE
D   SearchFor                   50A   OPTIONS(*VARSIZE) CONST
D   FieldLen                     5U 0 VALUE
D Arr1            S              1A   DIM(7) CTDATA PERRCD(7)
D Arr2            S             10A   DIM(3) CTDATA
D Elem            S              5U 0
 * Call Search to search an array of 7 elements of length 1 with
 * a search argument of length 1.  Since the '*' is in the 5th
 * element of the array, Elem will have the value 5.
C                   EVAL      Elem = Search(Arr1 :
C                                           %SIZE(Arr1) : %ELEM(Arr1) :
C                                           '*' : 1)
 * Call Search to search an array of 3 elements of length 10 with
 * a search argument of length 4.  Since 'Pink' is not in the
 * array, Elem will have the value 0.
C                   EVAL      Elem = Search(Arr2 :
C                                           %SIZE(Arr2) : %ELEM(Arr2) :
C                                           'Pink' : 4)
C                   RETURN
```

図 *147. OPTIONS(*VARSIZE)* の使用

```
 *-------------------------------------------------------------
 * Search:
 *   Searches for SearchFor in the array SearchIn.  Returns
 *   the element where the value is found, or 0 if not found.
 *   The character parameters can be of any length or
 *   dimension since OPTIONS(*VARSIZE) is specified for both.
 *-------------------------------------------------------------
P Search          B
D Search          PI             5U 0
D   SearchIn                    50A   OPTIONS(*VARSIZE)
D                                     DIM(100) CONST
D   ArrayLen                     5U 0 VALUE
D   ArrayDim                     5U 0 VALUE
D   SearchFor                   50A   OPTIONS(*VARSIZE) CONST
D   FieldLen                     5U 0 VALUE
D I               S              5U 0
 * Check each element of the array to see if it the same
 * as the SearchFor.  Use the dimension that was passed as
 * a parameter rather than the declared dimension.  Use
 * %SUBST with the length parameter since the parameters may
 * not have the declared length.
C     1             DO        ArrayDim      I                 5 0
 * If this element matches SearchFor, return the index.
C                   IF        %SUBST(SearchIn(I) : 1 : ArrayLen)
C                             = %SUBST(SearchFor : 1 : FieldLen)
C                   RETURN    I
C                   ENDIF
C                   ENDDO
 * No matching element was found.
C                   RETURN    0
P Search          E

Compile-time data section:

**CTDATA ARR1
A2$@*jM
**CTDATA ARR2
Red
Blue
Yellow
```

OPTIONS(*EXACT)

OPTIONS(*EXACT) が指定された場合、コンパイラーは渡される可能性のあるパラメーターに対してより厳密に適用されます。

参照によって渡されるパラメーターに対する OPTIONS(*EXACT)

OPTIONS (*EXACT) がパラメーターに指定されている場合、呼び出されるプロシージャーが、渡されるパラメーターと同じ値を受け取るようにするために、追加の規則が適用されます。

参照によって渡されるパラメーターで OPTIONS(*EXACT) が指定される場合、渡されるパラメーターには以下の追加規則が適用されます。

- 英数字、UCS-2 およびグラフィックの各パラメーターの定義された長さは、プロトタイプ・パラメーターと同じである必要があります。
- 英数字、UCS-2 およびグラフィックの各パラメーターの CCSID は一致している必要があります。
- オブジェクト・パラメーターの Java クラスは一致している必要があります。
- ASCEND キーワードまたは DESCEND キーワードには、渡されるパラメーターとプロトタイプ・パラメーターの両方で同じものを指定する必要があります。
- 配列の次元は同じである必要があります。
- プロトタイプ・パラメーターがデータ構造である場合、渡されるパラメーターは、LIKEDS または LIKEREC によってプロトタイプ・パラメーターに関連付けられている必要があります。
- LIKE キーワードを使用してプロトタイプ・パラメーターが定義されている場合、渡されるパラメーターは LIKE キーワードによってプロトタイプ・パラメーターに関連付けられている必要があります。

次の例は、参照によって渡されるパラメーターを使用して OPTIONS(*EXACT) を指定した場合の効果を示しています。 プロトタイプ noOptionExact_ref は、そのパラメーターに OPTIONS(*EXACT) を指定しません。プロトタイプ optionExact_ref は、そのパラメーターに OPTIONS(*EXACT) を指定します。 noOptionExact_ref の呼び出しは、呼び出されるプロシージャーが受け取る情報が予想より少なかったり、呼び出されるプロシージャーが正しくないタイプの情報を受け取ったりしたために、予期しない結果を引き起こす可能性があるパラメーターを渡します。

どちらのプロシージャーの呼び出しも同じですが、noOptionExact_ref の呼び出しでは、正しくない可能性があるパラメーターが渡されるようになります。 いずれの場合も、noOptionExact_ref の呼び出しではパラメーターが許可され、optionExact_ref の呼び出しではパラメーターが許可されません。

1. プロトタイプ・パラメーターの長さは 5 で、渡されるパラメーターの長さは 10 です。 プロシージャー noOptionExact_ref は最初の 5 文字のみを受け取ります。
2. プロトタイプ・パラメーターでは配列を降順にする必要がありますが、渡されるパラメーターは昇順になっています。
3. プロトタイプ・パラメーターは 3 つの要素を持つ配列であり、渡されるパラメーターには 4 つの要素があります。 プロシージャー noOptionExact_ref は最初の 3 つの要素のみを受け取ります。
4. プロトタイプ・パラメーターは LIKEDS(orderInfo_t) で定義され、渡されるパラメーターは LiKEDS(customerInfo_t) で定義されます。
5. プロトタイプ・パラメーターは、LIKE(price_t) で定義されます。これは品目の価格です。 渡されるパラメーターは、顧客口座の残高を参照します。 ここでは *accountBalance* が *price_t* と同じように定義されているので、noOptionExact_ref の呼び出しが可能です。

```
DCL-PR noOptionExact_ref;
   char5Parm CHAR(5);
   arrayDescendParm CHAR(5) DIM(3) DESCEND;
   arrayDim3Parm CHAR(5) DIM(3);
   orderInfoParm LIKEDS(orderInfo_t);
   itemPriceParm LIKE(price_t);
END-PR;
DCL-PR optionExact_ref;
   char5Parm CHAR(5) OPTIONS(*EXACT);
   arrayDescendParm CHAR(5) DIM(3) DESCEND OPTIONS(*EXACT);
   arrayDim3Parm CHAR(5) DIM(3) OPTIONS(*EXACT);
   orderInfoParm LIKEDS(orderInfo_t) OPTIONS(*EXACT);
   itemPriceParm LIKE(price_t) OPTIONS(*EXACT);
END-PR;
DCL-S char10 CHAR(10);
DCL-S arrayAscend CHAR(5) DIM(3) ASCEND;
DCL-S arrayDim4 CHAR(5) DIM(4);
DCL-DS customerInfo LIKEDS(customerInfo_t);
DCL-S accountBalance PACKED(7:2);

noOptionExact_ref
             (char10            // char5Parm  1
            : arrayAscend       // arrayDescendParm  2
            : arrayDim4         // arrayDim3Parm  3
            : customerInfo      // orderInfoParm  4
            : accountBalance);  // itemPriceParm  5
optionExact_ref
             (char10            // Error: char5Parm  1
            : arrayAscend       // Error: arrayDescendParm  2
            : arrayDim4         // Error: arrayDim3Parm  3
            : customerInfo      // Error: orderInfoParm  4
            : accountBalance);  // Error: itemPriceParm  5
```

図 *148.* 参照によって渡されるパラメーターを指定した *OPTIONS(*EXACT)* の使用

VALUE および CONST を指定した OPTIONS(*EXACT)

値または定数参照によって渡されるパラメーターで OPTIONS(*EXACT) が指定される場合、渡されるパラメーターには以下の追加規則が適用されます。

注 : この例では CONST キーワードが使用されていますが、VALUE キーワードが使用されている場合も規則はまったく同じです。

- データ・タイプは同じである必要があります。 OPTIONS(*EXACT) が指定されていない場合、タイプが英数字、UCS-2、またはグラフィックのパラメーターは、タイプが英数字、UCS-2、またはグラフィックのいずれかのプロトタイプ・パラメーターに渡すことができ、コンパイラーは、プロトタイプ・パラメーターのデータ・タイプへの暗黙的な変換を実行します。 場合によっては、パラメーターが渡されるときにデータが失われることがあります。これは、変換されたパラメーターの長さがプロトタイプ・パラメーターよりも長いか、渡されるパラメーターの文字をプロトタイプ・パラメーターのデータ・タイプに変換できないことが原因です。

  以下の例では、フィールド *ucs2Fld* をパラメーター *parmNoExact* に渡すことはできますが、プロシージャーが受け取るデータは *ucs2Fld* のデータと同じにならない可能性があります。 OPTIONS (*EXACT) を指定して定義された *parmExact* にフィールド *ucs2Fld* を渡すことはできません。このフィールドのタイプは UCS-2 であり、パラメーターのタイプは英数字であるためです。

```
DCL-PR pr1;
   parmNoExact CHAR(5) CONST;
   parmExact CHAR(5) CONST OPTIONS(*EXACT);
END-PR;
DCL-S ucs2Fld UCS2(5);
```

- CCSID は同じである必要があります。 OPTIONS(*EXACT) が指定されていない場合、それぞれ CCSID が異なるパラメーターをプロトタイプ・パラメーターに渡すことができ、コンパイラーは、プロトタイプ・パラメーターの CCSID への暗黙的な変換を実行します。 場合によっては、パラメーターが渡されるときにデータが失われることがあります。これは、変換されたパラメーターの長さがプロトタイプ・パラメーターよりも長いか、渡されるパラメーターの文字をプロトタイプ・パラメーターの CCSID に変換できないことが原因です。

  以下の例では、フィールド *utf8Fld* をパラメーター *parmNoExact* に渡すことはできますが、プロシージャーが受け取るデータは *utf8Fld* のデータと同じにならない可能性があります。 フィールド *utf8Fld* の CCSID が異なるため、OPTIONS(*EXACT) で定義された *parmExact* にそのフィールドを渡すことができません。

```
DCL-PR pr2;
   parmNoExact CHAR(5) CONST;
   parmExact CHAR(5) CONST OPTIONS(*EXACT);
END-PR;
DCL-S utf8Fld CHAR(5) CCSID(*UTF8);
```

- 英数字、UCS-2、またはグラフィックの場合、渡されるパラメーターの可変長属性は、プロトタイプ・パラメーターと同じである必要はありません。 ただし、渡されるパラメーターの定義された長さは、プロトタイプ・パラメーター以下である必要があります。 渡されるパラメーターがリテラルまたは式の場合、値の長さは、プロトタイプ・パラメーターに指定された長さ以下である必要があります。

  以下の例のフィールド *char4Fld*、*varchar4Fld*、*char5Fld*、および *varchar5Fld* は、プロシージャー pr3 のすべてのパラメーターに渡すことができます。定義された長さが、プロトタイプ・パラメーターの定義された長さ以下であるためです。

  フィールド *char6Fld* および *varchar6Fld* は *parmNoExact* および *parmVaryingNoExact* に渡すことができますが、呼び出されるプロシージャーは、渡されるパラメーター内の一部のデータを受信しない可能性があります。 フィールド *char6Fld* および *varchar6Fld* は、定義された長さがプロトタイプ・パラメーターの定義された長さよりも長いため、OPTIONS(*EXACT) で定義された *parmxact* および *parmVaryingExact* に渡すことができません。

```
DCL-PR pr3;
   parmNoExact         CHAR(5) CONST;
   parmVaryingNoExact  VARCHAR(5) CONST;
   parmExact           CHAR(5) CONST OPTIONS(*EXACT);
   parmVaryingExact    VARCHAR(5) CONST OPTIONS(*EXACT);
END-PR;
DCL-S char4Fld CHAR(4);
DCL-S varchar4Fld VARCHAR(4);
DCL-S char5Fld CHAR(5);
DCL-S varchar5Fld VARCHAR(5);
DCL-S char6Fld CHAR(6);
DCL-S varchar6Fld VARCHAR(6);
```

- ASCEND キーワードまたは DESCEND キーワードには、渡されるパラメーターとプロトタイプ・パラメーターの両方で同じものを指定する必要があります。
- 配列の次元は同じである必要があります。 プロトタイプに OPTIONS(*EXACT) が指定されている場合、変動次元配列を渡すことができません。
- 小数点以下の桁数は、プロトタイプ・パラメーターの小数点以下の桁数以下である必要があります。

  次の例のフィールド *packed_5_2* および *zoned_3_1* は、プロシージャー pr4 の両方のパラメーターに渡すことができます。定義されている小数点以下の桁数が、プロトタイプ・パラメーターの定義された小数点以下の桁数の 2 以下であるためです。

  フィールド *packed_5_3* をパラメーター *parmNoExact* に渡すことはできますが、呼び出されるプロシージャーがパラメーターと同じ値を受け取ることはできません。 *packed_5_3* の値が 12.345 の場合、呼び出されるプロシージャーは、値 12.34 を受け取ります。

  フィールド *packed_5_3* は、パラメーター *parmExact* に渡すことができません。定義されている小数点以下の桁数が、プロトタイプ・パラメーターの定義されている小数点以下の桁数より大きいためです。

```
DCL-PR pr4;
   parmNoExact PACKED(5:2) CONST;
   parmExact   PACKED(5:2) CONST OPTIONS(*EXACT);
END-PR;
DCL-S packed_5_2 PACKED(5:2);
DCL-S zoned_3_1 ZONED(3:1);
DCL-S packed_5_3 PACKED(5:3);
```

- 渡されるパラメーターの整数の桁数は、プロトタイプ・パラメーターの整数の桁数以下である必要があります。

  以下の例のフィールド *packed_4_2* の整数桁数は 2 つ、*int_3* は 3 つ、*packed_7_3* は 4 つです。

  フィールド *packed_4_2* および *int_3* は、プロシージャー pr4 のすべてのパラメーターに渡すことができます。定義されている整数桁数が、プロトタイプ・パラメーターの定義されている整数桁数以下であるためです。

  フィールド *packed_7_3* はパラメーター *parmNoExact* に渡すことができますが、*packed_7_3* の値が 999.99 より大きいか、-999.99 より小さい場合、呼び出しは失敗し、数値オーバーフロー例外が発生する可能性があります。

  フィールド *packed_7_4* は、パラメーター *parmExact* に渡すことができません。定義されている整数の桁数が、プロトタイプ・パラメーターの定義されている整数の桁数より大きいためです。

```
DCL-PR pr5;
   parmNoExact PACKED(5:2) CONST;
   parmExact   PACKED(5:2) CONST OPTIONS(*EXACT);
END-PR;
DCL-S packed_4_2 PACKED(4:2);
DCL-S int_3 INT(3);
DCL-S packed_7_3 PACKED(7:3);
```

- タイム・スタンプ・パラメーターの長さは、プロトタイプに指定されている長さ以下である必要があります。
- 浮動小数点パラメーターの長さは、プロトタイプ・パラメーターの長さ以下である必要があります。プロトタイプが浮動小数点として定義されている場合、渡されるパラメーターも浮動小数点である必要があります。 プロトタイプ・パラメーターが浮動小数点として定義されていない場合、渡されるパラメーターを浮動小数点にすることはできません。
- 日付パラメーターの年の範囲は、プロトタイプに指定された日付形式の年の範囲内である必要があります。 例えば、*YMD 形式の年の範囲は 1940 から 2039 であり、*ISO 形式の年の範囲より狭くなっています。
- *USA 形式の時刻パラメーターは、プロトタイプ・パラメーターも *USA 形式である場合にのみ有効です。
- オブジェクト・パラメーターの Java クラスは一致している必要があります。
- プロトタイプ・パラメーターがデータ構造である場合、渡されるパラメーターは、LIKEDS または LIKEREC によってプロトタイプ・パラメーターに関連付けられている必要があります。

  以下の例では、データ構造 *customerInfo* をパラメーター *orderInfoNoExact* に渡すことができますが、呼び出されるプロシージャーが LIKEDS(orderInfo_t) で定義されたデータ構造として解釈される場合、*customerInfo* のデータが無効になる可能性があります。

  データ構造 *customerInfo* をパラメーター *orderInfoExact* に渡すことはできません。LIKEDS キーワードによって、OPTIONS(*EXACT) で定義されたプロトタイプ・パラメーターに関連付けられていないためです。

```
DCL-PR pr5;
   orderInfoNoExact LIKEDS(orderInfo_t) CONST;
   orderInfoExact   LIKEDS(orderInfo_t) CONST OPTIONS(*EXACT);
END-PR;
DCL-DS customerInfo LIKEDS(customerInfo_t);
```

- LIKE キーワードを使用してプロトタイプ・パラメーターが定義されている場合、渡されるパラメーターは LIKE キーワードによってプロトタイプ・パラメーターに関連付けられている必要があります。

  以下の例では、フィールド *accountBalance* をパラメーター *itemPriceNoExact* に渡すことができますが、口座の残高を保持する変数内のデータが、品目の価格であると想定されるパラメーターに対して有効でない可能性があります。

  フィールド *accountBalance* は、LIKE キーワードによってプロトタイプ・パラメーターに関連付けられていないため、パラメーター *itemPriceExact* に渡すことができません。

```
DCL-S price_t PACKED(7:2) TEMPLATE;
DCL-PR pr6;
   itemPriceNoExact LIKE(price_t);
   itemPriceExact   LIKE(price_t) OPTIONS(*EXACT);
END-PR;

DCL-S accountBalance PACKED(7:2);
```

OPTIONS(*STRING)

値または定数参照によって渡される基底ポインター・パラメーター に OPTIONS(*STRING) を指定する場合、ポインターまたは文字式のいずれかを 渡してもかまいません。 文字式を渡す場合、後ろに ヌル文字終了記号 (x'00') が 付いている文字式の値を含んだ一時値が作成 されます。 この一時値のアドレスは、呼び出されるプログラムまたはプロシージャーに 渡されます。

次の例は、ヌル文字で終了するストリング・パラメーターを使用する プロトタイプおよびプロシージャーをコーディングするため に OPTIONS(*STRING) を使用する方法を示しています。

```
 * The following prototype describes a procedure that expects
 * a null-terminated string parameter.  It returns the length
 * of the string.
D StringLen       PR             5U 0
D   Pointer                       *   VALUE OPTIONS(*STRING)
D P               S               *
D Len             S              5U 0
 * Call StringLen with a character literal.  The result will be
 * 4 since the literal is 4 bytes long.
C                   EVAL      Len = StringLen('abcd')
 * Call StringLen with a pointer to a string.  Use ALLOC to get
 * storage for the pointer, and use %STR to initialize the storage
 * to 'My stringÂ¬' where 'Â¬' represents the null-termination
 * character x'00'.
 * The result will be 9 which is the length of 'My string'.
C                   ALLOC     25              P
C                   EVAL      %STR(P:25) = 'My string'
C                   EVAL      Len = StringLen(P)
 * Free the storage.
C                   DEALLOC                   P
C                   RETURN
 *---------------------------------------------------------
 * StringLen:
 *   Returns the length of the string that the parameter is
 *   pointing to.
 *---------------------------------------------------------
P StringLen       B
D StringLen       PI             5U 0
D   Pointer                       *   VALUE OPTIONS(*STRING)
C                   RETURN    %LEN(%STR(Pointer))
P StringLen       E
```

図 *149. OPTIONS(*STRING)* の使用

OPTIONS(*RIGHTADJ)

プロトタイプ内の CONST パラメーターまたは VALUE パラメーター に OPTIONS(*RIGHTADJ) を指定すると、文字、図形、または UCS-2 パラメーター値 は右寄せされます。 このキーワードは、プロシージャー・プロトタイプ内の可変長パラメーターに は使用できません。 可変長値は、対応のパラメーターが OPTIONS(*RIGHTADJ) を指定して 定義されているプロシージャー呼び出し上で、パラメーターとして 渡すことができます。

OPTIONS(*TRIM)

文字、UCS-2、またはグラフィック・タイプの CONST または VALUE パラメーターに OPTIONS(*TRIM) を指定した場合、渡されたパラメーターは、先行および後書きブランクなしで一時ファイルにコピーされます。 このパラメーターが可変長パラメーターでない場合、トリミングされた値にはブランクが 埋め込まれます (OPTIONS(*RIGHTADJ) が指定されている場合には左側に、 それ以外の場合には右側に埋め込まれます)。 そして、この一時ファイルがオリジナル・パラメーターの代わりに渡されます。 OPTIONS(*TRIM) を指定すると、パラメーターは、 プロシージャーに対するそれぞれの呼び出しで %TRIM がコーディングされた場合とまったく同様に渡されます。

ポインター・タイプの CONST または VALUE パラメーターに OPTIONS(*STRING : *TRIM) を 指定した場合、文字パラメーターまたはポインター・パラメーターの %STR が、先行または後書きブランクなしで一時フ

ァイルにコピーされ、 ヌル終止符が一時ファイルに追加されて、その一時ファイルのアドレスが渡されます。

```
* The following prototype describes a procedure that expects
* these parameters:
*   1. trimLeftAdj   - a fixed length parameter with the
*                      non-blank data left-adjusted
*   2. leftAdj       - a fixed length parameter with the
*                      value left-adjusted (possibly with
*                      leading blanks)
*   3. trimRightAdj  - a fixed length parameter with the
*                      non-blank data right-adjusted
*   4. rightAdj      - a fixed length parameter with the
*                      value right-adjusted (possibly with
*                      trailing blanks)
*   5. trimVar       - a varying parameter with no leading
*                      or trailing blanks
*   6. var           - a varying parameter, possibly with
*                      leading or trailing blanks
D trimProc        PR
D   trimLeftAdj                 10a   const options(*trim)
D   leftAdj                     10a   const
D   trimRightAdj                10a   value options(*rightadj : *trim)
D   rightAdj                    10a   value options(*rightadj)
D   trimVar                     10a   const varying options(*trim)
D   var                         10a   value varying
* The following prototype describes a procedure that expects
* these parameters:
*   1. trimString    - a pointer to a null-terminated string
*                      with no leading or trailing blanks
*   2. string        - a pointer to a null-terminated string,
*                      possibly with leading or trailing blanks
```

図 *150. OPTIONS(*TRIM)* の使用

```
D trimStringProc  PR
D   trimString                   *   value options(*string : *trim)
D   string                       *   value options(*string)
D ptr             s              *
 /free
       // trimProc is called with the same value passed
       // for every parameter
       //
       // The called procedure receives the following parameters
       //   trimLeftAdj    'abc       '
       //   leftAdj        ' abc      '
       //   trimRightAdj   '       abc'
       //   rightAdj       '      abc '
       //   trimVar        'abc'
       //   var            ' abc '

       callp trimProc (' abc ' : ' abc ' : ' abc ' :
                                ' abc ' : ' abc ' : ' abc ' );

       // trimStringProc is called with the same value passed
       // for both parameters
       //
       // The called procedure receives the following parameters,
       // where Â¬ represents x'00'
       //   trimstring     pointer to 'abcÂ¬'
       //   string         pointer to ' abc Â¬'

       callp trimStringProc (' abc ' : ' abc ');

       // trimStringProc is called with the same pointer passed
       // to both parameters
       //
       // The called procedure receives the following parameters,
       // where Â¬ represents x'00'
       //   trimstring     pointer to 'xyzÂ¬'
       //   string

       pointer to ' xyz Â¬'
       ptr = %alloc (6);
       %str(ptr : 6) = ' xyz ';
       callp trimStringProc (ptr : ptr);
```

OPTIONS(*NULLIND)

OPTIONS(*NULLIND) がパラメーターに指定された場合、そのパラメーターと共にヌル・バイト・マップが渡され、着呼側プロシージャーが発呼者のパラメーターのヌル・バイト・マップに直接アクセスできるようになります。 OPTIONS(*NULLIND) の以下の規則に注意してください。

- ALWNULL(*USRCTL) が有効になっていなければなりません。
- OPTIONS(*NULLIND) は、値によって受け渡されるパラメーターでは無効です。
- OPTIONS(*NULLIND) は、NULLIND キーワードで定義されたパラメーターには無効です。
- OPTIONS(*NULLIND) と一緒に指定可能なその他のオプションは、*NOPASS および *OMIT のみです。
- OPTIONS(*NULLIND) が指定されている場合に、変数のみパラメーターとして受け渡し可能で、 変数は CONST が指定されている場合でも正確に一致する必要があります。
- パラメーターがデータ構造になっている場合、受け渡されたパラメーターは、プロトタイプ化されているパラメーターと同じ親 LIKEDS または LIKEREC で定義されている必要があります。 また、プロトタイプ・パラメーターのヌル機能、および受け渡されたパラメーターは、 正確に一致している必要があります。
- プロトタイプ化されたデータ構造パラメーターには、ヌル可能サブフ ィールドが存在している場合でもしていない場合でも、OPTIONS(*NULLIND) を指定可能です。
- 非データ構造プロトタイプ化パラメーターが OPTIONS(*NULLIND) で定義される場合、プロシージャー・インターフェース内のパラメーターは、ヌル可能として定義されます非データ構造プロトタイプ化パラメーターが OPTIONS(*NULLIND) で定義される場合、 プロシージャー・インターフェース内のパラメーターは、ヌル可能として定義されます。

- 呼び出し側のプロシージャーまたは呼び出されるプロシージャーが ILE RPG を使用して作成されていない場合の OPTIONS(*NULLIND) の使用については、*Rational Development Studio for i: ILE RPG* プログラマーの手引きを参照してください。

```
 *-----------------------------------
 * DDS for file NULLFILE
 *-----------------------------------

A          R TESTREC
A            NULL1         10A         ALWNULL
A            NOTNULL2      10A
A            NULL3         10A         ALWNULL


 *-----------------------------------
 * Calling procedure
 *-----------------------------------

 * The externally-described data structure DS, and the
 * data structure DS2 defined LIKEDS(ds) have
 * null-capable fields NULL1 and NULL3.

D ds             E DS                  EXTNAME(nullFile)
D ds2              DS                  LIKEDS(ds)
 * Procedure PROC specifies OPTIONS(*NULLIND) for all its
 * parameters.  When the procedure is called, the
 * null-byte maps of the calling procedure's parameters
 * will be passed to the called procedure allowing the
 * called procedure to use %NULLIND(parmname) to access the
 * null-byte map.

D proc             PR
D   parm                               LIKEDS(ds)
D                                      OPTIONS(*NULLIND)
D   parm2                        10A   OPTIONS(*NULLIND)
D   parm3                        10A   OPTIONS(*NULLIND) CONST

 /free
     // The calling procedure sets some values
     // in the parameters and their null indicators

     %nullind(ds.null1) = *on;
     ds.notnull2 = 'abcde';
     ds.null3 = 'fghij';
     %nullind(ds.null3) = *off;
     ds2.null1 = 'abcde';
     %nullind(ds2.null1) = *on;
     %nullind(ds3.null3) = *off;
     // The procedure is called (see the code for
     // the procedure below

     proc (ds : ds2.null1 : ds2.null3);

     // After "proc" returns, the calling procedure
     // displays some results showing that the
     // called procedure changed the values of
     // the calling procedure's parameters and
     // their null-indicators

     dsply (%nullind(ds.null1));  // displays '0'

     dsply ds2.null2;             // displays 'newval'

     dsply (%nullind(ds2.null2)); // displays '0'

 /end-free
```

図 *151. OPTIONS(*NULLIND)* の使用

```
 *----------------------------------
 * Called procedure PROC
 *----------------------------------

P                 B
D proc            PI
D   parm                              LIKEDS(ds)
D                                     OPTIONS(*NULLIND)
D   parm2                       10A   OPTIONS(*NULLIND)
D   parm3                       10A   OPTIONS(*NULLIND) CONST
 /free
     if %NULLIND(parm.null1);
        // This code will be executed because the
        // caller set on the null indicator for
        // subfield NULL1 of the parameter DS

     endif;

     if %NULLIND(parm3);
        // PARM3 is defined as null-capable since it was
        // defined with OPTIONS(*NULLIND).
        // This code will not be executed, because the
        // caller set off the null-indicator for the parameter

     endif;

     // Change some data values and null-indicator values
     // The calling procedure will see the updated values.

     parm2 = 'newvalue';
     %NULLIND(parm2) = *OFF;
     %NULLIND(parm.null1) = *OFF;
     parm.null1 = 'newval';
     return;
 /end-free
P                 E
```

OVERLAY(名前 {: 開始位置 | *NEXT })

OVERLAY キーワードは、固定形式および自由形式で使用できます。 これは、データ構造サブフィールドに対してのみ使用できます。

OVERLAY キーワードは、1 つのサブフィールドの記憶域を別の記憶域によってオーバーレイするか、 または、固定形式定義では、データ構造自体の記憶域によってオーバーレイします。

名前パラメーターで指定された記憶域が、開始位置パラメーターで指定された位置で、 「名前」記入項目のサブフィールドによってオーバーレイされます。 開始位置が 指定されていない場合、1 にデフォルト設定されます。

注 : サブフィールドのタイプに関係なく、開始位置パラメーターはバイト 単位です。

OVERLAY(名前:*NEXT) を指定すると、サブフィールドを、オーバーレイされたフィールド 内の次に使用可能な位置に置きます。 (これは、このサブフィールドより前にあり、同じサブフィールドをオーバーレイ する他のすべてのサブフィールドを過ぎた後の、最初のバイトです。)

キーワード OVERLAY には次の規則が適用されます。

1. 名前パラメーターは、現行データ構造内の事前定義済みのサブフィールドの名前である必要があります。
2. 固定形式定義では、現行データ構造の名前をその名前に指定することもできます。 自由形式定義では、別のサブフィールド内のサブフィールドの位置指定に使用できるのは OVERLAY キーワードのみです。 データ構造内のサブフィールドの位置を指定するには、POS キーワードを使用してください。
3. データ構造が修飾されている場合は、OVERLAY キーワードに対する最初のパラメーターは、データ構造名を修飾せずに指定する必要があります。 以下の例では、サブフィールド MsgInfo.MsgPrefix がサブフィールド MsgInfo.MsgId をオーバーレイします。

```
D MsgInfo         DS                  QUALIFIED
D   MsgId                        7
D     MsgPrefix                  3    OVERLAY(MsgId)
```

4. 開始位置パラメーターは (指定される場合は)、小数点以下の桁を持たない 0 より大きい 値でなければなりません。 これは、数値リテラル、数値を戻す組み込み関数、または数値定数とすること ができます。 開始位置が名前付き定数の場合、この指定の前にその定数が定義されている必要があります。
5. OVERLAY キーワードは「開始位置」記入項目がブランクでない時には使用す ることができません。
6. 名前パラメーターがサブフィールドである場合、定義中のサブフィールドは、 名前パラメーターによって指定されたサブフィールドの中に完全に含まれていなければなりません。

   2 つのサブフィールドを他の方法で関連付けずに、あるサブフィールドをもう一方のサブフィールドと同じ開始位置に置くには、SAMEPOS キーワードを使用します。
7. OVERLAY キーワードを使用して定義したサブフィールドの位置合わせは、手操 作で行なわなければなりません。 正しく位置合わせされなかった場合には、警告メッセージが出されます。
8. OVERLAY キーワードの最初のパラメーターとして指定されたサブフィールド が配列の場合には、OVERLAY キーワードは配列の各要素に適用されます。 すなわち、定義中のフィールドは同じ要素数を持つ配列として定義されます。 この配列の最初の要素は オーバーレイされる配列の最初の要素にオーバーレイし、この 配列の 2 番目の要素はオーバーレイされる配列の 2 番目の要素にオーバーレイし、以下同様となります。 この状況では、OVERLAY キーワードを持つサブフィールドに配列キーワードを指 定することはできません。 (469 ページの図 152 を参照 してください) 881 ページの『SORTA (配列の分類)』も参照してください。

   OVERLAY キーワードの最初のパラメーターとして指定されたサブフィールド名が 配列で、その要素の長さが定義中のサブフィールドの長さより長い場合には、定 義中のサブフィールドの配列要素は連続して記憶されません。 このような配列は、PARM 命令の結果のフィールドとして、または MOVEA 命令の 演算項目 2 または結果のフィールドの中では使用することができません。
9. データ構造に ALIGN キーワードを指定する と、OVERLAY(名前:*NEXT) を指定して定義されたサブフィールド は、その任意の位置に位置合わせされます。 ポインター・サブフィールドは、常に 16 バイト境界に位置合わせされます。
10. オーバーレイしているサブフィールド をもつサブフィールドが定義されていない場合、そのサブフィールド は次のように暗黙的に定義されます。
    - 開始位置は、データ構造内で最初に使用可能な位置です。
    - 長さは、オーバーレイしているサブフィールドをすべて含むことが できる最低の長さです。 そのサブフィールドが配列として定義されている 場合、オーバーレイしているサブフィールドがすべて適切に位置合わせされる ように、長さが延長されます。

例

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
D DataStruct      DS
D   A                            10    DIM(5)
D     B                           5    OVERLAY(A)
D     C                           5    OVERLAY(A:6)
```

図 *152.* キーワード *DIM* および *OVERLAY* を持つサブフィールドの記憶域割り振り

記憶域でのフィールドの割り振り:

A(1)		A(2)		A(3)		A(4)		A(5)	
B(1)	C(1)	B(2)	C(2)	B(3)	C(3)	B(4)	C(4)	B(5)	C(5)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D DataStruct      DS
D   A                             5
D     B                           1    OVERLAY(A) DIM(4)
```

図 *153.* キーワード *DIM* および *OVERLAY* を持つサブフィールドの記憶域割り振り

記憶域でのフィールドの割り振り:

A				
B(1)	B(2)	B(3)	B(4)	

次の例は、サブフィールドのオーバーレイ位置を定義するための等価の方法を 2 つ示しています。 1 つは (名前:開始位置) を使用して明示的に行う方法で、もう 1 つは (名前:*NEXT) を使用して暗黙的に行う方法です。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
 * Define subfield overlay positions explicitly
D DataStruct      DS
D   PartNumber                  10A
D     Family                     3A   OVERLAY(PartNumber)
D     Sequence                   6A   OVERLAY(PartNumber:4)
D     Language                   1A   OVERLAY(PartNumber:10)
```

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... *
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
 * Define subfield overlay positions with *NEXT
D DataStruct      DS
D   PartNumber
D     Family                     3A   OVERLAY(PartNumber)
D     Sequence                   6A   OVERLAY(PartNumber:*NEXT)
D     Language                   1A   OVERLAY(PartNumber:*NEXT)
```

図 *154. NEXT* を使用したサブフィールド・オーバーレイ位置の定義

以下の例は、自由形式で定義された同じデータ構造を示しています。

```
// Define subfield overlay positions explicitly
DCL-DS DataStruct;
   PartNumber CHAR(10);
      Family CHAR(3) OVERLAY(PartNumber);
      Sequence CHAR(6) OVERLAY(PartNumber:4);
      Language CHAR(1) OVERLAY(PartNumber:10);
END-DS;

// Define subfield overlay positions with *NEXT
DCL-DS DataStruct;
   PartNumber CHAR(10);
      Family CHAR(3) OVERLAY(PartNumber);
      Sequence CHAR(6) OVERLAY(PartNumber:*NEXT);
      Language CHAR(1) OVERLAY(PartNumber:*NEXT);
END-DS;
```

図 *155.* 自由形式でのサブフィールドのオーバーレイ位置の定義

OVERLOAD(プロトタイプ 1 { : プロトタイプ 2 ...})

OVERLOAD キーワードは、OVERLOAD キーワードが指定されたプロトタイプの名前を使用して呼び出すことができる他のプロトタイプのリストを定義します。 OVERLOAD キーワードが指定されたプロトタイプ

が呼び出し操作で使用される場合、コンパイラーは、呼び出しに指定されたパラメーターを使用して、OVERLOAD キーワードにリストされているプロトタイプのうちどれを呼び出すかを決定します。

以下の例の *FORMAT* は、OVERLOAD キーワードで定義されています。

1. *FORMAT* の最初の呼び出しでは、パラメーターに日付タイプが指定されているため、*FORMAT_DATE* が呼び出されます。
2. *FORMAT* の 2 回目の呼び出しでは、両方のパラメーターに文字タイプが指定されているので、*RETRIEVE_MESSAGE* が呼び出されます。

```
DCL-PR format_date VARCHAR(100);
   dateParm DATE(*ISO) CONST;
END-PR;
DCL-PR retrieve_message VARCHAR(100);
   msgid CHAR(7) CONST;
   replacement_text VARCHAR(100) CONST;
END-PR;
DCL-PR format VARCHAR(100) OVERLOAD(format_date : retrieve_message);
DCL-S result VARCHAR(50);
DCL-S filename CHAR(10);

result = format(%date());              // 1
result = format('MSG0100' : filename); // 2
```

この説明で使用されている用語

- 「プロトタイプ」という用語は、同じモジュール内にプロトタイプなしで定義されているプロシージャーの、明示的プロトタイプと暗黙的プロトタイプの両方を指します。
- 「多重定義プロトタイプ」という用語は、OVERLOAD キーワードが指定されたプロトタイプを指します。
- 「候補プロトタイプ」という用語は、OVERLOAD キーワードにリストされているプロトタイプを指します。
- 上記の例では、*FORMAT* は「多重定義プロトタイプ」であり、*FORMAT_TIME* と *FORMAT_DATE* は「候補プロトタイプ」です。

OVERLOAD キーワードの規則

- OVERLOAD キーワードが指定されたプロトタイプには、パラメーターは指定されません。
- OVERLOAD キーワードが指定されたプロトタイプは、END-PR で終了しません。
- すべての候補プロトタイプは、多重定義プロトタイプと同じ戻り値のタイプ (または戻り値なし) である必要があります。
- 多重定義プロトタイプに使用できるその他のキーワードは、戻り値のデータ・タイプに関連するもののみです。
- 候補プロトタイプには、任意のタイプのプロトタイプを使用できます。 すべてが同じタイプでなければならないというわけではありません。 例えば、プログラム、プロシージャー、および Java メソッドを多重定義プロトタイプの候補プロトタイプに使用できます。

RPG コンパイラーが呼び出すプロシージャーまたはプログラムを決定する方法

特定の呼び出し操作のために渡されたパラメーターごとに、コンパイラーは、渡されたパラメーターがそれぞれの候補プロトタイプに対して有効かどうかを検査します。 渡されたパラメーターが特定の候補プロトタイプに対して有効でない場合、コンパイラーはそのプロトタイプをその特定の呼び出しの候補として考慮しなくなります。 コンパイラーがすべてのパラメーターの検査を終えた時点で、すべてのパラメーターに対して有効なプロトタイプがちょうど 1 つのみ残っているはずです。

特定のパラメーターに最も適合するプロトタイプがどのプロトタイプにあるかについては考慮されません。 例えば、渡されたパラメーターがタイプ PACKED(8)で定義されており、1 つの候補プロトタイプがキ

ーワード PACKED(5:2) および CONST でそのパラメーターを定義し、別の候補プロトタイプがそのパラメーターをキーワード PACKED(15:5) および CONST で定義していると想定します。 この呼び出しでは、渡された PACKED(8) のパラメーターとの適合性が高いのは PACKED(15:5) パラメーターですが、コンパイラーは、パラメーターがこれら両方の候補プロトタイプに適合するとみなします。

コンパイラー・リスト内の「多重定義プロトタイプ」セクション

リストの「多重定義プロトタイプ」セクションを使用して、多重定義プロトタイプを呼び出すたびに、どの候補プロトタイプが呼び出されるかを調べることができます。 多重定義プロトタイプごとに、すべての候補プロトタイプがリストされ、そのプロトタイプが実際に呼び出しで使用されるステートメントのリストが表示されます。

/OVERLOAD 指示を指定することによって、使用する候補プロトコルをコンパイラーが決定した方法を示す詳細リストを入手できます。 「/OVERLOAD DETAIL」が有効になっているときに指定された多重定義プロトタイプの呼び出しは、リストの「多重定義プロトタイプ」セクションにも詳細な情報が表示されます。この詳細情報には、渡されたパラメーターとプロトタイプ・パラメーターが一致するかどうかをコンパイラーが検査しているときに、コンパイラーによって内部的に出されるすべてのエラー・メッセージが含まれます。

```
DCL-PR format_date VARCHAR(100);
   dateParm DATE(*ISO) CONST;
END-PR;
DCL-PR format_time VARCHAR(100);
   timeParm TIME(*ISO) CONST;
END-PR;
DCL-PR format_message VARCHAR(100);
   msgid CHAR(7) CONST;
   replacement_text VARCHAR(100) CONST;
END-PR;
DCL-PR format VARCHAR(100)
             OVERLOAD(format_time : format_date : format_message);
DCL-S result varchar(50);

/OVERLOAD DETAIL
result = format(%date());
result = format('MSG0102');
```

上記の例のリストにある「多重定義プロトタイプ」セクションを以下に示します。

```
     Calls to prototypes for FORMAT
         Called prototype                    References (D=Details below)
           FORMAT_TIME
           FORMAT_DATE                           16D
           FORMAT_MESSAGE
           No prototype selected                 17D
     Detailed determination for calls to FORMAT
       Call at statement 16 column 18
         Error messages issued for parameter 1 for FORMAT_TIME
*RNF7536 30     16 001600  The type of parameter 1 specified for the call does
                           match the prototype.
         Error messages issued for parameter 1 for FORMAT_MESSAGE
*RNF7536 30     16 001600  The type of parameter 1 specified for the call does
                           match the prototype.
         Selected prototype: FORMAT_DATE
       Call at statement 17 column 18
         Error messages issued for parameter 1 for FORMAT_TIME
*RNF7536 30     17 001700  The type of parameter 1 specified for the call does not
                           match the prototype.
         Error messages issued for parameter 1 for FORMAT_DATE
*RNF7536 30     17 001700  The type of parameter 1 specified for the call does not
         The call had too few parameters for these prototypes:
            FORMAT_MESSAGE
         Selected prototype: *N
```

/OVERLOAD 指示について詳しくは、83 ページの『/OVERLOAD DETAIL | NODETAIL』を参照してください。

PACKED(桁数 {: 小数点以下の桁数})

PACKED キーワードは、数字データ・タイプのキーワードの 1 つです。 これは自由形式定義で使用されて、項目 の形式がパック 10 進数形式であることを示します。

これは最初のキーワードでなければなりません。

最初のパラメーターは必須です。 これは総桁数を指定します。 1 から 63 までの値を指定できます。

2 番目のパラメーターは任意指定です。 これは小数点以下の桁数を指定します。 ゼロから桁数までの範囲の 値を指定できます。 デフォルトはゼロです。

各パラメーターにはリテラルまたは名前付き定数を指定できます。 名前付き定数である場合、定義ステートメントの前にその定数が定義されている必要があります。

次に例を示します。

- フィールド *salary* は、桁数が 5 で小数点以下の桁数が 2 のパック 10 進数フィールド として定義されています。
- フィールド *age* は、桁数が 3 で小数点以下の桁数がデフォルトの 0 であるパック 10 進数フィールド として定義されています。
- フィールド *price* は、桁数が 7 で小数点以下の桁数が 3 のパック 10 進数フィールド として定義されています。 小数点以下の桁数は、名前付き定数 *NUM_DEC_POS* を使用して定義されています。

```
DCL-S salary PACKED(5 : 2);
DCL-S age PACKED(3);
DCL-C NUM_DEC_POS 3;
DCL-S price PACKED(7 : NUM_DEC_POS);
```

PACKEVEN

PACKEVEN キーワードは、パック 10 進数フィールドまたは配列の桁数が偶数にな ることを指示します。 このキーワードが有効なのは、「開始位置/終了位置」の桁を使用して定義され たパック形式のプログラム記述データ構造サブフィールドの場合だけです。 長さが N のフィールドまたは配列要素の場合には、PACKEVEN キーワードが指定さ れなければ桁数は 2N -1 になり、PACKEVEN が指定されれば桁数は 2(N-1) にな ります。

PERRCD(数値定数)

PERRCD キーワードによって、コンパイル時または実行時前配列またはテーブル用 のレコード当りの要素の数を指定することができます。 PERRCD キーワードが指定されない場合には、レコード当りの要素の数のデフォルトの 値として 1 が使用されます。

この数値定数パラメーターは、小数点以下の桁数のない 0 より大きい値でな ければなりません。 これは、数値リテラル、数値を戻す組み込み関数、または数値定数とすること ができます。パラメーターが名前付き定数の場合、この指定の前に定義されている必要はありません。

PERRCD キーワードが有効なのは、キーワード FROMFILE、TOFILE、または CTDATA が指定されている場合だけです。

POINTER{(*PROC)}

POINTER キーワードは、自由形式定義で使用されて、 項目のタイプが基底ポインター またはプロシージャー・ポインターであることを示します。

これは最初のキーワードでなければなりません。

パラメーターは任意指定です。 パラメーターを指定する場合は、項目がプロシージャー・ポインターであることを示す *PROC でなければなりません。

パラメーターが指定されていない場合、項目は基底ポインターです。

次に例を示します。

- フィールド *userspace* は、基底ポインター・フィールドとして定義されています。
- フィールド *callback* は、プロシージャー・ポインター・フィールドとして定義されています。

```
DCL-S userspace POINTER;
DCL-S callback POINTER(*PROC);
```

POS(開始位置)

POS キーワードは、自由形式サブフィールド定義で使用されて、 データ構造内のサブフィールドの開始位置を指定します。

開始位置パラメーターは、1 からデータ構造の長さまでの範囲内の値でなければなりません。

パラメーターにはリテラルまたは名前付き定数を指定できます。 名前付き定数である場合、定義ステートメントの前にその定数が定義されている必要があります。

次の例は、INDDS データ構造の定義です。 POS キーワードを使用することによって、データ構造内の特定の位置に標識が置かれています。

```
DCL-DS indds LEN(99);
   exit IND POS(3);
   refresh IND POS(5);
   cancel IND pos(12);
   sflclr IND pos(55);
   sfldsp IND pos(56);
END-DS;
```

次の例は、API で使用するためにデータ構造を定義 しています。 POS キーワードを使用することによって、 API 文書に指定されている位置にサブフィールドが置かれています。

```
DCL-DS Qwc JOBI0100_t QUALIFIED;
   Job_Type CHAR(1) POS(61);
   Job_Subtype CHAR(1) POS(62);
   Default_Wait INT(10) POS(73);
END-DS;
```

PREFIX(接頭部 {:置き換えられる文字数 })

PREFIX キーワードによって、定義中の外部記述データ構造のサブフィールド名の接頭部となる文字ストリングまたは文字リテラルを指定することができます。 さらに、既存の名前の中で置き換えられる文字 (それがあった場合) の数を指示する 数値をオプションで指定することができます。 パラメーター「nbr_of_char_replaced」を指定しない場合には、名前の先頭にストリングが付加されます。それぞれの名前の先頭から文字を削除するには、PREFIX(':削除する数) のように、空ストリングを最初のパラメーターとして指定してください。

「置き換えられる文字数」を指定する場合には、小数点以下の桁数のない 0 から 9 の数 値を表していなければなりません。 ゼロの値の指定は、「置き換えられる文字数」をまったく指定しないことと同じです。 たとえば、PREFIX(YE:3) の指定によって、フィールド名 'YTDTOTAL' が 'YETOTAL' に変更されます。

「置き換えられる文字数」パラメーターは、数値リテラル、数値を戻す組み込み関数、 または数値定数とすることができます。 それが名前付き定数の場合には、PREFIX キーワードが含まれる指定に先立っ て定数が定義されていなければなりません。 さらに、それが組み込み関数の場合は、その組み込み関数のすべての パラメーターが、PREFIX キーワードが含まれる指定に先立って定義されていなければなりません。

適用される規則は次のとおりです。

- EXTFLD キーワードを使用して明示的に名前が変更されたサブフィールドは このキーワードの影響を受けません。
- 接頭部を適用した後の名前の合計長が RPG フィールド名の最大長を超えては なりません。
- 接頭部が付けられる名前の中の文字数が「置き換えられる文字数」パラメーター によって表された値より小さいかまたは等しい場合には、その名前全体が接頭 部ストリングによって置き換えられます。
- 接頭部はピリオドで終わることはできません。
- 接頭部が文字リテラルの場合は英大文字でなければなりません。

PREFIX キーワードと ALIAS キーワードの相互の影響については、ALIAS キーワードを参照してください。

次の例は、外部記述データ構造 DS1 および DS2 で PREFIX(':2) を使用します。 ファイル FILE1 のフィールドは、すべて文字 X4 で始まり、ファイル FILE2 のフィールドは、すべて文字 WR で始まります。 2 つのファイルに、最初の 2 文字を除いて同じ名前のフィールドがある場合は、 外部記述データ構造に PREFIX(':2) を指定することによって、サブフィールドの名前は RPG プログラム内で同一になります。 これにより、そのサブフィールドは、EVAL-CORR 命令を使用して割り当てることができます。

```
Ffile1     if   e             disk
Ffile2     o    e             disk
D ds1           e ds                    extname(file1) prefix(':2)
D                                       qualified
D ds2           e ds                    extname(file2) prefix(':2)
D                                       qualified
 /free
     read file1 ds1;      // Read into data structure

     eval-corr ds2 = ds1; // Assign fields with same name

     write file2 ds2;     // Write from data structure

 /end-free
```

図 *156. PREFIX* を使用して、名前から文字を削除する

詳しくは、381 ページの『PREFIX(接頭部 {:置き換えられる文字数 })』を参照してください。

PROCPTR

PROCPTR キーワードは、項目をプロシージャー・ポインターとして定義します。 内部 データ・フィールド (40 桁目) には * を 入れなければなりません。

プロシージャー・ポインターを使用してプロシージャーを呼び出す方法については、433 ページの『EXTPROC{({*CL|*CWIDEN|*CNOWIDEN|*JAVA:クラス名:}名前|*DCLCASE)}』を参照してください。

注 : PROCPTR キーワードは、自由形式定義では使用されません。 代わりに、キーワード POINTER(*PROC) を指定して、項目をプロシージャー・ポインターとして定義します。

PSDS

PSDS キーワードは、自由形式のデータ構造定義で使用されて、 データ構造がプログラム状況データ構造であることを示します。

QUALIFIED

QUALIFIED キーワードは、データ構造のサブフィールドが、データ構造名の後にピリオドとサブフィールド名を続けて指定することによってアクセスされることを示します。 データ構造には名前が必要です。

サブフィールドは任意の有効な名前が可能で、プログラム内の他の場所で使われている名前でもかまいません。 これを次の例で説明します。

```
  * In this example, FILE1 and FILE2 are the names of files.  FILE1 and FILE2 are
  * also subfields of qualified data structure FILESTATUS.  This is valid,
  * because the subfields FILE1 and FILE2 must be qualified by the data structure
  * name: FILESTATUS.FILE1 and FILESTATUS.FILE2.
Ffile1     if   e             disk
Ffile2     if   e             disk

D fileStatus      ds                  qualified
D   file1                        N
D   file2                        N

C                   open(e)   file1
C                   eval      fileStatus.file1 = %error
```

REQPROTO(*NO)

REQPREXP 制御キーワードまたは REQPREXP コマンド・パラメーターによって、エクスポートされるプロシージャーのプロトタイプがない場合に警告またはエラーを発行することが示されていても、サイクル・メイン・プロシージャーのプロシージャー・インターフェースにキーワード REQPROTO(*NO) を指定することで、プロトタイプが必要ないことを示すことができます。

注 : サイクル・メイン・プロシージャーのプロシージャー・インターフェースは、自由形式のコードでは最初の DCL-PROC ステートメント、固定形式のコードでは Procedure-Begin ステートメントの前にソース・コードに表示されるプロシージャー・インターフェースです。 プロシージャー・インターフェースに名前があり、プロシージャー・インターフェースの前に同名のプロトタイプが指定されている場合、そのプロトタイプはサイクル・メイン・プロシージャーのプロトタイプであるとみなされます。

***NO**
REQPROTO(*NO) が指定されているときに、メイン・プロシージャーにプロトタイプがない場合には、警告もエラーも出されません。

REQPROTO キーワードは、サイクル・メイン・プロシージャー以外のプロシージャーのプロシージャー・インターフェースでは使用できません。 それ以外のプロシージャーでは、プロシージャーの最初のステートメントに REQPROTO キーワードが指定されています。 538 ページの『REQPROTO(*NO)』を参照してください。

例

- 以下の自由形式の例の場合:
  1. 制御キーワード REQPREXP(*REQUIRE) が指定されています。
  2. プロシージャー・インターフェースが名前なしで指定されています。 プロシージャー・インターフェースに名前がない場合、サイクル・メイン・プロシージャーでプロトタイプを持つことはできません。
  3. コンパイル・リストにエラー・メッセージが出されます。 そのエラー・メッセージは「外部プロシージャーおよび外部プログラムにはプロトタイプが指定されていません。REQPREXP(*REQUIRE) が指定されています。(No prototype is specified for an external procedure or program, and REQPREXP(*REQUIRE) is specified.)」というものです。

```
CTL-OPT REQPROTO(*REQUIRE); // 1

DCL-PI *N END-PI;           // 1
===> Error:                 // 1
```

- 以下の自由形式の例の場合:
  1. 制御キーワード REQPREXP(*REQUIRE) が指定されています。
  2. プロシージャー・インターフェースが名前なしで指定されていますが、キーワード REQPROTO(*NO) が指定されています。

キーワード REQPROTO(*NO) は プロトタイプが必要でないことを示すため、コンパイル・リストにエラー・メッセージは出されません。

```
CTL-OPT REQPROTO(*REQUIRE); // 1

DCL-PI *N REQPROTO(*NO) END-PI;  // 1
```

- 以下の固定形式の例の場合:
  1. 制御キーワード REQPREXP(*WARN) が指定されています。
  2. プロシージャー・インターフェースは名前付きで指定されます。 ただし、プロシージャー・インターフェースの前にその名前のプロトタイプは指定されません。
  3. コンパイル・リストに警告メッセージが表示されます。 その警告メッセージは、「警告: 外部プロシージャーまたは外部プログラムに対してプロトタイプが指定されていません。(warning: No prototype is specified for an external procedure or program.)」というものです。

```
H REQPREXP(*WARN)          1
D MYPGM           PI       2
===> Warning:              3

C                   RETURN
```

- 以下の固定形式の例の場合:
  1. 制御キーワード REQPREXP(*WARN) が指定されています。
  2. プロシージャー・インターフェースは名前付きで指定されます。 ただし、プロシージャー・インターフェースの前にその名前のプロトタイプは指定されません。

     プロトタイプが必要ないことを示すために、キーワード REQPROTO(*NO) が指定されます。

```
H REQPREXP(*WARN)          1
D MYPGM           PI       REQPROTO(*NO)  2

C                   RETURN
```

RTNPARM

RTNPARM キーワードは、プロシージャーの戻り値が、 参照によって渡される、 定義済みの戻り値と同じタイプのパラメーターとして内部的に処理されることを指定します。

RTNPARM を使用すると、大きな値を戻す場合にパフォーマンスが向上する可能性があります。

RTNPARM キーワードによるパフォーマンスへの影響は、小さなマイナスの影響から、大きなプラスの影響までさまざまです。 プロトタイプ化された戻り値が、整数や小さいデータ構造などのように比較的小さい場合には、小さなマイナスの影響が出る可能性があります。 プロトタイプ化された戻り値が、32767 バイトのデータ構造などのように、大きい値の場合にはパフォーマンスは多少向上します。 プロトタイプ化された戻り値が大きい可変長ストリングで、実際に戻された値が比較的小さい場合に、パフォーマンスの向上が最も顕著になります。 例えば、プロトタイプで戻り値を 1,000,000 バイトの可変長文字ストリングとして定義していて、値 「abc」が戻されるような場合です。

プロシージャー・プロトタイプに RTNPARM を使用すると、そのプロシージャーの呼び出しを含む他のプロシージャーに必要な自動記憶域の量が減る場合もあります。 例えば、プロシージャー MYCALLER に、大きな値を戻すプロシージャー MYPROC の呼び出しが含まれている場合、 プロシージャー MYCALLER は、(MYCALLER が実行時にプロシージャー MYPROC を実際に呼び出さなくても) 追加の自動記憶域を必要とします。 場合によっては、必要な自動記憶域が多すぎるために、プロシージャー MYCALLER がコンパイル

されなくなります。また、呼び出しスタックの合計自動記憶域が最大限度を超えてしまい、MYCALLER を呼び出せなくなることもあります。 RTNPARM を使用すると、追加の自動記憶域によるこの問題が回避されます。

注 :

1. 追加のパラメーターは、最初のパラメーターとして渡されます。
2. %PARMS 組み込み関数と %PARMNUM 組み込み関数のパラメーター・カウントには、追加のパラメーターが含まれます。 RTNPARM キーワードが指定されると、%PARMNUM によって戻される値は、見かけ上のパラメーター番号より 1 つ大きくなります。
3. パラメーター番号を必要とする API (CEEDOD または CEETSTA など) を呼び出すときには、 追加の最初のパラメーターを考慮してください。 例えば、プロシージャーに 3 つのパラメーターがあり、パラメーター・リストで 3 番目に出現するパラメーターの長さを取得したい場合には、 4 番目のパラメーターについて情報を要求します。 %PARMNUM 組み込み関数を使用してこれらの API を呼び出すための正しいパラメーター番号を戻す場合には、 正しいパラメーター番号を手動で決定する必要はありません。
4. 呼び出し元のプロシージャーが RPG 以外の言語で作成されている場合、 呼び出し元では、プロシージャーに戻り値がなく、RPG の戻り値と同じタイプの、参照によって渡される追加の最初のパラメーターがあるものとして、呼び出しをコーディングする必要があります。
5. 同様に、呼び出し先のプロシージャーが RPG 以外の言語で作成されている場合、 そのプロシージャーは戻り値なしで、RPG の戻り値と同じタイプの、参照によって渡される追加の最初のパラメーターがある状態としてコーディングする必要があります。
6. RTNPARM がプロシージャーに指定された場合、プロトタイプ・パラメーターの最大数は 398 です。
7. RTNPARM キーワードは、Java メソッド呼び出しには使用できません。

RTNPARM キーワードは、プロトタイプ定義とプロシージャー・インターフェース定義の両方に適用されます。

```
1. The prototype for the procedure
D center          pr        100000a   varying
D                                     rtnparm
D   text                     50000a   const varying
D   len                         10i 0 value


2. Calling the procedure
D title           s            100a   varying
 /free
    title = center ('Chapter 1' : 20);
    // title = '     Chapter 1     '

3.The procedure
P center          b                   export
D center          pi        100000a   varying
D                                     rtnparm
D   text                     50000a   const varying
D   len                         10i 0 value
D blanks          s          50000a   inz(*blanks)
D numBlanks       s             10i 0
D startBlanks     s             10i 0
D endBlanks       s             10i 0
 /free
    if len < %len(text);
       ... handle invalid input
    endif;
    numBlanks = len - %len(text);
    startBlanks = numBlanks / 2;
    endBlanks = numBlanks - startBlanks;
    return %subst(blanks : 1 : startBlanks)
         + text
         + %subst(blanks : 1 : endBlanks);
 /end-free
P center          e
```

図 *157. RTNPARM* キーワードを指定したプロシージャーの例

```
D proc            pi              a   len(16773100) varying
D                                     rtnparm opdesc
D   p1                            10a
D   p2                            10a   options(*varsize)
D   p3                            10a   options(*omit : *nopass)

D num_parms       s               10i 0
D parm_len        s               10i 0
D desc_type       s               10i 0
D data_type       s               10i 0
D desc_info1      s               10i 0
D desc_info2      s               10i 0
D CEEDOD          pr
D   parm_num                      10i 0 const
D   desc_type                     10i 0
D   data_type                     10i 0
D   desc_info1                    10i 0
D   desc_info2                    10i 0
D   parm_len                      10i 0
D   feedback                      12a   options(*omit)
 /free
    // Get information about parameter p2
    callp CEEDOD(%parmnum(p2) : desc_type : data_type
                : desc_info1 : desc_info2
                : parm_len : *omit);
    if parm_len < 10;
       // The parameter passed for p2 is shorter than 10
    endif;

    // Find out the number of parameters passed
    num_parms = %parms();
    // If all three parameters were passed, num_parms = 4

    // test if p3 was passed
    if num_parms >= %parmnum(p3);
       // Parameter p3 was passed
       if %addr(p3) <> *null;
          // Parameter p3 was not omitted
       endif;
    endif;
```

図 *158. RTNPARM* キーワードを指定した *%PARMS* の使用と *CEEDOD* の呼び出し

```
1. The RPG prototype

D myproc          pr           200a   rtnparm
D   name                        10a   const

2. A CL module calling this RPG procedure

dcl &retval type(*char) len(200)

callprc myproc parm(&retval 'Jack Smith')
```

図 *159.* 別の言語からの *RTNPARM* キーワードを指定したプロシージャーの呼び出し

```
1. CL procedure GETLIBTEXT

PGM PARM(&retText &lib)

DCL &retText type(*char) len(50)
DCL &lib     type(*char) len(10)

/* Set &retText to the library text */

rtvobjd obj(&lib) objtype(*lib) text(&retText)
return

2. RPG procedure calling this CL procedure using the RTNPARM keyword

D getLibText      pr            50a   rtnparm
D   name                        10a   const
 /free
      if getLibText('MYLIB') = *blanks;
          ...
```

図 *160.* 別の言語で作成された *RTNPARM* キーワードを指定したプロシージャーの呼び出し

SAMEPOS(サブフィールド)

SAMEPOS キーワードは定義内で使用して、サブフィールドの開始位置が、パラメーターに指定されたサブフィールドと同じであることを指定します。

このパラメーターは、同じレベルのデータ構造内の前に指定したサブフィールドでなければなりません。

SAMEPOS で定義された複数のサブフィールドを持つデータ構造

以下の例では、SAMEPOS キーワードを使用して複数のサブフィールドを定義します。

1. サブフィールド A は 1 から 10 桁目にあります。
2. サブフィールド B は 11 桁目にあります。
3. サブフィールド C はキーワード SAMEPOS(B) で定義されているので、 B と同じ 11 桁目で開始されます。 これは 30 桁目で終了します。
4. データ構造サブフィールド SUB_DS は、データ構造の次の桁である 31 桁目で開始します。
5. サブフィールド N は、データ構造サブフィールド SUB_DS の 1 から 5 桁目にあります。
6. サブフィールド X1 は、データ構造サブフィールド SUB_DS の 6 桁目にあります。
7. サブフィールド X2 は、データ構造サブフィールド SUB_DS の 7 桁目にあります。
8. サブフィールド X3 は、データ構造サブフィールド SUB_DS の 8 桁目にあります。
9. 配列のサブフィールド ARR_X はキーワード SAMEPOS(X1) で定義されているので、データ構造サブフィールド SUB_DS の 6 から 8 桁目にあります。 配列はサブフィールド X1、X2、および X3 をオーバーレイします。
10. サブフィールド ERROR_1 はキーワード SAMEPOS(a) で定義されます。 これは、サブフィールド A がデータ構造サブフィールド SUB_DS に含まれていないためエラーになります。
11. サブフィールド ERROR_2 はキーワード SAMEPOS(last) で定義されます。 これは、サブフィールド LAST がサブフィールド ERROR_2 の前に指定されていないためエラーになります。

```
DCL-DS ds1 QUALIFIED;
   a CHAR(10);                          //  1
   b CHAR(1);                           //  2
   c CHAR(20) SAMEPOS(b);               //  3
   DCL-DS sub_ds;                       //  4
      n  CHAR(5);                       //     5
      x1 CHAR(1);                       //     6
      x2 CHAR(1);                       //     7
      x3 CHAR(1);                       //     8
      arr_x CHAR(1) DIM(3) SAMEPOS(x1); //     9
      error_1 CHAR(1) SAMEPOS(a);       //     10
      error_2 CHAR(1) SAMEPOS(last);    //     11
      last CHAR(1);
   END-DS;
END-DS;
```

外部記述データ構造での複数の反復フィールドにわたる配列の定義

以下の例では、ファイル SALESFILE に 4 つのフィールド SALESQ1、SALESQ2、SALESQ3、および SALESQ4 が含まれ、 間にフィールドを入れずに、すべて同じように定義されています。

```
     A          R REC
     A            AGENT         50A
     A            SALESQ1        9P 2
     A            SALESQ2        9P 2
     A            SALESQ3        9P 2
     A            SALESQ4        9P 2
     A            SALESTOTAL    10P 2
```

SALES 配列サブフィールドは SALESQ1、SALESQ2、SALESQ3、および SALESQ4 の各サブフィールドにまたがって位置付けされます。

```
DCL-DS ds EXTNAME('SALESFILE');
   sales LIKE(salesq1) DIM(4) SAMEPOS(salesq1);
END-DS;
DCL-S i INT(10);
FOR i to %ELEM(sales);
   DSPLY sales(i);
ENDFOR;
```

複数の反復データ構造サブフィールドにわたる配列サブフィールドの定義

以下の例は、複数の同一データ構造のサブフィールドの後に、同じデータ構造の配列が続き、最初のサブフィールドと同じ開始位置に配列を位置づけるために SAMEPOS キーワードを使用して定義しています。

```
DCL-DS info QUALIFIED;
   other_subfield CHAR(11);
   DCL-DS info1;
      age INT(10) INZ(5);
      name CHAR(10) INZ('Mary');
   END-DS;
   DCL-DS info2;
      age INT(10) INZ(7);
      name CHAR(10) INZ('Tom');
   END-DS;
   DCL-DS info3;
      age INT(10) INZ(6);
      name CHAR(10) INZ('Sally');
   END-DS;
   DCL-DS info_arr DIM(3) SAMEPOS(info1);
      age INT(10);
      name CHAR(10);
   END-DS;
END-DS;
DCL-S i INT(10);

FOR i = 1 TO %ELEM(info_arr);
   DSPLY ('Name: ' + info.info_arr(i).name + ' '
        + 'Age: ' + %char(info.info_arr(i).age));
ENDFOR;
```

STATIC{(*ALLTHREAD)}

STATIC キーワードは次の目的で使用します。

- ローカル変数が静的記憶域に保管されることを指定するため。
- 同じ静的変数のコピーを、マルチスレッド環境内のすべてのスレッドで使用できるように指定するため。
- Java メソッドを静的メソッドとして定義するため。

サブプロシージャーのローカル変数の場合、STATIC キーワードは、データ項目が静的記憶域に記憶され、それにより、データ項目が定義されているプロシージャーに対する呼び出しの間はその値が保留されることを指定します。 このキーワードは、サブプロシージャーの中でのみ使用することができます。 グローバル・フィールドはすべて静的です。

データ項目は、それが含まれているプログラムまたはサービス・プログラムが初 めて活動化された時に初期化されます。 通常のサイクル処理の一部としてグローバル定義について再初期化が行なわれ た場合であっても、データ構造が再び再初期化されることはありません 。

STATIC が定義されない場合には、ローカルに定義されたすべてのデータ項目は自 動記憶域に記憶されます。 自動記憶域に記憶されたデータは、すべての呼び出しごとにその始めに初期化さ れます。 プロシージャーが反復して呼び出された場合には、各呼び出しごとに記憶域の固有 のコピーが得られます。

制御仕様書で *THREAD(*CONCURRENT)* が指定されているモジュール内の変数の場合、
STATIC(*ALLTHREAD) を使って、静的変数の同じインスタンスをすべてのスレッドで使用するように指定します。 並行スレッド・モジュール内の静的変数に *ALLTHREAD が指定されていない場合、その変数はスレッド・ローカル記憶域に置かれます。 つまり、各スレッドが、その変数のインスタンスを個別に持つことになります。

THREAD(*CONCURRENT) キーワードの有無にかかわらずソースにコピーできるコピー・ファイルがある場合は、STATIC(*ALLTHREAD) キーワードを組み込むかどうかを制御するために事前定義条件 *THREAD_CONCURRENT を調べることができます。 詳しくは、88 ページの『制御仕様書キーワードに関連する条件』 を参照してください。

STATIC(*ALLTHREAD) キーワードの使用時には、以下の規則が適用されます。

- 制御仕様書で THREAD(*CONCURRENT) が指定されていない限り、STATIC(*ALLTHREAD) を使用することはできません。
- グローバル変数の場合、STATIC キーワードは暗黙に指定されます。 *ALLTHREAD がパラメーターとして指定されていない限り、STATIC キーワードをグローバル変数に指定することはできません。

- STATIC(*ALLTHREAD) で定義されている変数を初期化して、STATIC(*ALLTHREAD) で定義されていない変数のアドレスにすることはできません。

注意 : すべてのスレッドで使用される静的変数がスレッド・セーフな方法で処理されるようにするのは、ユーザー自身の責任となります。 「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」の『マルチスレッド化に関する考慮事項』のセクションを参照してください。

ヒント : すべてのスレッドの静的変数について命名規則を設けて、メンテナンス・プログラマーおよびコード・レビューアーに対して、それらの変数には特別な処理が必要であることを気付かせるようにすることをお勧めします。 例えば、STATIC(*ALLTHREAD) を指定して定義されている変数名のすべてに接頭部 ATS_ を付ける、という方法があります。

***Java* メソッドの場合**、STATIC キーワードは、そのメソッドが静的であることを指定します。 STATIC が指定されていないと、そのメソッドはインスタンス・メソッドであると想定されます。 Java メソッドに「静的」属性が設定されている場合に限り、STATIC キーワードをプロトタイプにコーディングする必要があります。 プロトタイプに STATIC キーワードが指定されている場合には、*ALLTHREAD パラメーターを使用することはできません。

*STATIC(*ALLTHREAD)* についての追加の考慮事項

ヌル値可能フィールド: ヌル値可能フィールド STATIC(*ALLTHREAD) のヌル標識を保持するために使用される内部変数も、STATIC(*ALLTHREAD) として定義します。 あるスレッドによって変数のヌル標識の値が変更されると、その変更はすべてのスレッドに対して可視になります。 ヌル標識値へのアクセスは同期化されません。

テーブルと複数オカレンス・データ構造: STATIC(*ALLTHREAD) によって定義されているテーブルまたは複数オカレンス・データ構造の、現行のオカレンスを保持するために使用される内部変数は、スレッド・ローカル記憶域で定義されます。 各スレッドは、現行オカレンス変数のインスタンスを個別に持ちます。

TEMPLATE

TEMPLATE キーワードは、該当する定義を LIKE 定義または LIKEDS 定義でも使用するように指示するものです。 TEMPLATE キーワードは、データ構造定義および独立フィールド定義で有効です。

定義仕様書の *TEMPLATE* キーワードに関する規則*:*

1. ある定義に TEMPLATE キーワードが指定されている場合、テンプレート名およびそのテンプレート名のサブフィールドは、以下の形でのみ使用することができます。
   - LIKE キーワードのパラメーターとして
   - LIKEDS キーワードのパラメーターとして (テンプレートがデータ構造の場合)
   - %SIZE 組み込み関数のパラメーターとして
   - %ELEM 組み込み関数のパラメーターとして
   - 定義仕様書の %LEN 組み込み関数のパラメーターとして (名前付き定数、初期設定値など)
   - 定義仕様書の %DECPOS 組み込み関数のパラメーターとして (名前付き定数、初期設定値など)
2. INZ キーワードは、テンプレート・データ構造に使用することができます。 これによって、INZ(*LIKEDS) キーワードを介して、テンプレートの LIKEDS 定義で使用される初期設定値を設定できるようになります。

```
 * Define a template for the type of a NAME
D standardName    S            100A   VARYING TEMPLATE

 * Define a template for the type of an EMPLOYEE
D employee_type   DS                   QUALIFIED TEMPLATE INZ
D   name                               LIKE(standardName)
D                                      INZ('** UNKNOWN **')
D   idNum                        10I 0 INZ(0)
D   type                          1A   INZ('R')
D   years                         5I 0 INZ(-1)

 * Define a variable like the employee type, initialized
 * with the default value of the employee type
D employee        DS                   LIKEDS(employee_type)
D                                      INZ(*LIKEDS)

 * Define prototypes using the template definitions
 *
 * The "id" parameter is defined like a subfield of a
 * template data structure.
D getName         PR                   LIKE(standardName)
D   idNum
D findEmp         PR              N
D   emp                                LIKEDS(employee_type)
D   id                                 LIKE(employee_type.idNum)
D                                      CONST
```

図 *161. TEMPLATE* 定義の例

TIME{(形式 {区切り記号 }) }

TIME キーワードは、自由形式定義で使用されて、 項目のタイプが時刻であることを示します。

これは最初のキーワードでなければなりません。

パラメーターは任意指定です。 これは時刻形式と区切り記号を指定します。 時刻項目のデフォルト形式については、275 ページの『時刻データ・タイプ』を参照してください。

次の例では、フィールド *time_dft* は、モジュールのデフォルト形式の時刻フィールドとして定義されており、フィールド *time_hms* は、形式が *HMS、区切り記号がピリオドとして定義されています。

```
DCL-S time_dft TIME;
DCL-S time_hms TIME(*HMS.);
```

TIMESTAMP { (秒の小数部の桁数) }

TIMESTAMP キーワードは、自由形式定義で使用されて、 項目のタイプがタイム・スタンプであることを示します。

オプション・パラメーターは、秒の小数部の桁数を指定します。 パラメーターが指定されない場合、秒の小数部の桁数は 6 にデフォルト設定されます。

これは最初のキーワードでなければなりません。

次の例には、秒の小数部の桁数が異なるいくつかのタイム・スタンプが示されています。

```
DCL-S TS0 TIMESTAMP(0);    // YYYY-MM-DD-hh-mm-ss
DCL-S TS1 TIMESTAMP(1);    // YYYY-MM-DD-hh-mm-ss.f
DCL-S TS6A TIMESTAMP;      // YYYY-MM-DD-hh-mm-ss.ffffff
DCL-S TS6B TIMESTAMP(6);   // YYYY-MM-DD-hh-mm-ss.ffffff
DCL-S TS12 TIMESTAMP(12);  // YYYY-MM-DD-hh-mm-ss.ffffffffffff
```

TIMFMT(形式 {区切り記号 })

TIMFMT キーワードによって、時刻タイプの独立フィールド、データ構造サブフ ィールド、プロトタイプ・パラメーター、あるいはプロトタイプまたはプロシー ジャー・インターフェース定義上の戻り値のいずれの項目についても内部時刻形 式を指定し、また、オプションで時刻区切り記号を指定することができます。このキーワードは、時刻タイプの外部記述データ構造サブフィールドの場合には 自動的に生成されます。

TIMFMT が指定されていない場合、時刻フィールドの時刻形式と区切り記号 は、モジュールの現行デフォルト時刻形式で指定されたものになります。 モジュールのデフォルト時刻形式は、 制御ステートメントで TIMFMT キーワードを使用することによって初期設定されます。 それを一時的に別の値に設定することが、/SET 指示および /RESTORE 指示で TIMFMT キーワードを使用することによって可能です。 時刻形式のデフォルトは *ISO です。81 ページの『/SET』を参照してください。

注 : TIMFMT キーワードは、自由形式定義では使用されません。 代わりに、時刻形式は TIME キーワードのパラメーターとして指定されます。

有効な形式および区切り記号については、276 ページの表 74 を参照してください。内部形式について詳しくは、247 ページの『内部形式および外部形式』を参照してください。

TOFILE(ファイル名)

TOFILE キーワードによって、実行時前またはコンパイル時配列またはテーブルを 書き出すターゲット・ファイルを指定することができます。

配列またはテーブルを書き出す場合には、出力ファイルまたは入出力共用ファイ ルをキーワード・パラメーターとして指定してください。 このファイルは、ファイル仕様書にも定義されていなければなりません。配列またはテーブルは 1 つの出力装置にだけ書き出すことができます。

配列またはテーブルが出力ファイルに割り当てられた場合には、プログラムの終 了時に LR 標識がオンの場合に自動的に書き出されます。 配列またはテーブルは、他のすべてのレコードがファイルに書き出された後に書 き出されます。

配列またはテーブルを読み取ったファイルと同じファイルに書き出す場合には、 FROMFILE パラメーターとして指定されたのと同じファイル名を TOFILE パラメー ターとして指定しなければなりません。 このファイルは入出力共用ファイル (ファイル仕様書 の 17 桁目に C) として定義されていなければなりません。

UCS2(length)

UCS2 キーワードは、自由形式定義で使用されて、 項目が固定長 UCS-2 であることを示します。

これは最初のキーワードでなければなりません。

パラメーターは、長さを 2 バイト文字単位で指定します。 1 から 8,386,552 までの値を指定できます。

パラメーターにはリテラルまたは名前付き定数を指定できます。 名前付き定数である場合、定義ステートメントの前にその定数が定義されている必要があります。

次に例を示します。

- フィールド *cust_name* は、100 文字 の固定長 UCS-2 フィールドとして定義されています。
- フィールド *message* は、5000 文字 の固定長 UCS-2 フィールドとして定義されています。 長さは、名前付き定数 *MSG_LEN* を使用して定義されています。

```
DCL-S cust_name UCS2(100);
DCL-C MSG_LEN 5000;
DCL-S message UCS2(MSG_LEN);
```

可変長 UCS-2 項目の定義については、487 ページの『VARUCS2(length {:2 | 4})』を参照してください。

UNS(桁数)

UNS キーワードは、数字データ・タイプのキーワードの 1 つです。 これは自由形式定義で使用されて、項目の形式が符号なし整数形式であることを示します。

これは最初のキーワードでなければなりません。

パラメーターは長さを桁数で指定します。 これは、3、5、10、または 20 (それぞれ、1 バイト、2 バイト、4 バイト、8 バイトを占めます) のいずれかでなければなりません。

パラメーターにはリテラルまたは名前付き定数を指定できます。 名前付き定数である場合、定義ステートメントの前にその定数が定義されている必要があります。

次に例を示します。

- フィールド *num_elems* は、10 桁の符号なし整数フィールドとして定義されています。 これは記憶域で 4 バイトを占めます。

```
DCL-S num_elems UNS(10);
```

VALUE

VALUE キーワードは、パラメーターが参照によってではなく、値によって渡され ることを指示します。 パラメーターを値によって渡すことができるのは、それらが関連づけられたプロ シージャーがプロシージャー呼び出しを使用して呼び出された時です。

そのプロトタイプが EXTPGM キーワードを使用して定義されていた場合には、VALUE キーワードをパラメーターに対して指定することはできません。 プログラムに対する呼び出しには、パラメーターが参照によって渡されることが必 要です。

呼び出されるプロシージャーに値パラメーターとして渡すことができるものに関 する規則は、EVAL 命令を使用して割り当てることができるものに関する規則と同 じです。 プロシージャーによって受け取られたパラメーターは式の左辺と対応し、渡され たパラメーターは式の右辺と対応しています。 詳しくは、771 ページの『EVAL (式の評価)』を参照してください。

VARCHAR(長さ {:2 | 4})

VARCHAR キーワードは、自由形式定義で使用されて、 項目が可変長文字であることを示します。

これは最初のキーワードでなければなりません。

パラメーターは長さをバイト単位で指定します。 1 から 16,773,102 までの値を指定できます。

各パラメーターにはリテラルまたは名前付き定数を指定できます。 名前付き定数である場合、定義ステートメントの前にその定数が定義されている必要があります。

2 番目のパラメーターは任意指定です。 これは、可変長項目の現在の長さを保管するために 使用されるバイト数を指定します。254 ページの『可変長項目の長さ接頭部のサイズ』を参照してください。

各パラメーターにはリテラルまたは名前付き定数を指定できます。 名前付き定数である場合、定義ステートメントの前にその定数が定義されている必要があります。

次に例を示します。

- フィールド *cust_name* は、最大長が 50 文字の 可変長文字フィールドとして定義されています。
- フィールド *message* は、最大長が 500 文字で接頭部サイズが 4 バイト の可変長文字フィールドとして定義されています。

```
DCL-S cust_name VARCHAR(50);
DCL-S message VARCHAR(500 : 4);
```

固定長文字項目の定義については、419 ページの『CHAR(長さ)』 を参照してください。

VARGRAPH(長さ {:2 | 4})

VARGRAPH キーワードは、自由形式定義で使用されて、 項目が可変長グラフィックであることを示します。

これは最初のキーワードでなければなりません。

パラメーターは長さをバイト単位で指定します。 1 から 8,386,550 までの値を指定できます。

パラメーターにはリテラルまたは名前付き定数を指定できます。 名前付き定数である場合、定義ステートメントの前にその定数が定義されている必要があります。

2 番目のパラメーターは任意指定です。 これは、可変長項目の現在の長さを保管するために 使用されるバイト数を指定します。254 ページの『可変長項目の長さ接頭部のサイズ』を参照してください。

各パラメーターにはリテラルまたは名前付き定数を指定できます。 名前付き定数である場合、定義ステートメントの前にその定数が定義されている必要があります。

次に例を示します。

- フィールド *cust_name* は、最大長が 50 文字の 可変長グラフィック・フィールドとして定義されています。
- フィールド *message* は、最大長が 500 文字で接頭部サイズが 4 バイト の可変長グラフィック・フィールドとして定義されています。

```
DCL-S cust_name VARGRAPH(50);
DCL-S message VARGRAPH(500 : 4);
```

固定長グラフィック項目の定義については、440 ページの『GRAPH(長さ)』 を参照してください。

VARUCS2(length {:2 | 4})

VARUCS2 キーワードは、自由形式定義で使用されて、 項目が可変長 UCS-2 であることを示します。

これは最初のキーワードでなければなりません。

最初のパラメーターは必須です。 これは長さをバイト単位で指定します。 1 から 8,386,550 までの値を指定できます。

2 番目のパラメーターは任意指定です。 これは、可変長項目の現在の長さを保管するために 使用されるバイト数を指定します。254 ページの『可変長項目の長さ接頭部のサイズ』を参照してください。

各パラメーターにはリテラルまたは名前付き定数を指定できます。 名前付き定数である場合、定義ステートメントの前にその定数が定義されている必要があります。

次に例を示します。

- フィールド *cust_name* は、最大長が 50 文字 の可変長 UCS-2 フィールドとして定義されています。
- フィールド *message* は、最大長が 500 文字で接頭部サイズが 4 バイト の可変長 UCS-2 フィールドとして定義されています。

```
DCL-S cust_name VARUCS2(50);
DCL-S message VARUCS2(500 : 4);
```

固定長 UCS-2 項目の定義については、485 ページの『UCS2(length)』 を参照してください。

VARYING{(2 | 4)}

VARYING キーワードは、定義仕様書に定義されている文字フィールド、図形フィールド、または UCS-2 フィールドが可変長形式であることを示します。 文字フィールド、図形フィールド、または UCS-2 フィールドにこのキーワードが指定されて いない場合、これらのフィールドは固定長として定義されます。

VARYING キーワードのパラメーターは、可変長項目の現行の長さを保存するために使用されるバイト数を表します。254 ページの『可変長項目の長さ接頭部のサイズ』を参照してください。

注 : VARYING キーワードは、自由形式定義では使用されません。 代わりに、VARCHAR、 VARGRAPH、 または VARUCS2 キーワード を使用して、項目のデータ・タイプを指定します。

詳しくは、「253 ページの『可変長の文字形式、図形形式および UCS-2 形式』」を参照してください。

ZONED(桁数 {: 小数点以下の桁数})

ZONED キーワードは、数字データ・タイプのキーワードの 1 つです。 これは自由形式定義で使用されて、項目の形式がゾーン 10 進数形式であることを示します。

これは最初のキーワードでなければなりません。

最初のパラメーターは必須です。 これは総桁数を指定します。 1 から 63 までの値を指定できます。

2 番目のパラメーターは任意指定です。 これは小数点以下の桁数を指定します。 ゼロから桁数までの範囲の 値を指定できます。 デフォルトはゼロです。

各パラメーターにはリテラルまたは名前付き定数を指定できます。 名前付き定数である場合、定義ステートメントの前にその定数が定義されている必要があります。

以下に例を示します。

- フィールド *salary* は、桁数が 5 で小数点以下の桁数が 2 のゾーン 10 進数フィールド として定義されています。
- フィールド *age* は、桁数が 3 で小数点以下の桁数がデフォルトの 0 であるゾーン 10 進数フィールド として定義されています。
- フィールド *price* は、桁数が 7 で小数点以下の桁数が 3 のゾーン 10 進数フィールド として定義されています。 小数点以下の桁数は、名前付き定数 *NUM_DEC_POS* を使用して定義されています。

```
DCL-S salary ZONED(5 : 2);
DCL-S age ZONED(3);
DCL-C NUM_DEC_POS 3;
DCL-S price ZONED(7 : NUM_DEC_POS);
```

定義仕様書タイプごとの要約

489 ページの表 99 は、各定義仕様書タイプごとに、必要な記入項目および 使用可能な記入項目をリストしたものです。

489 ページの表 100 および 491 ページの表 101 は、各定義仕様書タイプごとに使 用可能なキーワードをリストしたものです。

これらの表のおのおので、**R** はそれらの桁の指定が必須であることを 示し、**A** はそれらの桁の指定が可能であることを示しています。

表 *99.* 各定義仕様書タイプごとに必要または使用可能な項目

タイプ	7 桁目から 21 桁目 名前	22 桁目 外部	23 桁目 DS タイプ	24 桁目から 25 桁目 定義 タイプ	26 桁目から 32 桁目 開始位置	33 桁目から 39 桁目 終了位置/長さ	40 桁目 データ・タイプ	41 桁目から 42 桁目 小数点以下の桁数	44 桁目から 80 桁目 キーワード
データ構造	A	A	A	R		A			A
データ構造サブフィールド	A				A	A	A	A	A
外部サブフィールド	A	R							A
独立フィールド	R			R		A	A	A	A
名前付き固定情報	R			R					R
Prototype	R			R		A	A	A	A
Prototype パラメーター	A					A	A	A	A
プロシージャーインターフェース	A			R		A	A	A	A
プロシージャーインターフェースパラメーター	R					A	A	A	A

表 *100.* データ構造、独立フィールド、および名前付き定数のキーワード

キーワード	データ構造	データ構造サブフィールド	外部サブフィールド	独立フィールド	名前付き固定情報
ALIGN	A				
ALT		A	A	A	
ALTSEQ	A	A	A	A	
ASCEND		A	A	A	
BASED	A			A	
BINDEC[7]		A		A	
CHAR[7]		A		A	

表 *100.* データ構造、独立フィールド、および名前付き定数のキーワード *(*続き*)*

キーワード	データ構造	データ構造サブフィールド	外部サブフィールド	独立フィールド	名前付き固定情報
CCSID	A[8]	A		A	
CLASS[6]				A	
CONST[1]					R
CTDATA[2]		A	A	A	
DATE[7]		A		A	
DATFMT[6]		A		A	
DESCEND		A	A	A	
DIM	A	A	A	A	
DTAARA[2]	A	A		A	
EXPORT[2]	A			A	
EXT[5]	A				
EXTFLD			A		
EXTFMT		A	A	A	
EXTNAME[4]	A				
FLOAT[7]		A		A	
FROMFILE[2]		A	A	A	
GRAPH[7]		A		A	
IMPORT[2]	A			A	
IND[7]		A		A	
INT[7]		A		A	
INZ	A	A	A	A	
LEN	A	A		A	
LIKE		A		A	
LIKEDS[5]	A	A			
LIKEREC	A	A			
NOOPT	A			A	
NULLIND	A[4]	A		A	
OBJECT[7]		A		A	
OCCURS	A				
OVERLAY		A			
PACKED[7]		A		A	
PACKEVEN[6]		A			
PERRCD		A	A	A	
POINTER[7]		A		A	

表 *100.* データ構造、独立フィールド、および名前付き定数のキーワード *(*続き*)*

キーワード	データ構造	データ構造サブフィールド	外部サブフィールド	独立フィールド	名前付き固定情報
POS[5]		A			
PREFIX[4]	A				
PROCPTR[6]		A		A	
PSDS	A				
QUALIFIED	A				
STATIC[3]	A			A	
TEMPLATE	A			A	
TIME[7]		A		A	
TIMESTAMP[7]		A		A	
TIMFMT[6]		A		A	
TOFILE[2]		A	A	A	
UCS2[7]		A		A	
UNS[7]		A		A	
VARCHAR[7]		A		A	
VARGRAPH[7]		A		A	
VARUCS2[7]		A		A	
VARYING[6]		A		A	
ZONED[7]		A		A	

注 :

1. 名前付き定数を定義する場合には、キーワードは任意指定ですが、 キーワードに対するパラメーターは必須です。 たとえば、名前付き定数に値 '10' を割り当てるためには、CONST('10') か '10' のいずれかを指定すること ができます。
2. このキーワードはグローバル定義にのみ適用されます。
3. このキーワードはローカル定義にのみ適用されます。
4. このキーワードは、外部記述データ構造にのみ適用されます。
5. このキーワードは、プログラム記述データ構造にのみ適用されます。
6. このキーワードは固定形式定義にのみ適用されます。
7. このキーワードは自由形式定義にのみ適用されます。
8. このキーワードは、外部記述データ構造と EXTNAME または LIKEREC キーワードで定義されたデータ構造にのみ適用されます。 さらに、*NULL を EXTNAME または LIKEREC キーワードの抜き出しタイプとして指定することはできません。

表 *101.* プロトタイプ、プロシージャー・インターフェース、およびパラメーターのキーワード

キーワード	プロトタイプ (PR)	プロシージャー・インターフェース (PI)	PR または PI パラメーター
ALTSEQ	A	A	A
ASCEND			A

表 *101.* プロトタイプ、プロシージャー・インターフェース、およびパラメーターのキーワード *(*続き*)*

キーワード	プロトタイプ (PR)	プロシージャー・インターフェース (PI)	PR または PI パラメーター
BINDEC[1]	A	A	A
CCSID	A	A	A
CHAR[1]	A	A	A
CLASS[2]	A	A	A
CONST			A
DATE[1]	A	A	A
DATFMT[2]	A	A	A
DESCEND			A
DIM	A	A	A
EXTPGM	A	A	
EXTPROC	A	A	
FLOAT[1]	A	A	A
GRAPH[1]	A	A	A
IND[1]	A	A	A
INT[1]	A	A	A
LEN	A	A	A
LIKE	A	A	A
LIKEFILE			A
LIKEDS	A	A	A
LIKEREC	A	A	A
NOOPT			A
NULLIND			A
OBJECT[1]	A	A	A
OPDESC	A	A	
OPTIONS			A
PACKED[1]	A	A	A
POINTER[1]	A	A	A
PROCPTR[2]	A	A	A
RTNPARM	A	A	
STATIC	A	A	
TIME[1]	A	A	A
TIMESTAMP[1]	A	A	A
TIMFMT[2]	A	A	A
UCS2[1]	A	A	A

表 101. プロトタイプ、プロシージャー・インターフェース、およびパラメーターのキーワード (続き)			
キーワード	**プロトタイプ (PR)**	**プロシージャー・インターフェース (PI)**	**PR または PI パラメーター**
UNS[1]	A	A	A
VALUE			A
VARCHAR[1]	A	A	A
UNS[1]	A	A	A
VARCHAR[1]	A	A	A
VARGRAPH[1]	A	A	A
VARUCS2[1]	A	A	A
VARYING[2]	A	A	A
ZONED[1]	A	A	A
注: 1. このキーワードは自由形式定義にのみ適用されます。 2. このキーワードは固定形式定義にのみ適用されます。			

入力仕様

プログラム記述入力ファイルの場合には、入力仕様によって、ファイル内のレコードのタイプ、レコードのタイプの順序、レコード内 のフィールド、フィールド内のデータ、フィールドの内容に基づいた標識、制御フィールド、レコードの突き合わせに使用されるフィールド、および順序検査に使用 されるフィールドが記述されます。 外部記述ファイルの場合には、入力仕様は任意指定であり、RPG IV の機能を外部記述に追加するために使用することができます。

入力仕様は、プログラムのすべてのファイルに使用されるわけではありません。 ファイルによっては、入力操作の結果フィールドでデータ構造をコーディングする必要があります。 プログラムの以下のファイルでは、入力仕様を使用しません。

- サブプロシージャーで定義されるファイル
- QUALIFIED キーワードを指定して定義されるファイル
- TEMPLATE キーワードを指定して定義されるファイル
- LIKEFILE キーワードを指定して定義されるファイル

入力仕様の詳細については、以下に示されています。

- プログラム記述ファイルの記入項目
- 外部記述ファイルの記入項目

入力仕様ステートメント

入力仕様の一般的なレイアウトは次のとおりです。

- 入力仕様コード (I) は 6 桁目に入れられます。
- 仕様書の注記以外の部分は 7 から 80 桁目です。
- 仕様書の注記部分は 81 から 100 桁目です。

プログラム記述

プログラム記述ファイルの場合には、入力仕様の記入項目は次のカテゴリー に分けられます。

- ファイル中の入力レコードと他のレコードとの関係を記述するレコード識別 項目 (7 から 46 桁目)。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+...
9 ...+... 10
IFilename++SqNORiPos1+NCCPos2+NCCPos3+NCC.............................Comments++++++++
++++
I.........And..RiPos1+NCCPos2+NCCPos3+NCC.............................Comments++++++++
++++
```

図 *162.* プログラム記述レコードのレイアウト

- レコード内のフィールドを記述するフィールド記述項目 (31 から 74 桁目)。 各フィールドは、その対応するレコード識別項目の下の別個の行に記述されます。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+...
9 ...+... 10
I.......................Fmt+SPFrom+To+++DcField+++++++++L1M1FrPlMnZr......Comments++++++++
++++
```

図 *163.* プログラム記述フィールドのレイアウト

外部記述

外部記述ファイルの場合には、入力仕様の記入項目は次のカテゴリーに分け られます。

- RPG IV の機能を追加する先のレコード (外部記述レコード様式) を識別するレコード識別項目 (7 から 16 桁目および 21 から 22 桁目)。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+...
9 ...+... 10
IRcdname+++....Ri.....................................................Comments++++++++
++++
```

図 *164.* 外部記述レコードのレイアウト

- レコード内のフィールドに追加される RPG IV の機能を記述するフィールド記述項目 (21 から 30 桁目、49 から 66 桁目、および 69 から 74 桁目)。 フィールド記 述項目は、対応するレコード識別項目の次の行に記入されます。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+...
9 ...+... 10
I..............Ext-field+.................Field+++++++++L1M1..PlMnZr......Comments++++++++
++++
```

図 *165.* 外部記述フィールドのレイアウト

プログラム記述ファイル

6 桁目 (仕様書コード)

この行が入力仕様ステートメントであることを識別するために、 6 桁目には I がなければなりません。

レコード識別項目

プログラム記述ファイルのレコード識別項目 (7 から 46 桁目) では、ファイル中 の入力レコードと他のレコードとの関係を記述します。

7 から 16 桁目 (ファイル名)

記入
説明

有効なファイル名
入力ファイルのファイル仕様書に表されたファイル名と同じもの。

これらの桁には、記述するファイルの名前を記入します。 この名前は、ファイル仕様書でこのファイルについて定義したのと同じ名前でなければなりませ ん。 このファイル、入力ファイル、更新ファイル、または入出力共用ファイルの いずれかでなければなりません。 ファイル名は各ファイルごとに最初のレコード 識別行に入れなければなりませんが、そのファイルに関する後続のレコード識別 行にも入れることができます。 1 つの入力ファイルを記述するすべての記入項目 はまとめて表す必要があり、他のファイルの記入項目と混用することはできま せん。

16 から 18 桁目 (論理関係)

記入
説明

AND
4 つ以上の識別コードが使用されます。

または
2 つ以上のレコード・タイプに共通のフィールドがあります。

使用することができる AND/OR 行の数に制限はあり ません。 詳しくは、499 ページの『AND 関係』および 499 ページの『OR 関係』を参照してくださ

17 から 18 桁目 (順序)

記入
説明

任意の 2 文字の英字
プログラムでは特別な順序について検査されません。

任意の 2 桁の数値
グループ内の特別な順序についてプログラムで検査されます。

番号 (19 桁目) とオプション (20 桁目) を組み合わせた数値による順序の指 定によって、プログラムは、ファイル内の入力レコードの順序を検査します。 順序が正しくない場合には、RPG IV 例外/エラー処理ルーチンに制御が渡されます。 AND または OR 行を指定した場合には、その AND または OR 行ではなく、グループ の主要レコード行で順序の指定を行います。

英字および数値は同じファイル中の異なるレコード (異なるレコード識別行) について指定できますが、英字項目のレコードは数値項目のレコードより前に指 定しなければなりません。

英字項目

順序検査を行わない場合には、これらの桁に英字の任意の 2 文字を入れてく ださい。 実際の一般的なプログラミングでは、プログラムの文書化に役立つよう にこれらのコードが順に指定されます。 ただし、固有の英字項目を使用する必要はありません。

数値項目

ファイル中のあるレコード・タイプを別のレコードより前に読み取る必要があ った場合には、17 から 18 桁目に固有の数値コードを入れてください。 数値項目は 01 から始まる昇順になっていなければなりませんが、連続している必要はありま せん。 数値項目を使用した場合には、19 から 20 桁目にも適切な指定を行わなけれ ばなりません。

順序検査を指定するためには、各レコード・タイプにレコード識別コードが含 まれ、レコード・タイプにはそれらが現れるべき順に番号が付けられていなけ ればなりません。 レコードが読み取られた時に、この順序が検査されます。 レコード・タイプの順序が違っていた場合には、RPG IV 例外/エラー処理ルー チンに制御が渡されます。

順序番号では、各レコード・タイプのすべてのレコードが、レコード・タイプ の順序番号がより高いレコードに先行するかどうかが確認されるだけです。 順序番号によって、レコード・タイプ内のレコードが一定の順序になっているか どうかの確認は行われません。 順序番号は制御レベルとは関係がなく、レコードのフィールド内のデータが特別 な順序であるかどうかの検査は行われません。 レコードのフィールド内のデー タが特別な順序であるかどうかを検査する必要があることを指示するためには、65 から 66 桁目 (突き合わせフィールド) を使用してください。

19 桁目 (番号)

記入
説明

ブランク
プログラムは (17 から 18 桁目に英字の指定がある) 特別な順序についてはレコ ード・タイプを検査しません。

1
順次グループにはこのタイプのレコードを 1 つだけ存在させることができま す。

N
順次グループにはこのタイプのレコードを 1 つまたは複数存在させることがで きます。

この記入項目は、17 から 18 桁目に数値を指定した場合に使用しなければなりま せん。 17 から 18 桁目に英字を指定した場合には、この記入項目はブランクでなければなり ません。

20 桁目 (オプション)

記入
説明

ブランク
順序検査を指定した場合にはこのレコード・タイプがなければなりません。

O
順序検査を指定した場合でもレコード・タイプは任意指定です (すなわち、あ ってもなくてもかまいません)。

17 から 18 桁目に英字項目が含まれている場合には、この項目はブランクでなけ ればなりません。

ファイル内のすべてのレコード・タイプを任意として指定した (17 から 18 桁目 に英字の指定または 20 桁目に O の指定がある) 場合には、レコード・タイプの 順序検査は意味がありません。

21 から 22 桁目 (レコード識別標識、または **)

記入
説明

ブランク
標識は使用されません。

01 から 99
一般標識。

L1 から L9 または LR
レコード識別標識に使用される制御レベル標識。

H1-H9
停止標識。

U1-U8
外部標識。

RT
戻り標識。

******
先読みレコード (標識ではありません)。 先読みは、1 次または 2 次ファイルでのみ使用することができます。

これらの桁に指定された標識は、レコード識別コード (23 から 46 桁目) と一緒に 使用されます。

標識

21 から 22 桁目は、標識をこの行で定義されたレコード・タイプと関連付けるも のです。 通常の指定は標識 01 から 99 の 1 つですが、制御レベル標識 L1 から L9 および LR を 使用して、特定の合計ステップを処理することができます。 制御レベル標識を指定しても、より低い制御レベル標識はオンに設定されません。 停止標識 H1 から H9 を使用して、処理を停止することができます。 戻り標識 (RT) は、呼び出し側プログラムに戻るために使用されます。

レコードが処理のために選択され、レコード識別コードによって示された条件を 満たしている場合には、該当するレコード識別標識がオンに設定されます。 この標識は、演算および出力命令を条件付けするために使用することができます。 レコード識別標識は、プログラマーがオンまたはオフに設定することができます。 しかし、サイクルの終わりでは、別のレコードが選択される前にすべてのレコード 識別標識がオフに設定されます。

先読みフィールド

** の記入項目は、先読み機能のために使用されます。 この機能によって、ファイ ル内の次のレコードの中の情報を見ることができます。 現在処理するために選択 されているファイルだけでなく、存在はしていてもこのサイクルでは選択されて いない他のファイルも見ることができます。

フィールド記述行には、レコード内のフィールドの開始位置と終了位置、フィー ルド名、およびフィールドが数値フィールドの場合には小数点以下の桁数が含ま れていなければなりません。 先読みフィールドは、入力仕様にフィールド名として、定義仕様書にデータ構 造名として、あるいは演算仕様書に結果のフィールドとして指定することはでき ないことに注意してください。

17 から 18 桁目には英字項目が含まれていなければなりません。 先読みフィールドは、21 から 22 桁目に ** が入っている行の次の行の 49 から 62 桁目に定義されます。 63 から 80 桁目はブランクでなければなりません。

レコード内のフィールドは、どれでもまたはすべて先読みフィールドとして定義 することができます。 この定義は、ファイル内のすべてのレコードにそのタイプ と無関係に適用されます。 フィールドを先読みフィールドと通常の入力フィール ドの両方として使用する場合には、異なる名前で 2 回定義しなければなりません。

先読み機能は、1 次および 2 次ファイルについてのみ、1 つ のファイルに一度だけ指定することができます。 これを 全手順ファイルに対して使用することはできず、AND 行および OR 行で使用することもできません。

入出力共用ファイルまたは更新ファイルのレコードが処理されている場合には、 先読みフィールドのデータは次のレコードのデータではなく、処理中のレコード のデータと同じになります。

先読み機能によって、ファイル情報データ構造内の情報は、現行 1 次 レコードと関連したデータではなく、先読みレコードと関連したデータにより更 新されます。

配列要素を先読みフィールドとして指定すると、配列全体が先読みフィールドと して分類されます。

ファイルの終わりが認識できるように、先読みフィールドには、ファイル内のすべ てのレコードが処理された時に特殊値が埋め込まれます。 文字フィールドの場合には、この値はすべてが '9' となり、その他すべてのデー タ・タイプの場合には、この値は *HIVAL と同じになります。

23 から 46 桁目 (レコード識別コード)

23 から 46 桁目の記入項目は、入力ファイルの各レコード・タイプを識別します。 各 仕様行に入れることができるのは 1 から 3 個のレコード識別コードです。 4 つ以上 のレコード識別コードは、追加の行に AND/OR 関係を使用して指定することがで きます。 ファイルに 1 つのレコード・タイプしか入っていない

場合には、識別コードをブ ランクのままにしておくことができますが、レコード識別項目 (21 から 22 桁目) お よび順序 (17 から 18 桁目) は指定しなければなりません。

注 : レコード識別コードは図形データ・タイプまたは UCS-2 データ・タイプ の処理には適用されません。 レコード識別が行われるのは単一バイトの桁に 対してだけです。

23 から 46 桁目には、23 から 30 桁目、31 から 38 桁目、および 39 から 46 桁目の 3 組の指定 をすることができます。 各組は、桁、否定、コード部分、および文字の 4 つのグループに分けられます。

以下の表は、各組のどのカテゴリーでどの桁が使用されるかを示しています。

カテゴリー	**23 から 30**	**31 から 38**	**39 から 46**
位置指定	23 から 27	31 から 35	39 から 43
Not	28	36	44
コード部分	29	37	45
文字	30	38	46

これらの組の指定は一定の順序である必要はありません。 例えば、23 から 30 桁目に指定する必要がなくとも、31 から 38 桁目に指定することができます。 ファイル内の入力レコードが同じタイプであった場合には、レコード識別コード 記入項目は必要ありません。 レコード識別コードが含まれていない入力仕様では 、ファイルの最終レコード・タイプが定義されるので、未定義のレコード・タイ プも処理できるようになります。 レコード識別コードが満たされない場合には、RPG IV 例外/エラー処理ルーチンに制御が渡されます。

23 から *27*、*31* から *35*、および *39* から *43* 桁目 (桁)

記入
説明

ブランク
レコード識別コードはありません。

1 から 32766
レコードの中でレコード識別コードが入っている位置。

これらの桁には、各レコードの中でレコード識別コードが入っている位置を記 入してください。 コードが入っている位置は、ファイルについて指定されたレコ ード長の範囲内になければなりません。 この指定は右寄せしなければなりませんが、先行ゼロは省略することができます。

28、*36*、および *44* 桁目 (否定)

記入
説明

ブランク
レコード識別コードがなければなりません。

N
レコード識別コードがあってはなりません。

指定したレコードの位置に記述したコードがあってはならない場合には、この 桁に N を記入してください。

29、*37*、および *45* 桁目 (コード部分)

記入
説明

C
文字全体。

Z
文字のゾーン部分。

D
文字の数字部分。

この記入項目では、レコード識別コードの中でテストされる文字の部分を指定 します。

文字 *(C)*

C の指定は、文字の完全な構造 (ゾーンおよび数字) がテストされることを指 示します。

ゾーン *(Z)*

Z の指定は、文字のゾーン部分がテストされることを指示します。 ゾーンの指 定によって、「文字」記入項目の上位 4 ビットが「位置」記入項目に指定された レコード位置の文字のゾーン部分と比較されます。 しかし、以下の 3 つの特殊な場合は例外です。

- & (アンパーサンド) の 16 進数表記は 50 です。 しかし、アンパーサンドを「文字」記入項目にコーディングした場合には、その 16 進数表記が C0、つまり、A から I と同じゾーンを持っているかのように扱われます。 入力データ中のアンパーサンドは、16 進数の 5 ゾーンおよ び 16 進数の C ゾーンに対する 2 つのゾーン検査を満たします。
- - (マイナス符号) の 16 進数表記は 60 です。 しかし、マイナス符号 を「文字」記入項目にコーディングした場合には、その 16 進数表記 が D0、つまり、J から R と同じゾーンを持っているかのように扱われます。 入力データ中のマイナス符号は、16 進数 の 6 ゾーンおよび 16 進数の D ゾーンに対する 2 つのゾーン検査を満たします。
- ブランクの 16 進数表記は 40 です。 しかし、ブランクを「文字」記入項目にコーディングした場合には、その 16 進数表記が F0、つまり、0 から 9 と 同じゾーンを持っているかのように扱われます。 入力データ中のブランク は、16 進数の 4 ゾーンおよび 16 進数の F ゾーンに対する 2 つのゾーン検査を満たします。

数字 *(D)*

D の指定は、文字の数字部分がテストされることを指示します。 その文字の下 位 4 ビットが「位置」記入項目に指定された文字と比較されます。

30、*38*、および *46* 桁目 (文字)

この桁には、入力レコードの中で指定した位置の文字と比較される識別文字を記 入します。

レコード・タイプの検査は、常に最初に指定されたレコード・タイプから始ま ります。 レコード内のデータが 2 組以上のレコード識別コードを満たしている場 合には、満たしている最初のレコード・タイプによってレコード・タイプが決定されます。

ファイルに複数のレコード・タイプを指定する場合には、各入力レコードごと に固有の識別コードの組になるようにレコード識別コードをコーディングしなけ ればいけません。

AND 関係

AND 関係は、4 つ以上のレコード識別コードによってレコードが識別される場 合に使用されます。

AND 関係を使用するためには、最初の行に少なくとも 1 つのレコード識別コー ドを入れ、残りのレコード識別コードはその後の行に続けて入れますが、使用す る各追加行の 16 から 18 桁目に AND をコーディングしてください。 16 から 18 桁目に AND が入っている各行の 7 から 15、19 から 20、および 46 から 80 桁目はブランクでなけ ればなりません。 順序およびレコード識別標識は、グループの最初の行に指定され、追加の行に指定することはできません。

入力仕様で使用できる AND/OR 行の数に制限はありません。

OR 関係

OR 関係は、2 つ以上のレコード・タイプに共通のフィールドがある場合に使 用されます。

OR 関係を使用するためには、16 から 17 桁目に OR を入れてください。 7 から 15、18 から 20、および 46 から 80 桁目はブランクでなければなりません。 レコード識別標識は 21 から 22 桁目に記入することができます。 標識の指定があり、OR 行にコーディングされたレコード識別コードが満たされた 場合には、

その行の 21 から 22 桁目に指定された標識がオンに設定されます。 標識が指定されていなければ、前の行の標識がオンに設定されます。

入力仕様で使用できる AND/OR 行の数に制限はありません。

フィールド記述項目

フィールド記述項目 (31 から 74 桁目) は、各ファイルごとにレコード識別項目 (7 から 46 桁目) の後に続けなければなりません。

6 桁目 (仕様書コード)

この行が入力仕様ステートメントであることを識別するために、 6 桁目には I がなければなりません。

7 から 30 桁目 (予約語)

7 から 30 桁目はブランクでなければなりません。

31 から 34 桁目 (データ属性)

31 から 34 桁目は、日付、時刻、または可変長の文字フィールド、図形フィールド、または UCS-2 フィールドの外部形式を指定します。

日付または時刻フィールドのこの記入項目がブランクであった場合には、ファイルについて (DATFMT か TIMFMT の いずれか、またはその両方によって) 指定された形式/区切り記号が使用されます。 ファイルについて指定された外部の日付または時刻の形式がない場合には、 エラー・メッセージが出されます。 有効な日時形式については、274 ページの表 71 および 276 ページの表 74 を参照してください。

文字データ、図形データ、または UCS-2 データの場合、可変長入力フィールドを指定するために、*VAR データ属性を使用します。 文字データ、図形データ、または UCS-2 データのこの記入項目がブランクの場合、外部形式は固定長でなければいけません。 フィールドをプログラム 内の別の場所で定義する場合、内部形式と外部形式が一致していなければ なりません。可変長フィールドについて詳しくは、253 ページの『可変長の文字形式、図形形式および UCS-2 形式』 を参照してください。

外部形式について詳しくは、247 ページの『内部形式および外部形式』 を参照してください。

35 桁目 (日付/時刻区切り記号)

35 桁目は、日付/時刻フィールドに使用される区切り文字を指定します。 ブランクの区切り記号を指定するためには、& (アンパーサンド) を使用する ことができます。 日付と時刻の形式およびそのデフォルトの区切り記号については、274 ページの表 71 および 276 ページの表 74 を参照してください。

このフィールドに指定した場合には、31 から 34 桁目 (日付/時刻の外部形式) にも 指定しなければなりません。

36 桁目 (データ形式)

記入
説明

ブランク
入力フィールドはゾーン 10 進数形式かまたは文字フィールドです。

A
文字フィールド (固定長形式または可変長形式 [1])

C
UCS-2 フィールド (固定長形式または可変長形式 [1])

G
図形フィールド (固定長形式または可変長形式 [1])

B
数値フィールド (2 進-10 進形式)

F
数値フィールド (浮動形式)

I
数値フィールド (整数形式)

L
先行 (左側) プラスまたはマイナス符号がある数値フィールド (ゾーン 10 進数形式)

N
文字フィールド (標識形式)

P
数値フィールド (パック 10 進数形式)

R
後書き (右側) プラスまたはマイナス符号がある数値フィールド (ゾーン 10 進数形式)

S
数値フィールド (ゾーン 10 進数形式)

U
数値フィールド (符号なし形式)

D
日付フィールド - 日付フィールドは 31 から 34 桁目に指定された外部形式また はデフォルトのファイル日付形式となります。

T
時刻フィールド - 時刻フィールドは 31 から 34 桁目に指定された外部形式また はデフォルトのファイル時刻形式となります。

Z
タイム・スタンプ・フィールド

注 :

1. 可変長形式の場合、31 から 34 桁目に *VAR を指定します。

36 桁目の記入項目は、プログラム記述ファイル中のデータのデータ・タイプ、お よび数値の場合はそのデータの外部データ形式を指定します。

37 から 46 桁目 (フィールドの位置)

記入
説明

2 つの 1 から 5 桁の数
フィールドの開始 (始め) およびフィールドの終了 (終わり)。

この記入項目は、入力レコードの各フィールドの位置およびサイズを指定します。 37 から 41 桁目はフィールドの開始桁の位置を指定し、42 から 46 桁目はフィールドの終了桁の位置を指定します。 1 桁のフィールドを定義するためには、37 から 41 桁目 と 42 から 46 桁目に同じ数を入れてください。 数値の指定は右寄せしなければなりませんが、先行ゼロは省略することができま す。

フィールドの各タイプごとの入力レコードにおける最大桁数は以下のとおりです。

位置
フィールドのタイプ

63
ゾーン 10 進数値 (63 桁)

32
パック形式の数値 (63 桁)

4
2 進-10 進 (9 桁)

8
整数 (20 桁)

8
符号なし (20 桁)

8
浮動 (8 バイト)

64
先行または後書き符号付きの数値 (63 桁)

10
日付

8
時刻

26
タイム・スタンプ

32766
文字 (32766 文字)

32766
図形または UCS-2 (16383 個の 2 バイト文字)

32766
可変長文字 (32764 文字)

32766
可変長の図形または UCS-2 (16382 個の 2 バイト文字)

32766
データ構造

プログラム記述入力フィールドとして指定する 文字またはデータ構造フィールドの 最大サイズは、ファイルの最大レコード長である 32766 になります。

可変長の文字、図形、または UCS-2 の入力フィールドを指定する場合、 長さには 2 バイトの接頭部が含まれます。

配列の場合には、37 から 41 桁目に配列の開始位置および 42 から 46 桁目に終了位 置を記入してください。 配列の長さは、要素の長さの整数倍でなければなりません。 配列内のすべての要 素について開始および終了位置を考慮する必要はありません。 配列へのデータの 配置は最初の要素から始まります。

47 から 48 桁目 (小数点以下の桁数)

記入
説明

ブランク
文字、図形、UCS-2、浮動、日付、時刻、またはタイム・スタンプ・フィールド。

0 から 63
数値フィールドの小数点以下の桁数。

この記入項目は 36 桁目のデータ形式の記入項目と併用され、フィールドの形 式を記述します。 このフィールドの指定は、入力フィールドを数値として識別します。すなわち、 フィールドが数値の場合には、これを指定しなければなりません。 数値フィールドについて指定される小数点以下の桁数は、フィールドの長さ を超えることはできません。

49 から 62 桁目 (フィールド名)

記入
説明

記号名
フィールド名、データ構造名、データ構造サブフィールド名、配列名、配列 要素、PAGE、PAGE1 から PAGE7、*IN、*INxx、 または *IN(xx)。

これらの桁は、RPG IV プログラムで使用される入力レコードのフィールドに名前を付けます。 この名前は、記号名に関する規則に従うものでなければなりません。

入力仕様で配列全体を参照するためには、49 から 62 桁目に配列名を入れてく ださい。 49 から 62 桁目に配列名を入れた場合には、制御レベル (63 から 64 桁目)、 突き合わせフィールド (65 から 66 桁目)、およびフィールド標識 (67 から 68 桁目) はブ ランクでなければなりません。

配列の要素を参照するためには、配列名とその後に括弧で囲んだ指標を指定 してください。 指標とは、小数点以下の桁数のない数値フィールドか、あるいは使用する配列要 素の実際の番号のことです。 指標の値は 1 から、配列内の要素の数である n の間で変えることができます。

63 から 64 桁目 (制御レベル)

記入
説明

ブランク
このフィールドは制御フィールドではありません。 制御レベル標識を全手順ファイルで使用することはできません。

L1-L9
このフィールドは制御フィールドです。

63 から 64 桁目は、制御フィールドとして使用されるフィールドを識別します。 制御 フィールドの内容を変更すると、その制御レベル標識およびそれより低いレベル のすべての標識によって条件付けされているすべての操作が処理されることにな ります。

分割制御フィールドとは、それぞれが同じ制御レベル標識を持つ複数のフィール ドから構成される制御フィールドのことです。 その制御レベル標識によって指定された最初のフィールドが分割制御フィールドの最高位の桁に入れられ、同じ制御レベル標識が指定された最後のフィールドが 分割制御フィールドの最低位の桁に入れられます。

2 進、浮動、整数、可変長文字、可変長図形、UCS-2 および符号なしフィールドは制御フィールドには使用できません。

65 から 66 桁目 (突き合わせフィールド)

記入
説明

ブランク
このフィールドは突き合わせフィールドではありません。

M1-M9
このフィールドは突き合わせフィールドです。

この記入項目は、あるファイルのレコードを他のファイルのレコードと突き合わ せるため、または 1 つのファイル内の突き合わせフィールドの順序検査をするために 使用されます。 突き合わせフィールドを指定することができるのは、1 次および 2 次ファイルのフィールドの場合だけです。

2 進、浮動、整数、可変長文字、可変長図形、UCS-2、および符号なしフィールドは突き合わせフィールドには使用できません。

レコード内の突き合わせフィールドは、該当するフィールド記述仕様行の 65 から 66 桁 目に記入された M1 から M9 のコードによって指定されます。 最大 9 つの突き合わせフィールドを指定することができます。

突き合わせフィールドのコード M1 から M9 は、任意の順序で割り当てていることができます。 例えば、M3 を M1 より前の行に定義するか、あるいは M1 がまったく 定義されていなくてもかまいません。

レコードに複数の突き合わせフィールド・コードを使用した場合には、すべてのフィ ールドを 1 つの大きなフィールドと見なすことができます。 M1 または使用した最低のコードがフィールドの右端、すなわち最低位の桁となり ます。 M9 または使用した最高のコードがフィールドの左端、すなわち最高位の桁 となります。

制御仕様書に ALTSEQ (代替照合順序) および FTRANS (ファイル変換) キー ワードを指定して、突き合わせフィールドの照合順序を変更することができます。

単一の順次ファイル (入力、更新、または入出力共用) だけに突き合わせフィールド を指定した場合には、ファイル内の突き合わせフィールドの順序が検査されます。 MR 標識はオンに設定されず、プログラムで使用することはできません。 順序が違っているレコードがあると、RPG IV 例外/エラー処理ルーチンに制御が与えられることになります。

突き合わせフィールドは、順序検査のほかに、1 次ファイルのレコードと 2 次ファイルのレコードとの突き合わせにも使用されます。

67 から 68 桁目 (フィールドとレコードの関連)

記入
説明

ブランク
フィールドはすべてのレコード・タイプに共通です。

01 から 99
一般標識。

L1-L9
制御レベル標識。

MR
突き合わせレコード標識。

U1-U8
外部標識。

H1-H9
停止標識。

RT
戻り標識。

フィールドとレコードの関連標識は、特定のレコード・タイプが OR 関係にあ るいくつかのうちの 1 つである場合に、フィールドをそのレコード・タイプに関 連づけるために使用されます。この記入項目によって、書き込む必要がある行数が減ります。

行に記述されたフィールドは、67 から 68 桁目にコーディングされた標識がオンの場合、あるいはその 67 から 68 桁目がブランクである場合にのみ、RPG IV プログラムによってレコードから抜き出されます。 67 から 68 桁目がブランクの場合には、このフィールドは OR 関係によって定義され たすべてのレコード・タイプに共通です。

フィールドとレコードの関連標識は、制御レベル・フィールド (63 から 64 桁目) および突き合わせフィールド (65 から 66 桁目) と一緒に使用することができます。

69 から 74 桁目 (フィールド標識)

記入
説明

ブランク
標識の指定はありません。

01 から 99
一般標識

H1-H9
停止標識。

U1-U8
外部標識

RT
戻り標識。

69 から 74 桁目の記入項目は、フィールドまたは配列要素がプログラムに読み込 まれる時にその状況をテストするものです。 フィールド標識は、テストされるフィールドと同じ行に指定されます。 フィールドの状況 (プラス、マイナス、ゼロ、またはブランク) に応じて、該当 する標識がオンに設定されるので、それより後の仕様を条件付けするために使 用することができます。 同じ標識を 2 個所に指定することができますが、3 個所すべてに使用してはなり ません。 指標付きでない配列または先読みフィールドでフィールド標識を使用することは できません。

69 から 70 桁目 (プラス) および 71 から 72 桁目 (マイナス) は、数値フィールド の場合にのみ有効です。 73 から 74 桁目を使用して、数値フィールドがゼロであるかどうか、および 文字フィールド、図形フィールド、または UCS-2 フィールドがブランクで あるかどうかをテストすることができます。

フィールド標識は、レコードの読み取り時にフィールドまたは配列要素が指定され た条件を満たした場合にオンに設定されます。 各フィールド標識は、1 つのレコード・タイプとだけ関連しており、したがってその標識は、関連したレコードが再び読み取られるか、あるいは標識が他の仕様 に定義されるまでリセット (オンまたはオフ) されません。

外部記述ファイル

6 桁目 (仕様書コード)

この行を入力仕様ステートメントとして識別するために、6 桁目に I がなけれ ばなりません。

レコード識別項目

外部記述ファイルの記述がコンパイラーによって検索される場合には、レコー ド定義も検索されます。 レコード定義を参照するためには、プログラムの入力、 演算、および出力の各仕様にレコード様式名を指定してください。 外部記述ファ イルの入力仕様は、次の場合に必要になります。

- レコード識別標識を指定する場合。
- レコード内のフィールドの名前をそのプログラムに合わせて変更する場合。
- 制御レベル標識または突き合わせフィールド標識を使用する場合。
- フィールド標識を使用する場合。

外部記述ファイルでは、フィールド記述仕様はレコード識別仕様のすぐ後に続 けなければなりません。

外部記述ファイルのレコード行によって、レコードの一時変更仕様の始まりが 定義されます。 別のレコード様式名またはファイル名が入力仕様の 7 から 16 桁目 で見付かるまで、そのレコード行に続くすべての仕様がレコードの一時変更の一 部となります。 外部記述ファイルに属するすべてのレコード行は 1 つにまとまっ ていなければならず、他のファイルに対する指定と混合させることはできません。

7 から 16 桁目 (レコード名)

次のいずれかを入力してください。

- レコード様式の外部名。 (外部記述ファイルにはファイル名を使用できません。)
- 外部レコード様式の名前が変更された場合には、ファイル仕様書の RENAME キーワードによって指定された RPG IV の名前。 プログラムの入力仕様の 7 から 16 桁目には、レコード様式名を一度だけ表すことができます。

17 から 20 桁目 (予約語)

17 から 20 桁目はブランクでなければいけません。

21 から 22 桁目 (レコード識別標識)

これらの桁のレコード識別標識の指定はオプションですが、指定する場合は、この章の前半の 494 ページの『プログラム記述ファイル』で説明されている規則に従います。先読みの指定は外部記述ファイルで許可されていないので該当しません。

23 から 80 桁目 (予約語)

23 から 80 桁目はブランクでなければなりません。

フィールド記述項目

外部記述ファイルのフィールド記述仕様を使用して、レコード内のフィールド の名前をプログラムに合わせて変更したり、制御レベル、フィールド標識、およ び突き合わせフィールド機能を指定することができます。 フィールド定義 (属性) は 外部記述ファイルから検索されるので、プログラムで変更することはできません。 フィールドの属性が RPG IV プログラムに対して有効ではない場合、そのフィールドを使用することはできません。 外部レコード様式に含まれているフィールドについても、ソース・ステートメント の場合と同じに診断検査が行われます。

通常、外部記述入力フィールドは、プログラム内の別の場所で実際に使用される 場合のみ、入力操作時に読み取られます。 DEBUG または DEBUG(*YES) を指定した場合には、すべての外部記述入力フィールドが、プログラムで使用されていない場合でも、読み取られます。

7 から 20 桁目 (予約語)

7 から 20 桁目はブランクでなければなりません。

21 から 30 桁目 (外部フィールド名)

外部記述ファイルのレコードの中のフィールドの名前を変更する場合には、こ れらの桁にそのフィールドの名前を記入してください。 名前がプログラムに指定 されたフィールド名と同じであると、2 つの異なる名前が必要であるため、フィ ールドの名前を変更することが必要になることがあります。

注 : この入力フィールドが、PREFIX キーワードがコーディングされたファイルのためのフィールドであり、同じレコードに関する以前の入力仕様の「フィールド名 (Field Name)」記入項目 (桁 49 から 62) に 接頭部付きの名前がすでに指定されている場合には、 接頭部付きの名前を外部名として使用する必要があります。 詳しくは、381 ページの『PREFIX(接頭部 {:置き換えられる文字数 })』 を参照してください。

31 から 48 桁目 (予約語)

31 から 48 桁目はブランクでなければなりません。

49 から 62 桁目 (フィールド名)

このフィールド名は、外部記述に RPG IV の機能 (制御レベルなど) を追加する必要がある場合にのみ指定します。 フィールド名の指定には次の 1 つが含ま れます。

- 外部レコード記述で定義されたとおりのフィールドの名前 (10 桁以内の場合)。
- プログラムで使用するために指定され、21 から 30 桁目に指定した外部名に置き 換わる名前。

フィールド名は、記号名の使用に関する規則に従うものでなければなりません。

標識はヌル値可能にすることはできません。

63 から 64 桁目 (制御レベル)

この記入項目は、フィールドがプログラム内で制御フィールドとして使用され るかどうかを指示します。

記入
説明

ブランク
このフィールドは制御フィールドではありません。

L1-L9
このフィールドは制御フィールドです。

制御フィールドとして、ヌル可能フィールドおよび UCS-2 フィールドは使用できません。

注 : 外部記述ファイルの場合に、分割制御フィールドは、入力仕様に指定され たフィールドの順序ではなく、データ記述仕様 (DDS) に指定されたフィールドの 順序で結合されます。

65 から 66 桁目 (突き合わせフィールド)

この記入項目は、フィールドがプログラム内で突き合わせフィールドとして使用さ れるかどうかを指示します。

記入
　説明

ブランク
　このフィールドは突き合わせフィールドではありません。

M1-M9
　このフィールドは突き合わせフィールドです。

突き合わせフィールドとして、ヌル可能フィールドおよび UCS-2 フィールドは使用できません。

突き合わせフィールドについて詳しくは、503 ページの『65 から 66 桁目 (突き合わせフィールド)』 を参照してください。

67 から 68 桁目 (予約語)

67 から 68 桁目はブランクでなければなりません。

69 から 74 桁目 (フィールド標識)

記入
　説明

ブランク
　標識の指定はありません。

01 から 99
　一般標識

H1-H9
　停止標識

U1-U8
　外部標識

RT
　戻り標識。

フィールド標識は、ALWNULL(*USRCTL) キーワードが 制御仕様書またはコマンド・パラメーターとして指定されている 場合にのみ、ヌル値可能フィールドに使用することができます。

フィールドがヌル値可能フィールドで、値がヌル値である場合、標識は オフに設定されます。

詳しくは、504 ページの『69 から 74 桁目 (フィールド標識)』を参照してください。

75 から 80 桁目 (予約語)

75 から 80 桁目はブランクでなければなりません。

演算仕様書

宣言演算仕様書のみがメイン・ソース・セクションで許可されます。

メイン・ソース・セクション内の演算仕様書は、以下の順番でグループ化されている必要があります。

- 明細演算
- 合計演算

- サブルーチン

サブプロシージャーの演算仕様書には、以下の 2 つのグループが含まれています。

- サブプロシージャーの本体
- サブルーチン

グループ内の演算は、実行する順番で指定する必要があります。

注 : キーワード MAIN または NOMAIN が制御仕様書に定義されている場合は、メイン・ソース・セクションでは宣言演算仕様書のみが許可されます。

演算仕様書は、以下の 3 つの異なるフォーマットで指定できます。

- 508 ページの『従来型の構文』
- 514 ページの『拡張演算項目 2 の構文』
- 515 ページの『自由形式の演算ステートメント』.

演算仕様書の記入項目を個々の命令コードごとに指定する方法について詳しく は、710 ページの『命令コード』を参照してください。

演算仕様書は、SQL ステートメントを ILE RPG プログラムに入力するために使用することもできます。 詳しくは、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」および『IBM i Information Center データベースおよびファイル・システム』のカテゴリーを参照してください。

従来型の構文

演算仕様書の一般的なレイアウトは次のとおりです。

- 演算仕様書タイプ (C) は 6 桁目に入れられます。
- 仕様書の注記以外の部分は、7 から 80 桁目です。 これらの位置は、以下を指定する 3 つの部分に分割されます。
  - 演算の実行時期*:*

    7 から 11 桁目に指定されている制御レベル標識および条件付け標識は、演算がいつどのような条件の下で実行されるかを決定します。
  - 実行する演算の種類*:*

    12 から 70 桁目 (拡張演算項目 2 を使用する命令の場合は 12 から 80 桁。514 ページの『拡張演算項目 2 の構文』および 596 ページの『式』を参照) に指定されている記入項目は、実行する演算の種類、その命令を実行するデータ (フィールドまたはファイルなど)、および演算結果を含むフィールドを指定します。
  - 命令の結果実行する検査*:*

    71 から 76 桁目に指定されている標識は、演算結果の検査で使用され、 以降の演算または出力命令の条件を決定できます。 結果標識位置の使用方法は、命令コードによって異なります。 これらの位置の使用法については、710 ページの『命令コード』の個々の命令コードを参照してください。
- 仕様書の注記部分は 81 から 100 桁目です。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+...
9 ...+... 10
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....Comments++++++++++
++
CL0N01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++Comments++++++++++
++
```

図 *166.* 演算仕様書のレイアウト

演算仕様書拡張演算項目 2 継続記入行

「拡張演算項目 2 (Extended Factor-2)」フィールドは、次のように後続行に継続できます。

- 継続記入行の 6 桁目には C が入っていなければなりません。
- 継続記入行の 7 から 35 桁目はブランクでなければなりません。
- 指定は 36 桁目以降から継続されます。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+...
9 ...+... 10
C.............................Extended-factor2-continuation+++++++++++++++++Comments++++++++++
++
```

図 *167.* 演算仕様書拡張演算項目 *2* 継続記入行

6 桁目 (仕様書コード)

この行が演算仕様書ステートメントであることを識別するために、6 桁目には C がなければなりません。

7 から 8 桁目 (制御レベル)

記入
説明

ブランク
演算命令は、9 から 11 桁目の標識で許可されている場合、または演算がサブルーチンの一部の場合、それぞれのプログラム・サイクルごとに明細演算時間で実行されます。 ブランクは、宣言命令コード用にも使用します。

L0
演算命令は、それぞれのプログラム・サイクルごとに、合計演算時間で実行されます。

L1-L9
演算命令は、制御レベル標識がオンになっている場合、合計演算時間で実行されます。 標識は、レベルの切れ目によって、または入力や演算命令の結果として設定されます。

LR
演算命令は、最後のレコードが処理された後、または LR 標識がオンに設定された後に実行されます。

SR
演算命令は RPG IV サブルーチンの一部です。 ブランクの記入項目は、サブルーチンの一部である演算でも有効です。

AN、OR
複数の行の標識で、演算に条件を与えます。

制御レベル標識

L0 記入項目は、演算が必ず合計演算時間中に実行されるということを示す ために 7 から 8 桁目で使用されます。

標識 L1 から L9 が 7 から 8 桁目に指定されている場合、演算は、 指定されている標識がオンになっている場合のみ、合計演算時間で処理されます。 L1 から L9 が制御の切れ目によってオンに設定されている場合、すべてのより低いレベル標識もオンに設定されます。 7 から 8 桁目がブランクであれば、演算は明細時に実行されるか、サブルーチン 内のステートメントであるか、または宣言ステートメントであるか、または継続行です。

次の命令 PLIST、PARM、KLIST、KFLD、TAG、DEFINE、および ELSE は、7 から 8 桁目がブランクの状態で、 合計演算内で指定できます。 (9 から 11 桁目の条件付け標識は、これらの命令では許可されていません。) また、TAG および ELSE を除くすべての事前命令は、1 つのサブルーチンの ENDSR 命令および次のサブルーチンの BEGSR 命令の間、または最後のサブルーチンの ENDSR 命令の後を含む、演算内のどの場所でも指定できます。

注 : 制御標識は、サブプロシージャーには指定できません。

最終レコード標識

LR 標識が 7 から 8 桁目に指定されている場合は、演算は最後の合計演算時間中に実行されます。 LR 標識は、サブプロシージャーでは指定できないことに注意してください。

プログラムに 1 次ファイルが存在し、2 次ファイルが存在しない場合は、LR 標識は、 最後の入力レコードが読み込まれ、レコードに対して指定されている演算が実行され、 最後のレコード読み込みの明細出力が完了した後に設定されます。

複数の入力ファイル (1 次および 2 次) が存在している場合、プログラマーは、 ファイル記述仕様書の 19 桁目に E を入力することによって、どのファイルのファイルの終わりを検査するかを決定します。 ファイルの終わりが指定されているすべてのファイルの読み込み完了時、 これらのファイルの最後のレコードに対する明細出力完了時、およびすべての一致する 2 次レコードの処理後に、LR はオンに設定されます。

最後の入力レコードが読み込まれた後に LR 標識がオンに設定されている場合は、プログラムに対して定義されているすべての制御標識 L1 から L9 もオンに設定されます。

サブルーチン *ID*

オプションで、7 から 8 桁目の SR 記入項目は、文書補助としてサブルーチン内の命令に対して使用できます。 サブルーチン行は、合計演算仕様書の後に表示する必要があります。 命令コード BEGSR および ENDSR は、サブルーチンの区切り文字として機能します。

AND/OR 行 *ID*

7 から 8 行目には、AN または OR を入力して、演算の追加の標識 (9 から 11 桁目) を定義できます。

1 つの AND/OR 行または AND/OR 行のグループの直前行の 7 から 8 桁目の記入項目は、いつ演算が処理されるのかを決定します。 グループの最初の行の 7 から 8 桁目の記入項目は、そのグループのすべての AND/OR 行に適用されます。 制御レベル標識 (L1 から L9、L0、または LR) は、合計演算用に入力され、SR またはブランクはサブルーチン用、およびブランクは明細演算用です。

9 から 11 桁目 (標識)

記入
説明

ブランク
演算は、それぞれのレコードで処理されます。

01 から 99
一般標識。

KA から KN、KP から KY
機能キー標識。

L1-L9
制御レベル標識。

LR
最終レコード標識。

MR
突き合わせレコード標識。

H1-H9
停止標識。

RT
戻り標識。

U1-U8
外部標識。

OA から OG、OV
オーバーフロー標識。

10 から 11 桁目には、特定の演算を処理する必要があるかどうかを判別するために検査する標識が含まれています。 9 桁目のブランクは、演算を実行するためにその標識がオンになっている必要があるということを指定します。 9 桁目の N は、演算を実行するために関連付けされている標識がオフになっている必要があるということを指定します。

12 から 25 桁目 (演算項目 1)

演算項目 1 は、フィールドに名前を付けるか、命令が実行される実際のデータ (リテラル) を指定するか、または命令を実行する方法に関する追加情報を提供する RPG IV 特殊語 (例えば *LOCK) を含みます。 記入項目は、12 桁目から始まっている必要があります。 演算項目 1 で有効な記入項目は、26 から 35 桁目に指定された命令コードによって異なります。特定の命令コードの演算項目 1 に対する特定の記入項目については、710 ページの『命令コード』を参照してください。 いくつかの命令コードと一緒に、2 つのオペランドがコロンで分離して指定されている場合があります。

26 から 35 桁目 (命令および拡張)

26 から 35 桁目は、演算項目 1、演算項目 2、および結果フィールド記入項目を使用して実行する命令の種類を指定します。 命令コードは、26 桁目から開始する必要があります。 命令コードについて詳しくは、541 ページの『操作』および 710 ページの『命令コード』を参照してください。命令コード拡張について詳しくは、511 ページの『命令拡張』を参照してください。

命令拡張

記入
説明

ブランク
命令拡張は指定されていません。

A
DUMP 命令で、DEBUG オプションが H 仕様で設定されているかどうかとは関係なく、 この命令が必ず実行されるということを示すために使用します。

H
数値操作の四捨五入 (丸め) 結果

N
レコードは読み込まれますがロックされません

DEALLOC が正常に行われた後、ポインターを *NULL に設定する

P
結果フィールドにブランクの埋め込みを行います。

D
バインド呼び出し時に操作記述子を渡す

日付フィールド

T
時刻フィールド

Z
タイム・スタンプ・フィールド

M
デフォルトの精度規則

R
"結果の小数点以下の桁数" 精度規則

E
エラー処理

命令拡張は、添えられている命令に追加属性を提供します。 命令拡張は、演算仕様書の 26 から 35 桁目に指定されています。 命令拡張は、命令コードの右側から始まり、括弧で囲まれている必要があります。 読みやすさのためにブランクを使用することができます。 例えば、MULT(H)、MULT (H)、MULT (H) はすべて有効な記入項目です。

複数の命令拡張が指定できます。 例えば、CALLP 命令は、CALLP(EM) を使用して、エラー処理およびデフォルトの精度規則の両方を指定できます。

H は、結果フィールドの内容が四捨五入 (丸め) されるかどうかを指示します。 結果の標識は、四捨五入が行われた後で、結果のフィールドの値に応じて設定されます。

更新ディスク・ファイル上の READ、READE、READP、READPE、または CHAIN 命令の N は、 レコードが読み取り用であること、ただしロックされていないことを示しています。 値が指定されていない場合は、ロッキングのデフォルトのアクションが発生します。

DEALLOC 命令の N は、結果フィールド・ポインターが、正常な割り振り解除後に *NULL に設定されることを示しています。

P は、結果フィールドが命令結果よりも長い場合、指示の実行後に結果フィールドが埋め込まれることを示しています。

D が CALLB 命令コードに指定されている場合、操作記述子が組み込まれていることを示しています。

D、T、および Z 拡張は、日付、時刻、またはタイム・スタンプ・フィールドを示すために、TEST 命令コードで使用できます。

M および R は、単一自由形式の式の精度用に指定されています。 詳しくは、「608 ページの『数値演算の精度の規則』」を参照してください。

M は、デフォルトの精度規則が使用されることを示しています。

R は、10 進の中間の精度が、 小数位の数が割り当て結果の小数点以下の桁数より少なくならないように計算されることを示しています。

E は、命令に関するエラーが組み込み関数 %ERROR を使用して検査されることを示しています。

36 から 49 桁目 (演算項目 2)

演算項目 2 は、フィールド、レコード・フォーマット、またはファイルに名前を付けるか、 命令が実行される実際のデータを指定するか、または実行する命令に関する追加情報を提供する特殊語 (例えば*ALL) を含みます。 記入項目は、36 桁目から始まっている必要があります。 演算項目 2 で有効な記入項目は、26 から 35 桁目に指定された命令コードによって異なります。 いくつかの命令コードと一緒に、2 つのオペランドがコロンで分離して指定されている場合があります。特定の命令コードの演算項目 2 に対する特定の記入項目については、710 ページの『命令コード』を参照してください。

50 から 63 桁目 (結果フィールド)

結果フィールドは、26 から 35 桁目に指定されている演算命令の結果を含むフィールドまたはレコード・フォーマットに名前を付けます。 指定されているフィールドは、変更可能でなければなりません。 例えば、先読みフィールドまたはユーザー日付フィールドにすることはできません。 いくつかの命令コードと一緒に、2 つのオペランドがコロンで分離して指定されている場合があります。 個別の命令コードに関する結果フィールド規則については、710 ページの『命令コード』を参照してください。

64 から 68 桁目 (フィールド長)

記入
説明

1 から 63
数字フィールド長。

1 から 99999
文字フィールド長。

ブランク
結果フィールドはどこか別の場所に定義されているか、この命令コードを使用してフィールドを定義できません。

64 から 68 桁目は、結果フィールドの長さを指定します。 この記入項目はオプションですが、プログラム内の他の場所で定義されていない 数値または文字フィールドを定義するために使用できます。 フィールド記入項目のこれらの定義は、結果フィールドにフィールド名が含まれている場合に許可されます。 他の

データ・タイプは、*LIKE DEFINE 命令を使用して、定義仕様書、または演算仕様書で定義する必要があります。

記入項目は、結果フィールド用に予約する桁数を指定します。 記入項目は右寄せする必要があります。 数字フィールドは、アンパック長 (桁数) で指定する必要があります。

結果フィールドがプログラムの他の場所に定義されている場合は、長さの指定は必要ありません。 ただし、長さが指定されていて結果フィールドが他の場所に定義されている場合は、 長さは事前に定義された長さと同じになっている必要があります。

69 から 70 桁目 (小数点以下の桁数)

記入
説明

ブランク
結果フィールドは文字データで、プログラム内の他の場所に定義されているか、フィールド名が指定されていません。

0 から 63
数値結果フィールドの小数点以下の桁数。

69 から 70 桁目は、数値結果フィールドの小数部の右側の桁数を示しています。 数値結果フィールドに小数点以下の桁数が含まれていない場合は、'0' (ゼロ) を入力します。 この桁は、結果フィールドが文字データの場合、またはフィールド長が指定されていない場合はブランクになっている必要があります。 指定した小数点以下の桁数は、フィールド長を超えられません。

71 から 76 桁目 (結果標識)

これらの桁は、例えば命令の完了後に結果フィールドの値を検査するために、 またはファイルの終わり、エラー、またはレコードが見つからない、などの状態を示すために使用できます。 命令によっては、3 つの結果標識の別の組み合わせを指定することによって、 命令が実行される方法を制御できます (例えば LOOKUP)。 結果標識の桁の使用方法は、指定した命令コードによって異なります。 関連する結果標識の説明については、710 ページの『命令コード』の個々の命令コードを参照してください。 算術演算の場合、結果フィールドは、フィールドが切り捨てられ、四捨五入が実行された (指定されている場合) 後にのみ検査されます。 標識の設定は、指定した検査結果によって異なります。

記入
説明

ブランク
結果標識は指定されていません。

01 から 99
一般標識

KA から KN、KP から KY
機能キー標識

H1-H9
停止標識

L1-L9
制御レベル標識

LR
最終レコード標識

OA から OG、OV
オーバーフロー標識

U1-U8
外部標識

RT
戻り標識。

結果フィールドが非索引配列を使用している場合には、結果標識を使用することはできません。

同じ標識が複数の演算仕様書で結果標識として使用されている場合は、 最新に処理された仕様書が、標識の状況を決定します。

結果標識の指定にあたっては、以下の点に留意してください。

- 演算命令が処理されている時には、指定した結果標識がオフに設定され、 結果標識によって指定されている条件が満たされた場合に、その標識がオンに設定されます。
- 制御レベル標識 (L1 から L9) がオンに設定されている場合、より低いレベル標識はオンに設定されません。
- 停止標識 (H1 から H9) がオンに設定されたときに、 または標識が検査される前に停止標識がオフに設定されなければ、RETURN 命令が処理されるときに、 プログラムはそのサイクル内の次の *GETIN で異常終了します。

拡張演算項目 2 の構文

特定の命令コードでは、拡張演算項目 2 フィールドで式を使用できます。

7 から 8 桁目 (制御レベル)

509 ページの『7 から 8 桁目 (制御レベル)』を参照してください。

9 から 11 桁目 (標識)

510 ページの『9 から 11 桁目 (標識)』を参照してください。

12 から 25 桁目 (演算項目 1)

演算項目 1 はブランクでなければなりません。

26 から 35 桁目 (命令および拡張)

26 から 35 桁目は、拡張演算項目 2 フィールドの式を使用して実行する命令の種類を指定します。 命令コードは、26 桁目から開始する必要があります。 命令コードについて詳しくは、541 ページの『操作』および 710 ページの『命令コード』を参照してください。命令コード拡張について詳しくは、514 ページの『命令拡張』を参照してください。

プログラムは、演算仕様書書式で指定されている順番で命令を処理します。

命令拡張

記入
説明

ブランク
命令拡張は指定されていません。

H
数値操作の四捨五入 (丸め) 結果

M
デフォルトの精度規則

R
"結果の小数点以下の桁数" 精度規則

E
エラー処理

算術計算 EVAL および RETURN 命令では、H 拡張を使用して四捨五入を指定することができます。

CALLP、DOU、DOW、EVAL、IF、RETURN、および WHEN 命令では、M または R 拡張を使用して、精度のタイプを指定することができます。

CALLP 命令では、E 拡張を使用して、エラー処理を指定することができます。

36 から 80 桁目 (拡張演算項目 2)

このフィールドは、自由形式構文を使用します。 このフィールドはオペランドおよびオペレーターで構成され、オプションで複数の行に継続できます。 複数行を利用して指定した場合は、7 から 35 桁目の継続行はブランクでなければなりません。

拡張演算項目 2 を使用する命令は以下のとおりです。

- 721 ページの『CALLP (プロトタイプ・プロシージャーまたはプログラムの呼び出し)』
- 740 ページの『DATA-GEN (変数からの文書の生成)』
- 743 ページの『DATA-INTO (文書の変数への構文解析)』
- 760 ページの『DOU (条件が真になるまでの繰り返し)』
- 763 ページの『DOW (条件が真の間繰り返し)』
- 769 ページの『ELSEIF (ELSE IF)』
- 771 ページの『EVAL (式の評価)』
- 774 ページの『EVAL-CORR (対応するサブフィールドの代入)』
- 773 ページの『EVALR (式の評価、右寄せ)』
- 784 ページの『FOR (For)』
- 786 ページの『FOR-EACH (それぞれの場合)』
- 789 ページの『IF (If)』
- 838 ページの『ON-ERROR (エラーの時)』
- 839 ページの『ON-EXIT (終了時)』
- 867 ページの『RETURN (呼び出し元への戻し)』
- 881 ページの『SORTA (配列の分類)』
- 902 ページの『WHEN (真の場合に選択)』
- 908 ページの『XML-INTO (XML 文書の変数への構文解析)』
- 947 ページの『XML-SAX (XML 文書の構文解析)』

詳しくは、特定の命令コードを参照してください。 継続行の符号化について詳しくは、313 ページの『継続の規則』を参照してください。

自由形式の演算ステートメント

自由形式ステートメントで使用可能な桁については、307 ページの『自由形式ステートメント』を参照してください。

自由形式ステートメントでは、命令コードは使用可能な桁内の特定の位置で始まる必要はありません。 すべての拡張は、同一行上の命令コードの直後に、括弧で囲む必要があります。 命令コードおよび拡張の間には、ブランクを入れないでください。 命令コードおよび拡張の次に、演算項目 1、演算項目 2、結果フィールド・オペランドをブランクで分離して指定します。 これらのいずれかの項目が命令に必要ない場合、空白のままにしておきます。 ステートメントの残りの部分は、ブランクおよび継続行を自由に使用できます。 それぞれのステートメントは、セミコロンで終了する必要があります。 セミコロンの後ろのレコードの残り部分は、ブランクになっているか、ま たは行末コメントを含んでいる必要があります。

命令コード EVAL または CALLP において、拡張が不要で、変数またはプロトタイプの名前が命令コードの名前と異なる場合には、命令コードを省略できます。 例えば、以下の 2 つのステートメントは等価です。

```
eval pos = %scan (',': name);
pos = %scan (',': name);
```

自由形式演算ブロック内のすべてのレコードは、6 から 7 桁目をブランクにしておく必要があります。

コンパイラー指示は、以下の制約事項に沿って、自由形式演算ブロック内に指定できます。

- コンパイラー指示は、行の最初の項目でなければなりません。 次の行に継続することはできません。
- 桁制約付きコードでは、指示は 7 桁目以降の任意の位置で開始することができます。
- コンパイラー指示は、ステートメントでは許可されていません。 指示は、1 つのステートメントの終了後、次のステートメントが始まる前の新規行に表示する必要があります。

自由形式のオペランドは、14 文字よりも長くすることができます。 以下はサポートされていません。

- 数値リテラルの継続
- フィールド名の定義
- 結果標識。 (結果標識を持つ命令コードを使用する必要があるほとんどの場合、代わりに等価の組み込み関数を使用できます。)

合計演算の開始を示すには、7 から 8 桁目に指定されている制御レベルを使用して、 固定形式演算仕様書をコード化します。 完全自由形式モードでは、この固定形式ステートメントはコピー・ファイルに指定しなければなりません。 合計演算は、自由形式演算構文を使用して指定することができます。 自由形式演算仕様書には、制御レベルの記入項目が含まれていないため、 特定のレベルの中断で実行される演算は、「IF *INLx;」ステートメントを使用して条件付けを行う必要があります。

1. 明細演算
2. 合計演算の開始
3. 特定の制御レベル標識による演算の条件付け

```
    items += 1;                    1
CL0   Total         TAG            2
    IF *INL1;                      3
       EXCEPT orderTotal;
       orders += 1;
       totalItems += items;
       items = 0;
    ENDIF;
```

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....

        read  file;              // Get next record
        dow not %eof(file);      // Keep looping while we have
                                 // a record
            if %error;
                dsply 'The read failed';
                leave;
            else;
                chain(n) name database data;
                time = hours * num_employees
                       + overtime_saved;
                pos = %scan (',': name);
                name = %xlate(upper:lower:name);
                exsr handle_record;
                read file;
            endif;
        enddo;

     begsr handle_record;
        eval(h) time = time + total_hours_array (empno);
        temp_hours = total_hours - excess_hours;
        record_transaction();
     endsr;
```

図 *168.* 自由形式演算仕様書の例

桁制限付きソースでは、自由形式の演算仕様書と従来型の演算仕様書を同じプログラム内で結合することができます。

```
C                   testb     OPEN_ALL      flags                   10
           if *in10;
            openAllFiles();
           endif;
```

図 *169.* 従来型演算仕様書および自由形式演算仕様書を結合する例

自由形式演算

自由形式ステートメントで使用可能な桁については、307 ページの『自由形式ステートメント』を参照してください。

自由形式構文でサポートされている命令を入力します。 命令コード (EVAL および CALLP はオプション) の次にオペランドまたは式をコード化します。 命令はオプションで複数の行に継続できます。 新規継続文字は不要です。それぞれのステートメントはセミコロン (;) で終了します。ただし、既存の継続規則も適用できます。

自由形式の構文を使用できる命令コードのリストについては、542 ページの表 103 を参照してください。 自由形式構文を使用できない命令については、710 ページの『命令コード』の詳細な説明を参照して、推奨されている代替構文があるかどうか確認します。 継続行の符号化について詳しくは、313 ページの『継続の規則』を参照してください。

出力仕様

出力仕様は、プログラム記述出力ファイルのレコード、 フィールドの形式、およびレコードの書き出し時点を記述します。 外部記述ファイルの場合には、出力仕様は任意指定です。 制御仕様書に MAIN または NOMAIN が コーディングされている場合には、例外出力だけを実行することができます。

出力仕様は、プログラムのすべてのファイルに使用されるわけではありません。 ファイルによっては、出力操作の結果フィールドでデータ構造をコーディングする必要があります。 プログラムの以下のファイルでは、出力仕様を使用しません。

- サブプロシージャーで定義されるファイル
- QUALIFIED キーワードを指定して定義されるファイル
- TEMPLATE キーワードを指定して定義されるファイル
- LIKEFILE キーワードを指定して定義されるファイル

出力仕様は、2 つのカテゴリーに分けることができます。すなわち、レコード 識別と制御 (7 から 51 桁目) およびフィールド記述と制御 (21 から 80 桁目) です。 出力仕様の各カテゴリーの詳細については、以下に示されています。

- プログラム記述ファイルの記入項目
- 外部記述ファイルの記入項目

出力仕様ステートメント

出力仕様の一般的なレイアウトは次のとおりです。

- 6 桁目には出力仕様コード (O) が入れられます。
- 仕様書の注記以外の部分は 7 から 80 桁目です。
- 仕様書の注記部分は 81 から 100 桁目です。

プログラム記述

プログラム記述ファイルの場合には、出力仕様の記入項目は次の 2 つのカテゴ リーに分けられます。

- レコード識別および制御 (7 から 51 桁目)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+...
9 ...+... 10
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+.............................Comment+++++++++
++++
OFilename++DAddN01N02N03Excnam++++.....................................Comment+++++++++
++++
O.........And..N01N02N03................................................Comment+++++++++
++++
```

図 *170.* プログラム記述レコードのレイアウト

- フィールド記述および制御 (21 から 80 桁目)。 各フィールドは、その対応するレコード識別項目の下の別個の行に記述されます。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+...
9 ...+... 10
O.............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat++Comment+++++++++
++++
O.............................................Constant/editword-ContinutioComment+++++++++
++++
```

図 *171.* プログラム記述フィールドのレイアウト

外部記述

外部記述ファイルの場合には、出力仕様の記入項目は次のカテゴリーに分け られます。

- レコード識別および制御 (7 から 39 桁目)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+...
9 ...+... 10
ORcdname+++D...N01N02N03Excnam++++.....................................Comment+++++++++
++++
ORcdname+++DAddN01N02N03Excnam++++.....................................Comment+++++++++
++++
O.........And..N01N02N03Excnam++++.....................................Comment+++++++++
++++
```

図 *172.* 外部記述レコードのレイアウト

- フィールド記述および制御 (21 から 43, および 45 桁目)

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+...
9 ...+... 10
O.............N01N02N03Field+++++++++.B.............................Comment+++++++++
++++
```

図 *173.* 外部記述フィールドのレイアウト

プログラム記述ファイル

6 桁目 (仕様書コード)

6 桁目には、この行を出力仕様ステートメントとして識別する O がなけ ればなりません。

レコード識別および制御項目

7 から 51 桁目の記入項目は、ファイルを構成する出力レコードを識別し、印刷報 告書のために正しいスペーシングを行い、また、レコードが書き出される条件 を決定します。

7 から 16 桁目 (ファイル名)

記入
説明

有効なファイル名
出力ファイルに対するファイル仕様書の指定と同じファイル名。

ファイル名は、ファイルの出力レコードを定義する最初の行に指定してくださ い。 指定するファイル名は、ファイル仕様書で出力、更新、または入出力共 用ファイルに割り当てられたファイル名と同じでなければなりません。 ファイル からのレコードが出力仕様上に散在している場合には、ファイルが変わるたびにファイル名を指定しなければなりません。

ADD と一緒に出力、 更新、入出力共用 または 入力 として指定されたファイルの場合には、データ構造名を結果のフィールドに指定した 明示のファイル命令コードを演算で使用することがない限り、少なくとも 1 つの出力仕様が必要です。 例えば、WRITE 命令に出力仕様は必要ありません。

16 から 18 桁目 (論理関係)

記入
説明

AND または OR
AND/OR は、出力標識の行相互間の関係を指示します。 AND/OR 行は出力レコードには有効ですが、フィールドには有効でありません。

16 から 18 桁目は、出力命令の場合の AND/OR 行を指定します。 この関係を指定するためには、ファイル名が入っている行に続く追加の行の 16 から 18 桁目に それぞれ AND/OR を記入してください。 各 AND 行には、少なくとも 1 つの標識を指定しなければなりません。 AND 関係では、18 桁目のフェッチ・オーバーフロー・ルーチンは 最初の記入行 (ファイル名の行) にだけ指定しなければなりません。 フェッチ・オーバーフローの指定が必要なのは、 フェッチ・オーバーフロー・ルーチンを必要とするレコード・タイプの OR 行の場合です。

AND/OR を指定した時には、7 から 15 桁目はブランクでなければなりません。

出力仕様に指定できる AND/OR 行の数に制限はありません。

17 桁目 (タイプ)

記入
説明

H または D
明細レコードには、通常、入力レコードから直接もたらされたデータか、ま たは明細時に処理された演算の結果であるデータが入っています。 見出しレコードには、通常、タイトル、欄見出し、ページ番号、および日付など の情報を識別する定数が入っています。 見出しレコードと明細レコードの間の区別はありません。 H または D の指定は、 プログラマーがプログラムを文書化する場合に役立つために使用することがで きます。

T
合計レコードには、通常、いくつかの明細レコードについての特定の演算の 最終結果であるデータが入っています。

E
例外レコードは、演算の実行中に書き出されます。 例外レコードを指定でき るのは、命令コード EXCEPT が使用されている場合だけです。 EXCEPT 命令コードについて詳しくは、779 ページの『EXCEPT (演算時出力)』 を参照してください。

17 桁目は、書き出されるレコードのタイプを識別します。 すべての出力レコードについて 17 桁目の指定が必要です。 見出し (H) および明細 (D) 行は、どちらも明細レコードとして処理されます。 出力レコード

をコーディングする場合には、特別な順序は必要ありませんが、行 はそれらのレコード・タイプに基づいてプログラム・サイクル内の別の時点で処 理されます。 サイクル出力が実行される時点について詳しくは、103 ページの図 10 および 104 ページの図 11 を参照してください。

注 : 制御仕様書に MAIN または NOMAIN が コーディングされている場合には、例外出力だけを実行することができます。

18 から 20 桁目 (レコードの追加/削除)

記入
説明

ADD
ファイルまたはサブファイルにレコードを追加します。

DEL
ファイルから最後に読み取られたレコードを削除します。 削除済みレコード を検索することはできません。レコードはシステムから削除されます。

ADD の指定は入力、出力、または更新ファイルに有効です。 DEL が有効なのは更新 DISK ファイルの場合だけです。 ADD を指定する場合には、対応するファイル仕様書の 20 桁目に A がなければ なりません。

18 から 20 桁目がブランクであった場合には、出力ファイルではレコードが追加され 、更新ファイルではレコードが更新されます。

レコードの追加/削除の指定は、レコード・タイプ (H、D、T、E) の仕様 (17 桁 目) が含まれている同じ行になければなりません。 ADD または DEL の指定に続けて AND/OR 行を使用した場合には、この指定がその AND/OR 行にも適用されます。

表 *102.* ファイルの処理機能

機能	仕様書			
	自由形式 USAGE キーワード	固定形式ファイル記述		出力
		17 桁目	20 桁目	18 から 20 桁目
新しいファイルの作成[1] または既存のファイルへのレコードの追加	USAGE(*OUTPUT)	O O	ブランク A	ブランク ADD
ファイルの処理	USAGE(*INPUT)	I	ブランク	ブランク
ファイルの処理および既存のファイルへのレコードの追加	USAGE(*INPUT : *OUPUT)	I	A	ADD
ファイルの処理およびレコードの更新 (更新または削除)	USAGE(*OUTPUT) または USAGE(*DELETE)	U	ブランク	ブランク
ファイルの処理および既存のファイルへの新しいレコードの追加	USAGE(*UPDATE : *OUTPUT)	U	A	ADD
ファイルの処理およびファイルからの既存のレコードの削除	USAGE(*DELETE : *OUTPUT)	U	ブランク	DEL

注 : RPG では、新しいファイルの作成 という用語は、新たに 作成したファイルへレコードを追加することを意味します。 したがって、この表の 最初の 2 つの項目では同じ機能が実行されます。 その機能の指定方法は 2 つあることを示すために、 両方がリストされています。

18 桁目 (フェッチ・オーバーフロー・ルーチン/解放)

LIKEFILE キーワードが指定されている場合、この項目はブランクでなければなりません。 親ファイルのファイル指定が使用されます。

記入
説明

ブランク
印刷装置ファイル (ファイル仕様書の 36 から 42 桁目に PRINTER が指定さ れたファイル) を除くすべてのファイルの場合に、ブランクでなければなりません。 印刷装置ファイルで 18 桁目がブランクであった場合には、オーバーフローは取 り出されません。

F
フェッチ・オーバーフロー・ルーチン。

R
出力の後で装置 (ワークステーション) を解放します。

フェッチ・オーバーフロー

18 桁目の F は、この行に定義された印刷装置ファイルについてフェッチ・オーバーフロー・ルーチンを指定します。 このファイルは、オーバーフロー行があ る印刷装置ファイルでなければなりません。 フェッチ・オーバーフロー・ルーチ ンが処理されるのは、オーバーフローが起こり、21 から 29 桁目の標識によって指定 されたすべての条件が満たされた場合だけです。 オーバーフロー標識は、フェッチ・オーバーフロー・ルーチンとして同じ行に指定することはできません。

印刷装置ファイルのファイル仕様書で OFLIND キーワードと一緒にオ ーバーフロー標識が指定されなかった場合には、コンパイラーによってこのファ イルに 1 つの標識が割り当てられます。 ファイルに他の出力レコード存在していない場合、または印刷装置で外部記述デ ータが使用される場合を除いて、 このファイルについてオーバーフロー行がコンパイラーによって生成されます。 このコンパイラー生成のオーバーフローを取り出すことができます。

オーバーフロー行は、明細、合計、または例外出力時に書き出すことができま す。 フェッチ・オーバーフロー・ルーチンを指定した場合には、処理される取り出 しが含まれているファイルと関連したオーバーフロー出力だけが出力となります。 フェッチ・オーバーフローの記入項目 (F) の指定が必要なのは、 オーバーフロー・ルーチンを必要とするレコード・タイプの各 OR 行の場合です。 フェッチ・オーバーフロー・ルーチンでは、用紙が自動的に送られることはありません。 オーバーフロー・ルーチンについて詳しくは、111 ページの『オーバーフロー・ルーチン』 および 110 ページの図 13 を参照してください。

用紙の長さおよびオーバーフロー行は、印刷装置ファイルの中でファイル仕様書に FORMLEN および OFLIND キーワードを使用するか、あるいは IBM i 一時変更コマンドによって指定することができます。

リリース

対応する出力仕様の 18 桁目に R を指定した場合には、出力操作が完了した 後に、その操作に使用された装置が解放されます。 装置の解放について詳しくは、861 ページの『REL (解放)』 命令を参照してください。

21 から 29 桁目 (出力条件付け標識)

記入
説明

ブランク
レコード (見出し、明細、合計、または例外) が出力用に検査されるたびに 、行またはフィールドが出力されます。

01 から 99
結果標識、フィールド標識、またはレコード識別標識として使用される一 般標識。

KA から KN、KP から KY
機能キー標識。

L1-L9
制御レベル標識。

H1-H9
停止標識。

U1-U8
プログラムの実行前に、または演算命令の結果として設定される外部標 識。

OA から OG、OV
このファイルに事前に割り当てられているオーバーフロー標識。

MR
突き合わせレコード標識。

LR
最終レコード標識。

RT
戻り標識。

1P
1 ページ目標識。 見出し行または明細行でのみ有効です。

出力行には、条件付け標識は必須ではありません。 条件付け標識が指定されていない場合には、レコードが出力用に検査されるたび にその行が出力されます。 1 つの仕様行には、レコード内の特定のフィールドが書き出される時点を制御す るための標識を 3 つまで入れることができます。 出力を条件付けする 標識は、22 から 23、25 から 26、および 28 から 29 桁目にコーディングされます。 21、24、 または 27 桁目に N を入れる場合には、書き出される行またはフィールドと関連した桁の 標識がオフになっていなければなりません。 そうでない場合には、行またはフィ ールドを書き出すためには標識がオンになっていなければなりません。 出力標識が PAGE フィールドに与える影響については、525 ページの『PAGE、PAGE1 から PAGE7』 を参照してください。

1 つの行に複数の標識を指定した場合には、標識はすべて AND 関係にあると見なされます。

AND 関係の 4 つ以上の標識によって出力レコードを条件付けしなければならない 場合には、次の行の 16 から 18 桁目に英字 AND を入れて、その行の 21 から 29 桁目に 追加の標識を指定してください。

AND 関係の場合には、フェッチ・オーバーフロー (18 桁目) は最初の行にのみ指 定することができます。 40 から 51 桁目 (スペースとスキップ) はすべての AND 行でブランクになっていな ければなりません。

オーバーフロー標識を条件付け標識として使用する場合には、それを OFLIND キーワードによって事前にファイル仕様書で定義しておかなければな りません。 1 つの行をオーバーフロー行として条件付けする場合には、オーバー フロー標識が主要仕様行または OR 行になければなりません。 オーバーフロー標識が AND 行で使用された場合には、その行はオーバーフロー行 としては取り扱われません が、その行が書き出される前にオーバーフ ロー標識が検査されます。 この場合オーバーフロー標識は、他のすべての出力標識と同じように扱われます。

2 つ以上の組の条件のどれか 1 つが存在する場合 (OR 関係) に出力レコード が書き出されるようにする場合には、次の仕様行の 16 から 18 桁目に英字 OR を入 れ、その行に追加の OR 標識を指定してください。

印刷装置ファイルについて OR 行を指定した場合には、スキップとスペースの 指定 (40 から 51 桁目) をすべてブランクとすることができますが、この場合には、 前の行のスペースとスキップの指定が使用されます。 前の行と異なっている場合 には、スペースとスキップの指定を OR 行に入れてください。 フェッチ・オーバーフロー (18 桁目) を使用する場合には、各 OR 行にそれを指 定する必要があります。

30 から 39 桁目 (EXCEPT 名)

レコード・タイプが (17 桁目の E によって指示される) 例外レコードであっ た場合には、レコード行のこれらの桁に名前を入れることができます。 出力され るレコードのグループに割り当てられる名前は、EXCEPT 命令で指定することがで きます。 この名前は EXCEPT 名と呼ばれます。 EXCEPT 名は、XXX の使用に関する規則に従うものでなければなりません。 任意の数の出力レコードによるグループに同じ EXCEPT 名を使用することができ ますが、それらのレコードが連続したレコードである必要はありません。

EXCEPT 名を指定せずに EXCEPT 命令を指定した場合には、EXCEPT 名のない例 外レコードだけが検査されて、条件付け標識が満たされた場合に書き出されます。

EXCEPT 命令で EXCEPT 名を指定した場合には、その名前を持つ例外レコード だけが検査されて、条件付け標識が満たされた場合に書き出されます。

EXCEPT 名は主要レコード行に指定され、すべての AND/OR 行に適用されます。

EXCEPT 名を持つ例外レコードがオーバーフロー標識によって条件付けされた場合、そのレコードは、RPG IV サイクルのオーバーフロー部分またはフェッチ・オーバーフロー時にのみ書き込まれます。 EXCEPT 命令の処理時にはレコードは書き出されません。

フィールドのない EXCEPT 命令を使用して、ファイル中のレコード・ロックを 解放することができます。 UNLOCK 命令もこの目的で使用することができます。 523 ページの図 174 では、ファイル RCDA 内のレコード・ロックは EXCEPT 命令によって解放されます。詳しくは、「*ILE Application Development Example*」*(SC41-5602)* を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C*
C     KEY           CHAIN     RCDA
C                   EXCEPT    RELEASE
ORcdname+++D...N01N02N03Excnam++++.....................................
O
O*
ORCDA      E            RELEASE
O*                    (no fields)
```

図 *174. EXCEPT* 命令によるファイルのレコード・ロックの解放

40 から 51 桁目 (スペースとスキップ)

印刷装置ファイルの行のスペースおよびスキップを指定するためには、40 から 51 桁目を 使用してください。 スペーシングとは一度に 1 行ずつ行を進めること、スキップとはある印刷行から別の印刷行へジャンプすることです。

スペースとスキップを同じ行に指定した場合には、スペースおよびスキップ操 作は次の順序で処理されます。

- 印刷前スキップ
- 印刷前スペース
- 行の印刷
- 印刷後スキップ
- 印刷後スペース

PRTCTL (印刷装置制御オプション) キーワードがファイル仕様書に指定され ていない場合で、装置が PRINTER である場合には、40 から 42 (印刷前 スペース)、43 から 45 (印刷後スペース)、46 から 48 (印刷前スキップ)、または 49 から 51 (印刷後 スキップ) の桁のいずれか 1 つを指定しなければなりません。 スペース/スキップ記入項目をブランクのままにしておいた場合には、ブランク の指定によって特定の機能 (印刷前スペースまたは印刷後スペースなど) は行 われません。 40 から 42 桁目 (印刷前スペース) または 46 から 51 桁目 (印刷前スキッ プおよび印刷後スキップ) を指定し、43 から 45 桁目 (印刷後スペース) を指定しな かった場合には、印刷後にスペースはとられません。 PRTCTL を指定した場合に、 それは 40 から 51 桁目にブランクの指定があるレコードについてのみ使用されます。

新しいページの行について印刷前スキップまたは印刷後スキップを指定しても 、印刷装置がその行にあった場合には、スキップは行われません。

40 から 42 桁目 (印刷前スペース)

記入
説明

0 またはブランク
スペースなし

1 から 255
スペースの値

43 から 45 桁目 (印刷後スペース)

記入
説明

0 またはブランク
スペースなし

1 から 255
スペースの値

46 から 48 桁目 (印刷前スキップ)

記入
説明

ブランク
スキップしません。

1 から 255
スキップの値

49 から 51 桁目 (印刷後スキップ)

記入
説明

1 から 255
スキップの値

フィールド記述および制御項目

これらの記入項目によって、レコードのフィールドが書き出される条件および形 式が決まります。

各フィールドが別個の行に記述されます。 フィールドのフィールド記述および制御情報は、レコード識別行の次の行から始 められます。

21 から 29 桁目 (出力標識)

PAGE 予約フィールドを除き、フィールド記述行に指定された標識によって、 そのフィールドが出力レコードに含まれるかどうかが決まります。 出力標識が PAGE フィールドに与える影響については、525 ページの『PAGE、PAGE1 から PAGE7』 を参照してください。フィールドの制御には、レコードの制御に使用されたものと同じタイプの標識を使用することができます。521 ページの『21 から 29 桁目 (出力条件付け標識)』 を参照してください。 フィールド記述行の条件付けに使用される標識を AND/OR 関係の中で指定するこ とはできません。 条件付け標識は、プログラム記述ワークステーション・ファイルの形式名の仕様 (528 ページの『53 から 80 桁目 (定数、編集語、データ属性、形式名)』 を参照) に指定することはできません。

30 から 43 桁目 (フィールド名)

30 から 43 桁目には以下の記入項目の 1 つを使用して、書き出すフィールドをそ れぞれ指定してください。

- フィールド名
- 53 から 80 桁目に定数を指定する場合にはブランク
- テーブル名、配列名、または配列要素
- 名前のついた定数
- RPG IV 予約語 PAGE、PAGE1 から PAGE7、*PLACE、UDATE、*DATE、UDAY、*DAY、UMONTH、*MONTH、UYEAR、*YEAR、*IN、*INxx、または *IN(xx)
- データ構造名またはデータ構造サブフィールド名

注 : ポインター・フィールドは有効な出力フィールドではありません。すなわち、 ポインター・フィールドを書き出すことはできません。

フィールド名、ブランク、テーブル、および配列

使用するフィールド名はプログラムで定義しておかなければなりません。 53 から 80 桁目に定数または編集語を使用する場合には、フィールド名は記入しないでください。 30 から 43 桁目にフィールド名を入れた場合には、7 から 20 桁目はブランクでなければなりません。

出力レコードに現れるフィールドの順序は 47 から 51 桁目の指定によって決ま るため、フィールドは任意の順序で指定することができます。 フィールドがオー バーラップする場合には、最後に指定されたフィールドだけが完全なフィールド として書き出されることになります。

指標付きでない配列名を指定した場合には、その配列全体が書き出されます。 定数指標または変数指標がある配列名では、1 つの要素が書き出されます。 テーブル名を指定した場合には、798 ページの『LOOKUP (テーブルまたは配列要素の検索)』 命令で最後に見つかった要素が書き出されます。 LOOKUP 命令が正常に実行されなかった場合には、テーブルの最初の要素が書き出 されます。

レコードおよびフィールドに含まれている条件は、フィールドが書き出される 前に満たされていなければなりません。

PAGE、PAGE1 から PAGE7

自動ページ番号付けを使用するためには、30 から 43 桁目に出力フィールドの名 前として PAGE をコーディングしてください。 21 から 29 桁目に指定された標識は、フィールドを 印刷するかどうかではなく、PAGE フィールドの リセットについて条件付けをします。 PAGE フィールドは常に 1 ずつ増やされて、印刷されます。 条件付け標識が満たされた場合には、PAGE フィールドは 1 ずつ増やされる前にゼ ロにリセットされ、印刷されます。 ページ番号が複数の出力ファイルに必要な場合 (または 1 つのファイル内の異なる番号付けの場合) には、PAGE1 から PAGE7 の記入項目を使用することができます。 PAGE フィールドでは、Z 編集コードによって自動的にゼロが消去されます。

予約語 PAGE について詳しくは、75 ページの『特別な機能を持つ RPG IV の用語/予約語』を参照してください。

**PLACE*

*PLACE は、出力レコードのデータを繰り返すために使用される RPG IV の予約語です。 前の仕様行に指定したフィールドまたは定数は、新しい仕様行 にフィールドの名前およびその終了位置を指定しなくとも、出力レコードの中で 繰り返すことができます。 30 から 43 桁目に *PLACE をコーディングした場合には、その出力レコードのフィー ルドについて前に指定された最初の桁と最高位の終了位置の間のすべてのデータ が、*PLACE 仕様行の出力レコードに指定された終了位置に達するまで、反復され ます。 *PLACE 仕様行に指定される終了位置は、複写されるフィールドのグループの最高位の終了位置の少なくとも 2 倍は必要です。*PLACE は任意のタイプの出力に使用することができます。 後で消去 (45 桁目)、編集 (44、53 から 80 桁目)、データ形式 (52 桁目)、およ び相対終了位置は *PLACE と一緒に使用することができません。

ユーザー日付の予約語

ユーザー日付の予約語 (UDATE、*DATE、UDAY、*DAY、UMONTH、*MONTH、 UYEAR、*YEAR) によって、プログラマーは、実行時にプログラムに日付を指定することができます。ユーザー日付の予約語について詳しくは、78 ページの『ユーザー日付に関する規則』を参照してください。

**IN, *INxx, *IN(xx)*

予約語 *IN、*INxx、および *IN(xx) によって、プログラマーは RPG IV 標識をデータとして参照して扱うことができます。

44 桁目 (編集コード)

記入
説明

ブランク
編集コードは使用されません。

1 から 9、A から D、J から Q、X、Y、Z
数値フィールドのゼロは消去され、編集語を使用せずに事前に定義されたパ ターンに従って区切られます。

44 桁目は、数値フィールドの先行ゼロを消去する編集コードを指定するか、 あるいは数値フィールドを編集語を使用しないで区切るために使用されます。 使用できる記入項目は、1 から 9、A から D、J から Q、X、Y、Z、およびブランクです。

注 : 記入項目は、浮動出力フィールドを書き出している場合にはブランクに しておく必要があります。

編集コードについて詳しくは、292 ページの『数値フィールドの編集』を参照してください。

編集コードの 5 から 9 はユーザー定義の編集コードであり、IBM i の機能によって外部で定義されます。 編集コードはコンパイル時に判別されます。 ユーザー定義の編集コードを後から変更しても、プログラムを再コンパイルしない限り、RPG IV コンパイラーによる編集に影響することはありません。

45 桁目 (後で消去)

記入
説明

ブランク
このフィールドはリセットされません。

B
出力操作が完了した後で、30 から 43 桁目に指定されたフィールドがブランクま たはゼロ、あるいはデフォルトの日付/時刻/タイム・スタンプの値にリセットされます。

45 桁目は、数値フィールドをゼロに、または文字フィールド、図形フィールド、 または UCS-2 フィールドをブランクにリセットするために使用 されます。 日付、時刻、およびタイム・スタンプ・フィールドはデフォルトの値にリセットされます。

フィールドが 21 から 29 桁目の標識によって条件付けされている場合には、後で消 去も条件付けされます。 先読み、ユーザー日付の予約語、*PLACE、名前付き定数、およびリテラルの 場合には、この桁はブランクでなければなりません。

フィールドのゼロへのリセットは、合計を累算して、プログラム内の各制御グループごとに書き出す時の合計出力に有用な場合があります。 合計が累算されて 1 つの制御グループについて書き出された後、次の制御グルー プの合計について累算が始まる前に合計フィールドをゼロにリセットすることが できます。

2 回以上書き出されるフィールドに後で消去 (45 桁目) を指定した場合には、そ のフィールドの出力を指定する最後の行に B を記入する必要があります。そうし ないと、後で消去が一度実行された後では、指定したフィールドのすべての行に 後で消去の値があるものとして印刷されてしまいます。

47 から 51 桁目 (終了位置)

記入
説明

1 から n
終了位置

K1-K10
ワークステーション・ファイルの形式名の長さ

47 から 51 桁目は、出力レコードにおけるフィールドまたは定数の終了位置 、あるいはプログラム記述 ワークステーション・ファイルのデータ記述仕様レコード様式名 の長さを定義します。

K はその指定を終了位置ではなく長さとして識別するもので、K の後の数によ ってレコード様式名の長さが指示されます。 例えば、形式名が CUSPMT であった場合には、50 から 51 桁目の指定は K6 となり ます。 K の後には先行ゼロが許されていますが、右寄せで指定しなければなりません。

終了位置として有効な指定は、ブランク、+nnnn、-nnnn、および nnnnn です。 これらの桁での指定は、すべて 51 桁目で終っていなければなりません。 フィールドまたは定数の右端の文字の位置を入れてください。 終了処置がファイルのレコード長を超えることはできません。

配列全体を書き出す場合には、47 から 51 桁目に配列内の最後の要素の終了位置 を記入してください。 配列を編集する場合には、編集済みのすべての要素を十分 に書き出せる桁数となるように、終了位置を慎重に指定してください。 各要素は 、編集コードまたは編集語に従って編集されます。

+nnnn または -nnnn の記入項目は、前のフィールドの終了位置との相対関係 でのフィールドまたは定数の配置を指定します。 数字 (nnnn) は右寄せしなければなりませんが、先行ゼロは不要です。 符号は、この記入項目フィールドの数字の左側の任意に位置に入れてください。 終了位置の計算には次の式を使用します。

```
EP = PEP +nnnn + FL
```

```
EP = PEP -nnnn + FL
```

EP は計算された終了位置です。 PEP は前の終了位置です。 レコード内の最初のフィールド仕様の場合には、PEP はゼロと等しくなります。 FL は編集後のフィールドの長さ、またはこの仕様に指定した定数の長さです。 +nnnn を使用することは、フィールドの間に nnnn 桁を入れることと同じです。 -nnnn では、フィールドが nnnn 桁だけオーバーラップします。 例えば、前の終了位置 (PEP) が 6、フィールドの間に入れる桁数 (nnnn) が 5 で、 フィールド長 (FL) が 10 であるとすると、終了位置 (EP) は 21 になります。

*PLACE を使用した場合には、実際の終了位置を指定しなければならず、ブラ ンクまたは変位とすることはできません。

ブランクの指定は +0000 の指定として取り扱われます。 フィールドが桁数に よって分けられることはありません。

52 桁目 (データ形式)

記入

説明

ブランク

- 数値フィールドの場合は、データはゾーン 10 進数形式で書き出されます。
- 浮動数値フィールドの場合、データは、外部表示表現で書き出されます。
- 図形フィールドの場合、データは SO/SI ブラケットを付けて書き出されます。
- UCS-2 フィールドの場合、データは UCS-2 形式で書き出されます。
- 日付、時刻、およびタイム・スタンプ・フィールドの場合は、データは形式を変 換しないで書き出されます。
- 文字フィールドの場合、データは保管されているとおりに書き出されます。

A

文字フィールドは、*VAR データ属性の有無に従って、 固定長形式または可変長形式で 書き出されます。

C

UCS-2 フィールドは、*VAR データ属性の有無に従って、固定長形式 または可変長形式で書き出されます。

G

図形フィールド (SO/SI 大括弧がない) は、*VAR データ属性 の有無に従って、固定長形式または可変長形式で 書き出されます。

B

数値フィールドは、2 進-10 進形式で書き出されます。

F

数値フィールドは、浮動形式で書き出されます。

I
数値フィールドは、整数形式で書き出されます。

L
数値フィールドは、先行する (左側の) プラスまたはマイナス符号と 一緒にゾーン 10 進数形式で書き出されます。

N
文字フィールドは標識形式で書き出されます。

P
数値フィールドは、パック 10 進数形式で書き出されます。

R
数値フィールドは、後書きの (右側の) プラスまたはマイナス符号と一緒に ゾーン 10 進数形式で書き出されます。

S
数値フィールドは、ゾーン 10 進数形式で書き出されます。

U
数値フィールドは、符号なし整数形式で書き出されます。

D
日付フィールド ― 日付フィールドは 53 から 80 桁目に指定された形式またはデフォルトのファイル日付形式に変換されます。

T
時刻フィールド ― 時刻フィールドは 53 から 80 桁目に指定された形式またはデフォルトのファイル時刻形式に変換されます。

Z
タイム・スタンプ・フィールドだけに有効。

編集を指定した場合には、この桁はブランクでなければなりません。

52 桁目の記入項目は、ファイル内のレコードの中のデータの 外部形式を 指定します。 この指定によって、プログラムでの出力フィールドの内部処理に使用される形式 が影響を受けることはありません。

数値フィールドの場合には、出力レコード内に必要なバイト数は、次の形式によ って異なります。 例えば、5 桁の数値フィールドが必要であった場合には、次 のようになります。

- ゾーン形式で書き出される場合は 5 バイト
- パック形式で書き出される場合は 3 バイト
- L か R のいずれかの形式で書き出される場合は 6 バイト
- 2 進数形式で書き出される場合は 4 バイト
- I か U のいずれかの形式で書き出される場合は 2 バイト。 値が 2 バイトの整 数または符号なしフィールドの最大値より大きい場合には、これによって実行時 にエラーが起こることがあります。 5 桁のフィールドの場合には、2 進-10 進形式がより適しています。

  データ形式記入項目をブランクにして書き出された浮動数値フィールド は、出力レコードのうちの 14 桁または 23 桁 (それぞれ 4 バイトおよび 8 バイト 浮動フィールドの場合) を占めます。

プログラム記述ファイル中の図形フィールドについては、「G」またはブランク を指定しなければなりません。 「G」が指定された場合には、データは SO/SI なしに出力されます。 プログラム記述出力の場合にこの欄がブランクであった場合には、フィールドの タイプが図形であれば、コンパイラーによって出力レコード中のフィールドの前 後に SO/SI の対が入れられます。 出力レコードにデータと SO/SI 文字の両方を入れる十分な余地があることを確認 しておかなければなりません。

53 から 80 桁目 (定数、編集語、データ属性、形式名)

53 から 80 桁目は、プログラム記述ファイルについて、 定数、 編集語、 データ属性、 または 形式名を 指定するために使用されます。

定数

定数は、プログラムのある処理から次の処理の間で変更されない文字デー タ (リテラル) から構成されます。 定数とは、データの位置を表す名前ではなく、出力レコードで使用される 実際のデータのことです。

定数は 53 から 80 桁目に入れることができます。 定数は、54 桁目 (ア ポストロフィは 53 桁目) から始め、数値だけを含む場合であってもアポストロ フィで終わらせなければなりません。 定数の中でアポストロフィが使用され る場合には、それを 2 回指定しなければなりませんが、その定数が書き出さ れる場合には 1 つのアポストロフィだけが現れます。 フィールド名 (30 から 43 桁 目) はブランクでなければなりません。 定数は継続させることができます (継続の規則については、313 ページの『継続の規則』を参照)。 定数を指定する代わりに、名前付き定数を使用することができます。

図形リテラルおよび UCS-2 リテラルまたは名前付き定数を編集語として使用することはできませんが、定数として指定することはできます。

編集語

編集語は、円記号、コンマ、ピリオド、および符号状況の印刷を含めて、数値 フィールドの句読点を指定します。 詳しくは、300 ページの『編集語の各部分』を参照してください。

編集語は文字リテラルまたは名前付き定数でなければなりません。 図形リテラル、UCS-2 リテラル、または 16 進数リテラルと名前付き定数は使用できません。

データ属性

データ属性は、日付、時刻、または可変長の文字フィールド、図形フィールド、また は UCS-2 フィールドの場合の外部形式を指定します。

日付および時刻データの場合、形式を指定しないと、ファイルについて指定された 形式/区切り記号 (DATFMT か TIMFMT のいずれか、あるいはその両方) が使用されます。 ファイルについて指定された外部の日付または時刻の形式がない場合には、 エラー・メッセージが出されます。 有効な日付および時刻の形式については、274 ページの表 71 および 276 ページの表 74 を参照してください。

文字データ、図形データ、および UCS-2 データの場合、可変長出力フィールドを指定するために、*VAR データ属性を使用します。 文字データ、図形データ、および UCS-2 データのこの記入項目がブランクの場合、外部形式は固定長になります。可変長フィールドについて詳しくは、253 ページの『可変長の文字形式、図形形式および UCS-2 形式』を参照してください。

注 : 出力レコードに占めるバイト数は指定された形式によって異なります。 例えば、*MDY 形式で書き出される日付には 8 バイトが必要ですが、 *ISO 形式で書き出される日付には 10 バイトが必要です。

外部形式について詳しくは、247 ページの『内部形式および外部形式』を参照してください。

レコード様式名

プログラム記述 ワークステーション・ファイルによって使用されるデータ記述仕様レコード様式の名前は 53 から 62 桁目に指定しなければなりません。 ワークステーション・ファイルの 各出力レコードに 1 つの形式名が必要ですが、1 つのレコードに複数の形式名を 指定することはできません。 プログラム記述 ワークステーション・ファイルの形式名仕様に条 件付け標識を指定することはできません。 形式名はアポストロフィで囲まなければ なりません。 47 から 51 桁目に Kn も入れなければなりません。この場合の n は形式名の長さです。例えば、形式名が「CUSPMT」である場合には、50 から 51 桁目に K6 を入れてください。 名前付き定数も使用することができます。

外部記述ファイル

6 桁目 (仕様書コード)

6 桁目には、この行を出力仕様ステートメントとして識別する O がなけ ればなりません。

レコード識別および制御項目

外部記述ファイルの場合には、出力仕様は任意指定です。 レコード識別行の 7 から 39 桁目の指定によって、レコード様式が識別され、レコ ードが書き出される条件が決定されます。

7 から 16 桁目 (レコード名)

記入
説明

有効なレコード様式名
外部記述ファイルにはレコード様式名を指定しなければなりません。

16 から 18 桁目 (論理関係)

記入
説明

AND または OR
AND/OR は、出力標識の行相互間の関係を指示します。 AND/OR 行は出力レコードには有効ですが、フィールドには有効でありません。

詳しくは、519 ページの『16 から 18 桁目 (論理関係)』 を参照してください。

17 桁目 (タイプ)

記入
説明

H または D
明細レコード

T
合計レコード

E
例外レコード

17 桁目は、書き出されるレコードのタイプを識別します。 詳しくは、519 ページの『17 桁目 (タイプ)』 を参照してください。

18 桁目 (解放)

記入
説明

R
出力後に装置が解放されます。

詳しくは、521 ページの『リリース』 を参照してください。

18 から 20 桁目 (レコードの追加)

記入
説明

ADD
ファイルにレコードを追加します。

DEL
ファイルから既存のレコードを削除します。

レコードの追加について詳しくは、520 ページの『18 から 20 桁目 (レコードの追加/削除)』 を参照してください。

21 から 29 桁目 (出力標識)

外部記述ファイルの出力標識は、プログラム記述ファイルの場合と同じ方法で指定されます。 外部記述ファイルの場合には、オーバーフロー標識 OA から OG、OV は正しくありません。出力標識について詳しくは、521 ページの『21 から 29 桁目 (出力条件付け標識)』 を参照してください。

30 から 39 桁目 (EXCEPT 名)

EXCEPT 名は例外レコード行のこれらの桁に指定することができます。 詳しくは、522 ページの『30 から 39 桁目 (EXCEPT 名)』 を参照してください。

フィールド記述および制御項目

外部記述ファイルの場合に有効なフィールド記述は、出力標識 (21 から 29 桁目)、 フィールド名 (30 から 43 桁目)、および後で消去 (45 桁目) だけです。

21 から 29 桁目 (出力標識)

フィールド記述行に指定された標識によって、そのフィールドが出力レコード に含まれるかどうかが決まります。 フィールドの制御には、レコードの制御に使用されたものと同じタイプの標識を 使用することができます。 詳しくは、521 ページの『21 から 29 桁目 (出力条件付け標識)』 を参照してください。

30 から 43 桁目 (フィールド名)

記入
説明

有効なフィールド名
外部記述ファイルについて指定するフィールド名は、その外部名がプログラ ムに合わせて変更される場合を除いて、外部記述の中になければなりません。

***ALL**
すべてのフィールドがレコードに含まれることを指定します。

外部記述ファイルの場合には、指定されたフィールドのみが出力レコードに含まれます。*ALL を指定して、すべてのフィールドをレコードに含めることができます。 *ALL を指定した場合には、そのレコードについて他のフィールド記述行を 指定することはできません。 とくに、45 桁目に B (後で消去) を指定することは できません。

更新レコードの場合には、出力フィールド仕様に指定されたフィールドの中で 出力標識によって指定された条件を満たすものだけが、再書き出しされる出力レコ ードに入れられます。 他のすべてのフィールドの再書き出しには、前に読み取られている値が使用されます。

新しいレコードの作成 (18 から 20 桁目に ADD の指定) の場合には、指定されたフ ィールドが出力レコードに入れられます。 指定されていないフィールド、または出力標識によって指定された条件を満たし ていないフィールドは、外部記述に指定されたデータ形式に従って、ゼロまたは ブランクとして書き出されます。

45 桁目 (後で消去)

記入
説明

ブランク
このフィールドはリセットされません。

B
出力操作が完了した後で、30 から 43 桁目に指定されたフィールドがブランクま たはゼロ、あるいはデフォルトの日付/時刻/タイム・スタンプの値にリセットされます。

45 桁目は、数値フィールドをゼロに、または文字フィールド、図形フィールド、 または UCS-2 フィールドをブランクにリセットするために使用 されます。 日付、時刻、およびタイム・スタンプ・フィールドはデフォルトの値にリセットされます。

フィールドが 21 から 29 桁目の標識によって条件付けされている場合には、後で消 去も条件付けされます。 先読み、ユーザー日付の予約語、*PLACE、名前付き定数、およびリテラルの 場合には、この桁はブランクでなければなりません。

フィールドのゼロへのリセットは、合計を累算して、プログラム内の各制御グループごとに書き出す時の合計出力に有用な場合があります。 合計が累算されて 1 つの制御グループについて書き出された後、次の制御グルー プの合計について累算が始まる前に合計フィールドをゼロにリセットすることが できます。

2 回以上書き出されるフィールドに後で消去 (45 桁目) を指定した場合には、そ のフィールドの出力を指定する最後の行に B を記入する必要があります。そうし ないと、後で消去が一度実行された後では、指定したフィールドのすべての行に 後で消去の値があるものとして印刷されてしまいます。

プロシージャー仕様書

プロシージャー仕様書を使用して、メイン・ソース・セクションの後に 指定されるプロトタイプ・プロシージャーを定義します。定義しない場合、プロトタイプ・プロシージャーはサブプロシージャーとして認識されます。

サブプロシージャーのプロトタイプは、サブプロシージャー定義が含まれている モジュールのメイン・ソース・セクションに定義できます。 プロトタイプが指定されないと、プロシージャー・インターフェースの情報を使用して暗黙的にプロトタイプが定義されます。 プロシージャー・インターフェースも定義されない場合、 戻り値もパラメーターもないデフォルトのプロトタイプが暗黙的に定義されます。

サブプロシージャーには以下のものがあります。

1. プロシージャー開始ステートメント。(自由形式で DCL-PROC 命令コードを使用するか、固定形式プロシージャー仕様書の 24 桁目に B を指定します)。
2. プロシージャー・インターフェース定義。これは、 戻り値およびパラメーターがあった場合に、それらを 指定します。 サブプロシージャーから値が戻されず、また、そのサブプロシージャ ーに渡されるパラメーターがない場合には、プロシージャー・インターフェースの定義はオプションです。 プロシージャー・インターフェースは、プロトタイプが指定されていれば、対応するプロトタイプに一致していなければなりません。
3. 他の定義 (サブプロシージャーで必要とされる、ファイル、変数、定数、およびプロトタイプなど)。 これらの定義はローカル定義です。
4. プロシージャーのタスクを実行するためには、任意の演算仕様書が必要です。 サブプロシージャーの中に組み込まれたサブルーチンは、いずれもローカルです。 それらをサブプロシージャーの外で使用することはできません。 サブプロシージャーから値が戻される場合には、そのサブプロシージャーに RETURN 命令がコーディングされていなければなりません。 プロシージャーの終わりに達する前に RETURN 命令が実行されることを確認するこ とが必要です。
5. プロシージャー終了ステートメント (自由形式で DCL-PROC 命令コードを使用するか、 固定形式プロシージャー仕様書の 24 桁目に E を指定します)。

プロシージャー・インターフェース定義は他のローカル定義内のどこにでも入れることができますが、 それ以外のサブプロシージャーは上記の順序でコーディングする必要があります。

サブプロシージャーの例については、93 ページの図 7 を参照してください。

自由形式のプロシージャー・ステートメント

自由形式のプロシージャー開始ステートメントは、DCL-PROC で始まり、 その後にプロシージャー名が続き、さらにその後にキーワード が続き、最後はセミコロンで終わります。 プロシージャーのプロトタイプがなく、 EXTPROC キーワードのプロシージャー名パラメーターに *DCLCASE が 指定されている場合、 プロシージャーの外部名は、DCL-PROC ステートメントに指定された名前と 同じであり、大/小文字も同じです。

自由形式のプロシージャー終了ステートメントは、END-PROC で始まり、 その後に任意指定のプロシージャー名が続き、 最後はセミコロンで終わります。 名前が指定される場合は、プロシージャー開始ステートメントに指定された 名前と同じでなければなりません。

自由形式のプロシージャー・ステートメントの内部で使用を許可されている指示は、 /IF、/ELSEIF、/ELSE、および /ENDIF のみです。

```
DCL-PROC getCustName
    /IF DEFINED(EXPORT_ALL_PROCEDURES)
            EXPORT
    /ENDIF
            ;
```

プロシージャー・ステートメントの例

- 次の例では、END-PROC ステートメントには名前は指定されていません。

  ヒント: プロシージャーが大規模であり、DCL-PROC ステートメント と END-PROC ステートメントの両方をエディターで一緒に表示できない場合、 END-PROC ステートメントに名前を指定すると役立つことがあります。

```
DCL-PROC cleanup;
   CLOSE *ALL;
   UNLOCK *ALL;
   deleteTempUsrspc();
END-PROC;
```

- 次の例では、プロシージャー *getNextOrder* がプロトタイプなし で定義されています。 プロシージャー・インターフェースは EXTPROC(*DCLCASE)2を指定しています。したがって、プロシージャーの外部名は、DCL-PROC ステートメント 1 に指定された名前とまったく同じ「getNextOrder」です。1.

  注: 実行時に、外部名は、プロシージャーに誤りがあった場合にジョブ・ログ内に 示されるか、または、ジョブを表示したときにプログラム・スタック内に示されます。

```
DCL-PROC getNextOrder;            1
   DCL-PI *N IND
             EXTPROC(*DCLCASE);   2
      order LIKEDS(order_t);
   END-PI;

   DCL-F orders STATIC;

   READ orders order;

   RETURN %EOF(orders);
END-PROC getNextOrder;
```

従来型のプロシージャー仕様書ステートメント

プロシージャー仕様書の一般的なレイアウトは次のとおりです。

- 6 桁目にはプロシージャー仕様書コード (P) が入れられます。
- 仕様書の注記以外の部分は 7 から 80 桁目です。
  - 固定形式の記入項目は 7 から 24 桁目です。
  - キーワードの記入項目は 44 から 80 桁目です。
- 仕様書の注記部分は 81 から 100 桁目です。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+...
9 ...+... 10
PName++++++++++..B..................Keywords+++++++++++++++++++++++++++++Comments++++++++++
++
```

図 *175.* プロシージャー仕様書のレイアウト

プロシージャー仕様書のキーワード継続記入行

自由形式構文	自由形式ステートメントで使用可能な桁については、307 ページの『自由形式ステートメント』を参照してください。

キーワードに追加のスペースが必要な場合には、 次のようにキーワード・フィールドを後続の行に継続させることができます。

- 継続記入行の 6 桁目には P が入っていなければなりません。
- 継続記入行の 7 から 43 桁目はブランクでなければなりません。
- 指定は 44 桁目以降から継続されます。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+...
9 ...+... 10
P.....................................Keywords+++++++++++++++++++++++++++++Comments++++++++++
++
```

図 *176.* プロシージャー仕様書のキーワード 継続記入行のレイアウト

プロシージャー仕様書の継続名前行

自由形式構文	名前の最後の部分に省略符号 (...) をコーディングしないでください。532 ページの『自由形式のプロシージャー・ステートメント』を参照してください。 自由形式ステートメントで使用可能な桁については、307 ページの『自由形式ステートメント』を参照してください。

15 文字までの名前は、プロシージャー仕様書の名前記入項目 に指定することができるので継続は必要ありません。 部分名の終わりに省略符号 (...) をコーディングすると、任意の名前を (15 文字以下の名前であっても) 複数行に継続できます。 名前定義は、次の部分から構成されます。

1. ゼロまたはそれ以上の継続名前行。 継続名前行は、その記入項目 中の最後の非ブランク文字として省略記号を持つものとして識別されます。 名前 は、7 から 21 桁目の中で開始する必要があり、77 桁目まで (80 桁目で 終了する省略記号を付けて) の任意の位置で終了することができます。 名前の開始と 省略記号の間にはブランクを挿入することはできません。 これらの 条件のいずれかが真とならない場合、その行は主要プロシージャー名行で あると解析されます。
2. 名前、開始/終了プロシージャー、およびキーワードを含んでいる 主要プロシージャー名行。 継続名前行がコーディングされた場合、主要プロシージャー名行の名前記入項目はブランクのままになる場合があります。
3. ゼロまたはそれ以上のキーワード継続記入行。

```
*.. 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+...
9 ...+... 10
PContinuedName+++++++++++++++++++++++++++++++++++++++++++++++++++++++++Comments++++++++++
++
```

図 *177.* プロシージャー仕様書の 継続名前行のレイアウト

6 桁目 (仕様書コード)

自由形式構文	DCL-PROC または END-PROC 命令コード。532 ページの『自由形式のプロシージャー・ステートメント』を参照してください。

プロシージャー仕様書にはこの桁に P を入れなければなりません。

7 から 21 桁目 (名前)

自由形式構文	532 ページの『自由形式のプロシージャー・ステートメント』を参照してください。

記入
説明

名前
定義するサブプロシージャーの名前。

定義中のサブプロシージャーの名前を指定するためには、7 から 21 桁目を使用して ください。 その名前が 15 文字より長い場合、継続名前行の 7 から 80 桁目に名前が 指定されます。 RPG IV に関する通常の規則が適用されます。予約語を使用することはできません (73 ページの『記号名』を参照)。 名前は、指定されたスペースのどの桁からでも始めることができます。

指定する名前は、プロトタイプが指定されていれば、そのプロシージャーを記述するプロトタイプの名前と同じでなければなりません。 プロトタイプが指定されないと、プロシージャー仕様書で指定された名前と、プロシージャー・インターフェースで指定された情報を使用して暗黙的に定義されます。

24 桁目に E が含まれる場合には、名前は任意指定です。

24 桁目 (プロシージャーの始め/終わり)

自由形式構文	• プロシージャーを開始するには、DCL-PROC 命令コードを指定します。 • プロシージャーを終了するには、END-PROC 命令コードを指定します。 532 ページの『自由形式のプロシージャー・ステートメント』を参照してください。

記入
説明

B
この指定は、定義中のサブプロシージャーの始めをマークします。

E
この指定は、定義中のサブプロシージャーの終わりをマークします。

サブプロシージャーのコーディングは、少なくともプロシージャーの始めの指定 とプロシージャーの終わりの指定から構成されます。 これらのプロシージャーの指定の間に、そのサブプロシージャーに関するその他 の定義および演算だけでなく、任意のパラメーターおよび戻り値が指定されます。

44 から 80 桁目 (キーワード)

自由形式構文	自由形式ステートメントで使用可能な桁については、307 ページの『自由形式ステートメント』を参照してください。

44 から 80 桁目は、プロシージャー仕様書のキーワードのために用意されています。 キーワードを指定することができるのは、プロシージャーの始めの指定 (24 桁目 に B) だけです。

プロシージャー仕様書のキーワード

EXPORT

EXPORT キーワードの指定によって、プログラム内の別のモジュールからプロシー ジャーを呼び出すことができます。 7 から 21 桁目の名前が大文字形式でエクスポートされます。

注 : プロシージャー名は IMPORT キーワードによってはインポートされません。 プロシージャー名は、プロシージャーへの結合呼び出しを 行ったり、プロシージャー名を使用してプロシージャー・ポインターを初期化 する、プログラム内の任意のモジュールによって暗黙的にインポートされます。

EXPORT キーワードを指定しない場合には、プロシージャーをモジュールの中から だけ呼び出すことができます。

PGMINFO(*YES | *NO)

PGMINFO キーワードを使用すれば、モジュールの作成時にプロシージャーへのインターフェースをプログラム・インターフェース情報に組み込むかどうかを制御することができます。

プロシージャーのプロシージャー仕様書に PGMINFO キーワードを指定しないと、モジュールを作成するときに、メイン・プロシージャーおよび Java ネイティブ・メソッドではないすべてのエクスポートされたサブプロシージャーに対してプログラム・インターフェース情報が生成されます。

***YES**
モジュール内の 1 つ以上のプロシージャーに対して PGMINFO(*YES) が指定されると、PGMINFO(*YES) が指定されていないプロシージャーに対してはプログラム・インターフェース情報は生成されません。 モジュール内にサイクル・メイン・プロシージャーがある場合は、メイン・プロシージャーに対してはプログラム・インターフェース情報は生成されません。

***NO**
モジュール内の 1 つ以上のプロシージャーに対して PGMINFO(*NO) が指定されると、PGMINFO キーワードを持っていないプロシージャーに対してのみプログラム・インターフェース情報が生成されます。 モジュール内にサイクル・メイン・プロシージャーがある場合は、メイン・プロシージャーに対してプログラム・インターフェース情報が生成されます。

注 :

1. PGMINFO キーワードは、モジュールが作成されるときのみ有効です。 プログラムが作成されるときは、メイン・プロシージャーに対してのみプログラム・インターフェース情報が生成されます。 プログラムの作成時にメイン・プロシージャーに対してプログラム・インターフェース情報が生成されないようにしたい場合は、制御ステートメント内で PGMINFO(*NO) キーワードを使用すると、プログラム・インターフェース情報が生成されなくなります。 下記の例を参照してください。
2. モジュール内のすべての PGMINFO キーワードは同じ値 (*YES または *NO のいずれか) を持っていなければなりません。
3. モジュール内にサイクル・メイン・プロシージャーがある場合は、エクスポートされるサブプロシージャーの 1 つ以上で PGMINFO(*YES) を指定することによって、モジュールの作成時にメイン・プロシージャーに対してプログラム・インターフェース情報が生成されないようにすることができます。
4. コマンドの PGMINFO パラメーターでも、モジュールの制御ステートメント内の PGMINFO キーワードでも、プログラム・インターフェース情報を要求しない場合は、PGMINFO キーワードは無視されます。
5. PGMINFO(*NO) を使用してプログラム・インターフェース情報が一部のプロシージャーに指定されないようにする場合は、プログラム・インターフェース情報が生成されていないプロシージャー (Java ネイティブ・メソッドなど) や、エクスポートされないプロシージャーに対して PGMINFO(*NO) を指定する必要はありません。

例

1. 次の例では、PROC2 には PGMINFO(*YES) キーワードが指定されていないので、プログラム・インターフェース情報はプロシージャー PROC1 と PROC3 に対してのみ生成されます。

```
CTL-OPT PGMINFO(*PCML : *MODULE);
CTL-OPT NOMAIN;

DCL-PROC PROC1 EXPORT PGMINFO(*YES);
END-PROC;

DCL-PROC PROC2 EXPORT;
END-PROC;

DCL-PROC PROC3 EXPORT PGMINFO(*YES);
END-PROC;
```

2. 次の例では、PROC2 には PGMINFO(*NO) キーワードが指定されているので、プログラム・インターフェース情報はプロシージャー PROC1 と PROC3 に対してのみ生成されます。

```
CTL-OPT PGMINFO(*PCML : *MODULE);
CTL-OPT NOMAIN;

DCL-PROC PROC1 EXPORT;
END-PROC;

DCL-PROC PROC2 EXPORT PGMINFO(*NO);
END-PROC;

DCL-PROC PROC3 EXPORT;
END-PROC;
```

3. 次の例では、コマンドによって PGMINFO(*NO) が指定されると、プロシージャー PROC1 および PROC3 に対して PGMINFO(*YES) が指定されても、プログラム・インターフェース情報は生成されません。

```
CTL-OPT NOMAIN;

DCL-PROC PROC1 EXPORT PGMINFO(*YES);
END-PROC;

DCL-PROC PROC2 EXPORT;
END-PROC;

DCL-PROC PROC3 EXPORT PGMINFO(*YES);
END-PROC;
```

4. 次の例は、メイン・プロシージャーに対してプログラム・インターフェース情報が生成されないようにする方法を示しています。

   a. 制御仕様書キーワード PGMINFO(*NO) は、プログラムの作成時にプログラム・インターフェース情報が作成されないようにします。

   b. プロシージャー仕様書キーワード PGMINFO(*NO) は、モジュールの作成時にメイン・プロシージャー *myPgm* に対してプログラム・インターフェース情報が生成されないようにします。

```
CTL-OPT MAIN(myPgm);
/IF DEFINED(*CRTBNDRPG)
   CTL-OPT PGMINFO(*NO);    a
/ENDIF

DCL-PROC myPgm PGMINFO(*NO); b
END-PROC;

DCL-PROC PROC1 EXPORT PGMINFO(*NO);
END-PROC;

DCL-PROC PROC2 EXPORT;
END-PROC;
```

5. 次の例では、サブプロシージャーの 1 つに対して PGMINFO(*YES) が指定されているので、サイクル・メイン・プロシージャーに対してはプログラム・インターフェース情報が生成されません。

```
CTL-OPT PGMINFO(*PCML : *MODULE);

// Cycle main procedure
RETURN:

// Subprocedures
DCL-PROC PROC1 EXPORT PGMINFO(*YES);
END-PROC;
```

REQPROTO(*NO)

REQPREXP 制御キーワードまたは REQPREXP コマンド・パラメーターによって、エクスポートされるプロシージャーのプロトタイプがない場合に警告またはエラーを発行することが指示されていても、エクスポートされるプロシージャーにキーワード REQPROTO(*NO) を指定することで、プロシージャーにプロトタイプが必要ないことを指示することができます。

***NO**
REQPROTO(*NO) の指定時にプロシージャーにプロトタイプがない場合には、警告もエラーも出されません。

REQPROTO キーワードは、エクスポートされたプロシージャーの場合にのみ意味を持ちますが、どのプロシージャーに対しても使用できます。

注 : このキーワードは、サイクル・メイン・プロシージャーのプロシージャー・インターフェースでも使用できます。 476 ページの『REQPROTO(*NO)』を参照してください。

例

次に例を示します。

1. REQPREXP(*REQUIRE) が制御ステートメントに指定されています。これは、エクスポートされたプロシージャーにプロトタイプを指定する必要があることを示しています。
2. プロシージャー P1 および P2 にプロトタイプが指定されています。 プロシージャー P3 および P4 にはプロトタイプが指定されていません。
3. キーワード EXPORT および REQPROTO(*NO) が P1 に指定されています。

   キーワード REQPROTO(*NO) は許可されていますが、P1 にプロトタイプが指定されているので、これは必要ありません。 P1 ではコンパイル・エラーは出されません。
4. P2 にはキーワード EXPORT が指定されています。 P2 にプロトタイプが指定されているため、P2 に対してコンパイル・エラーは出されません。
5. キーワード EXPORT および REQPROTO(*NO) が P3 に指定されています。

P3 にはプロトタイプが指定されていませんが、REQPROTO(*NO) はプロトタイプが必要でないことを指示するので、P3 ではコンパイル・エラーは出されません。

6. P4 に対してキーワード EXPORT は指定されていますが、キーワード REQPROTO(*NO) は指定されていません。
7. P4 に対してプロトタイプが指定されていないため、P4 ではコンパイル・エラーが発行されます。

   そのエラー・メッセージは「外部プロシージャーおよび外部プログラムにはプロトタイプが指定されていません。REQPREXP(*REQUIRE) が指定されています。(No prototype is specified for an external procedure or program, and REQPREXP(*REQUIRE) is specified.)」というものです。

```
CTL-OPT REQPREXP(*REQUIRE);       // 1
DCL-PR P1 END-PR;                 // 2
DCL-PR P2 END-PR;                 // 2

DCL-PROC P1 EXPORT REQPROTO(*NO); // 3
END-PROC;

DCL-PROC P2 EXPORT;               // 4
END-PROC;

DCL-PROC P3 EXPORT REQPROTO(*NO;  // 5
END-PROC;

DCL-PROC P4 EXPORT;               // 6
===> Error:                      // 7
END-PROC;
```

SERIALIZE

SERIALIZE キーワードを並行スレッド・モジュールで指定すると、プロシージャーでスレッドを常に 1 つだけ実行することができます。 あるスレッドがプロシージャーで実行されているときに、他のスレッドがそのプロシージャーを呼び出した場合、最初のスレッドがプロシージャーで実行されなくなるまで、2 番目のスレッドはプロシージャーの実行を待機します。 あるスレッドがプロシージャーで実行されているときに、そのプロシージャーに対する再帰呼び出しが実行された場合、そのスレッドがプロシージャーに対するすべての再帰呼び出しから戻らない限り、別のスレッドがそのプロシージャーで実行されることはありません。

制御仕様書で THREAD(*CONCURRENT) が指定されている場合にのみ、SERIALIZE キーワードを使用することができます。

あるプロシージャーに対して SERIALIZE を指定することは、制御仕様書で THREAD(*SERIALIZE) を指定することに似ています。 ただし、異なる点もあります。 制御仕様書で THREAD(*SERIALIZE) を指定した場合は、複数のスレッドからモジュール内のすべてのプロシージャーに対するアクセスが制限されるのに対し、SERIALIZE キーワードをプロシージャーに指定した場合は、そのプロシージャーへのアクセスのみが制限されます。

SERIALIZE キーワードが指定された複数のプロシージャーが、あるモジュール内に存在する場合、それらのプロシージャーは独立して実行されます。 逐次化されたプロシージャーでスレッドを実行しながら、同じモジュール内の別の逐次化されたプロシージャーで別のスレッドを実行することができます。 例えば、同じモジュール内のプロシージャー PROCA と PROCB の両方に SERIALIZE キーワードが指定されている場合、あるスレッドで PROCA で実行しながら別のスレッドで PROCB を実行することができます。シリアライズされたプロシージャーの使用について詳しくは、346 ページの『THREAD(*CONCURRENT | *SERIALIZE)』を参照してください。

命令、式、および関数

データを操作するために、命令コード、式、および組み込み関数が使用されます。

この部では、データまたは装置を取り扱うことができる各種の方法について説明 します。 主要なトピックは次のとおりです。

- 命令コードまたは組み込み関数を使用して実行できる命令
- 式およびそれらに適用される規則
- 組み込み関数
- 命令コード

操作

RPG IV プログラミング言語によって、ユーザーのデータにさまざまなタイプの多くの命令を実行することができます。 命令を実行するには、命令コードまたは組み込み関数のいずれかが使用できます。

この章では、使用可能な命令コードと組み込み関数について要約しています。 また、命令コードと組み込み関数をカテゴリー別にまとめてあります。

特定の命令コードまたは組み込み関数について詳しくは、710 ページの『命令コード』または613 ページの『組み込み関数』を参照してください。

命令コード

以下の表では、それぞれの命令コードごとに自由形式構文を示します。

- 拡張

(A)
DEBUG(*NO) が指定されている場合であっても、必ずダンプを実行する

(A)
昇順ソート

(D)
バインド呼び出し時に操作記述子を渡す

(D)
日付フィールド

(D)
降順ソート

(E)
エラー処理

(H)
四捨五入 (数値の結果の丸め)

(M)
デフォルトの精度規則

(N)
レコードをロックしない

(N)
DEALLOC が正常に行われた後、ポインターを *NULL に設定する

(N)
データを不揮発性の記憶域に強制的に入れない

(P)
結果にブランクまたはゼロの埋め込みを行う

(R)
"結果の小数点以下の桁数" 精度規則

(T)
時刻フィールド

(Z)
タイム・スタンプ・フィールド

表 *103.* 自由形式構文における命令コード

コード	自由形式構文
ACQ[1]	ACQ{(E)} 装置名 ワークステーション・ファイル
BEGSR	BEGSR サブルーチン名
CALLP	{CALLP{(EMR) } } 名前({パラメーター *1*{:パラメーター *2*... } })
CHAIN	CHAIN {(ENHMR) } 検索引数 ファイルまたはレコード名 {データ構造 }
CLEAR	CLEAR {*NOKEY } {*ALL } 名前
CLOSE	CLOSE{(E)} ファイル名
COMMIT	COMMIT{(E) } {境界 }
DATA-GEN	DATA-GEN{(EH)} ソース文書パーサー
DATA-INTO	DATA-INTO{(EH)} ターゲットまたはハンドラー文書パーサー
DEALLOC [1]	DEALLOC {(EN) } ポインター名
DELETE	DELETE{(EHMR)} {検索引数} ファイル名またはレコード名
DOU	DOU {(MR) } 標識式
DOW	DOW {(MR) } 標識式
DSPLY	DSPLY {(E) } {メッセージ {メッセージ待ち行列 {応答 } } }
DUMP[1]	DUMP {(A) } {識別コード }
ELSE	ELSE
ELSEIF	ELSEIF {(MR) } 標識式
ENDDO	ENDDO
ENDFOR	ENDFOR
ENDIF	ENDIF
ENDMON	ENDMON
ENDSL	ENDSL
ENDSR	ENDSR {戻り点 }
EVAL	{EVAL {(HMR) } } 結果 = 式
EVALR	EVALR {(MR) } 結果 = 式
EVAL-CORR	EVAL-CORR{(EH)} ターゲット・データ構造 = ソース・データ構造
EXCEPT	EXCEPT {例外名 }
EXFMT	EXFMT{(E)} 様式名 *{データ構造}*
EXSR	EXSR サブルーチン名
FEOD	FEOD{(EN)} ファイル名

表 *103.* 自由形式構文における命令コード *(*続き*)*	
コード	**自由形式構文**
FOR	FOR {(MR) } 索引 {= 開始 } {BY 増分 } {TO|DOWNTO 限界 }
FOR-EACH	FOR-EACH{(H)} 項目 IN 配列 | %LIST | %SUBARR
FORCE	FORCE ファイル名
IF	IF{(MR)} 標識式
IN [1]	IN{(E)} {*LOCK} データ域名
ITER	ITER
LEAVE	LEAVE
LEAVESR	LEAVESR
MONITOR	MONITOR
NEXT[1]	NEXT{(E)} プログラム装置 ファイル名
ON-ERROR	ON-ERROR {例外 *ID1* {:例外 *ID2* ...}}
ON-EXIT	ON-EXIT {*status*}
OPEN	OPEN{(E)} ファイル名
OTHER	OTHER
OUT[1]	OUT{(E)} {*LOCK} データ域名
POST [1]	POST{(E)} {プログラム装置} ファイル名
READ	READ {(EN) } ファイルまたはレコード名 {データ構造 }
READC	READC {(E) } レコード名 {データ構造 }
READE	READE {(ENHMR) } 検索引数|*KEY ファイルまたはレコード名 {データ構造 }
READP	READP {(EN) } 名前 {データ構造 }
READPE	READPE {(ENHMR) } 検索引数|*KEY ファイルまたはレコード名 {データ構造 }
REL [1]	REL {(E) } プログラム装置 ファイル名
RESET [1]	RESET{(E)} {*NOKEY} {*ALL} 名前
RETURN	RETURN{(HMR)} 式
ROLBK	ROLBK {(E) }
SELECT	SELECT
SETGT	SETGT{(EHMR)} 検索引数 ファイル名またはレコード名
SETLL	SETLL{(EHMR)} 検索引数 ファイルまたはレコード名
SORTA	SORTA{(AD)} 配列名またはキー付きデータ構造配列
TEST [1]	TEST {(EDTZ) } {*dtz* 形式 } フィールド名
UNLOCK [1]	UNLOCK{(E)} 名前
UPDATE	UPDATE {(E) } ファイルまたはレコード名 {データ構造 | %FIELDS(名前 {:名前... }) }
WHEN	WHEN{(MR)} 標識式
WRITE	WRITE {(E) } ファイルまたはレコード名 {データ構造 }

表 *103.* 自由形式構文における命令コード *(*続き*)*	
コード	**自由形式構文**
XML-INTO	XML-INTO{(EH)} ターゲットまたはハンドラーの *XML* 文書
XML-SAX	XML-SAX{(E)} ハンドラーの *XML* 文書

注 :

1. この命令コードでは複合修飾名を使用できません。

次の表には、従来型の構文における各命令コードの仕様をまとめてあります。

- 空白の欄は、そのフィールドがブランクでなければならないことを示して います。
- 下線の付いたフィールドは必須のフィールドです。
- 下線の付いたスペースは、その位置に結果の標識がないことを表します。
- 記号

  +
  プラス

  -
  マイナス
- 拡張

  (A)
  DEBUG(*NO) が指定されている場合であっても、必ずダンプを実行する

  (A)
  昇順ソート

  (D)
  バインド呼び出し時に操作記述子を渡す

  (D)
  日付フィールド

  (D)
  降順ソート

  (E)
  エラー処理

  (H)
  四捨五入 (数値の結果の丸め)

  (M)
  デフォルトの精度規則

  (N)
  レコードをロックしない

  (N)
  DEALLOC が正常に行われた後、ポインターを *NULL に設定する

  (P)
  結果にブランクまたはゼロの埋め込みを行う

  (R)
  "結果の小数点以下の桁数" 精度規則

  (T)
  時刻フィールド

  (Z)
  タイム・スタンプ・フィールド
- 結果標識の記号

BL
ブランク (複数の場合もある)

BN
ブランク (複数の場合もある) の後で数値

BOF
ファイルの始め

EOF
ファイルの終わり

EQ
等しい

ER
エラー

FD
検索済み

HI
より大きい

IN
標識

LO
より小さい

LR
最後のレコード

NR
レコードが見付からなかった

NU
数値

OF
オフ

ON
オン

Z
ゼロ

ZB
ゼロまたはブランク

表 *104.* 従来型の構文における命令コード

コード	演算項目 1	演算項目 2	結果フィールド	結果標識		
				71-72	73-74	75-76
ACQ (E[7])	装置名	ワークステーション・ファイル			ER	
ADD (H)	加数	加数	和	+	-	Z
ADDDUR (E)	日付/時刻	期間: 期間コード	日付/時刻		ER	
ALLOC (E)		長さ	ポインター		ER	
ANDxx	被比較値	被比較値				
BEGSR	サブルーチン名					
BITOFF		ビット数	文字フィールド			

表 *104.* 従来型の構文における命令コード *(*続き*)*

コード	演算項目 1	演算項目 2	結果フィールド	結果標識		
				71-72	73-74	75-76
BITON		ビット数	文字フィールド			
CABxx	被比較値	被比較値	ラベル	HI	LO	EQ
CALL (E)		プログラム名	PLIST 名		ER	LR
CALLB (D E)		プロシージャー名または プロシージャー・ポインター	PLIST 名		ER	LR
CALLP (E M/R)		名前{ (パラメーター 1 {:パラメーター 2 ...}) }				
CASxx	被比較値	被比較値	サブルーチン名	HI	LO	EQ
CAT (P)	ソース・ストリング 1	ソース・ストリング 2: ブランクの数	ターゲット・ストリング			
CHAIN (E N)	検索引数	名前 (ファイルまたはレコード様式)	データ構造	NR[2]	ER	
CHECK (E)	比較ストリング	基本ストリング: 開始	左端の位置 (複数の場合もある)		ER	FD[2]
CHECKR (E)	比較ストリング	基本ストリング: 開始	右端の位置 (複数の場合もある)		ER	FD[2]
CLEAR	*NOKEY	*ALL	名前 (変数またはレコード様式)			
CLOSE (E)		ファイル名 または *ALL			ER	
COMMIT (E)	境界				ER	
COMP[1]	被比較値	被比較値		HI	LO	EQ
DEALLOC (E/N)			ポインター名		ER	
DEFINE	*LIKE	被参照フィールド	定義されるフィールド			
DEFINE	*DTAARA	外部データ区域	内部フィールド			
DELETE (E)	検索引数	名前 (ファイルまたはレコード様式)		NR[2]	ER	
DIV (H)	被除数	除数	商	+	-	Z
DO	開始値	限界値	指標値			
DOU (M/R)		標識式				
DOUxx	被比較値	被比較値				
DOW (M/R)		標識式				
DOWxx	被比較値	被比較値				

表 *104.* 従来型の構文における命令コード *(*続き*)*

コード	演算項目 1	演算項目 2	結果フィールド	結果標識		
				71-72	73-74	75-76
DSPLY (E)[4]	メッセージ	メッセージ・キュー	応答		ER	
DUMP (A)	ID					
ELSE						
ELSEIF (M/R)		標識式				
END		増分値				
ENDCS						
ENDDO		増分値				
ENDFOR						
ENDIF						
ENDMON						
ENDSL						
ENDSR	ラベル	戻り点				
EVAL (H M/R)		結果 = 式				
EVALR (M/R)		結果 = 式				
EVAL-CORR		EVAL-CORR ターゲット・データ構造 = ソース・データ構造				
EXCEPT		例外名				
EXFMT (E)		レコード様式名	データ構造		ER	
EXSR		サブルーチン名				
EXTRCT (E)		日付/時刻:期間コード	ターゲット・フィールド		ER	
FEOD (EN)		ファイル名			ER	
FOR		索引名 = 開始値 BY 増分 TO|DOWNTO 限界				
FORCE		ファイル名				
GOTO		ラベル				
IF (M/R)		標識式				
IFxx	被比較値	被比較値				
IN (E)	*LOCK	データ域名			ER	
ITER						
KFLD			キー・フィールド			
KLIST	KLIST 名					
LEAVE						
LEAVESR						
LOOKUP[1] **(配列)**	検索引数	配列名		HI	LO	EQ[6]

表 *104.* 従来型の構文における命令コード *(*続き*)*

コード	演算項目 1	演算項目 2	結果フィールド	結果標識		
				71-72	73-74	75-76
LOOKUP[1] (テーブル)	検索引数	テーブル名	テーブル名	HI	LO	EQ[6]
MHHZO		ソース・フィールド	ターゲット・フィールド			
MHLZO		ソース・フィールド	ターゲット・フィールド			
MLHZO		ソース・フィールド	ターゲット・フィールド			
MLLZO		ソース・フィールド	ターゲット・フィールド			
MONITOR						
MOVE(P)	データ属性	ソース・フィールド	ターゲット・フィールド	+	-	ZB
MOVEA (P)		ソース	ターゲット	+	-	ZB
MOVEL(P)	データ属性	ソース・フィールド	ターゲット・フィールド	+	-	ZB
MULT (H)	被乗数	乗数	積	+	-	Z
MVR			剰余	+	-	Z
NEXT (E)		ファイル名			ER	
OCCUR (E)	オカレンス値	データ構造	オカレンス値		ER	
ON-ERROR		状況コード				
ON-EXIT		状況				
OPEN (E)		ファイル名			ER	
ORxx	被比較値	被比較値				
OTHER						
OUT (E)	*LOCK	データ域名			ER	
PARM	ターゲット・フィールド	ソース・フィールド	パラメーター			
PLIST	PLIST 名					
POST (E)[3]	プログラム装置	ファイル名	INFDS 名		ER	
READ (E N)		名前 (ファイルまたはレコード様式)	データ構造		ER	EOF[5]
READC (E)		レコード名	データ構造		ER	EOF[5]
READE (E N)	検索引数	名前 (ファイルまたはレコード様式)	データ構造		ER	EOF[5]
READP (E N)		名前 (ファイルまたはレコード様式)	データ構造		ER	BOF[5]

表 *104.* 従来型の構文における命令コード *(*続き*)*

コード	演算項目 1	演算項目 2	結果フィールド	結果標識		
				71-72	73-74	75-76
READPE (E N)	検索引数	名前 (ファイルまたはレコード様式)	データ構造		ER	BOF[5]
REALLOC (E)		長さ	ポインター		ER	
REL (E)		ファイル名			ER	
RESET (E)	*NOKEY	*ALL	名前 (変数またはレコード様式)		ER	
RETURN (H M/R)		式				
ROLBK (E)					ER	
SCAN (E)	比較ストリング:長さ	基本ストリング: 開始	左端の位置 (複数の場合もある)		ER	FD[2]
SELECT						
SETGT (E)	検索引数	名前 (ファイルまたはレコード様式)		NR[2]	ER	
SETLL (E)	検索引数	名前 (ファイルまたはレコード様式)		NR[2]	ER	EQ[6]
SETOFF[1]				OF	OF	OF
SETON[1]				ON	ON	ON
SHTDN				ON		
SORTA(A/D)		配列名またはキー付きデータ構造配列				
SQRT (H)		値	平方根			
SUB (H)	被減数	減数	差	+	-	Z
SUBDUR (E) (期間)	日付/時刻/タイム・スタンプ	日付/時刻/タイムスタンプ	期間: 期間コード		ER	
SUBDUR (E) (新しい日付)	日付/時刻/タイム・スタンプ	期間: 期間コード	日付/時刻/タイム・スタンプ		ER	
SUBST (E P)	取り出す長さ	基本ストリング: 開始	ターゲット・ストリング		ER	
TAG	ラベル					
TEST (E)[8]			日付/時刻またはタイムスタンプ・フィールド		ER	
TEST (D E)[8]	日付の形式		文字または数値フィールド		ER	
TEST (E T)[8]	時刻の形式		文字または数値フィールド		ER	

表 *104.* 従来型の構文における命令コード *(*続き*)*

コード	演算項目 1	演算項目 2	結果フィールド	結果標識		
				71-72	73-74	75-76
TEST (E Z)[8]	タイム・スタンプ形式		文字または数値フィールド		ER	
TESTB[1]		ビット数	文字フィールド	OF	ON	EQ
TESTN[1]			文字フィールド	NU	BN	BL
TESTZ[1]			文字フィールド	AI	JR	XX
TIME			ターゲット・フィールド			
UNLOCK (E)		名前 (ファイルまたはデータ域)			ER	
UPDATE (E)		名前 (ファイルまたはレコード様式)	データ構造		ER	
WHEN (M/R)		標識式				
WHENxx	被比較値	被比較値				
WRITE (E)		名前 (ファイルまたはレコード様式)	データ構造		ER	EOF[5]
XFOOT (H)		配列名	和	+	-	Z
XLATE (E P)	変換元:変換先	ストリング:開始	ターゲット・ストリング		ER	
XML-INTO		XML-INTO ターゲットまたはハンドラー *XML* 文書				
XML-SAX		XML-SAX{(E)} ハンドラーの *XML* 文書				
Z-ADD (H)		加数	和	+	-	Z
Z-SUB (H)		減数	差	+	-	Z

注 :

1. 少なくとも 1 つの結果の標識を指定しなければなりません。
2. NR または FD 結果標識を指定する代わりに、%FOUND 組み込み関数 を使用することができます。
3. 演算項目 2 または結果のフィールドを指定しなければなりません。 両方を指定することもできます。
4. 演算項目 1 または結果のフィールドを指定しなければなりません。 両方を指定することもできます。
5. EOF または BOF 結果標識を指定する代わりに、%EOF 組み込み関数 を使用することができます。
6. SETLL および LOOKUP 命令をテストするために %EQUAL 組み込み関数 を使用することができます。
7. 拡張 'E' が付いているすべての命令コードでは、拡張 'E' または ER エラー標識を指定することができますが、両方を指定することはできません。
8. TEST 命令の場合は、拡張 'E' またはエラー標識を指定する 必要があります。

組み込み関数

組み込み関数は、指定されたデータに対して命令を実行 する点で命令コードに類似しています。 組み込み関数は、式の中で使用することができます。 また、定数値組み込み関数は名前付き定数においても使用できます。 これらの名前付き定数はどの仕様書でも使用できます。

すべての組み込み関数は、最初の文字としてパーセント記号 (%) で始まります。 組み込み関数の構文は次のとおりです。

```
function-name{(argument{:argument...})}
```

関数の引数は、変数、定数、式、プロトタイプ・プロシージャー、または他の組み込み関数とすることができます。 式の引数には組み込み関数を含めることができます。 以下の例は これを例示しています。

```
CL0N01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++
 *
 * This example shows a complex expression with multiple
 * nested built-in functions.
 *
 * %TRIM takes as its argument a string.  In this example, the
 * argument is the concatenation of string A and the string
 * returned by the %SUBST built-in function.  %SUBST will return
 * a substring of string B starting at position 11 and continuing
 * for the length returned by %SIZE minus 20.  %SIZE will return
 * the length of string B.
 *
 * If A is the string '     Toronto,' and B is the string
 * '  Ontario, Canada            ' then the argument for %TRIM will
 * be '     Toronto, Canada    ' and RES will have the value
 * 'Toronto, Canada'.
 *
C                   EVAL      RES = %TRIM(A + %SUBST(B:11:%SIZE(B) - 20))
```

図 *178.* 組み込み関数の引数の例

使用できる引数の詳細については、個々の組み込み関数の説明を参照してください。

命令コードと異なり、組み込み関数は、結果のフィールドに値を入れるのではなく 、値を戻してきます。 以下の例はこの相違を例示しています。

```
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 * In the following example, CITY contains the string
 * 'Toronto, Ontario'.  The SCAN operation is used to locate the
 * separating blank, position 9 in this illustration.  SUBST
 * places the string 'Ontario' in field TCNTRE.
 *
 * Next, TCNTRE is compared to the literal 'Ontario' and
 * 1 is added to CITYCNT.
 *
C     ' '           SCAN      CITY          C
C                   ADD       1             C
C                   SUBST     CITY:C        TCNTRE
C    'Ontario'      IFEQ      TCNTRE
C                   ADD       1             CITYCNT
C                   ENDIF
 *
 * In this example, CITY contains the same value, but the
 * variable TCNTRE is not necessary since the %SUBST built-in
 * function returns the appropriate value.  In addition, the
 * intermediary step of adding 1 to C is simplified since
 * %SUBST accepts expressions as arguments.
 *
C     ' '           SCAN      CITY          C
C                   IF        %SUBST(CITY:C+1) = 'Ontario'
C                   EVAL      CITYCNT = CITYCNT+1
C                   ENDIF
```

図 *179.* 組み込み関数の例

この例で使用されている引数 (変数 CITY および式 C+1) は SUBST 命令の演算項 目の値と似ていることに注意してください。 関数そのものの戻り値は結果と類似しています。 一般に、組み込み関数の引数は、命令コードの演算項目 1 および演算項目 2 フィール ドと類似しています。

組み込み関数が役立つもう 1 つの機能は、定義仕様書に使用された時にコードの保守 が単純化される点です。 以下の例はこの機能を例示しています。

```
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
 *
 * In this example, CUSTNAME is a field in the
 * externally described data structure CUSTOMER.
 * If the length of CUSTNAME is changed, the attributes of
 * both TEMPNAME and NAMEARRAY would be changed merely by
 * recompiling.  The use of the %SIZE built-in function means
 * no changes to your code would be necessary.
 *
D CUSTOMER      E DS
D                 DS
D TEMPNAME                          LIKE(CUSTNAME)
D NAMEARRAY                   1     OVERLAY(TEMPNAME)
D                                   DIM(%SIZE(TEMPNAME))
```

図 *180.* 組み込み関数による単純化された保守

組み込み関数は、拡張演算項目 2 演算仕様書で定義仕様書のキーワードと一緒に式 の中で使用することができます。 定義仕様書キーワードと一緒に使用する時には、組み込み関数の値がコンパイル 時に分かっていなければならず、引数を式とすることはできません。

次の表には、組み込み関数、その引数、および戻り値がリストされています。

表 *105.* 組み込み関数

名前	引数	戻り値
%ABS	数値式	式の絶対値
%ADDR	変数名 {: *DATA }	変数のアドレス、または可変長変数のデータ部のアドレス

表 105. 組み込み関数 (続き)		
名前	**引数**	**戻り値**
%ALLOC	割り振るバイト数	割り振られる記憶域へのポインター
%BITAND	文字、数値	すべての引数のビットの論理積 (ビット単位)
%BITNOT	文字、数値	引数のビットの反転 (ビット単位)
%BITOR	文字、数値	すべての引数のビットの論理和 (ビット単位)
%BITXOR	文字、数値	2 つの引数のビットの排他論理和 (ビット単位)
%CHAR	• 文字、グラフィック、UCS-2 式 { : ccsid } • 数値式 • 日付、時刻、タイム・スタンプ式 {: 日付、時刻、またはタイム・スタンプの形式 }	• 指定された CCSID の文字形式の値 • ジョブ CCSID の文字形式の値 • ジョブ CCSID の文字形式の値
%CHECK	コンパレーター・ストリング:チェックされるストリング {:開始位置 }	コンパレーター・ストリング内でない文字の最初の位置。 または、見付からない場合はゼロ
%CHECKR	コンパレーター・ストリング:チェックされるストリング {:開始位置 }	コンパレーター・ストリング内でない文字の末尾の位置。 または、見付からない場合はゼロ
%DATA	文書 { : オプション }	該当せず
%DATE	{値 {: 日付の形式 } }	指定された値 に対応する日付。 または、指定されていなければ現在のシステム日付。
%DAYS	日数	期間としての日数
%DEC	• 数値式 {:数字:小数点以下の桁数 } • 文字式: 数字:小数点以下の桁数 • 日付、時刻、またはタイム・ スタンプ式 {:形式 }	パック数値形式の値
%DECH	数値式または文字式: 数字:小数点以下の桁数	パック数値形式の四捨五入値
%DECPOS	数値式	10 進数の数
%DIFF	日付または時刻式: 日付または時刻式: 単位 {:秒の小数部の桁数 }	2 つの日付、時刻、またはタイム・スタンプの差を指定された単位で示したもの
%DIV	被除数: 除数	2 つの引数の割り算からの商
%EDITC	非浮動数値式:編集コード {:*CURSYM | *ASTFILL | 通貨記号 }	編集済みの値を表すストリング
%EDITFLT	数値式	浮動の文字外部表示表現
%EDITW	非浮動数値式:編集語	編集済みの値を表すストリング
%ELEM [1]	配列、テーブル、または複数オカレンス・データ構造名	要素または繰り返しの数

表 105. 組み込み関数 (続き)		
名前	**引数**	**戻り値**
%EOF	{file name}	サブファイル (指定されている場合は特定のファイル) に対して最後に実行された反復入力、読み取り操作、または書き出しが、ファイルの終わり条件またはファイルの先頭条件で終了した場合は「1」。
		それ以外の場合は「0」です。 **注 :** パラメーターが指定される場合は、CHAIN、OPEN、SETGT、および SETLL の各命令も %EOF に影響を与えます。 638 ページの『%EOF (ファイルの終わりまたは先頭条件の戻し)』を参照してください。
%EQUAL	{file name}	'1' 最後に実行された SETLL (指定された場合は特定のファイルに対する) または LOOKUP 命令が等しい項目を見付けた場合
		'0' それ以外の場合
%ERROR		'1' 最後に実行された、拡張 'E' が指定された命令コードの結果がエラーである場合
		'0' それ以外の場合
%FIELDS	• 更新されるフィールドのリスト • ソートするためのサブフィールドのリスト	該当せず
%FLOAT	数値式または文字式	浮動形式の値
%FOUND	{file name}	'1' 最後に実行された関係のある命令 (指定された場合は特定のファイルに対する) がレコード (CHAIN、DELETE、SETGT、SETLL)、要素 (LOOKUP)、または等しい項目 (CHECK、CHECKR、SCAN) を見付けた場合
		'0' それ以外の場合
%GEN	生成プログラム { : オプション }	該当せず
%GRAPH	文字、グラフィック、または UCS-2 式 { : ccsid }	指定された CCSID のグラフィック形式の値
%HANDLER	処理プロシージャー: 通信域	該当せず
%HOURS	時間数	期間としての時間数
%INT	数値式または文字式	整数形式の値
%INTH	数値式または文字式	整数形式の四捨五入値
%KDS	キーを含むデータ構造 {: number of keys}	該当せず
%LEN	任意の式	数字または文字の長さ
%LIST	item1 { : item2 { : item3 ... } }	オペランドから作成される配列
%LOOKUPxx	引数: 配列 {:開始指標 {:要素の数 } }	突き合わせる要素の配列指標

表 *105.* 組み込み関数 *(*続き*)*

名前	引数	戻り値
%LOWER	ストリング {:開始{:長さ}}	ストリング (サブストリングを小文字に変換します)
%MAX	item1 : item2 { : item3 { : item4 ... } }	オペランドの最大値
%MAXARR	配列 { : 開始要素 { : 要素の数 } }	配列内の最大値の指標
%MIN	item1 : item2 { : item3 { : item4 ... } }	オペランドの最小値
%MINARR	配列 { : 開始要素 { : 要素の数 } }	配列内の最小値の指標
%MINUTES	分数	期間としての分数
%MONTHS	月数	期間としての月数
%MSECONDS	マイクロ秒数	期間としてのマイクロ秒数
%NULLIND	ヌル値可能フィールド名	ヌル値可能フィールド用のヌル標識設定値を示す 標識形式の値
%OCCUR	複数オカレンス・データ構造名	複数オカレンス・データ構造の現行のオカレンス
%OPEN	ファイル名	'1' 指定されたファイルがオープンされている場合
		'0' 指定されたファイルがクローズされている場合
%PADDR	プロシージャーまたはプロトタイプ名	プロシージャーまたはプロトタイプのアドレス
%PARMS	なし	プロシージャーに渡されたパラメーターの数
%PARMNUM	プロシージャー・インターフェース・パラメーター名	プロシージャー・インターフェース・パラメーターの番号
%PARSER	パーサー { : オプション }	該当せず
%PROC		現在のプロシージャーの外部名
%RANGE	item1 : item2	該当せず
%REALLOC	ポインター: 数値式	割り振られる記憶域へのポインター
%REM	被除数: 除数	2 つの引数の割り算からの剰余
%REPLACE	置換ストリング: ソース・ストリング {:開始 位置 {:置換するソースの長さ } }	開始位置から開始し、指定された文字数を置換して、 置換ストリングをソース・ストリングに挿入することによって 生成されるストリング
%SCAN	検索引数:検索されるストリング{:開始位置{:長さ}}	ストリング内の検索引数の最初の桁またはそれが見付からない場合はゼロ
%SCANR	検索引数:検索されるストリング{:開始位置{:長さ}}	ストリング内の検索引数の最後の位置、またはゼロ (その検索引数が見つからない場合)

表 *105.* 組み込み関数 *(*続き*)*

名前	引数	戻り値
%SCANRPL	スキャン・ストリング: 置換ストリング: ソース・ストリング {: スキャン開始位置 {:scan length}}	ソース・ストリング内で、開始位置から指定された長さの分を走査して、走査ストリングを置換ストリングで置換して生成したストリング
%SECONDS	秒数	期間としての秒数
%SHTDN		'1' システム・オペレーターがシャットダウンを要求している場合
		'0' それ以外の場合
%SIZE [1]	変数、配列、またはリテラル {:*ALL }	変数またはリテラルのサイズ
%SPLIT	ストリング{:区切り文字}	サブストリングの配列
%SQRT	数値	数値の平方根
%STATUS	{file name}	'0' 最後に実行された、拡張 'E' が指定された命令コード以降、 プログラム・エラーまたはファイル・エラーが発生していない場合
		エラーが発生した場合、プログラムまたはファイル状況 に関して設定された最新の値
		ファイルが指定されている場合、戻り値は、そのファイルに関する最新の状況
%STR	ポインター {:最大長 }	最初の x'00' までの (ただしその値を含まない) ポインター引数 によってアドレスされる文字
%SUBARR	配列名:開始指標 {:要素の数 }	配列サブセット
%SUBDT	日付または時刻の式: 単位	日付または時刻の値の指定された部分を含む、符号なし数値
%SUBDT	日付または時刻式: 単位 {桁数 : { 秒の小数部の桁数 } }	
%SUBST	ストリング:開始位置 {:長さ }	サブストリング
%THIS		ネイティブ・メソッド用のクラス・インスタンス
%TIME	{値 {: 時刻の形式 } }	指定された値 に対応する時刻。 または、指定されていなければ現在のシステム時刻。
%TIMESTAMP	{(値 {: タイム・スタンプ形式 {: 秒の小数部の桁数 } }) }	指定された英数字または数字の値 に対応する タイム・スタンプ、または、何も指定されていなければ現在のシステム・タイム・スタンプ。
%TIMESTAMP	{(値 {: 秒の小数部の桁数 }) }	指定された日付またはタイム・スタンプの値 に対応する タイム・スタンプ、または、何も指定されていなければ現在のシステム・タイム・スタンプ。
%TLOOKUPxx	引数: 検索テーブル {: 代替テーブル }	'*ON' 一致するものがある場合
		'*OFF' それ以外

表 105. 組み込み関数 (続き)		
名前	**引数**	**戻り値**
%TRIM	ストリング {: トリミング対象文字 }	左および右のブランクまたは指定文字がトリミングされたストリング
%TRIML	ストリング {: トリミング対象文字 }	左のブランクまたは指定文字がトリミングされたストリング
%TRIMR	ストリング {: トリミング対象文字 }	右のブランクまたは指定文字がトリミングされたストリング
%UCS2	文字、グラフィック、または UCS-2 式 { : ccsid }	指定された CCSID の UCS-2 形式の値
%UNS	数値式または文字式	符号なし形式の値
%UNSH	数値式または文字式	符号なし形式の四捨五入値
%UPPER	ストリング {:開始{:長さ}}	ストリング (サブストリングを大文字に変換します)
%XFOOT	配列式	要素の合計
%XLATE	変換元文字: 変換先文字: ストリング {: 開始位置 }	変換元文字が変換先文字で置換されたストリング
%XML	XML 文書 { : オプション }	該当せず
%YEARS	年数	期間としての年数

注 :

1. 複合修飾名は許可されません。

算術演算

算術演算を次の表に示します。

表 106. 算術演算		
命令	**従来型の構文**	**自由形式構文**
絶対値		613 ページの『%ABS (式の絶対値)』
Add	710 ページの『ADD (加算)』	+ 演算子
DIVIDE	758 ページの『DIV (除算)』	/ 演算子または 634 ページの『%DIV (商の戻り整数部分)』
除算の剰余	834 ページの『MVR (剰余の転送)』	679 ページの『%REM (戻り整数剰余)』
MULTIPLY	833 ページの『MULT (乗算)』	* 演算子
平方根	886 ページの『SQRT (平方根)』	691 ページの『%SQRT (式の平方根)』
減算	887 ページの『SUB (減算)』	- 演算子
ゼロにして加算	961 ページの『Z-ADD (ゼロにして加算)』	(許可されていない)
ゼロにして減算	961 ページの『Z-SUB (ゼロにして減算)』	(許可されていない)

算術演算の例については、560 ページの図 181 を参照してください。

算術演算の指定にあたっては次のことを忘れないでください。

- 算術演算を実行できるのは、数値 (数値サブフィールド、数値配列、数値 配列要素、数値テーブル要素、名前付き数値定数、数値形象定数、および 数値リテラルを含む) に対してだけです。

- 一般に、算術演算はパック 10 進数形式で実行されます。 このことは、 算術演算を実行する前に、フィールドが最初にパック 10 進数形式に変換 され、次に結果のフィールドに結果を入れる前に (必要があれば) 指定の 形式に戻されることを意味しています。

  ただし、以下の例外に注意してください。

  - すべてのオペランドが符号なしの場合には、符号なしの数値で算術演算が 行われます。
  - すべてのオペランドが整数、または整数と符号なしの数字 の場合には、整数の数値で算術演算が行われます。
  - いずれかのオペランドが浮動の場合、残りのオペランドは浮動に 変換されます。

  ただし、DIV 命令は、その演算にパック 10 進数または浮動形式を 使用します。整数および符号なしの数値演算について詳しくは、559 ページの『整数および符号なしの演算』を参照してください。
- 小数点の位置合わせはすべての算術演算で行われます。 切り捨てが行われても、結果のフィールドの小数点の位置には影響がありません。
- 算術演算の結果、結果のフィールドに入っていたデータは置き換えられます。
- 演算項目 1 と演算項目 2 は、それらが結果のフィールドと同じである 場合を除いて、算術演算によっては変更されません。
- DIV および MVR で条件付け標識を使用する場合には、ユーザーの責任において DIV 命令が MVR 命令の直前に行われることを確認してください。 DIV の条件付け標識によって直前の DIV 命令が実行されないときに MVR 命令が 実行されると、好ましくない結果が起こる場合があります。
- 算術演算での配列の使用方法については、241 ページの『演算での配列の指定』を参照してください。

精度の確認

- 算術演算で指定するフィールドの長さは 63 桁を超えることができません。 フィールドに代入される値が 63 桁を超えると、小数点の位置によって、いずれか一方または両方の末端から数字が脱落します。
- TRUNCNBR オプション (コマンド・パラメーターまたは制御仕様書上 のキーワード) によって、数値オーバーフローで切り捨てが左から起こるか、 あるいは実行時エラーになるかが決まります。 TRUNCNBR は、式の中で行われる計算には適用されないことに注意してくだ さい。 式の計算の中でオーバーフローが起こると、実行時メッセージが出されます。 また、TRUNCNBR は整数または符号なしの形式で行われる算術演算にも 適用されません。
- 四捨五入は、 結果のフィールドに最後に指定された小数点の 1 桁右に 5 (フィールドが負の場合には -5) を 加えて行われます。
- 四捨五入の指定は、以下の命令で許可されます。
  - 算術演算。ただし、MVR 命令や MVR 命令が後に続く DIV 命令では指定できません。
  - EVAL 命令および FOR-EACH 命令 (ターゲット・フィールドが数値の場合)。
  - RETURN 命令 (戻り値が数値の場合)。
  - DATA-INTO 命令および XML-INTO 命令。 これらの命令では、四捨五入は数値フィールドまたはサブフィールドへの代入に影響します。
  - CHAIN、DELETE、READE、READPE、SETGT、および SETLL 命令 (検索引数または %KDS のリストが指定されている場合)。 579 ページの『ファイル命令のキー』を参照してください。
- 四捨五入が結果に影響を与えるのは、計算結果の小数点以下の桁数が結果の フィールドの小数点以下の桁数より大きい場合だけです。 四捨五入は、この命令の後で結果のフィールドに結果が入れられる前に行われます。
- 四捨五入は、浮動フィールドには影響しません。
- 結果の標識は、四捨五入が行われた後で、結果のフィールドの値に応じて設定されます。

パフォーマンスに関する考慮事項

一番速い算術演算のパフォーマンス時間は、すべてのオペランドが整数または 符号なしの形式のときのものです。 次に速いパフォーマンス時間は、すべてのオペランドがパック 10 進数である ときのものです。この場合には、共通形式に変換する必要がないためです。

整数および符号なしの演算

すべての算術演算 (式に入っているものを除く) で、演算項目 1、演算項目 2、 および結果のフィールドが符号なしの形式で定義されている場合には、 命令は符号なしの形式で実行されます。 同じように、演算項目 1、演算項目 2、および結果のフィールドが整数または 符号なしの形式のいずれかとして定義されている場合には、命令は整数形式で 実行されます。 いずれかのフィールドが整数または符号なしの形式でない場合には、 命令はデフォルトの形式 (パック 10 進数) で実行されます。

次の項目は、整数および符号なしの 算術演算のみに適用されます。

- すべての整数および符号なし命令は 8 バイト形式で実行されます。
- 整数および符号なしの値は 1 つの命令で一緒に使用することができます。 ただし、演算項目 1、演算項目 2、または結果のフィールドが整数の場合 には、符号なしの値はすべて整数に変換されます。 必要な場合には、数値オーバーフローを生じにくくするために、1 バイト、2 バイト、 または 4 バイトの符号なしの値が、より大きなサイズの整数値に変換されます。
- あるリテラルが、20 またはそれ以下の桁数であり、小数点以下の桁数がゼロであって、 整数および符号なしのフィールドに指定できる範囲内に 入っている場合、そのリテラルは、負の値であるのか正の値であるのかに応じて、 それぞれ整数または符号なしの形式でロードされます。

注 : 整数または符号なしの演算では、パフォーマンスが向上します。 ただし、 整数または符号なしの数値形式を使用する場合には、パックまたはゾーン 10 進数形式を使用する場合に比べて、 数値オーバーフローの可能性は高くなります。

算術演算の例

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
C*
C*   In the following example, the initial field values are:
C*
D A               s              3p 0  inz(1)
D B               s              3p 1  inz(10.0)
D C               s              2p 0  inz(32)
D D               s              2p 0 inz(-10)
D E               s              3p 0 inz(6)
D F               s              3p 0 inz(10)
D G               s              3p 2 inz(2.77)
D H               s              3p 0 inz(70)
D J               s              3p 1 inz(0.6)
D K               s              2p 0 inz(25)
D L               s              2p 1 dim(3)
D V               s              5p 2
D W               s              5p 1
D X               s              8p 4
D Y               s              6p 2
D Z               s              5p 3

 /FREE
    L(1) = 1.0;
    L(2) = 1.7;
    L(3) = -1.1;

    A = A + 1;               // A = 002
    V = B + C;               // V = 042.00
    V = B + D;               // V = 0
    V = C;                   // V = 032.00
    E = E - 1;               // E = 005
    W = C - B;               // W = 0022.0
    W = C - D;               // W = 0042.0
    W = - C;                 // W = -0032.0
    F = F * E;               // F = 060
    X = B * G;               // X = 0027.7000
    X = B * D;               // X = -0100.0000
    H = H / B;               // H = 007
    Y = C / J;               // Y = 0053.33
    eval(r) Z = %sqrt(K);    // Z = 05.000
    Z = %xfoot(L);           // Z = 01.600

    dump(a);
    *inlr = *on;
 /END-FREE
```

図 *181.* 自由形式演算での算術演算

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....Comments
C*
C*   In the following example, the initial field values are:
C*
C*                          A = 1
C*                          B = 10.0
C*                          C = 32
C*                          D = -20
C*                          E = 6
C*                          F = 10.0
C*                          G = 2.77
C*                          H = 70
C*                          J = .6
C*                          K = 25
C*                          L = 1.0, 1.7, -1.1                          Result:
C*

C                   ADD       1             A                 3 0         A = 002
C     B             ADD       C             V                 5 2         V = 042.00
C     B             ADD       D             V                             V = -10.00
C                   Z-ADD     C             V                             V = 032.00
C                   SUB       1             E                 3 0         E = 005
C     C             SUB       B             W                 5 1         W = 0022.0
C     C             SUB       D             W                             W = 0052.0
C                   Z-SUB     C             W                             W = -0032.0
C                   MULT      E             F                 3 0         F = 060
C     B             MULT      G             X                 8 4         X = 0027.7000
C     B             MULT      D             X                             X = -0200.0000
C                   DIV       B             H                 3 0         H = 007
C     C             DIV       J             Y                 6 2         Y = 0053.33
C                   MVR                     Z                 5 3         Z = 00.002
C                   SQRT      K             Z                             Z = 05.000
C                   XFOOT     L             Z                             Z = 01.600
```

図 *182.* 固定形式演算での算術演算

配列命令

配列命令を次の表に示します。

表 *107.* 配列命令

命令	従来型の構文	自由形式構文
要素の検索	798 ページの『LOOKUP (テーブルまたは配列要素の検索)』	657 ページの『%LOOKUPxx (配列要素の検索)』 または 702 ページの『%TLOOKUPxx (テーブル要素の検索)』
要素の数		637 ページの『%ELEM (要素数の検索)』
配列の転送	817 ページの『MOVEA (配列の転送)』	(許可されていない)
配列のソート	881 ページの『SORTA (配列の分類)』	881 ページの『SORTA (配列の分類)』
配列のサブセット		695 ページの『%SUBARR (配列の部分の設定/入手)』
配列要素の合計	906 ページの『XFOOT (配列要素の合計)』	707 ページの『%XFOOT (配列式要素の合計)』
配列内の最大値または最小値の検索		663 ページの『%MAXARR および %MINARR (配列内の最大要素および最小要素)』

多くの命令は配列を処理しますが、これらの命令は特定の配列機能を実行します。 機能の説明については、それぞれの命令の項を参照してください。

ビット操作

ビット命令には次のものがあります。

- 616 ページの『%BITAND (ビット単位の AND 演算)』
- 617 ページの『%BITNOT (ビットの反転)』
- 617 ページの『%BITOR (ビット単位の OR 演算)』
- 618 ページの『%BITXOR (ビット単位の排他 OR 演算)』
- 714 ページの『BITOFF (ビットをオフに設定)』
- 715 ページの『BITON (ビットをオンに設定)』
- 894 ページの『TESTB (ビットのテスト)』.

表 *108.* ビット操作

命令	従来型の構文	自由形式構文
ビットをオンに設定	BITON	%BITOR
ビットをオフに設定	BITOFF	%BITAND と %BITNOT
ビットのテスト	TESTB	%BITAND (619 ページの図 204 の例を参照)

BITOFF および BITON 命令によって、結果のフィールドに指定したフィール ドの特定のビットをオフまたはオンに変えることができます。 結果フィールドは 1 桁の文 字フィールドでなければなりません。

TESTB は、演算項目 2 で示されたビットを、結果フィールドとして指定さ れたフィールド内の対応するビットと比較します。

1 バイトの中のビットは、左から右へ番号が付けられます。 一番左のビットがビット番号 0 です。これらの命令では、演算項目 2 にはビット・パターン (ビット番号) を指定し、結果のフィールドには命令が実行される 1 バイトの文字フィールドを指定します。 演算項目 2 にビット番号を指定するためには、1 バイトの 16 進数リテラルまた は 1 バイトの文字フィールドを使用することができます。 ビット番号は、この リテラルまたはフィールドの中でオンにされるビットによって示されます。 また、ビット番号が入っている文字リテラルを演算項目 2 に指定することもで きます。

BITAND 命令では、引数内の対応するビットがすべてがオンの場合、結果のビットはオンになり、それ以外の場合は、 結果のビットはオフになります。

BITNOT 命令では、引数内の対応するビットがオフの場合、結果のビットはオンになり、それ以外の場合は、結果のビット はオフになります。

BITOR 命令では、引数内の対応するビットのいずれかがオンの場合、結果のビットはオンになり、それ以外の場合は、結 果のビットはオフになります。

BITXOR 命令では、引数内の対応するビットの 1 つのみがオンの場合、結果のビットはオンになり、それ以外の場 合は、結果のビットはオフになります。

分岐命令

分岐命令を次の表に示します。

表 *109.* 分岐命令

操作	従来型の構文	自由形式構文
比較および分岐	717 ページの『CABxx (比較および分岐)』	(許可されていない)
GO TO	788 ページの『GOTO (演算命令のスキップ)』	(許可されていない)
ITERATE	792 ページの『ITER (繰り返し)』	792 ページの『ITER (繰り返し)』

表 109. 分岐命令 (続き)		
操作	**従来型の構文**	**自由形式構文**
LEAVE	796 ページの『LEAVE (Do/For グループからの抜け出し)』	796 ページの『LEAVE (Do/For グループからの抜け出し)』
サブルーチンからの抜け出し	797 ページの『LEAVESR (サブルーチンから抜け出す)』	797 ページの『LEAVESR (サブルーチンから抜け出す)』
タグ	892 ページの『TAG (タグ)』	(許可されていない)

GOTO 命令 (TAG 命令と一緒に使用した場合) によって分岐することが できます。 GOTO 命令が現れると、プログラムは指定されたラベルに分岐します。 このラベルは、GOTO 命令の前または後に指定することができます。 ラベルは、TAG または ENDSR 命令によって指定されます。

TAG 命令は、GOTO または CABxx 命令の宛先を識別するラベルを指定します。

ITER 命令は、DO グループの中から DO グループの ENDDO ステートメント に制御を渡します。

LEAVE 命令は ITER 命令に似ていますが、LEAVE は ENDDO 命令の**次の** ステートメントに制御を渡します。

LEAVESR 命令を使用すると、サブルーチンの ENDSR 命令に制御が渡されます。

機能の説明については、それぞれの命令の項を参照してください。

呼び出し命令

呼び出し命令を次の表に示します。

表 110. 呼び出し命令		
命令	**従来型の構文**	**自由形式構文**
プログラムまたはプロシージャーの呼び出し	• 719 ページの『CALL (プログラムの呼び出し)』 • 720 ページの『CALLB (バインド・プロシージャーの呼び出し)』 • 721 ページの『CALLP (プロトタイプ・プロシージャーまたはプログラムの呼び出し)』	721 ページの『CALLP (プロトタイプ・プロシージャーまたはプログラムの呼び出し)』
パラメーターの識別	• 846 ページの『PARM (パラメーターの識別)』 • 848 ページの『PLIST (パラメーター・リストの識別)』	PI または PR 定義仕様
パラメーターの数		672 ページの『%PARMS (パラメーター数の戻り)』
パラメーターの番号		674 ページの『%PARMNUM (パラメーター番号を戻す)』
Return	867 ページの『RETURN (呼び出し元への戻し)』	867 ページの『RETURN (呼び出し元への戻し)』

CALLP はプロトタイプ呼び出しの 1 つです。 もう 1 つのタイプは式の中からの呼び出しです。 **プロトタイプ呼び出し**は、呼び出しインターフェースに定義されたプロ トタイプがある呼び出しです。 プロトタイプは、プロトタイプ定義を使用して明示的に定義することも、あるいは、呼び出しと同じモジュールにプロシージャーが定義されている場合には、 プロシージャー・インターフェースからコンパイラーが暗黙的に定義することもできます。

呼び出し命令によって、RPG IV プロシージャーは他のプログラムまたはプロシージャーに制御を渡すことができます。 しかし、プロトタイプ呼び出しは自由形式の構文を使用できる点で、 CALL および CALLB 命令とは異なります。

RETURN 命令は、呼び出し側プログラムまたはプロシージャーに制御を戻し、値があ ればその値を戻します。 PLIST および PARM 命令は、CALL および CALLB 命令と一緒に使用して、 呼び出しに対してどのパラメーターを渡すかを指示することができます。 プロトタイプ呼び出しでは、呼び出しに対してユーザーがパラメーターを渡します。

プログラムまたはプロシージャー (任意の言語で書かれた) を 呼び出すために推奨できる方式は、プロトタイプ呼び出しをコーディングする方法です。

プロトタイプ呼び出し

プロトタイプ呼び出しでは、ユーザーは (同じ構文で) 以下を呼び出すことが できます。

- 実行時にシステム上にあるプログラム
- 同じプログラムまたはサービス・プログラムにバインドされている他の モジュールまたはサービス・プログラムにエクスポートされたプロシージャー
- 同じモジュールのサブプロシージャー

呼び出しと同じモジュールにプログラムまたはプロシージャーが定義されていない場合、 呼び出しを行うプログラムまたはプロシージャーの定義仕様書にプロトタイプが含まれていなければなりません。 これはプログラムやプロシージャーを正しく呼び出し、呼び出し元が正しいパラ メーターを確実に渡すように、コンパイラーによって使用されます。

呼び出しと同じモジュールにプロシージャーが定義されている場合には、 プロトタイプを明示的に定義する必要はありません。 プロトタイプは、プロシージャーのプロシージャー・インターフェースで指定された情報を使用して、 コンパイラーが暗黙的に定義することができます。

プログラムまたはプロシージャーがプロトタイプの場合には、そのプログラム またはプロシージャーに使用するデータ項目の名前を知っている必要はなく、パラメーターの数とタイプだけ知っていれば済みます。

プロトタイプは、プログラムまたはプロシージャー間の通信を改善します。 プロトタイプ呼び出しを使用する利点のいくつかは次のとおりです。

- PARM または PLIST 命令が必要ないので、構文が簡単になります。
- 一部のパラメーターでは、リテラルや式を渡すことができます。
- プロシージャーを呼び出す場合に、操作記述子が必要かどうかを覚えてい る必要がありません。
- 呼び出しが間違っていた場合、コンパイラーがコンパイル時にエラーを示すことによって、 正しいタイプ、形式、および長さのパラメーターを渡すのが容易になります。
- コンパイラーが実行時に変換を実行することによって、一部のタイプのパラメーターについて正しい形式および長さのパラメーターを渡すのが容易になります。

564 ページの図 183 に、プロトタイプ ProcName を使用して、3 つのパラメーターを渡す例を示します。 プロトタイプ ProcName は、プログラムまたはプロシージャーを参照すること ができます。 呼び出しを行う時にこのことを認識しているは重要でなく、プロトタイプを定義する時にのみ重要です。

```
/FREE
// The following calls ProcName with the 3
// parameters CharField, 7, and Field2:
     ProcName (CharField: 7: Field2);

// If you need to specify operation extenders, you must also
// specify the CALLP operation code:
     CALLP(e) ProcName (CharField: 7: Field2);
/END-FREE
```

図 *183. CALLP* 命令の例

式の中でプロシージャーを呼び出す場合には、指定された戻り値のデータ・タイプに合った方法でプロシージャー名を使用してください。 例えば、プロシージャーが数値を戻すように定義された場合には、式の中での プロシージャーの呼び出しは、数値の必要な個所でなければなりません。

プログラムおよびプロシージャーの呼び出しおよびパラメーターの受け渡しについて詳しくは、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」の該当する章を参照してください。プロトタイプおよびパラメーターの定義について詳しくは、224 ページの『プロトタイプおよびパラメーター』を参照してください。

操作記述子

プロシージャーにパラメーターを渡すことは必要ですが、それでも呼び出され たプロシージャーにデータ・タイプの詳細がわからない場合 (例えば、別の タイプのストリングなど) があります。 そのような場合には、パラメーターの形式に関係なく、操作記述子を使用して 呼び出されたプロシージャーに記述情報を提供することができます。 この追加情報によって、プロシージャーはストリングを正しく解釈することができます。 操作記述子を使用するのは、呼び出されたプロシージャーに必要な場合だけに してください。

操作記述子は、プロトタイプと非プロトタイプの両方のパラメーターに要求す ることができます。 プロトタイプ呼び出しの場合、キーワード OPDESC をプロトタイプ定義に指定します。 非プロトタイプ・パラメーターの場合には、CALLB 命令の命令コード拡張子と して (D) を指定します。 いずれの場合にも、操作記述子は呼び出しプロシージャーによって作成されて、 呼び出されたプロシージャーに隠されたパラメーターとして渡されます。

独自のプロシージャーに OPDESC キーワードを指定している場合には、 API を呼び出して、パラメーターの長さとタイプの情報を判別できます。 これらの API では、対象のパラメーターを識別するためのパラメーター番号を渡す必要があります。 通常は、プロトタイプまたはプロシージャー・インターフェースでパラメーターを単に数えることで、パラメーターの番号を入手できます。 ただし、RTNPARM キーワードが指定されている場合には、各パラメーターの番号が、見かけ上の番号より 1 つ大きくなります。 数値リテラルを使用せずに、特定パラメーターの番号を取得するには、%PARMNUM 組み込み関数を使用します。 詳しくは、454 ページの『OPDESC』、477 ページの『RTNPARM』、および 674 ページの『%PARMNUM (パラメーター番号を戻す)』を参照してください。

呼び出しのプログラム名の解析

プログラム名は、CALL 命令の演算項目 2 で、あるいは、プロトタイプまたはプロシージャー・インターフェースの EXTPGM キーワードのパラメーターとして指定します。ライブラリー名を指定する場合は、直後にスラッシュを付けてプログラム名を続ける必要があります (例えば「LIB/PROG」)。 ライブラリーを指定しない場合は、プログラムを見つけるためにライブラリー・リストが使用されます。*CURLIB はサポートされません。

次の規則に注意してください。

- スラッシュを含むフィールドまたは名前のついた定数のブランク以外 のデータの合計長は 21 桁を超えることができません。
- プログラムまたはライブラリー名が 10 桁を超えると、10 桁で切り捨てら れます。

プログラム名は、リテラル、フィールド、名前のついた定数、または配列 要素に指定されたとおりに正確に使用する、呼び出すプログラムが判別 されます。 特に、次のことに注意してください。

- 先行または後書きのブランクは無視されます。
- 項目の最初の文字がスラッシュの場合には、プログラムを見付けるのに、 ライブラリー・リストが使用されます。
- 項目の最後の文字がスラッシュの場合には、コンパイル時メッセージが 出されます。
- 小文字は大文字にシフトされません。
- 引用符に囲まれた名前 (例えば、「“ABC”」) には、常に呼び出されるプログラムの名前の一部として引用符が含まれます。

プログラム参照は、目的プログラムへの分析解決のオーバーヘッドを避けるた めに、グループ化されます。 名前のついた定数またはリテラルを使用する特定のプログラムに対する参 照は、すべてそのプログラムが

1 回だけ分析解決されて、 そのプログラムに対する (名前のついた定数またはリテラルのみによる) 以後のすべての参照で分析解決が行われないようにグループ化されます。

プログラム参照は、 プログラムとライブラリー名の両方が同じ場合には、グループ化されます。 変数名によるプログラム参照はすべて変数名によってグループ化されます。 プログラム参照が変数で行われる場合には、その現在の値がその変数を使用 した前のプログラム参照で使用された値と比較されます。 値が変わっていない 場合には、分析解決は行われません。 値が変わっている場合には、指定された新しいプログラムに対して分析解決が 行われます。 この規則が適用されるのは、変数名を使用する参照だけであることに注意して ください。 名前のついた定数またはリテラルを使用する参照は解決されず、変数によるプログラム参照が再び解決されるかどうかに影響を与えません。566 ページの図 184 に、プログラム参照のグループ化を示します。

プログラム *CALL* の例

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D Pgm_Ex_A        C                   'LIB1/PGM1'
D Pgm_Ex_B        C                   'PGM1'
D PGM_Ex_C        C                   'LIB/PGM2'
 *
```

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
C                   CALL      Pgm_Ex_A
 *
 * The following two calls will be grouped together because both
 * have the same program name (PGM1) and the same library name
 * (none). Note that these will not be grouped with the call using
 * Pgm_Ex_A above because Pgm_Ex_A has a different library
 * name specified (LIB1).
 *
C                   CALL      'PGM1'
C                   CALL      Pgm_Ex_B
 *
 * The following two program references will be grouped together
 * because both have the same program name (PGM2) and the same
 * library name (LIB).
 *
C                   CALL      'LIB/PGM2'
C                   CALL      Pgm_Ex_C
 *
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 * The first call in the program using CALLV below will result in
 * a resolve being done for the variable CALLV to the program PGM1.
 * This is independent of any calls by a literal or named constant
 * to PGM1 that may have already been done in the program.  The
 * second call using CALLV will not result in a resolve to PGM1
 * because the value of CALLV has not changed.
 *
C                   MOVE      'PGM1'         CALLV          21
C                   CALL      CALLV
C                   CALL      CALLV
```

図 *184.* プログラム参照のグループ化の例

システム組み込み名の解析

バインド済みの呼び出しで指定されたリテラルまたは名前の付いた定数が「CEE」または下線 ('_') で始まっている場合、コンパイラーはこれをシステム組み込みとして処理します (バインド済み呼び出しは、CALLB またはプロトタイプで EXTPGM が指定されていない プロトタイプ呼び出しのいずれかになります)。

実際にシステム組み込みでない場合には、リストに警告メッセージが現れます。 この警告は無視することができます。

API について詳しくは、IBM i Information Center「プログラミング」のカテゴリーを参照してください。システム提供の API との混同を避けるために、ユーザーのプロシージャーに "CEE" で始まる名前を付けないでください。

*ROUTINE の値

呼び出しが正常に行われないときには、プログラム状況データ構造 (PSDS) の *ROUTINE サブフィールドの内容が以下によって更新されます。

- 外部呼び出しでは、呼び出されたプログラムの名前 (すなわち、プログラム への CALL または CALLP の)。
- バインド済み静的呼び出しでは、呼び出されたプロシージャーの名前。
- プロシージャー・ポインター呼び出しでは、*N。

このサブフィールドのサイズは 8 バイトの長さしかないので、名前が切り捨て られる場合があることに注意してください。

比較命令

比較命令を次の表に示します。

表 *111.* 比較命令

命令	従来型の構文	自由形式構文
また	713 ページの『ANDxx (かつ)』	AND 演算子
比較	739 ページの『COMP (比較)』	=、<、>、<=、>=、または <> 演算子
比較および分岐	717 ページの『CABxx (比較および分岐)』	(許可されていない)
条件付きサブルーチン	724 ページの『CASxx (サブルーチンの条件付き呼び出し)』	789 ページの『IF (If)』 および 781 ページの『EXSR (サブルーチンの呼び出し)』
DO UNTIL	760 ページの『DOU (条件が真になるまでの繰り返し)』 または 761 ページの『DOUxx (条件までの繰り返し)』	760 ページの『DOU (条件が真になるまでの繰り返し)』
DO WHILE	763 ページの『DOW (条件が真の間繰り返し)』 または 763 ページの『DOWxx (条件が真の間繰り返し)』	763 ページの『DOW (条件が真の間繰り返し)』
IF	789 ページの『IF (If)』 または 790 ページの『IFxx (満たされた条件の処理)』	789 ページの『IF (If)』
または	843 ページの『ORxx (または)』	OR 演算子
WHEN	902 ページの『WHEN (真の場合に選択)』 または 903 ページの『WHENxx (真の場合に選択)』	902 ページの『WHEN (真の場合に選択)』

ANDxx、CABxx、CASxx、DOUxx、DOWxx、IFxx、ORxx、および WHENxx 命令に おける xx には、以下を使用することができます。

xx
意味

GT
演算項目 1 は演算項目 2 より大きい。

LT
演算項目 1 は演算項目 2 より小さい。

EQ
演算項目 1 は演算項目 2 と等しい。

NE
演算項目 1 は演算項目 2 と等しくない。

GE
演算項目 1 は演算項目 2 より大きいか等しい。

LE
演算項目 1 は演算項目 2 より小さいか等しい。

ブランク
無条件実行 (CASxx または CABxx)。

比較命令では、フィールドの命令で指定された条件がテストされます。 これらの命令ではフィールドの値は変更されません。 COMP、CABXX、および CASXX の場合には、71 桁と 76 桁に割り当てられた結果 の標識がこの命令の結果に従って設定されます。 すべてのデータ・タイプが、同じデータ・タイプのフィールドと比較されます。

比較命令の使用にあたっては、以下のことを忘れないでください。

- 数値フィールドが比較される場合には、長さが等しくないフィールドは暗 黙の小数点に位置合わせされます。 各フィールドには、フィールドの長さと小数点以下の桁数を同じにして比較で きるように、小数点の左または右にゼロが埋められます。
- 数値の比較はすべて代数で行われます。 プラス (+) の値は常にマイナス (-) の値より大きくなります。
- プログラムのコンパイルで FIXNBR(*ZONED) 制御仕様書の キーワードまたはコマンド・パラメーターが使用されている場合には、 ゾーン数値フィールド内のブランクはゼロと見なされます。
- 文字フィールド、図形フィールド、または UCS-2 フィールドが比較される場合には、長さが等しくないフィールドはその左端文字に位置合わせされます。 短いフィールドには、フィール ドの長さを同じにして比較できるように、長い方のフィールドの長さに等しく なるまでブランクが埋められます。
- 日付フィールドは、比較するときに共通の形式に変換されます。
- 比較する時に時刻フィールドは共通の形式に変換されます。
- 配列名は比較命令では指定できませんが、配列要素は指定することが できます。
- DOUxx、DOWxx、IFxx、および WHENxx 命令と一緒に ANDxx および ORxx 命 令を使用することができます。
- 長さがゼロの文字リテラル、図形リテラル、または UCS-2 リテラルを、ブランクを含んでいるフィールド (固定または可変) と比較した結果は、等しくなります。 値の長さが 0 であるかどうかをテストするには、%LEN 組み込み関数を使用してください。 例については、207 ページの図 58 を参照してください。

注意!

予測できない結果を避けたい場合には、特に次の点に注意してください。

- UCS-2 データにおける文字の順序は、文字または図形データに関するものであるため、必ずしも同じになるわけではありません。 例えば、UCS-2 の場合は '2' が 'A' より小さくなりますが、文字比較の場合は '2' が 'A' より大きくなります。 比較演算において UCS-2 への暗黙的な変換が実行される場合や、文字タイプまたは図形タイプではなく UCS-2 タイプとなるようにフィールドの一部を変更した場合には、大なり比較または小なり比較の結果が、期待したものとは異なることがあります。
- すべての図形および UCS-2 の比較はデータの 16 進数表現を使用して 実行されます。 代替順序は使用されません。
- 文字フィールドの比較に (制御仕様書の 322 ページの『ALTSEQ {(*NONE | *SRC | *EXT) }』 キーワードを使用して) 代替照合順序が指定されている場合、被比較数は代替照合順序に変換されてから比較されます。 比較に *HIVAL または *LOVAL が 使用されている場合には、比較操作の前に代替照合順序によって値が変更される 場合があります。 いずれかの被比較数が定義仕様書の ALTSEQ(*NONE) キーワードによって 定義されている場合、代替照合順序は使用されないことに注意してください。
- 基底ポインターを *NULL (または値が *NULL の基底ポインター) と比較する 場合に、予測可能な結果が得られる唯一の比較は等号および不等号についての 場合だけです。
- より小またはより大のポインターを比較する時に予測可能な結果が得られ るのは、ポインターが連続記憶域内のアドレスを指示している場合だけです。 例えば、すべてのポインターが 1 つの *USRSPC の複数

のアドレスに設定され ているか、またはすべてのポインターが 1 つの配列の複数の配列要素のアドレスに設定されている場合です。

- プロシージャー・ポインター・フィールドが等号または不等号以外のもの について比較されると、結果は予測できないものとなります。
- 浮動値は等号または不等号について比較しないでください。その理由は 浮動値の保管方法にあります。このような比較をする代わりに、2 つの値の差の絶対値を、非常に小さい値と比較します。

変換命令

次の組み込み関数は変換命令を実行します。

- 620 ページの『%CHAR (文字データへの変換)』
- 629 ページの『%DEC (パック 10 進数への変換)』
- 630 ページの『%DECH (四捨五入を伴うパック 10 進数形式への変換)』
- 634 ページの『%EDITC (編集コードを使用する編集値)』
- 636 ページの『%EDITFLT (浮動外部表現への変換)』
- 636 ページの『%EDITW (編集語を使用する編集値)』
- 643 ページの『%FLOAT (浮動形式への変換)』
- 647 ページの『%GRAPH (図形値への変換)』
- 651 ページの『%INT (整数形式への変換)』
- 652 ページの『%INTH (四捨五入を伴う整数形式への変換)』
- 705 ページの『%UCS2 (UCS-2 値への変換)』
- 706 ページの『%UNS (符号なし形式への変換)』
- 706 ページの『%UNSH (四捨五入を伴う符号なし形式への変換)』

これらの組み込み関数は、従来型の構文と自由形式構文の両方で使用可能です。

従来の MOVE 命令コードおよび MOVEL 命令コードは、演算項目 2 と結果のフィールドが違うタイプのときに変換を実行します。 次を参照してください。

- 803 ページの『MOVE (転送)』
- 824 ページの『MOVEL (左につめて転送)』

組み込み関数を使用して文字値を数値に変換するための規則

%DEC、%DECH、%FLOAT、%INT、%INTH、%UNS、または %UNSH の最初のパラメーターが文字式である場合、次の規則が適用されます。

データが DATA-INTO および XML-INTO によって数値フィールドに対して処理される場合にも、この規則が適用されます。 以下の説明で、%FLOAT に関連する規則は浮動フィールドに適用されます。

- 無効な数値データが検出された場合、例外が発生し、状況コード 105 が戻されます。
- 符号は任意指定です。 「+」または「-」を使用できます。%FLOAT の場合、数値データの前に置く必要があります。 その他の組み込み関数の場合、数値データの前または後に置くことができます。
- 小数点は任意指定です。

  制御キーワード EXPROPTS(*USEDECEDIT) が指定されていない場合、小数点はピリオドまたはコンマのいずれかになります。

  制御キーワード EXPROPTS(*USEDECEDIT) が指定されている場合、小数点は DECEDIT 制御キーワードによって指定された文字である必要があります。
- ブランクはデータの任意の場所で使用できます。 たとえば、' + 3 ' は有効なパラメーターです。 ただし、制御キーワード EXPROPTS(*ALWBLANKNUM) が指定されていない限り、データを完全にはブランクにすることはできません。
- 2 番目と 3 番目のパラメーターは必須です。

- 浮動小数点データ (例えば「1.2E6」) は、%FLOAT に対してのみ許可されます。
- %FLOAT の場合、指数はオプションです。 'E' でも 'e' でもかまいません。 指数の符号は任意指定です。 指数の数値部分の前に存在する必要があります。
- 制御キーワード EXPROPTS(*USEDECEDIT) が指定されている場合は、桁区切り文字 (千単位の区切り文字など) を使用できます。 桁区切り文字は、DECEDIT 制御キーワードによって異なります。

  デフォルト、および DECEDIT('.') または DECEDIT('0.')、あるいはジョブ DECFMT の値が J でない場合の DECEDIT(*JOBRUN) では、ピリオドが小数点文字で、コンマは数字区切り文字です。

  DECEDIT(',') または DECEDIT('0,')、あるいはジョブ DECFMT の値が J の場合の DECEDIT(*JOBRUN) では、コンマが小数点文字で、ピリオドは桁区切り文字です。

  例えば、DECEDIT(',') が指定されている場合、小数点文字はコンマで、区切り文字はピリオドです。 EXPROPTS(*USEDECEDIT) も指定されている場合、%DEC('1.234.567,89') は 1234567.89 を戻します。

  以下の規則が桁区切り文字に適用されます。

  – 桁区切り文字はオプションです。
  – 桁区切り文字の前後は必ず数値とします。
  – 桁区切り文字は、小数点の前に置くことも、後に置くこともできます。
  – 例えば、小数点文字がピリオドで、桁区切り文字がコンマの場合は、次のようになります。
    -「1.2」、「1,2.3」、「1.23」は有効です。 桁区切り文字は、各ケースの 2 桁を区切ります。
    -「1 . 2」、「1 , 2 . 3」、「1 . 2 , 3」も有効です。 ブランクは無視されます。
    -「,1」は無効です。 桁区切り文字の前に数字が付いていないからです。
    -「1,」は無効です。 桁区切り文字の後に数字が続いていないからです。
    -「1.,2」は無効です。 桁区切り文字の前に数字が付いていないからです。
    -「1,.2」は無効です。 桁区切り文字の後に数字が続いていないからです。

例

以下の例では、キーワード EXPROPTS(*USEDECEDIT) が指定されていません。 ピリオド (.) およびコンマ (,) は、どちらの文字も小数点を表すと見なされます。

1. キーワード EXPROPTS(*USEDECEDIT) が指定されていません。
2. パラメーター「1.2」は 1.2 と解釈されます。
3. パラメーター「1,2」も 1.2 と解釈されます。

```
CTL-OPT;                // 1

DCL-S num PACKED(5:2);

num = %DEC('1.2');      // = 1.2 2
num = %DEC('1,2');      // = 1.2 3
```

以下の例では、キーワード EXPROPTS(*USEDECEDIT) が指定されています。 DECEDIT キーワードは指定されていませんが、デフォルトの DECEDIT('.') に設定されます。ピリオド (.) は小数点を表すと見なされ、コンマ (,) は桁区切り文字を表すと見なされます。

1. キーワード EXPROPTS(*USEDECEDIT) が指定されています。
2. パラメーター「1.2」は 1.2 と解釈されます。
3. パラメーター「1,2」は 12 と解釈されます。 コンマ (,) は桁区切り文字であるため、無視されます。

```
CTL-OPT EXPROPTS(*USEDECEDIT); // 1

DCL-S num PACKED(5:2);

num = %DEC('1.2');     // = 1.2 2
num = %DEC('1,2');     // = 12  3
```

以下の例では、キーワード EXPROPTS(*USEDECEDIT) および DECEDIT(',') が指定されています。 コンマ (,) は小数点を表すと見なされ、ピリオド (.) は桁区切り文字を表すと見なされます。

1. キーワード EXPROPTS(*USEDECEDIT) および DECEDIT(',') が指定されています。
2. パラメーター「1.2」は 12 と解釈されます。 ピリオド (.) は桁区切り文字であるため、無視されます。
3. パラメーター「1,2」は 1.2 と解釈されます。

```
CTL-OPT EXPROPTS(*USEDECEDIT) DECEDIT(','); // 1

DCL-S num PACKED(5:2);

num = %DEC('1.2');     // = 12  2
num = %DEC('1,2');     // = 1.2 3
```

データ域命令

データ域命令には以下のものがあります。

- 791 ページの『IN (データ域の検索)』
- 845 ページの『OUT (データ域の書き出し)』
- 899 ページの『UNLOCK (データ域のアンロックまたはレコードの解放)』.

これらの命令は、従来型の構文と自由形式構文の両方で使用可能です。

IN および OUT 命令では、演算項目 2 の指定によって、プログラム内の 1 つ またはすべてのデータ域の検索および書き出しを行うことができます。

また、IN および OUT 命令では、ユーザーがデータ域のロックやアンロック を制御することもできます。 データ域がロックされている場合には、他のプログラムまたはプロシージャー によって読み取ることはできますが、更新することはできません。

次のロック状態が使用されます。

- *LOCK が指定された IN 命令の場合には、データ域が読み取り可占有ロック 状態になります。
- *LOCK が指定された OUT 命令の場合には、データ域は書き出し操作の後で ロックされたままになります。
- ブランクが指定された OUT 命令の場合には、データ域は更新された後で アンロックされます。
- データ域をアンロックしてレコードのロックを解除するために UNLOCK を 使用した場合には、データ域またはレコード (あるいはその両方) が更新され ません。

データ域との実際のデータの転送中には、データ域にシステムの内部ロック があります。 複数のユーザーが同じデータ域に対して競合している場合には、ユーザーは、データ域が使用できないことを示すエラー・メッセージを受け取ります。

IN、OUT、および UNLOCK 命令を使用する場合には、次のことに留意してく ださい。

- データ域命令は、オペレーティング・システムに定義されていないデータ 域には実行することができません。
- データ域に対して IN、OUT、および UNLOCK 命令を実行する前に、そのデータ域の定義仕様書に DTAARA キーワードを指定するか、または *DTAARA DEFINE ステートメントの結果のフィールドにそのデータ域

を指定する必要があります。 (DEFINE ステートメントについて詳しくは、754 ページの『DEFINE (フィールド定義)』 を参照してください。)

- ロックされたデータ域を別の RPG プログラムで更新したりロックすること はできませんが、このデータ域は演算項目 1 をブランクにして IN 命令で検索 することができます。
- データ域の名前を、複数オカレンス・データ構造、入力レコード・フィールド、 配列、配列要素、またはテーブルの名前にすることはできません。
- データ域を、複数オカレンス・データ構造、データ域データ構造、プログラム 状況データ構造、ファイル情報データ構造 (INFDS)、または *DTAARA DEFINE ステートメントに現れるデータ構造のサブフィールドにすることはできませ ん。
- データ域の名前が実行時に判別される場合、 DTAARA(*VAR) キーワードが使用中であるため、その名前を含む変数は IN 命令の実行前に設定する必要があります。 以前に実行した *LOCK IN 命令が原因でデータ域がロックされる場合、そのデータ域に対する他の命令 (IN、OUT、UNLOCK) は、以前にロックされたデータ域を使用し、データ域の名前を含む変数を調べません。
- ライブラリー名が DTAARA キーワードで指定されていない場合には、 データ域を探すためにライブラリー・リストが使用されます。

データ域データ構造はプログラムの初期化時に自動的に読み取られてロックされ、プログラムが LR オンで終了したときにデータ構造の内容がデータ域に 書き込まれます。 データ域データ構造のためのデータ域が見付からない場合、 初期値としてブランクを使用して、データ域が作成されます。 データ域を見付けるためにライブラリー・リストが検索された場合、 新規データ域は QTEMP 内に作成されます。

自由形式では、DTAARA(*AUTO) キーワードは、データ構造がデータ域データ構造である ことを指定します。 固定形式では、定義仕様の 23 桁目に U を記入して定義された データ構造は、そのデータ構造がデータ域であることを示します。

場合によっては、 IN、OUT、および UNLOCK 命令コードを使用して、データ域に対してさらに命令を指定することができます。 自由形式の定義の場合、パラメーターとして *USRCTL も指定します。 固定形式の定義の場合、DTAARA キーワードを指定します。

データ域データ構造のためのデータ域が見付からない場合、 初期値としてブランクを使用して、データ域が作成されます。 データ域を見付けるためにライブラリー・リストが検索された場合、 新規データ域は QTEMP 内に作成されます。

内部データ域 (*LDA) を定義するために、次のいずれかを行うことができ ます。

- データ域の定義仕様書に DTAARA(*LDA) キーワードを指定する。
- データ域の固定形式の定義仕様書に UDS を指定し、名前をブランクにする。
- 自由形式の定義に名前として *N を指定し、名前なしで DTAARA キーワードを指定する。
- *DTAARA DEFINE ステートメントの演算項目 2 に *LDA を指定する。

*PDA を定義するためには、データ域の定義仕様書に DTAARA(*PDA) キーワー ドを指定するか、あるいは *DTAARA DEFINE ステートメントの演算項目 2 に *PDA を指定することができます。

日付命令

日付命令を次の表に示します。

表 *112.* 日付命令

命令	従来型の構文	自由形式構文
日付の加算	711 ページの『ADDDUR (期間の加算)』	+ 演算子
抽出	782 ページの『EXTRCT (日付/時刻/タイム・スタンプの抽出)』	697 ページの『%SUBDT (日付、時刻、またはタイム・スタンプの一部の取り出し)』
日付/時刻の期間の減算	887 ページの『SUBDUR (期間減算)』	- 演算子または 632 ページの『%DIFF (2 つの日付、時刻、またはタイム・スタンプ値の差)』

表 112. 日付命令 (続き)		
命令	**従来型の構文**	**自由形式構文**
日付/時刻/タイム・スタンプから文字への変換	803 ページの『MOVE (転送)』 または 824 ページの『MOVEL (左につめて転送)』	620 ページの『%CHAR (文字データへの変換)』
日付/時刻/タイム・スタンプから数値への変換	803 ページの『MOVE (転送)』 または 824 ページの『MOVEL (左につめて転送)』	629 ページの『%DEC (パック 10 進数への変換)』
文字/数値から日付への変換	803 ページの『MOVE (転送)』 または 824 ページの『MOVEL (左につめて転送)』	628 ページの『%DATE (日付への変換)』
文字/数値から時刻への変換	803 ページの『MOVE (転送)』 または 824 ページの『MOVEL (左につめて転送)』	700 ページの『%TIME (時刻への変換)』
文字/数値/日付からタイム・スタンプへの変換	803 ページの『MOVE (転送)』 または 824 ページの『MOVEL (左につめて転送)』	701 ページの『%TIMESTAMP (タイム・スタンプへの変換)』
日付/時刻のタイム・スタンプへの移動	803 ページの『MOVE (転送)』 または 824 ページの『MOVEL (左につめて転送)』	日付 + 時刻
テスト	893 ページの『TEST (日付/時刻/タイム・スタンプのテスト)』	893 ページの『TEST (日付/時刻/タイム・スタンプのテスト)』
年数		709 ページの『%YEARS (年数)』
月数		667 ページの『%MONTHS (月数)』
日数		629 ページの『%DAYS (日数)』
時間数		651 ページの『%HOURS (時間数)』
分数		667 ページの『%MINUTES (分数)』
秒数		686 ページの『%SECONDS (秒数)』
マイクロ秒数		668 ページの『%MSECONDS (マイクロ秒数)』

日付命令により、日付フィールド、時刻フィールド、およびタイム・スタンプ・フィールド、さらには日付、時刻、タイム・スタンプを表す文字フィールドあるいは数字フィールドを処理することができます。次のことができます。

- 年、月、日、時間、分、秒、あるいはマイクロ秒の単位での期間の加算または減算
- 2 つの日付、時刻、またはタイム・スタンプの間の期間の判別
- 日付、時刻、またはタイム・スタンプの値からの抽出 (例えば、日)
- 日付、時刻、またはタイム・スタンプの値が有効かどうかのテスト

期間を加算または減算するには、自由形式構文では + 演算子または - 演算子を使用でき、従来型の構文では ADDDUR 命令コードまたは SUBDUR 命令コードを使用できます。 下記の表では、自由形式構文で使用する組み込み関数と、従来型の構文で使用する期間コードを示します。

表 113. 組み込み関数と期間コード		
装置	**組み込み関数**	**期間コード**
年	%YEARS	*YEARS または *Y

表 113. 組み込み関数と期間コード (続き)		
装置	**組み込み関数**	**期間コード**
月	%MONTHS	*MONTHS または *M
日	%DAYS	*DAYS または *D
時間	%HOURS	*HOURS または *H
分	%MINUTES	*MINUTES または *MN
秒	%SECONDS	*SECONDS または *S
マイクロ秒	%MSECONDS	*MSECONDS または *MS

例えば、次のどちらの方法でも、既存の日付に 23 日を加算することができます。

```
C                   ADDDUR    23:*D         DUEDATE

 /FREE
     newdate = duedate + %DAYS(23)
 /END-FREE
```

2 つの日付、時刻、またはタイム・スタンプの間の期間を計算するには、自由形式構文では %DIFF 組み込み関数を使用でき、従来型の構文では SUBDUR 命令コードを使用できます。どちらの場合にも、573 ページの表 113 に示す期間コードのいずれかを指定する必要があります。

期間は完全な単位として渡され、剰余は廃棄されます。 59 分という期間を時間単位で表すと 0 になります。 61 分という期間を時間単位で表すと 1 になります。

次の表では、SUBDUR 命令コードを使用した追加の例を示します。 %DIFF 組み込み関数でも同じ結果を得られます。

表 114. *SUBDUR* を使用した結果の期間			
期間の単位	**演算項目 1**	**演算項目 2**	**結果**
月	1999-03-28	1999-02-28	1 月
	1999-03-14	1998-03-15	11 か月
	1999-03-15	1998-03-15	12 か月
年	1999-03-14	1998-03-15	0 年
	1999-03-15	1998-03-15	1 年
	1999-03-14-12.34.45.123456	1998-03-14-12.34.45.123457	0 年
時間	1990-03-14-23.00.00.000000	1990-03-14-22.00.00.000001	0 時間

予期しない結果

1 か月には 28、29、30、または 31 日を含めることができます。 1 年は 365 日または 366 日を含めることができます。 これらの不整合のために、次のような命令では予期しない結果となる場合があります。

- 日付に月の 29 日、30 日、31 日を指定しての月の数の加算または減算 (あるいは月単位の期間の計算)
- 2 月 29 日の日付が指定されている年の数の加算または減算 (あるいは年単位での期間の計算)

使用される規則は次のとおりです。

- 月または年を加算ないしは減算する時、日の部分は変更されないままということも起こりえます。 例えば、2000-03-15 + %MONTHS(1) は 2000-04-15 となります。
- 加算または減算によって実在しない日付が生成された場合 (例えば 4 月 31 日) は、代わりにその月の最後の日付が使用されます。

- 日の部分を変更する月または年の命令では、逆に戻すことは**できません**。 例えば、2000-03-31 + %MONTHS(1) は 2000-04-30 であり、日が 31 から 30 に変更されます。 1 か月を減算しても、元の 2000-03-31 に戻すことはできません。

  命令 2000-03-31 + %MONTHS(1) - %MONTHS(1) は 2000-03-30 になります。
- 2 つの日付の間の期間は、2 つめの日付から 1 か月を引くと 1 つめの日付になる場合に 1 か月になります。 例えば、2000-04-30 - %MONTHS(1) は (2000-03-31 ではなく) 2000-03-30 であるため、2000-03-31 と 2000-04-30 との間の期間は月単位 (端数切り捨て) では 0 となります。

宣言命令

宣言命令を次の表に示します。

表 *115.* 宣言命令

命令	従来型の構文	自由形式構文
フィールド定義	754 ページの『DEFINE (フィールド定義)』	定義仕様書における LIKE キーワードまたは DTAARA キーワード
キーの定義	• 794 ページの『KFLD (キーの構成部分定義)』 • 795 ページの『KLIST (複合キーの定義)』	(許可されていない)
パラメーターの識別	• 846 ページの『PARM (パラメーターの識別)』 • 848 ページの『PLIST (パラメーター・リストの識別)』	PR 定義仕様
タグ	892 ページの『TAG (タグ)』	(許可されていない)

宣言命令では (任意指定の演算項目 1 または 2 を持つ PARM の場合を除き) 処置は行われません。宣言命令は、演算中の任意の個所に指定することがで きます。 宣言命令は、フィールドの特性を宣言したり、プログラムの各部分にマークを 付けるために使用します。 制御レベルの指定 (7 から 8 桁目) はブランクにすることも、 プログラムの適切なセクション内のステートメントをグループにまとめる指定を入れる ことができます。

DEFINE 命令は、別のフィールドの属性 (長さおよび小数点以下の桁数) に 基づいてフィールドを定義するか、あるいはフィールドをデータ域として定義 します。

KLIST および KFLD 命令は、複合キー・フィールドを参照する名前および複 合キーを構成するフィールドを指示するために使用します。 複合キー とは、キー・フィールドのリストを含むキーのことです。 複合キーは、左から右へと作成され、最初に指定した KFLD が複合キーの左端 (高位) のフィールドになります。

PLIST および PARM 命令は、呼び出されたプログラムまたはプロシージャー が呼び出し側プログラムまたはプロシージャーからパラメーターにアクセスできる ように、CALL および CALLB 命令と一緒に使用されます。

TAG 命令は、GOTO や CABxx などの分岐命令の宛先を指定します。

エラー処理命令

例外処理命令コードには次のものがあります。

- 802 ページの『MONITOR (監視グループの始め)』
- 838 ページの『ON-ERROR (エラーの時)』
- ENDMON (770 ページの『ENDyy (構造化グループの終わり)』 を参照)
- 839 ページの『ON-EXIT (終了時)』

これらの命令コードは、従来型の構文と自由形式構文の両方で使用可能です。

MONITOR、ON-ERROR および ENDMON は、監視グループをコーディングするために使用されます。 監視グループは、監視ブロックに 1 つまたは複数の ON-ERROR ブロックが続き、さらに ENDMON が続いて構成されています。

監視ブロックには、エラーを発生させる可能性があると思われるコーディングを入れます。 ON-ERROR ブロックには、監視ブロック内で発生するエラーを処理するコーディングを入れます。

監視ブロックは、MONITOR 命令と、それに続く監視の対象となる命令で構成されます。 ON-ERROR ブロックは、状況コードのリストと、それに続けて、リストされている状況コードが監視ブロックの中でのエラーによって生成された場合に実行されるべき命令で構成されます。

監視ブロック内でエラーが発生し、命令に (E) 拡張またはエラー標識がある場合、エラーはその (E) 拡張またはエラー標識によって処理されます。 エラーを処理できる標識または拡張がない場合、制御は、そのエラーに対応する状況コードを含んでいる ON-ERROR ブロックに渡されます。 ON-ERROR ブロックが終了すると、制御は ENDMON に渡されます。 エラーを処理する ON-ERROR ブロックがない場合は、制御は例外処理の次のレベル (*PSSR または INFSR サブルーチン、またはデフォルトのエラー処理プログラム) に渡されます。

ON-EXIT は、正常終了か異常終了かに関わらずプロシージャーが終了した時点で実行されるコードのセクションを開始するために使用されます。

```
/free
   MONITOR;                          _
       OPEN   FILE;                   |
       DOW    getNextRecord ();       |
         X = X + 1;                    +-- This is the monitor block
         nameList(X) = name;           |
       ENDDO;                          |
       CLOSE  FILE;                  _|
   ON-ERROR   1216;                  _
       DSPMSG                         |
          ('Error opening file FILE'  |
         : %status);                   +-- First on-error block
       RETURN;                       _|
   ON-ERROR   121;                   _
       DSPMSG                         |
          ('Array NAME is too small'   +-- Second on-error block
         : %status);                   |
       RETURN;                       _|
   ON-ERROR   *ALL;                  _
       DSPMSG                         |
          ('Unexpected error'          +-- Final catch-all on-error block
         : %status);                   |
       RETURN;                       _|
   ENDMON;                           --- End of MONITOR group
/end-free
```

図 *185. MONITOR* ブロックおよび *ON-ERROR* ブロックの例

ファイル操作

ファイル命令コードには次のものがあります。

- 710 ページの『ACQ (獲得)』
- 728 ページの『CHAIN (ファイルからのランダム検索)』
- 737 ページの『CLOSE (ファイルのクローズ)』
- 738 ページの『COMMIT (コミット)』
- 757 ページの『DELETE (レコードの削除)』
- 779 ページの『EXCEPT (演算時出力)』
- 780 ページの『EXFMT (形式の書き出し、その後読み取り)』
- 783 ページの『FEOD (データの強制終了)』
- 788 ページの『FORCE (次のサイクルでのファイルの強制読み取り)』

- 834 ページの『NEXT (次の入力の取り出し)』
- 842 ページの『OPEN (処理のためのファイルのオープン)』
- 849 ページの『POST (転記)』
- 851 ページの『READ (レコードの読み取り)』
- 852 ページの『READC (次の変更レコードの読み取り)』
- 853 ページの『READE (等しいキーのレコードの読み取り)』
- 856 ページの『READP (前のレコードの読み取り)』
- 857 ページの『READPE (等しいキーの前のレコードの読み取り)』
- 861 ページの『REL (解放)』
- 869 ページの『ROLBK (ロールバック)』
- 873 ページの『SETGT (より大きい設定)』
- 876 ページの『SETLL (下限の設定)』
- 899 ページの『UNLOCK (データ域のアンロックまたはレコードの解放)』
- 900 ページの『UPDATE (既存のレコードの変更)』
- 905 ページの『WRITE (新しいレコードの作成)』.

ファイル組み込み関数には次のものがあります。

- 638 ページの『%EOF (ファイルの終わりまたは先頭条件の戻し)』
- 640 ページの『%EQUAL (完全な一致条件の戻し)』
- 644 ページの『%FOUND (検出条件の戻し)』
- 669 ページの『%OPEN (ファイル・オープン条件の戻し)』
- 691 ページの『%STATUS (ファイルまたはプログラム状況の戻し)』

これらの命令は、従来型の構文と自由形式構文の両方で使用可能です。

ほとんどのファイル命令は、プログラム記述ファイルと外部記述ファイルの両方に 使用することができます。

外部記述ファイルを特定のファイル命令で使用する場合には、演算項目 2 にファイル名でなくレコード様式名を指定することができます。 したがって、処理用命令コードでは、使用する演算命令コードの規則に従って、指定したタイプのレコード様式でファイルの検索または位置決め (あるいはその両方) が行われます。

OVRDBF (データベース・ファイルの一時変更) コマンドを MBR (*ALL) パラ メーターを指定して使用した場合には、SETLL、SETGT、および CHAIN 命令は、 現在オープンされているファイル・メンバーだけを処理します。 詳しくは、IBM i Information Center の「データベースおよびファイル・システム」のカテゴリーを参照してください。

CHAIN、READ、READC、READE、READP、および READPE 命令では、 結果データ構造を指定することができます。 これらの命令では、 データはファイルとデータ構造の間で直接転送され、そのファイルの入力仕様 は処理されません。 したがって、データ構造に対する入力操作の結果として、レコード識別標識 またはフィールド標識がオンに設定されることはありません。 ファイルに対するすべての入力操作で 結果データ構造が指定されている場合には、入力仕様は必要ありません。

演算項目 2 にプログラム記述ファイル名を指定する WRITE および UPDATE 命令では、結果のフィールドにデータ構造名が指定されていなければなり ません。 外部記述レコードに対する WRITE および UPDATE 命令では、結果データ構造を指定することができます。 これらの命令では、データはデータ構造とファイルの間で直接転送され、そのファイルの出力仕様 は処理されません。 ファイルに対するすべての出力命令で結果データ構造が指定されている場合には、出力仕様は必要ありません。

次のような場合、外部記述ファイル名またはレコード名に対する入出力命令を実行することによって、 データ構造名を生成することができます。

1. 入出力命令でレコード名を指定する場合、データ構造の生成元はそのレコードと一致していなければなりません。 つまり、そのデータ構造は LIKEREC(rec) または EXTNAME(file:rec) を使用して定義されて

いる必要があります (rec は、その命令で指定された形式名です)。LIKEREC キーワードまたは EXTNAME キーワードに *NULL を指定することはできません。

- 入力命令の場合は、*INPUT または *ALL を使用して結果データ構造 (または LIKEDS データ構造の基底構造) を定義しなければなりません。 データ構造が LIKEREC キーワードで定義された場合は、タイプを明示的に指定する必要はありません。
- PRINTER、SEQ、SPECIAL、または WORKSTN ファイルへの WRITE 命令の場合は、*OUTPUT または *ALL を使用して結果データ構造を定義する必要があります。
- DISK ファイルへの WRITE の場合は、*OUTPUT または *ALL を使用してこのファイルを定義することができます。 出力バッファーのレイアウトが入力バッファーのレイアウトと同一であり、タイプの指定されていない LIKEREC キーワードを使用してデータ構造が定義されている場合は、WRITE 命令にデータ構造を使用することができます。
- WORKSTN ファイルのサブファイル・レコードに対する UPDATE の場合は、*OUTPUT または *ALL を使用して結果データ構造を定義することができます。
- DISK ファイルに対する UPDATE の場合は、*INPUT、*OUTPUT、または *ALL を使用して結果データ構造を定義することができます。 データ構造が LIKEREC キーワードで定義された場合は、タイプを明示的に指定する必要はありません。

2. 命令コード CHAIN、READ、READE、READP、および READPE については、 レコード名に対する入出力命令のほかに、外部記述ファイル名に対する入出力命令でも結果データ構造を指定することができます。外部記述ファイルの名前を指定するときには、そのデータ構造には、入力可能フィールドを含むレコードごとに 1 つずつ、サブフィールド・データ構造が含まれている必要があります。データ構造に含めることのできるサブフィールド・データ構造は、規則 1 に従って定義されます。 それぞれのサブフィールド・データ構造は、1 桁目から開始される必要があります。 (通常、オーバーレイするサブフィールドはキーワード POS(1) または OVERLAY(ds:1) を使用して定義されます。) 特殊なケースとして、ファイルにレコードが 1 つしか含まれない場合は、 結果データ構造は規則 1 に従って定義することができます。
3. 結果データ構造は、LIKEDS(ds) を使用して定義することもできます (*ds* は、これらの規則に従うデータ構造です)。
4. 以下の場合には、*ALL を 2 番目のパラメーターとしてデータ構造を定義しなければ、個々のサブフィールドが別々に転送されるのではなく、結果データ構造と入力または出力バッファーとの間でデータが全体として転送されます。

   - ファイルに対して DATA(*NOCVT) が有効になっていて、データ構造が CCSID(*EXACT) を指定して定義されている場合
   - ファイルに対して DATA(*CVT) が有効になっていて、データ構造が CCSID(*EXACT) を指定して定義されていない場合
   - ファイルに対して DATA キーワードが有効でなく、データ構造体に対して CCSID キーワードが指定されていない場合

   上記以外の場合、データが結果データ構造と入力または出力バッファーとの間 で転送されるときに、英数字サブフィールドおよびグラフィック・サブフィールドの CCSID 変換が必要になることがあります。

   すべてのケースで、英数字でもグラフィックでもないタイプのサブフィールドのデータは、 データ・タイプに関係なく、入力または出力バッファーと結果データ構造との間で直接転送されます。

   366 ページの『DATA(*CVT | *NOCVT)』 および 419 ページの『CCSID(*EXACT | *NOEXACT)』 を参照してください。

レコードが見付からなかったか、操作中にエラーが起こったか、あるいは最後 のレコードがすでに検索されている (ファイルの終わり) ために入力操作 (CHAIN、EXFMT、READ、READC、READE、READP、READPE) でレコードが検索され なかった場合には、データは抜き出されずにプログラムのすべてのフィール ドが変更されないままとなります。

更新ディスク・ファイルに対する CHAIN、READ、READE、READP、または READPE 命令 の命令拡張として N を指定した場合には、レコードはロック なしで読み取られます。 ファイルが更新ディスク・ファイルの場合には、命令 拡張が指定されていないと、レコードはロックされます。

ファイル命令の実行中に発生した例外/エラーは、プログラマーが (エラー標識をコーディングするかまたはファイル・エラー処理サブルーチンを指定して) 処理するか、あるいは RPG IV エラー処理プログラムによって処理することができます。

注 : 入力仕様および出力仕様に関係するサブプロシージャーの入出力操作には、 同じ名前の内部変数があっても、常に大域名が使用されます。 例えば、サブプロシージャー内とメイン・ソース部分にフィールド名 TOTALS が 定義されている場合には、このサブプロシージャーの入出力操作にメイン・ソース 部分に定義されたフィールドが使用されます。

ヌル値可能フィールドを持つファイルの処理については、286 ページの『データベースのヌル値サポート』を参照してください。

プロトタイプ・プログラムまたはプロトタイプ・プロシージャーに、パラメーターとしてファイルを渡すことができます。 パラメーターとしてファイルを渡すと、ファイルの設定のうち、ファイル仕様書キーワードを使用して定義されたものは、そのファイルにアクセスするすべてのプロシージャーについて有効になります。 例えば、変数パラメーターに EXTFILE キーワードに指定されている時に、呼び出し先プロシージャーがそのファイルをオープンすると、呼び出し元の変数の値を使用して、オープンされるファイルの名前が設定されます。 ファイルに関連付けられている変数に対して、呼び出し先プロシージャーがキーワードを介してアクセスまたは変更を実行する必要がある場合、呼び出し元プロシージャーはそれらの変数をパラメーターとして渡さなければなりません。

ファイル・フィードバック組み込み関数の %EOF(ファイル名)、%EQUAL(ファイル名)、%FOUND(ファイル名)、%OPEN(ファイル名)、および %STATUS(ファイル名) を呼び出し先のプロシージャーまたはプログラムで使用して、ファイルの現在の状態を判別することができます。 この場合は、ファイル・パラメーターの名前をオペランドとして組み込み関数に指定してください。

ファイル・パラメーターについて詳しくは、447 ページの『LIKEFILE(ファイル名)』および 173 ページの『ファイルに関する一般的な考慮事項』を参照してください。

ファイル命令のキー

ファイル命令 CHAIN、DELETE、READE、READPE、SETGT、および SETLL において、検索引数 (検索引数) は、レコードの識別に使用する キーまたは相対レコード番号でなければなりません。 自由形式演算の場合、検索引数には次のものを適用できます。

1. 単一のフィールド名
2. klist 名
3. "(a:b:c+2)" などの値のリスト。 複合キーのそれぞれの部分には、式を使用することができます。

   デフォルトで、データ・タイプは対応するキー・フィールドと一致する必要がありますが、長さとデータ形式は一致する必要はありません。

   制御ステートメントにキーワード EXPROPTS(*STRICTKEYS) が指定されている場合、規則はより厳密になります。 335 ページの『*STRICTKEYS』を参照してください。
4. %KDS(ds{:数})

   複合キーは､指定されているデータ構造のサブフィールドから順に形成されます。 数 が指定されている場合、 これは複合キーで使用するサブフィールド数です。

   デフォルトで、データ・タイプは対応するキー・フィールドと一致する必要がありますが、長さ、データ形式、および CCSID は一致する必要はありません。 データを式の値からキー作成域に転送する規則は、 短い検索引数は右側にブランクが埋め込まれ、長い検索引数は文字タイプの場合切り捨てられるという 点では、命令コード EVAL の場合と同じです。

   制御ステートメントにキーワード EXPROPTS(*STRICTKEYS) が指定されている場合、規則はより厳密になります。 335 ページの『*STRICTKEYS』を参照してください。

自由形式演算ではない場合、フィールド名と klist 名のみが検索引数として使用できます。

命令拡張 H、M、および R は、検索引数のリストまたは %KDS が指定されている場合に、CHAIN、DELETE、READE、READPE、 SETGT、および SETLL で使用することができます。 これらの拡張は、個々の検索引数を検索引数の作成域へ転送する際に 適用されます。 558 ページの『精度の確認』を参照してください。

標識設定命令

標識設定命令コードには次のものがあります。

- 879 ページの『SETOFF (標識をオフに設定)』
- 880 ページの『SETON (標識をオンに設定)』

これらの命令コードは従来型の構文の中でのみ、使用可能です。 自由形式構文では、EVAL 命令を使用して、*INxx を *ON や *OFF に設定することができます。

次の標識設定組み込み関数は、従来型の構文と自由形式構文の両方で使用可能です。

- 668 ページの『%NULLIND (ヌル標識の照会または設定)』

SETON および SETOFF 命令は 71 から 76 桁目に指定された 標識を (オンまたはオフに) 設定します。 これらの桁には少なくとも 1 つの結果の標識を指定しなければなりません。 標識を設定する場合には次のことに留意してください。

- 1P、MR、KA から KN、および KP から KY の標識は SETON 命令ではオンに設定できません。
- 1P および MR 標識を SETOFF 命令でオフに設定することはできません。
- SETON または SETOFF 命令を用いて L1 から L9 をオンまたはオフに設定し ても、より低い制御レベル標識は自動的に設定されません。

情報命令

情報命令を次の表に示します。

表 *116.* 情報命令

命令	従来型の構文	自由形式構文
ダンプ	768 ページの『DUMP (プログラム・ダンプ)』	768 ページの『DUMP (プログラム・ダンプ)』
シャットダウン状況の入手	880 ページの『SHTDN (シャットダウン)』	686 ページの『%SHTDN (シャットダウン)』
時刻と日付の入手	897 ページの『TIME (時刻と日付の検索)』	• 628 ページの『%DATE (日付への変換)』 • 700 ページの『%TIME (時刻への変換)』 • 701 ページの『%TIMESTAMP (タイム・スタンプへの変換)』

DUMP 命令では、プログラム中で使用されるすべての標識、フィールド、デ ータ構造、配列、およびテーブルのダンプが実行されます。

SHTDN 命令では、システム操作員がシャットダウンを要求したかどうかをプログラムで 判別することができます。 要求した場合には、71 から 72 桁目に指定する必要が ある結果の標識がオンに設定されます。

TIME 命令では、プログラムは実行中の任意の時点でシステム時刻およびシ ステム日付にアクセスすることができます。

初期化命令

初期化命令では、構造 (レコード様式、データ構造、配列、またはテーブ ル) 中のすべての要素あるいは変数 (フィールド、サブフィールド、または標 識) の実行時消去およびリセットが行われます。

初期化命令には次のものがあります。

- 734 ページの『CLEAR (消去)』
- 862 ページの『RESET (リセット)』.

これらの命令は、従来型の構文と自由形式構文の両方で使用可能です。

CLEAR 命令では、構造中のすべての要素または変数が、フィールドの タイプ (数値、文字、図形、UCS-2、標識、ポインター、または日付/時刻/タイム・スタンプ) に 基づいて、それぞれのデフォルトの値に設定されます。

RESET 命令では、構造中のすべての要素または変数が、それぞれの初期値 (プログラム・サイクルの初期化ステップの終わりに持っていた値) に設定さ れます。

RESET 命令は、データ構造初期化および初期化サブルーチン (*INZSR) で使用されます。 変数の初期値の設定には、データ構造初期化と *INZSR の 両方を使用することができます。 初期値が、RESET 命令の結果のフィールドに 指定されると、これが変数の設定に使用されます。

これらの命令コードがレコード様式に適用される場合には、出力されるフィ ールドだけ (演算項目 2 がブランクの場合) が影響を受けるか、またはすべて のフィールド (演算項目 2 が *ALL の場合) が影響を受けます。 演算項目 1 に *NOKEY を指定すると、キー・フィールドが消去またはリセットされるのを防ぎます。

結果のフィールドにテーブル名、あるいは複数オカレンス・データ構造またはレ コード様式が入っている場合には、演算項目 2 に *ALL を指定することができ ます。 *ALL が指定された場合には、すべての要素または繰り返しが消去またはリセットされます。 詳しくは、734 ページの『CLEAR (消去)』および 862 ページの『RESET (リセット)』を参照してください

詳しくは、246 ページの『データ・タイプおよびデータ形式』を参照してください。

メモリー管理命令

メモリー管理命令を次の表に示します。

表 *117.* メモリー管理命令

命令	従来型の構文	自由形式構文
記憶域の割り振り	712 ページの『ALLOC (記憶域の割り振り)』	616 ページの『%ALLOC (記憶域の割り振り)』
記憶域の解放	752 ページの『DEALLOC (記憶域の解放)』	752 ページの『DEALLOC (記憶域の解放)』
記憶域の再割り振り	860 ページの『REALLOC (新しい長さでの記憶域の再割り振り)』	678 ページの『%REALLOC (記憶域の再割り振り)』
変数のアドレスの入手		614 ページの『%ADDR (変数のアドレスの検索)』
プロシージャーのアドレスの入手		670 ページの『%PADDR (プロシージャー・アドレスの検索)』

ALLOC 命令では、動的記憶域を割り振り、結果フィールド・ポインターを、 その記憶域を指し示すように設定します。 この記憶域は初期化されません。

REALLOC 命令は、結果フィールド・ポインターによって指し示されている 動的記憶域の長さを変更します。 新しい記憶域は、割り振られると、古い記憶域の値に初期化されます。 新しいサイズが古いサイズより小さい場合、データが切り捨てられます。 新しいサイズが古いサイズより大きい場合は、コピーされたデータの後の 記憶域は初期化されません。 古い記憶域は解放されます。 結果フィールド・ポインターは、新しい記憶域を指し示すように設定 されます。

DEALLOC 命令では、結果フィールド・ポインターが設定される 動的記憶域を解放します。 操作拡張 (N) が指定されている場合、ポインターは再割り振りが正常に 行われた後、*NULL に設定されます。

記憶域は、活動化グループが終了すると暗黙に解放されます。 LR をオンに設定すると、モジュールによって割り振られた 動的記憶域は解放されませんが、動的記憶域を指し示すポインターは失われます。

ヒープ記憶域には、単一レベルとテラスペースの 2 つのタイプがあります。 制御仕様書で ALLOC キーワードを使用して、 メモリー管理命令が使用するヒープ記憶域のタイプを制御できます。

ヒープ記憶域のそれぞれのタイプに、利点と欠点があります。

- 個別の割り振りまたは再割り振りの最大サイズは、テラスペース・ヒープ記憶域のほうが大きくなります。

  - %ALLOC 組み込み関数および %REALLOC 組み込み関数に対する RPG の許容最大サイズは、4294967295 バイトです。 単一レベル・ヒープ記憶域を使用するとき、RPG の許容最大サイズは 16776704 バイトです。
  - メモリー管理命令がテラスペース・ヒープ記憶域を使用することをコンパイラーがコンパイル時に検出できると、 ALLOC および REALLOC の命令コードに対する RPG の許容最大サイズは 4294967295 バイトに上がります。 RPG メモリー管理命令が単一レベル・ヒープ記憶域を使用する場合、 またはコンパイラーがコンパイル時にヒープ記憶域のタイプを検出できない場合には、 16776704 バイトという小さいほうの限度が有効になります。
  - 実行時にヒープ記憶域が使用可能であるかどうかによって、実際の割り振り可能最大サイズが、 RPG の許容最大サイズより小さくなることがあります。
- テラスペース・ヒープ記憶域の再割り振りと割り振り解除に RPG が使用するシステム関数では、 単一レベル・ヒープ記憶域およびテラスペース・ヒープ記憶域へのポインターを処理できます。 ポインターの再割り振りにテラスペース再割り振り関数が使用されると、 その新しい割り振りのヒープ記憶域タイプは、元の割り振りと同じになります。
- 単一レベル・ヒープ記憶域の再割り振りと割り振り解除に RPG が使用するシステム関数では、 単一レベル・ヒープ記憶域へのポインターしか処理できません。
- 単一レベル記憶域のほうが、テラスペース記憶域より保全性を高くすることができます。 例えば、記憶域オーバーランで影響を受ける可能性がある記憶域は、単一レベル記憶域を使用するとメガバイト単位ですが、 テラスペース記憶域の場合にはテラバイト単位になります。

さまざまなタイプのヒープ記憶域について詳しくは、「*ILE 概念*」*(SD88-5033)* の記憶域管理に関する章を参照してください。

動的記憶域の使用を誤ると、問題が発生する原因となります。 以下の例は、 それを回避するための手順を示したものです。

```
D Fld1         S                  25A    BASED(Ptr1)
D Fld2         S                   5A    BASED(Ptr2)
D Ptr1         S                   *
D Ptr2         S                   *
 ....
C                ALLOC     25              Ptr1
C                DEALLOC                   Ptr1
 * After this point, Fld1 should not be accessed since the
 * basing pointer Ptr1 no longer points to allocated storage.
C                CALL      'SOMEPGM'

 * During the previous call to 'SOMEPGM', several storage allocations
 * may have been done.  In any case, it is extremely dangerous to
 * make the following assignment, since 25 bytes of storage will
 * be filled with 'a'.  It is impossible to know what that storage
 * is currently being used for.
C                EVAL      Fld1 = *ALL'a'
```

以下は、さらに深刻な状況です。

- ポインターが、再割り振りまたは割り振り解除の前にコピーされた場合、 同様のエラーが起こる可能性があります。 ポインターを割り振り済みの記憶域にコピーする場合は、十分に注意して、 そのポインターが、記憶域の割り振り解除後や再割り振り後に使用されることが ないようにしてください。
- 動的記憶域を指し示すポインターがコピーされると、そのコピーが 記憶域の割り振り解除または再割り振りに使用される場合があります。 この場合、元のポインターは、それが新しい値に設定されるまでは使用してはなりません。
- 動的記憶域を指し示すポインターがパラメーターとして渡されると、 呼び出される側が、その記憶域を割り振り解除したり、再割り振りしたりする 可能性があります。 呼び出しが戻った後で、そのポインターを使用してその記憶域にアクセスしようと すると問題が起こります。
- 動的記憶域を指し示すポインターが *INZSR 内で設定されると、 それ以降にそのポインターが RESET されたときに、 もはや割り振られていない記憶域にそのポインターが設定される可能性が あります。
- 動的記憶域を指し示すポインターが失われた場合 (例えば、クリアされた ことによって、あるいは ALLOC 命令によって新しいポインターに設定 された場合など) には、また別の種類の問題が起こる可能性があります。 ポインターが失われてしまうと、それが指し示している記憶域は解放することができません。 シ

ステムは、この記憶域がもはやアドレス可能ではなくなっていることを認識していないために、この記憶域を割り振ることはできません。 この記憶域は、活動化グループが終了するまで解放されません。

メッセージ命令

メッセージ命令

- 765 ページの『DSPLY (メッセージ表示)』

によって、プログラムと操作員の間またはプログラムとそのプログラムを要求した表示装 置ワークステーションの間の対話通信が可能になります。

これらの命令は、従来型の構文と自由形式構文の両方で使用可能です。

移動命令

移動命令を次の表に示します。

表 *118.* 移動命令

命令	従来型の構文	自由形式構文
移動	803 ページの『MOVE (転送)』	773 ページの『EVALR (式の評価、右寄せ)』 または 変換 組み込み関数
配列の転送	817 ページの『MOVEA (配列の転送)』	(許可されていない)
左につめて転送	824 ページの『MOVEL (左につめて転送)』	771 ページの『EVAL (式の評価)』 または 変換 組み込み関数

移動命令では、演算項目 2 の全部または一部が結果のフィールドに転送され ます。 演算項目 2 は変更されません。

移動命令の転送元と転送先は同じタイプにすることも異なったタイプにする こともできますが、次のようないくつかの制約が適用されます。

- ポインターの転送の場合には、転送元と転送先は同じタイプでなければな らず、両方とも基底ポインターか、両方ともプロシージャー・ポインターのい ずれかでなければなりません。
- MOVEA を使用する場合には、転送元と転送先の両方が同じタイプでなけれ ばなりません。
- 日付、時刻、またはタイム・スタンプ・フィールドに MOVEA を使用することは できません。
- MOVE および MOVEL は、浮動フィールドまたはリテラルには使用できません。

結果の標識は、文字、図形、UCS-2、および数値の結果フィールドにだけ指定できます。 MOVE および MOVEL 命令の場合には、結果フィールドが指標なし配列 であると、結果の標識は指定できません。 MOVEA の場合には、 結果フィールドが 配列であれば、指標の有無に関係なく、結果の標識は指定できません。

命令拡張 P を指定できるのは、結果のフィールドが文字、図形、UCS-2、または 数値の場合だけです。

文字、図形、UCS-2、および数値データの転送

文字フィールドが数値の結果のフィールドに転送される場合には、それぞれの 文字の数字部分がそれに対応する数字に変換されてから結果のフィールドに転 送されます。 ブランクはゼロとして転送されます。 MOVE 命令の場合には、右端の文字のゾーン部分がそれに対応する符号に変換さ れて数値の結果のフィールドの右端の桁に転送されます。 これがそのフィールドの符号になります。 (例については、816 ページの図 351 を参照してください。) MOVEL 命令の場合には、右端の文字が移動命令に含まれているかどうかに関係 なく、演算項目 2 の右端の文字のゾーン部分が変換されて結果のフィールドの 符号として使用されます (ただし、演算項目 2 が結果のフィールドよりも短い 場合を除きます)。 (例については、827 ページの図 353 を参照してください。)

移動命令が数値フィールドの間に指定された場合には、演算項目 2 フィールド に指定された小数点以下の桁数は無視されます。 例えば、小数点以下の桁数が 1 桁の 3 桁の数値フィールドに 1.00 を転送す ると、結果は 10.0 になります。

文字または数値フィールドに転送する場合には、演算項目 2 に形象定数 *ZEROS を入れることができます。図形フィールドで同じ機能を実行するためには、*ALLG'oXXi' をコーディング しなければなりません (この場合に、'XX' は図形ゼロを表します)。

文字の転送元から図形フィールドにデータを転送する場合には、転送元が文字 リテラル、名前のついた定数、または *ALL であれば、コンパイラーは、 全体が 1 対のシフトアウト文字とシフトイン文字 (SO/SI) で囲まれている かどうかを確認する検査を行います。 コンパイラーは、文字転送元が偶数長で、最低 4 バイト (SO/SI 文字プラス 1 図形文字) であるかどうかも検査します。 16 進数リテラルまたは *ALLX から図形フィールドに転送する場合には、16 進数リ テラルの最初のバイトと最後のバイトまたは *ALLX 内のパターンは、0E (シフ トアウト) および 0F (シフトイン) であってはなりません。 この場合にも 16 進数リテラル (またはパターン) は偶数バイトを表していなければなりません。

文字フィールドが図形フィールドへの転送または図形フィールドからの転送に かかわっている場合には、コンパイラーは、文字フィールドが偶数長で、最低 4 バイトの長さであるかどうかを検査します。 実行時には、コンパイラーは文字 フィールドの内容を検査して、全体が 1 対の SO/SI だけで囲まれているかどうかを確認します。

図形フィールドから文字フィールドに転送する場合には、文字フィールドが 図形フィールドのバイト数に 2 バイトを加えた長さより大きい場合に、 図形データの直前と直後に SO/SI が追加されます。 これが、文字フィールドの残りのデータによって その文字フィールドの SO/SI の不均衡の原因になることがあり、これはコンパイラーでは診断されません。

移動命令を使用して文字フィールドから図形フィールドにデータを転送する場 合には、シフトアウト文字とシフトイン文字が除去されます。 図形フィー ルドから文字フィールドにデータを転送する場合には、転送先フィールドにシ フトアウト文字とシフトイン文字が挿入されます。

移動命令が、文字から UCS-2 または UCS-2 から文字へのデータの変換に 使用される場合、転送される文字数は、文字データにシフト文字と図形文字が 含まれる場合と含まれない場合があるために、変化します。例えば、5 つの UCS-2 文字は、次の文字に変換できます。

- 5 個の 1 バイト文字
- 5 個の 2 バイト文字
- モードを分離するシフト文字を含む、1 バイト文字と 2 バイト文字の 組み合わせ

結果のデータが長過ぎて、結果フィールドに入りきらない場合、そのデータは 切り捨てられます。 結果が 1 バイト文字である場合、その結果に完全な文字が含まれている こと、および突き合わせられた SO/SI のペアが含まれていることは、ユーザーが確認 してください。

移動命令に命令拡張 P を指定すると、 結果のフィールドには MOVEL および MOVEA の場合には右から、また MOVE の場合には左から埋め込みが行われます。 埋め込み文字は、文字の場合はブランク、図形の場合は 2 バイトのブランク、UCS-2 の場合は UCS-2 ブランク、数値の場合は 0、標識の場合は '0' です。 この埋め込みは操作の後で行われます。 フィールドを配列に転送するために MOVE または MOVEL を使用した場合には、 配列のそれぞれの要素に埋め込みが行われます。 これらの命令を使用して配列を配列に転送して結果に演算項目 2 の配列より多 くの要素が入る場合にも同じ埋め込みが行われますが、余分の要素は影響を受けません。 結果のフィールドに配列名がある MOVEA 命令では、 この命令によって影響を受ける最後の要素とそれ以後のすべての要素に埋め込みが行われます。

移動命令で結果の標識を指定した場合には、結果のフィールドによってどの 標識をオンに設定するかが決まります。 結果のフィールドが文字フィールド、図形フィールド、または UCS-2 フィールドの場合には、75 桁目と 76 桁目の結果の標識しか指定できません。 この標識は、結果のフィールドがすべてブランクの場合にオンに設定されます。 結果のフィールドが数値の場合には、 3 つの結果の標識すべてを使用することができます。 これらの標識は次の場合にオンに設定されます。

大きい (71 から 72)
結果のフィールドが 0 より大きい場合にオンに設定されます。

小さい (73 から 74)
結果のフィールドが 0 より小さい場合にオンに設定されます。

等しい (75 から 76)
結果のフィールドが 0 に等しい場合にオンに設定されます。

日付時刻データの転送

MOVE および MOVEL 命令コードは、日付、時刻、およびタイム・スタンプ・デー タ・タイプ・フィールドの転送に使用することができます。

MOVE および MOVEL 命令コードでは次の組み合わせを使用することができます。

- 日付から日付
- 時刻から時刻
- タイム・スタンプからタイム・スタンプ
- 日付からタイム・スタンプ
- 時刻からタイム・スタンプ (マイクロ秒を 000000 に設定)
- タイム・スタンプから日付
- タイム・スタンプから時刻
- 日付から文字または数値
- 時刻から文字または数値
- タイム・スタンプから文字または数値
- 文字または数値から日付
- 文字または数値から時刻
- 文字または数値からタイム・スタンプ

演算項目 1 は、転送元および転送先が両方とも、日付、時刻または タイム・スタンプ・フィールドである場合にはブランクでなければなりません。 演算項目 1 がブランクの場合、日付、時刻、またはタイム・スタンプ・フィールド の形式が使用されます。

そうでない場合には、演算項目 1 にこの命令の転送元または転送先の文字また は数値フィールドに対応する日付または時刻の形式が入ります。 有効な任意の形式を指定することができます。 273 ページの『日付データ・タイプ』、275 ページの『時刻データ・タイプ』、および 277 ページの『タイム・スタンプ・データ・タイプ』 を参照してください。

演算項目 1 に指定する場合には、次のことに留意してください。

- 時刻の形式 *USA は、時刻と数値フィールドの間の転送には使用することがで きません。
- 演算項目 1 には *LONGJUL、*CYMD、*CMDY、および *CDMY の各形式と特殊値 *JOBRUN を使用できます。 (詳しくは、275 ページの表 73 を参照してください。)
- ゼロ (0) が形式の最後に指定されると (例えば *MDY0)、 その文字フィールドに区切り記号が含まれていないことを示します。
- 2 桁の年の形式 (*MDY、*DMY、*YMD、*JUL および *JOBRUN) で 表すことができるのは、1940 から 2039 の範囲内の日付だけです。 3 桁の年の 形式 (*CYMD、*CMDY、*CDMY) で表すことができるのは、1900 から 2899 の範囲内の日付だけです。 これらの範囲外の日付で 2 桁または 3 桁の年の形式への変換が要求された場合には、 エラーが出されます。
- タイム・スタンプとの間で文字値または数値を転送するため に MOVE および MOVEL を使用する場合、文字値または数値はタイム・スタンプ であると想定されます。

演算項目 2 は必須で、文字、数値、日付、時刻、またはタイム・スタンプのいずれかの 値でなければなりません。 この項目には、変換するフィールド、配列、配列要素、 テーブル名、リテラル、または名前のついた定数のいずれかを入れます。

演算項目 2 には次の規則が適用されます。

- 区切り文字は指定された形式に対して有効なものでなければなりません。
- 演算項目 2 が日付または時刻に有効な表記でないか、あるいはその形式が 演算項目 1 に指定された形式と一致しない場合には、エラーが生成されます。
- 演算項目 2 に UDATE または *DATE が含まれている場合、 演算項目 1 はオプションで、見出し仕様 DATEDIT キーワードに 対応します。

- 演算項目 2 に UDATE が入っていて、演算項目 1 の記入項目がコーディング されている場合、年が 2 桁の日付形式でなければなりません。 演算項目 2 に *DATE が入っていて、演算項目 1 がコーディング されている場合、年が 4 桁の日付形式でなければなりません。

結果フィールドは日付、時刻、タイム・スタンプ、数値、または 文字の各変数でなければなりません。 このフィールドは、フィールド、配列、配列要素、または テーブル名のいずれでもかまいません。 日付または時刻は、結果フィールドに定義された形式または演算項目 1 に指定された形式コードに従って結果フィールドに入れられます。 結果フィールドが数値の場合には、命令の前に区切り文字が除去されます。 使用される長さは、区切り文字が除去された後の長さです。

日付からタイム・スタンプ・フィールドへ転送する場合には、タイム・スタンプの時刻 とマイクロ秒部分は影響を受けませんが、タイム・スタンプ全体が検査されて、正 しくなければエラーが生成されます。

時刻からタイム・スタンプ・フィールドに転送する場合には、タイム・スタンプのマイ クロ秒部分は 000000 に設定されます。 日付部分は影響を受けませんが、タイム・スタンプ全体が検査されて、正しくなければエラーが生成されます。

文字データまたは数値データが必要な長さより長い場合には、左端のデータ (MOVE 命令 の場合は右端) が使用されます。 演算項目 1 によって転送されるデータの長さが 決まることに留意してください。 例えば、数値日付からの MOVE 命令で、演算項目 1 の形式が *MDY の場合、演算項目 2 の右端の 6 桁のみが 使用されます。

文字フィールドから日付フィールドへの変換の例

587 ページの図 186 に、日付フィールド相互間または文字フィールドと日付フィールドの間で 2 桁と 4 桁の年の日付を定義して転送する方法について示します。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
 * Define two 8-byte character fields.
D CHR_8a          s              8a   inz('95/05/21')
D CHR_8b          s              8a   inz('abcdefgh')
 *
 * Define two 8-byte date fields.  To get a 2-digit year instead of
 * the default 4-digit year (for *ISO format), they are defined
 * with a 2-digit year date format, *YMD.  For D_8a, a separator (.)
 * is also specified.  Note that the format of the date literal
 * specified with the INZ keyword must be the same as the format
 * specified on the * control specification.  In this case, none
 * is specified, so it is the default, *ISO.
 *
D D_8a            s               d   datfmt(*ymd.)
D D_8b            s               d   inz(d'1995-07-31') datfmt(*ymd)
 *
 * Define a 10-byte date field.  By default, it has *ISO format.
D D_10            s               d   inz(d'1994-06-10')
 *
 *  D_10 now has the value 1995-05-21
 *
 *  Move the 8-character field to a 10-character date field D_10.
 *  It will contain the date that CHR_8a was initialized to, but
 *  with a 4-digit year and the format of D_10, namely,
 *  1995-05-21 (*ISO format).
 *
 *  Note that a format must be specified with built-in function
 *  %DATE to indicate the format of the character field.
 *
 /FREE
    D_10 = %DATE (CHR_8a: *YMD);
    //
    //  Move the 10-character date to an 8-character field CHR_8b.
    //  It will contain the date that was just moved to D_10, but with
    //  a 2-digit year and the default separator indicated by the *YMD
    //  format.
    //
    CHR_8b = %CHAR (D_10: *YMD);
    //
    //  Move the 10-character date to an 8-character date D_8a.
    //  It will contain the date that *  was just moved to D_10, but
    //  with a 2-digit year and a . separator since D_8a was defined
    //  with the (*YMD.) format.
    //
    D_8a = D_10;
    //
    //  Move the 8-character date to a 10-character date D_10
    //  It will contain the date that *  D_8b was initialized to,
    //  but with a 4-digit year, 1995-07-31.
    //
    D_10 = D_8b;
    //
    // After the last move, the fields will contain
    //  CHR_8b:  95/05/21
    //  D_8a:    95.05.21
    //  D_10:    1995-07-31
    //
    *INLR = *ON;
 /END-FREE
```

図 *186.* 文字および日付データの転送

次の例は CYYMMDD 形式の文字フィールドから *ISO 形式の日付フィールド に変換する方法を示しています。 これは、タイプ *DATE のコマンド・パラメータ ーを使用している場合に特に効果的です。

RPG プログラムは、コマンド・インターフェースを使用してのみ呼び出されるため、 プログラムのプロトタイプを指定する必要はありません。 プロトタイプは、プロシージャー・インターフェースの情報を使用して、 コンパイラーが暗黙的に定義します。

```
             CMD        PROMPT('Use DATE parameter')
             PARM       KWD(DATE) TYPE(*DATE)
```

図 *187.* 日付パラメーターを使用しているコマンドのソース

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
 * Procedure interface for this program (no prototype is necessary)
D FIG210          PI                  EXTPGM('FIG210')
D   DateParm                     7A

 * Declare a date type with date format *ISO.
D ISO_DATE        S                D   DATFMT(*ISO)

 * The format of the DateParm parameter is CYYMMDD, so code
 * *CYMD0 as the 2nd parameter of built-in function %DATE.
 /FREE
    ISO_DATE = %DATE (DateParm: *CYMD0);
 /END-FREE
```

図 *188. RPG IV* コマンド処理プログラムの一部

ゾーン移動命令

ゾーン移動命令には次のものがあります。

- 801 ページの『MHHZO (上位桁から上位桁へのゾーンの転送)』
- 801 ページの『MHLZO (上位桁から下位桁へのゾーンの転送)』
- 801 ページの『MLHZO (下位桁から上位桁へのゾーンの転送)』
- 801 ページの『MLLZO (下位桁から下位桁へのゾーンの転送)』.

これらの命令は従来型の構文の中でのみ、使用可能です。

ゾーン移動命令では 1 文字のゾーン部分だけが転送されます。

ゾーン移動命令で上位 の語を使用する場合には関係するフィー ルドは文字フィールドでなければならず、下位 の語を使用する場合 には関係するフィールドは文字または数値フィールドのいずれであってもかまいません。 ゾーン移動命令では浮動数値フィールドは使用することができません。

文字 J - R には D ゾーンがあるので、負の値を作るために使用することが できます。

```
(J = hexadecimal D1, ..., R = hexadecimal D9).
```

注 : 古いプログラムでこの使用法を見付けた場合には、この目的で 16 進数リ テラルを使用すればコーディングが明確になります。 正のゾーンを取り出すた めには X'F0' を、また負のゾーンを取り出すためには X'D0' を使用してくだ さい。

注 : 文字 (-) は 16 進数の 60 で表され、負の結果を取り出すために使用 することはできません。これは 6 のゾーンを持っていて、負の結果には "D" のゾーンが必要になるためです。

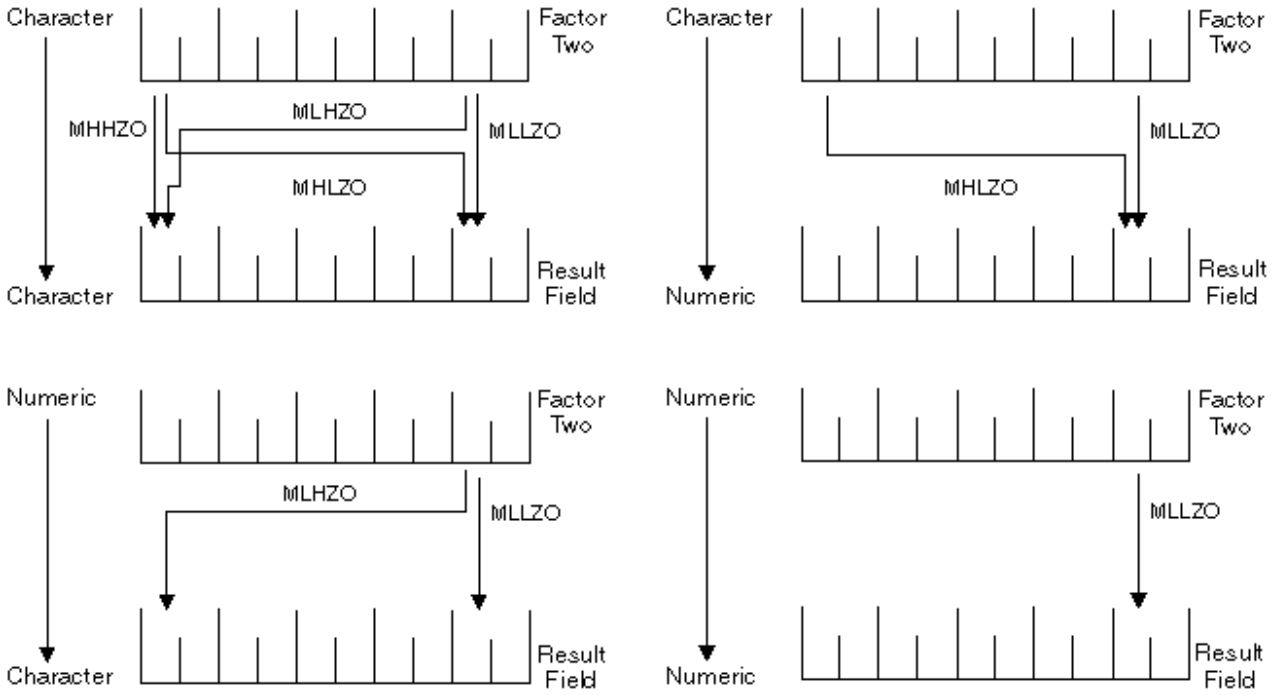


図 *189.* ゾーン移動命令の機能

結果命令

次の組み込み関数は、直前の命令の結果を処理します。

- 640 ページの『%EQUAL (完全な一致条件の戻し)』
- 644 ページの『%FOUND (検出条件の戻し)』
- 641 ページの『%ERROR (エラー条件の戻し)』
- 691 ページの『%STATUS (ファイルまたはプログラム状況の戻し)』

これらの組み込み関数は、従来型の構文と自由形式構文の両方で使用可能です。

サイズ変更命令

次の組み込み関数は、変数、フィールド、定数、配列、テーブル、またはデータ構造のサイズに関する情報を戻します。

- 632 ページの『%DECPOS (小数部の桁数の取得)』
- 653 ページの『%LEN (長さの入手または設定)』
- 687 ページの『%SIZE (サイズ (バイト数) の検索)』

これらの組み込み関数は、従来型の構文と自由形式構文の両方で使用可能です。

ストリング命令

ストリング命令を次の表に示します。

表 *119.* ストリング命令

命令	従来型の構文	自由形式構文
連結	725 ページの『CAT (2 つの文字ストリングの連結)』	+ 演算子
小文字に変換		660 ページの『%LOWER (小文字に変換)』
大文字に変換		707 ページの『%UPPER (大文字に変換)』

表 119. ストリング命令 (続き)		
命令	**従来型の構文**	**自由形式構文**
検査	730 ページの『CHECK (文字の検査)』	624 ページの『%CHECK (文字の検査)』
逆向きの検査	732 ページの『CHECKR (逆向きの検査)』	625 ページの『%CHECKR (逆向きの検査)』
作成		693 ページの『%STR (ヌル文字で終了するストリングの入手または保管)』
置換		679 ページの『%REPLACE (文字ストリングの置換)』
スキャン	870 ページの『SCAN (ストリングの走査)』	681 ページの『%SCAN (文字の走査)』
逆方向走査		683 ページの『%SCANR (文字の逆方向走査)』
走査と置換		684 ページの『%SCANRPL (文字の走査と置換)』
ストリングの分割		689 ページの『%SPLIT (ストリングをサブストリングに分割)』
サブストリング	890 ページの『SUBST (サブストリング)』	698 ページの『%SUBST (サブストリングの検索)』
変換	907 ページの『XLATE (変換)』	708 ページの『%XLATE (変換)』
ブランクのトリミング		703 ページの『%TRIM (両端の文字のトリミング)』、704 ページの『%TRIML (先行文字のトリミング)』、または 704 ページの『%TRIMR (末尾の文字のトリミング)』

ストリング命令には、連結、走査、サブストリング化、変換、および検査が あります。 ストリング命令を使用できるのは、文字フィールド、図形フィールド、または UCS-2 フィールドに対してだけです。

CAT 命令では、2 つのストリングが連結されて 1 つのストリングになります。

CHECK および CHECKR 命令では、演算項目 2 のそれぞれの文字が演算項目 1 の有効な文字の間にあるかどうかが検査されます。 CHECK では左から右に、CHECKR では右から左に検査されます。

SCAN 命令では、演算項目 1 に指定された別のストリングを見付けるために、 演算項目 2 の基本ストリングが走査されます。

SUBST 命令では、演算項目 2 の基本ストリングから指定されたストリングが抜き出されます。 抜き出されたストリングは、結果のフィールドに入れられます。

XLATE 命令では、演算項目 1 の変換元および変換先ストリングに従って、演算項目 2 の文字が変換されます。

注 : 演算項目 1、演算項目 2、または結果フィールドに形象定数を使用することは できません。 演算項目 1 と結果のフィールドまたは演算項目 2 と結果のフィールドに、デー タ構造のオーバーラップがあってはなりません。

ストリング命令では、演算項目 1 と演算項目 2 は 2 つの部分を持つことがで きます。 両方の部分を指定する場合には、コロンで区切らなければなりません。 このオプションは、CAT、CHECK、CHECKR、および SUBST 以外のすべての命令に 適用されます (これらの命令では演算項目 2 だけに適用されます)。

CAT、SUBST、または XLATE 命令の命令拡張として P を指定した場合には、 操作の後で結果のフィールドに右側からブランクが埋め込まれます。

詳細については、それぞれの命令の項を参照してください。

図形フィールドにストリング命令を使用する場合 には、演算項目 1、演算項目 2、および結果のフィールドのデータは すべて図形でなければなりません。 図形文字の長さ、開始位置、およびブランクの数に数値が指定されている場合 には、その値は 2 バイト文字を表します。

UCS-2 フィールドにストリング命令を使用する場合 には、演算項目 1、演算項目 2、および結果のフィールドのデータは すべて UCS-2 でなければなりません。 UCS-2 文字の長さ、開始位置、および ブランクの数に数値が指定されている場合には、その値は 2 バイト文字を表します。

混合モードの文字データの図形部分にストリング命令が使用されている場合 には、開始位置、長さ、およびブランクの数は 1 バイト文字を表します。 データの保全性の維持はユーザーの責任になります。

構造化プログラミング命令

構造化プログラミング命令を次の表に示します。

表 *120.* 構造化プログラミング命令

命令	従来型の構文	自由形式構文
また	713 ページの『ANDxx (かつ)』	AND 演算子
Do	758 ページの『DO (命令グループの開始)』	784 ページの『FOR (For)』
DO UNTIL	760 ページの『DOU (条件が真になるまでの繰り返し)』 または 761 ページの『DOUxx (条件までの繰り返し)』	760 ページの『DOU (条件が真になるまでの繰り返し)』
DO WHILE	763 ページの『DOW (条件が真の間繰り返し)』 または 763 ページの『DOWxx (条件が真の間繰り返し)』	763 ページの『DOW (条件が真の間繰り返し)』
Else	769 ページの『ELSE (他の場合)』	769 ページの『ELSE (他の場合)』
ELSE IF	769 ページの『ELSEIF (ELSE IF)』	769 ページの『ELSEIF (ELSE IF)』
終了	770 ページの『ENDyy (構造化グループの終わり)』	770 ページの『ENDyy (構造化グループの終わり)』
For	784 ページの『FOR (For)』	784 ページの『FOR (For)』
For-Each	786 ページの『FOR-EACH (それぞれの場合)』	786 ページの『FOR-EACH (それぞれの場合)』
IF	789 ページの『IF (If)』 または 790 ページの『IFxx (満たされた条件の処理)』	789 ページの『IF (If)』
ITERATE	792 ページの『ITER (繰り返し)』	792 ページの『ITER (繰り返し)』
LEAVE	796 ページの『LEAVE (Do/For グループからの抜け出し)』	796 ページの『LEAVE (Do/For グループからの抜け出し)』
または	843 ページの『ORxx (または)』	OR 演算子
OTHERWISE	844 ページの『OTHER (その他の場合の選択)』	844 ページの『OTHER (その他の場合の選択)』
選択	872 ページの『SELECT (選択グループの始め)』	872 ページの『SELECT (選択グループの始め)』
WHEN	902 ページの『WHEN (真の場合に選択)』 または 903 ページの『WHENxx (真の場合に選択)』	902 ページの『WHEN (真の場合に選択)』

DO 命令では、演算項目 1 の値から始めて、毎回対応する ENDDO 命令の値だ け増分して、演算項目 2 に指定された限界に達するまで 0 回または 1 回以上 演算グループを処理することができます。

DOU および DOUxx (条件が真になるまでの繰り返し) 命令では、 演算グループを 1 回または複数回処理することができます。 Do-Until 命令の終わりは ENDDO 命令で示されます。

DOW および DOWxx (条件が真の間繰り返し) 命令では、 演算グループを 0 回または 1 回以上処理することができます。 Do-While 命令の終わりは ENDDO 命令で示されます。

FOR 命令によって、演算のグループの反復処理が可能になります。 開始値は、索引名に割り当てられます。増分値および限界値も同様に指定できます。 開始値、増分値、限界値は、自由形式の式にすることができます。 ENDFOR 命令 は、FOR グループの終わりを示します。

FOR-EACH 命令では、配列または %LIST の各項目を一度に 1 つずつ処理することができます。 ENDFOR 命令は、FOR-EACH グループの終わりを示します。

LEAVE 命令は、制御のフローを早期に中断して、反復構造化グループの ENDDO 命令または ENDFOR 命令の後のステートメントに制御を渡します。 ITER 命令では、次のループの繰り返しが即時に行われます。

IF および IFxx 命令では、指定された条件が満たされた場合に、 演算グループを処理することができます。ELSE 命令では、条件が満たされない場合に処理する演算グループを指定するこ とができます。 ELSEIF 命令は、ELSE 命令と IF 命令を組み合わせたものです。 IF または IFxx グループの終わりは ENDIF で示されます。

SELECT、WHEN、WHENxx、および OTHER グループの命令は、 命令の複数の代替順序のうちの 1 つを条件付きで処理するために使用されます。 選択グループの始めは SELECT 命令で示されます。 WHEN および WHENxx 命令は、命令の処理順序を選択するために使用されます。 OTHER 命令は、WHENxx 条件がどれも満たされない時に処理する命令の順序を 示すために使用されます。 選択グループの終わりは ENDSL 命令で示されます。

ANDxx および ORxx 命令は DOUxx、DOWxx、WHENxx、および IFxx 命令と 一緒に使用され、IFxx 命令はさらに複雑な条件を指定するために使用 されます。 ANDxx 命令の方が ORxx 命令よりも優先順位が高くなります。 ただし、IF、 DOU、DOW、および WHEN 命令では、それらの xx が付いている命 令よりもさらに簡潔に、複雑な式をコーディングできることに注意してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 * In the following example, indicator 25 will be set on only if the
 * first two conditions are true or the third condition is true.
 *
 * As an expression, this would be written:
 * EVAL *IN25 = ((FIELDA > FIELDB) AND (FIELDA >= FIELDC)) OR (FIELDA < FIELDD)
 *
 *
C     FIELDA        IFGT      FIELDB
C     FIELDA        ANDGE     FIELDC
C     FIELDA        ORLT      FIELDD
C                   SETON                                        25
C                   ELSE
C                   SETOFF                                       25
C                   ENDIF
```

図 *190. AND/OR* の優先順位の例

DO、DOUxx、DOWxx、FOR、IFxx、MONITOR、または SELECT 命令 (ANDxx または ORxx 命令を伴う場合もある)、および ENDyy 命令は、構造化グループを区切ります。 ENDDO 命令は、それぞれの DO、DOUxx、および DOWxx グループを終了するか、 または指定された終了条件が満たされるまでその構造化グループを繰り返し処理 します。 ENDFOR 命令は、各 FOR グループを終了します。 SELECT は ENDSL で終了しなければなりません。 IFxx 命令 および ELSE を伴う IFxx 命令は、ENDIF 命令で終了しなければなり ません。

ANDxx、DOUxx、DOWxx、IFxx、ORxx、および WHENxx の各命令コードで比較を行う場合の規則は、567 ページの『比較命令』に示されている規則と同じです。

ANDxx、DOUxx、DOWxx、IFxx、ORxx、および WHENxx 命令で、xx は次の値に することができます。

xx
意味

GT
演算項目 1 は演算項目 2 より大きい。

LT
演算項目 1 は演算項目 2 より小さい。

EQ
演算項目 1 は演算項目 2 と等しい。

NE
演算項目 1 は演算項目 2 と等しくない。

GE
演算項目 1 は演算項目 2 より大きいか等しい。

LE
演算項目 1 は演算項目 2 より小さいか等しい。

ENDyy 命令で、yy は次の値にすることができます。

yy
意味

CS
CASxx 命令の終わり。

DO
DO、DOUxx、および DOWxx 命令の終わり。

FOR
FOR 命令の終わり。

IF
IFxx 命令の終わり。

SL
SELECT 命令の終わり。

ブランク
任意の構造化命令の終わり。

注 : ENDyy 命令の yy は任意指定です。

構造化グループ (この場合には DO グループ) に別の完全な構造化グループ が入っていると、ネストされた構造化グループが形成されます。 構造化グループは最大 100 レベルの深さまでネストすることができます。 次の例は、3 レベルの深さまでネストされた構造化グループです。

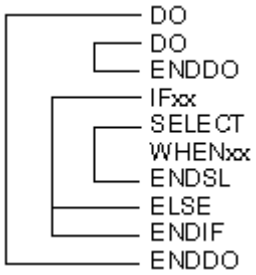


図 *191.* ネストされた構造化グループ

構造化グループを指定する場合には次のことに留意してください。

- ネストされたそれぞれの構造化グループは、外側のレベルの構造化グルー プ内に完全に含まれていなければなりません。
- それぞれの構造化グループには、DO、DOUxx、DOWxx、FOR、IFxx、または SELECT 命令の 1 つと、それに対応する ENDyy 命令が含まれている必要があります。
- 構造化グループは、明細演算、合計演算、またはサブルーチン演算に入れ ることができますが、それらの間で分割することはできません。

- 構造化グループの外から構造化グループに分岐すると、好ましくない結果 になることがあります。

サブルーチン命令

サブルーチン命令には次のものがあります。

- 714 ページの『BEGSR (サブルーチンの開始)』
- 771 ページの『ENDSR (サブルーチンの終了)』
- 781 ページの『EXSR (サブルーチンの呼び出し)』
- 797 ページの『LEAVESR (サブルーチンから抜け出す)』
- 724 ページの『CASxx (サブルーチンの条件付き呼び出し)』 (従来型の構文のみ)

CASxx を除くこれらの命令はすべて、従来型の構文と自由形式構文の両方で使用可能です。

サブルーチンとは、プログラムの中で複数回処理が可能な、そのプログラム内の演算仕様のグループです。サブルーチン指定は、1 つのプロシージャーで処理可能な他のすべての演算命令の後になければなりません。ただし、PLIST、PARM、 KLIST、KFLD、および DEFINE 命令は、ENDSR 命令 (1 つのサブルーチンの終わり) と BEGSR 命令 (別のサブルーチンの始め) の間、あるいはすべてのサブルーチンの後に指定することができます。 サブルーチンは、EXSR または CASxx 命令を使用して、演算仕様書のどこからで も呼び出すことができます。 サブルーチン行は、7 から 8 桁目の SR で識別することができます。 サブルーチン行の 7 から 8 桁目に有効な記入項目は、SR、AN、OR、またはブランクのみです。

サブルーチンのコーディング

RPG IV サブルーチンは、演算命令のどこからでも処理することができます。 すべての RPG IV 命令は 1 つのサブルーチン内で処理することができ、これらの命令は 9 から 11 桁目の有効な標識で条件を付けることができます。 7 から 8 桁目には SR またはブランクを指定することが できます。 これらの桁に制御レベル標識 (L1 から L9) を使用することはできません。 しかし、サブルーチン内の AND/OR 行は 7 から 8 桁目に指定することができます。

サブルーチンで使用するフィールドは、そのサブルーチン または残りのプロシージャーの中で定義することができます。 いずれの場合にも、フィールドはプロシージャーの本体とサブルーチンの両方 で使用することができます。

サブルーチンに別のサブルーチンを入れることはできません。 サブルーチンは別のサブルーチンを呼び出すことができます。 すなわち、サブルーチンには EXSR または CASxx を入れることができます。 しかし、サブルーチン内の EXSR または CASxx の指定で直接それ 自体を呼び出すことはできません。 別のサブルーチンを介してそれ自体を間接的に呼び出すことも、 予測できない結果になる場合があるので、行ってはなりません。 同じサブルーチン内の別の場所に分岐したい場合には、GOTO および TAG 命令コードを使用してください。

サブルーチンを、使用する順序に指定する必要はありません。 それぞれのサブルーチンは固有の記号名を持っていて、BEGSR および ENDSR ス テートメントを含んでいなければなりません。

サブルーチンの中では GOTO (分岐) 命令を使用することができます。 GOTO で はそのサブルーチンに対応する ENDSR 命令のラベルは指定できますが、BEGSR 命令の名前を指定することはできません。 GOTO が TAG または ENDSR と同じサブルーチン内にある場合を除き、サブルー チン内の TAG または ENDSR に GOTO を出すことはできません。 LEAVESR 命令を使用すると、サブルーチン内の任意のポイントからそのサブルーチンを終了できます。 制御は、そのサブルーチンの ENDSR 命令に渡されます。 LEAVESR はサブルーチン内からのみ使用できます。

サイクル・メイン・プロシージャーのサブルーチン内の GOTO は、同じサブルーチン、明細演算、または合計演算の中の TAG に出すことができます。 サブプロシージャーのサブ ルーチン内の GOTO は、同じサブルーチン内またはサブプロシージャーの本体 内の TAG に出すことができます。

サブルーチンのコーディングの例

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 * For a subroutine, positions 7 and 8 can be blank or contain SR.
 *
C                           :
C                           :
C                   EXSR      SUBRTB
C                           :
C                           :
C                           :
CL2                 EXSR      SUBRTA
C                           :
C                           :
C                           :
C     SUBRTA        BEGSR
C                           :
C                           :
C                           :
 *
 *  One subroutine can call another subroutine.
 *
C                   EXSR      SUBRTC
C                           :
C                           :
C                           :
C                   ENDSR
C     SUBRTB        BEGSR
C                           :
C                           :
C                           :
 *
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  GOTO and TAG operations can be used within a subroutine.
 *
C     START         TAG
C                           :
C                           :
C                           :
C   23              GOTO      END
C                           :
C                           :
C                           :
C   24              GOTO      START
C     END           ENDSR
C     SUBRTC        BEGSR
C                           :
C                           :
C                           :
C                   ENDSR
 *
```

図 *192.* サブルーチンのコーディング例

テスト命令

テスト命令には以下のものがあります。

- 893 ページの『TEST (日付/時刻/タイム・スタンプのテスト)』
- 894 ページの『TESTB (ビットのテスト)』
- 896 ページの『TESTN (数字のテスト)』
- 897 ページの『TESTZ (ゾーンのテスト)』.

TEST は、従来型の構文と自由形式構文の両方で使用可能です。 その他の命令は従来型の構文の中でのみ、使用可能です。 TESTB の機能の複製に %BITAND を使用する方法の例については、619 ページの図 204 を参照してください。

TESTx 命令では、結果のフィールドに指定されたフィールドをテストするこ とができます。 TEST は、日付、時刻、またはタイム・スタンプ・データの妥当性をテストします。 TESTB は、結果のフィールドのビット・パターンをテストします。 TESTN は、結果のフィールドに指定された文字フィールドにすべて数字または 先行ブランクのついた数字が入っているか、すべてブランクかをテストします。 TESTZ は、結果のフィールドに指定された文字フィールドの左端文字のゾーン 部分をテストします。 これらの命令の結果は結果の標識で示されます。

XML 命令

XML 命令には、XML 文書の SAX 構文解析および変数への直接読み込 みがあります。

XML 命令には次のものがあります。

- 947 ページの『XML-SAX (XML 文書の構文解析)』
- 908 ページの『XML-INTO (XML 文書の変数への構文解析)』
- 708 ページの『%XML (xmlDocument {:options})』
- 648 ページの『%HANDLER (handlingProcedure : communicationArea)』

%HANDLER および %XML の各組み込み関数は、値を戻さない特別な組み込み関数です。 これらの関数は、XML 命令コード XML-SAX および XML-INTO でのみ使用されます。

XML-SAX は、イベントを処理する SAX 処理プロシージャーを継続的に呼び出す SAX 構文解析を開始します。

XML-INTO は、XML 文書内の情報をプログラム変数にコピーします。

反復する多数の XML 要素がある XML 文書の場合、この命令を使用して XML-INTO 処理プロシージャーに要素を受け渡し、同時に処理する XML 要素の数を限定できます。

RPG プログラムにおける XML 文書の処理について詳しくは、「*Rational Development Studio for i: ILE RPG プログラマーの手引き*」を参照してください。

式

式とは、自由形式の構文を使用してプログラム論理を表す方法です。 式を使用すると、固定形式ステートメントより読みやすく、簡潔な方法 でプログラム・ステートメントを作成することができます。

式は、オペランドと演算子の単純なグループです。 たとえば、以下が有効な式です。

```
A+B*21
STRINGA + STRINGB
D = %ELEM(ARRAYNAME)
*IN01 OR (BALANCE > LIMIT)
SUM + TOTAL(ARRAY:%ELEM(ARRAY))
'The tax rate is ' + %editc(tax : 'A') + '%.'
```

式は、次のステートメントでコーディングされる場合があります。

- 721 ページの『CALLP (プロトタイプ・プロシージャーまたはプログラムの呼び出し)』
- 728 ページの『CHAIN (ファイルからのランダム検索)』 (自由形式演算のみ)
- 734 ページの『CLEAR (消去)』(自由形式演算のみ)
- 740 ページの『DATA-GEN (変数からの文書の生成)』
- 743 ページの『DATA-INTO (文書の変数への構文解析)』
- 757 ページの『DELETE (レコードの削除)』 (自由形式演算のみ)
- 765 ページの『DSPLY (メッセージ表示)』(自由形式演算のみ)
- 760 ページの『DOU (条件が真になるまでの繰り返し)』
- 763 ページの『DOW (条件が真の間繰り返し)』
- 769 ページの『ELSEIF (ELSE IF)』
- 771 ページの『EVAL (式の評価)』

- 773 ページの『EVALR (式の評価、右寄せ)』
- 774 ページの『EVAL-CORR (対応するサブフィールドの代入)』
- 784 ページの『FOR (For)』
- 786 ページの『FOR-EACH (それぞれの場合)』
- 789 ページの『IF (If)』
- 839 ページの『ON-EXIT (終了時)』
- 867 ページの『RETURN (呼び出し元への戻し)』
- 853 ページの『READE (等しいキーのレコードの読み取り)』 (自由形式演算のみ)
- 857 ページの『READPE (等しいキーの前のレコードの読み取り)』 (自由形式演算のみ)
- 873 ページの『SETGT (より大きい設定)』 (自由形式演算のみ)
- 876 ページの『SETLL (下限の設定)』 (自由形式演算のみ)
- 881 ページの『SORTA (配列の分類)』
- 902 ページの『WHEN (真の場合に選択)』
- 908 ページの『XML-INTO (XML 文書の変数への構文解析)』
- 947 ページの『XML-SAX (XML 文書の構文解析)』

598 ページの図 193 は、式の使用例をいくつか示しています。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
* The operations within the DOU group will iterate until the
* logical expression is true.  That is, either COUNTER is less
* than MAXITEMS or indicator 03 is on.
/FREE
    dou  counter < MAXITEMS or *in03;
    enddo;

    // The operations controlled by the IF operation will occur if
    // DUEDATE (a date variable) is an earlier date than
    // December 31, 1994.
    if DueDate < D'12-31-94';
    endif;

    // In this numeric expression, COUNTER is assigned the value
    // of COUNTER plus 1.
    Counter = Counter + 1;

    // This numeric expression uses a built-in function to assign the numb
    // of elements in the array ARRAY to the variable ARRAYSIZE.
    ArraySize = %elem (Array);

    // This expression calculates interest and performs half adjusting on
    // the result which is placed in the variable INTEREST.
    eval(h) Interest = Balance * Rate;

    // This character expression builds a sentence from a name and a
    // number using concatentation.  You can use built-in function
    // %CHAR, %EDITC, %EDITW or %EDITFLT to convert the numeric value
    // to character data.
    // This statement produces 'Id number for John Smith is 231 364'
    String = 'Id number for '
             + %trimr (First) + ' ' + %trimr (Last)
             + ' is ' + %editw (IdNum: '   &   ');

    // This expression adds a duration of 10 days to a date.
    DueDate = OriginalDate + %days(10);

    // This expression determines the difference in seconds between
    // two time values.
    Seconds = %diff (CompleteTime: t'09:00:00': *seconds);

    // This expression combines a date value and a time value into a
    // timestamp value.
    TimeStamp = TransactionDate + TransactionTime;
/END-FREE
```

図 *193.* 式の例

一般的な式の規則

以下は、すべての式に適用される一般的な規則です。

1. 式は、演算仕様書の拡張項目 2 記入項目でコーディングされるか、または自由形式演算では命令コードの後にコーディングされます。
2. 式は、複数の仕様書に継続することができます。 継続仕様書で使用可能な記入項目は、6 桁目の C と 拡張演算項目 2 記入項目だけです。

   特殊継続文字は、式がリテラルまたは名前の中で分割されるので ない限り、必要ありません。
3. ブランク (括弧など) が必要なのは、あいまいさを解決するためだけです。 しかし、ブランクを使用して読みやすさを向上させることができます。

   RPG は、式の各トークンを解析するときは、可能な限り多くの文字を読み取ることに 注意してください。 例えば、

   - X**DAY は X が DAY のべき乗になります。
   - X* *DAY は X に *DAY が乗じられます。

4. 式の中で実行される演算には、TRUNCNBR オプション (制御仕様書 のコマンド・パラメーターまたはキーワードとしての) は 適用されません。 式の命令中にオーバーフローがおこった場合 は、必ず例外が出されます。

式のオペランド

オペランドは、任意のフィールド名、名前付き定数、リテラル、 または値を戻しているプロトタイプ・プロシージャーにすることができます。 また、命令の結果を、別の命令に対するオペランドとして使用することもできます。 たとえば、式 A+B*21 では、B*21 の結果は追加の命令に対するオペランド です。

式の演算子

演算子にはいくつかのタイプがあります。

単項演算
単項演算は、後ろにオペランドを 1 つ付けて演算子を指定することによって コーディングされます。 単項演算子には次のものがあります。

+
単項のプラス演算は、数値オペランドの値を維持します。

-
単項のマイナス演算は、数値オペランドの値を否定します。 たとえば、NUMBER に値 123.4 がある 場合、 -NUMBER の値は -123.4 となります。

NOT
論理否定演算は、標識オペランドの値が '0' である場合は '1' を 戻し、標識オペランドが '1' である場合は '0' を戻します。 比較演算、あるいは演算 AND または OR の結果は、 タイプ標識の値であることを覚えておいてください。

2 項演算
2 項演算は、2 つのオペランド間に演算子を指定することによって コーディングされます。 2 項演算子には次のものがあります。

+
この演算の意味は、オペランドのタイプによって異なります。 これは次の場合に使用することができます。

1. 2 つの数値の加算
2. 日付、時刻、またはタイム・スタンプへの期間の加算
3. 2 つの文字値、2 つの図形値、または 2 つの UCS-2 値の連結
4. 基底ポインターへの数値オフセットの加算
5. 日付と時刻を結合してタイム・スタンプを作成

-
この演算の意味は、オペランドのタイプによって異なります。 これは次の場合に使用することができます。

1. 2 つの数値の減算
2. 日付、時刻、またはタイム・スタンプからの期間の減算
3. 基底ポインターからの数値オフセットの減算
4. 2 つのポインターの減算

*****
乗算演算は 2 つの数値を乗算するために使用されます。

/
除算演算は、2 つの数値を除算するために使用されます。

******
指数演算は、ある数を別の数のべき乗にする場合に使用されます。 たとえば、2**3 の値は 8 です。

=
等号演算は、2 つのオペランドが等しい場合には '1' を戻し、そうでない 場合は、'0' を戻します。

<>
非等号演算は、2 つのオペランドが等しい場合は '0' を戻し、そうでない 場合は '1' を戻します。

>
より大演算は、第 1 オペランドが第 2 オペランドより 大きい場合には '1' を戻します。

>=
より大または等号演算は第 1 オペランドが第 2 オペランドと 等しいか、それより大きい場合には '1' を戻します。

<
より小演算は、第 1 オペランドが第 2 オペランドより小さい 場合には '1' を戻します。

<=
より小または等号演算は、第 1 オペランドが第 2 オペランド と等しいか、それより小さい場合には '1' を戻します。

AND
論理 AND 演算は、両方のオペランドに標識 '1' がある場合には '1' を 戻します。

または
論理 OR 演算は、いずれかのオペランドに標識 '1' がある場合には '1' を 戻します。

IN
IN 命令は、第 1 オペランドが第 2 オペランドの要素と等しい場合、または第 1 オペランドが第 2 オペランドで指定された範囲内にある場合に、戻り値「1」を戻します。

代入演算

代入演算は、代入のターゲットを指定し、その後に代入演算子を指定し、 さらにそのターゲットに代入する式を指定することによってコーディングされます。 op= 形式 (例えば +=) の複合代入演算子では、別の演算のオペランドの 1 つとしてそのターゲットを使用し、その演算に代入を結合します。 = 代入演算子は、EVAL 演算および EVALR 演算で使用されます。 op= 複合代入演算子は、EVAL 演算でのみ使用されます。 代入演算子には次のものがあります。

- = 式がターゲットに代入されます。
- += 式がターゲットに加算されます。
- -= 式がターゲットから減算されます。
- *= ターゲットが式によって乗算されます。
- /= ターゲットが式によって除算されます。
- **= ターゲットに、ターゲットを式で累乗した値が代入されます。

組み込み関数
組み込み関数については、551 ページの『組み込み関数』で説明します。

ユーザー定義の関数
式の中で使用可能な値を戻すプロトタイプ・プロシージャー。 このプロシージャーへの呼び出しは、そのプロシージャーの戻り値と 同じタイプの値が使用される場所ならどこにでも入れることができます。 たとえば、プロシージャー MYFUNC が文字値を戻すとします。 以下は MYFUNC に対する 3 つの呼び出しを示しています。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/FREE
     if  MyFunc (string1) = %trim (MyFunc (string2));
         %subst(X(3))= MyFunc('abc');
     endif;
/END-FREE
```

図 *194.* 式でのプロトタイプ・プロシージャーの使用

ユーザー定義の関数について詳しくは、96 ページの『サブプロシージャーおよびサブルーチン』を参照してください。

IN 演算子

```
X IN Y
```

IN 演算子は 2 進条件ステートメントで使用され、IN 演算子の第 1 オペランドとして指定された項目が以下の条件を満たしているかどうかを判別します。

- %RANGE を使用した値の範囲内にある
- 配列またはサブ配列内のいずれかの要素に等しい
- %LIST 組み込み関数内の項目の 1 つと等しい

IN 演算子は FOR-EACH 命令コードと一緒に使用することもできます。

IN 演算子の例

以下の例の中の標識 *isValid* は、*requestDate* が *startDate* から *endDate* の範囲にある場合、*ON に設定されます。

```
isValid = requestDate IN %RANGE(startDate : endDate);
```

以下の例では、*item* が配列 *availableItems* 内にない場合、IF ステートメントは真になります。%SUBARR は、検査される配列要素数を制限するために使用されます。

```
IF NOT item IN %SUBARR(availableItems : 1 : numAvailabeItems);
   declineOrder (item);
ENDIF;
```

詳しくは、656 ページの『%LIST (項目 { : 項目 { : 項目 ... } })』および 677 ページの『%RANGE (下限 : 上限)』を参照してください。

演算の優先順位

演算の優先順位によって、式の中で実行される演算の順序が 決まります。 高い優先順位の演算は低い優先順位の演算の前に実行されます。

括弧は最高の優先順位を持つので、括弧の中の演算が常に最初に実行されま す。

優先順位が同じ演算 (たとえば、A+B+C) は、左から右の順序 で評価されます。ただし、** の場合は、右から左に評価されます。

(式は左から右に評価されますが、これは、オペランドも左から右に評価されることを意味するわけではありません。 詳しくは、613 ページの『評価の順序』を参照してください。)

以下のリストは、演算の優先順位を最高位から最低位まで示したものです。

1. ()
2. 組み込み関数、ユーザー定義の関数
3. 単項 +、単項 -、NOT
4. **
5. *, /
6. 2 項 +、2 項 -
7. =、<>、>、>=、<、<=、IN

8. AND
9. または

602 ページの図 195 は、優先順位の仕組みを示しています。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
* The following two operations produce different results although
* the order of operands and operators is the same.  Assume that
* PRICE = 100, DISCOUNT = 10, and TAXRATE = 0.15.
* The first EVAL would result in a TAX of 98.5.
* Since multiplication has a higher precedence than subtraction,
* DISCOUNT * TAXRATE is the first operation performed.  The result
* of that operation (1.5) is then subtracted from PRICE.

/FREE
    TAX = PRICE - DISCOUNT * TAXRATE;

    // The second EVAL would result in a TAX of 13.50.
    // Since parentheses have the highest precedence the operation
    // within parenthesis is performed first and the result of that
    // operation (90) is then multiplied by TAXRATE.

    TAX = (PRICE - DISCOUNT) * TAXRATE;
/END-FREE
```

図 *195.* 優先順位の例

データ・タイプ

式の中ではすべてのデータ・タイプが使用可能です。 ただし、演算によっては、オペランドとして特定のデータ・タイプしか サポートしないものがあります。 たとえば、* 演算は、オペランドとして数値しか許容しません。 関係および論理演算は、タイプ標識の値を戻すことに注意してください。 これは特殊なタイプの文字データです。 この結果、関係または論理結果を、文字オペランドを予期している演算に 対するオペランドとして使用することができます。

式のオペランドによってサポートされるデータ・タイプ

602 ページの表 121 は、各単項演算子に使用できるオペランドのタイプと結果のタイプを説明しています。602 ページの表 122 は、各 2 項演算子に使用できるオペランドのタイプと結果のタイプを説明しています。603 ページの表 123 は、各組み込み関数に使用できるオペランドのタイプと結果のタイプを説明しています。 プロトタイプ・プロシージャーは、プロトタイプ定義で定義されていればどのデータ・タイプもサポート します。

表 *121.* 単項演算にサポートされるタイプ

命令	オペランド・タイプ	結果のタイプ
- (否定)	数値	数値
+	数値	数値
NOT	標識	標識

表 *122. 2* 項演算にサポートされるオペランド

演算子	オペランド 1 タイプ	オペランド 2 タイプ	結果のタイプ
+ (加法)	数値	数値	数値
+ (加法)	日付	期間	日付
+ (加法)	時刻	期間	時刻
+ (加法)	タイム・スタンプ	期間	タイム・スタンプ

表 122. 2 項演算にサポートされるオペランド (続き)			
演算子	**オペランド 1 タイプ**	**オペランド 2 タイプ**	**結果のタイプ**
- (減算)	数値	数値	数値
- (減算)	日付	期間	日付
- (減算)	時刻	期間	時刻
- (減算)	タイム・スタンプ	期間	タイム・スタンプ
* (乗算)	数値	数値	数値
/ (除法)	数値	数値	数値
** (指数)	数値	数値	数値
+ (連結)	文字、図形、UCS-2	文字、図形、UCS-2	文字、図形、UCS-2
+ (ポインターへのオフの加算)	基底ポインター	数値	基底ポインター
- (減算ポインター)	基底ポインター	基底ポインター	数値
- (ポインターからのオフセットの減算)	基底ポインター	数値	基底ポインター
注 : 次の演算の場合は、オペランドは任意のタイプとすることができますが、 2 つのオペランドは同じタイプでなければなりません。			
= (等しい)	任意	任意	標識
>= (より大か等しい)	任意	任意	標識
> (より大)	任意	任意	標識
<= (より小さいか等しい)	任意	任意	標識
< (より小)	任意	任意	標識
<> (等しくない)	任意	任意	標識
AND (論理 AND)	標識	標識	標識
OR (論理 OR)	標識	標識	標識
IN	任意	任意のタイプの配列、または %LIST	標識

表 123. 組み込み関数にサポートされるタイプ		
命令	**オペランド**	**結果のタイプ**
%ABS	数値	数値
%ALLOC	数値	ポインター
%BITAND	文字:文字 {: 文字... }	文字
%BITAND	数値:数値 {: 数値... }	数値
%BITNOT	文字	文字
%BITNOT	数値	数値
%BITOR	文字:文字 {: 文字... }	文字

表 123. 組み込み関数にサポートされるタイプ (続き)

命令	オペランド	結果のタイプ
%BITOR	数値:数値 {: 数値... }	数値
%BITXOR	文字:文字	文字
%BITXOR	数値:数値	数値
%CHAR	文字、図形、UCS-2 {: ccsid }	指定された CCSID の可変長文字
%CHAR	数値	ジョブ CCSID の可変長文字
%CHAR	日付、時刻またはタイム・スタンプ {: 日付、時刻、またはタイム・スタンプの形式 }	ジョブ CCSID の可変長文字
%CHECK	文字、図形、または UCS-2 {: 数値 }	数値
%CHECKR	文字、図形、または UCS-2 {: 数値 }	数値
%DATE	{文字、数値、またはタイム・スタンプ {: 日付の形式 } }	日付
%DAYS	数値	数値 (期間)
%DEC	文字 : 数値定数 : 数値定数	数値 (パック)
%DEC	数値 {: 数値定数 : 数値定数 }	数値 (パック)
%DEC	日付、時刻、またはタイム・スタンプ {: 形式 }	数値 (パック)
%DECH	文字 : 数値定数 : 数値定数	数値 (パック)
%DECH	数値 : 数値定数 : 数値定数	数値 (パック)
%DECPOS	数値	数値 (符号なし)
%DIFF	日付、時刻、またはタイム・スタンプ : 日付、 時刻、またはタイム・スタンプ : 単位 {: 秒の小数部の桁数 }	数値 (期間) (双方に互換性のある)
%DIV	数値 : 数値	数値
%EDITC	非浮動数値 : 長さ 1 の 文字定数 {:*CURSYM | *ASTFILL | 文字通貨記号 }	文字 (固定長)
%EDITFLT	数値	文字 (固定長)
%EDITW	非浮動数値 : 文字定数	文字 (固定長)
%EOF	{File name}	標識
%EQUAL	{File name}	標識
%ERROR		標識
%FLOAT	文字	数値 (浮動)
%FLOAT	数値	数値 (浮動)
%FOUND	{File name}	標識
%GRAPH	文字、図形、または UCS-2 {: ccsid }	指定された CCSID のグラフィック
%HOURS	数値	数値 (期間)
%INT	文字	数値 (整数)
%INT	数値	数値 (整数)
%INTH	文字	数値 (整数)

表 123. 組み込み関数にサポートされるタイプ (続き)

命令	オペランド	結果のタイプ
%INTH	数値	数値 (整数)
%LEN	任意	数値 (符号なし)
%LIST	任意 { : 任意 { : 任意 ...}	配列 (タイプはオペランドによって異なる)
%LOOKUPxx	任意 : 任意の配列 {: 数値 {: 数値 } }	数値 (符号なし)
%LOWER	文字 : {: 数値 {: 数値}}	文字
%LOWER	UCS-2 : {: 数値 {: 数値}}	UCS-2
%MAX	任意 : 任意 { : 任意 ...}	任意 (オペランドに依存)
%MAXARR	任意の配列 {: 数値 {: 数値}}	数値
%MIN	任意 : 任意 { : 任意 ...}	任意 (オペランドに依存)
%MINARR	任意の配列 {: 数値 {: 数値}}	数値
%MINUTES	数値	数値 (期間)
%MONTHS	数値	数値 (期間)
%MSECONDS	数値	数値 (期間)
%OCCUR	複数オカレンス・データ構造	複数オカレンス・データ構造
%OPEN	ファイル名	標識
%PARMS		数値 (整数)
%REALLOC	ポインター : 数値	ﾎﾟｲﾝﾀｰ
%REM	数値 : 数値	数値
%REPLACE	文字 : 文字 {: 数値 {: 数値 } }	文字
%REPLACE	図形 : 図形 {: 数値 {: 数値 } }	ｸﾞﾗﾌｨｯｸ
%REPLACE	UCS-2 : UCS-2 {: 数値 {: 数値 } }	UCS-2
%SCAN	文字 : 文字 {: 数値 {: 長さ]}	数値
%SCAN	図形 : 図形 {: 数値 {: 長さ}}	数値
%SCAN	UCS-2 : UCS-2 {: 数値 {: 長さ}}	数値
%SCANR	文字 : 文字 {: 数値 {: 長さ]}	数値
%SCANR	図形 : 図形 {: 数値 {: 長さ}}	数値
%SCANR	UCS-2 : UCS-2 {: 数値 {: 長さ}}	数値
%SCANRPL	文字 : 文字 : 文字 {: 数値 {: 数値 } }	文字
%SCANRPL	図形 : 図形 : 図形 {: 数値 {: 数値 } }	ｸﾞﾗﾌｨｯｸ
%SCANRPL	UCS-2 : UCS-2 : UCS-2 {: 数値 {: 数値 } }	UCS-2
%SECONDS	数値	数値 (期間)
%SHTDN		標識
%SPLIT	文字 : {: 文字 }	文字配列
%SPLIT	グラフィック : {: グラフィック }	グラフィック配列

表 123. 組み込み関数にサポートされるタイプ (続き)

命令	オペランド	結果のタイプ
%SPLIT	UCS-2 : {: UCS-2 }	UCS-2 配列
%SQRT	数値	数値
%STATUS	{File name}	数字 (ゾーン 10 進)
%STR	基底ポインター {: 数値 }	文字
注 : %STR が式の左辺に付いている場合、第 2 オペランドが必要です。		
%SUBARR	任意: 数値 {:数値 }	任意 (第 1 オペランドと同じタイプ)
%SUBDT	日付、時刻、またはタイム・スタンプ : 単位	数値 (符号なし)
%SUBDT	日付、時刻、またはタイム・スタンプ : 単位 : 桁数 { : 秒の小数部の桁数 }	数値 (パック 10 進数)
%SUBST	文字 : 数値 {: 数値 }	文字
%SUBST	図形 : 数値 {: 数値 }	ｸﾞﾗﾌｨｯｸ
%SUBST	UCS-2 : 数値 {: 数値 }	UCS-2
%THIS		オブジェクト
%TIME	{文字、数値、あるいはタイム・スタンプ {: 時刻の形式 } }	時刻
%TIMESTAMP	{文字、数値 {: タイム・スタンプ形式 { : 秒の小数部の桁数 } } }	タイム・スタンプ
%TIMESTAMP	{*SYS、タイム・スタンプ、日付 { 秒の小数部の桁数 } }	タイム・スタンプ
%TLOOKUPxx	任意のテーブル: 任意のテーブル {: 任意 }	標識
%TRIM	文字 { : 文字 }	文字
%TRIM	図形 { : 図形 }	ｸﾞﾗﾌｨｯｸ
%TRIM	UCS-2 { : UCS-2 }	UCS-2
%TRIML	文字 { : 文字 }	文字
%TRIML	図形 { : 図形 }	ｸﾞﾗﾌｨｯｸ
%TRIML	UCS-2 { : UCS-2 }	UCS-2
%TRIMR	文字 { : 文字 }	文字
%TRIMR	図形 { : 図形 }	ｸﾞﾗﾌｨｯｸ
%TRIMR	UCS-2 { : UCS-2 }	UCS-2
%UCS2	文字、図形、または UCS-2 {: ccsid }	指定された CCSID の可変長 UCS-2 値
%UNS	文字	数値 (符号なし)
%UNS	数値	数値 (符号なし)
%UNSH	文字	数値 (符号なし)
%UNSH	数値	数値 (符号なし)
%UPPER	文字 : {: 数値 {: 数値}}	文字

表 123. 組み込み関数にサポートされるタイプ (続き)		
命令	**オペランド**	**結果のタイプ**
%UPPER	UCS-2 : {: 数値 {: 数値}}	UCS-2
%XFOOT	数値	数値
%XLATE	文字、図形、または UCS-2 : 文字、図形、または UCS-2 : 文字、図形、または UCS-2 {: 数値 }	文字、図形、または UCS-2
%YEARS	数値	数値 (期間)
注 : 次の組み込み関数の場合は、引数はリテラル、名前付き定数、または変 数でなければなりません。		
%PADDR	文字	プロシージャーまたはプロトタイプ・ポインター
%SIZE	任意 {: *ALL }	数値 (符号なし)
注 : 次の組み込み関数の場合は、引数は変数でなければなりません。 しかし、配列指標が指定された場合には、有効な任意の数値式とすることができ ます。		
%ADDR	任意	基底ポインター
%ELEM	任意	数値 (符号なし)
%NULLIND	任意	標識
注 : 次の組み込み関数は値を戻さないため、正確には組み込み関数ではありません。 これらは、一部の自由形式演算で使用されます。		
%FIELDS	任意{: 任意 {: 任意 ... }	適用除外
%GEN	文字またはプロシージャー・ポインター { : 任意}	適用除外
%HANDLER	プロトタイプ名: 任意	適用除外
%KDS	データ構造 {: 数値 }	適用除外
%PARSER	文字またはプロシージャー・ポインター { : 任意}	適用除外
%RANGE	任意 { : 任意}	適用除外
%XML	文字、または UCS-2 { : 文字 }	適用除外

数値中間結果の形式

2 項演算に数値フィールドが含まれている場合には、中間結果の形式はオペラン ドの形式によって決まります。

演算子 +、-、および * の場合:

- 少なくとも 1 つのオペランドが浮動形式であった場合には、結果は浮動形式になります。
- そうでない場合、少なくとも 1 つのオペランドがパック 10 進数、ゾーン 10 進数、または 2 進形式であった場合には、結果はパック 10 進数形式になります。
- そうでない場合、少なくとも 1 つのオペランドが整数であった場合には、結果は整数形式になります。
- これ以外の場合、結果は符号なし形式になります。
- 浮動形式でない数値リテラルは次のようになります。
  - リテラルが符号なし整数の範囲の中にある場合は、そのリテラルは符号なし整数であると想定されます。

- そうでない場合、リテラルが整数の範囲の中にある場合は、そのリテラルは整数であると想定されます。
- そうでない場合、リテラルはパック 10 進数であると想定されます。

/ 演算子の場合:

1 つのオペランドが浮動であるか、または制御仕様書で FLTDIV キーワードが指定されている場合は、/ 演算子の結果は浮動になります。 そうでなければ、結果はパック 10 進数になります。

** 演算子の場合:

結果は浮動形式で表されます。

数値演算の精度の規則

固定形式命令コード (プログラマーが個々の演算の結果を常に指定 しなければならない) の場合とは異なり、RPG が、式の中の各演算の 結果の形式および精度を決定しなければなりません。

演算に、浮動形式、整数形式、または符号なし形式の結果が ある場合、精度はその型式の最大サイズとなります。 整数および符号なし演算は、4 バイト値を生成し、浮動演算は 8 バイト値を 生成します。

ただし、演算にパック 10 進数、ゾーン 10 進数、または 2 進数型式が ある場合、結果の精度はオペランドの精度によって異なります。

10 進演算の精度規則をよく知っておくことは重要です。これは、比較的単純な式でも、予期しない結果が出る場合があるためです。 たとえば、乗算の 2 つのオペランドの大きさ十分である場合、乗算の結果 の小数位はゼロになります。 2 つの 40 桁の数を乗算している場合、その乗算で出される 可能性のあるすべての結果を収容するために、理想的には 80 桁が必要になります。 しかし、RPG は 63 桁までの数値しか サポートしないため、結果は 63 桁に調整されます。この場合、結果から最大 17 桁の数字が切り捨てられます。

中間結果のサイズを制御するために、次の 2 組の精度規則を使用する ことができます。

1. デフォルトの規則では、数値のオーバーフローの可能性を最小限に 抑えるために、できるだけ大きい中間結果を出します。 残念ながら、その結果があまりにも大きいと、場合によって小数位のない 結果が出される場合が生じます。
2. 「結果の小数点以下の桁数」精度規則は、デフォルトの規則と同じ働きを しますが、ステートメントに、数値変数への割り当て、または特定 の 10 進精度への変換が含まれる場合、中間結果の小数点以下の桁数は、 希望する結果の小数位より少なくなることは絶対にない、という点が 異なります。

   実際には、数値式のコーディング時にコンパイル・リストを調べる場合に は、正確な精度について心配する必要はありません。 診断メッセージが、中間結果の小数点以下の桁数が切り捨てられていることを示します。 式の中に割り当てが含まれている場合、命令コード拡張 (R) を コーディングすることによって、そのステートメントに関して「結果の 小数点以下の桁数」精度規則を使用すると、その小数点以下の桁数を 確保することができます。

   「結果の小数点以下の桁数」精度規則が使用できない場合 (たとえば、 関係式の中などで)、組み込み関数 %DEC を使用して、副次式の結果 を小さい精度に変換し、小数点以下の桁数が失われないように することができます。

デフォルトの精度規則の使用

デフォルトの精度規則を使用すると、式の中の 10 進数の中間の精度が、 数値オーバーフローの可能性を最小限に抑えるように計算されます。 ただし、その式に大きな 10 進数の演算がいくつか含まれている場合、小数点以下の桁がない中間となる可能性があります。 (特に、式に 2 つ以上のネストされた部分がある場合はそれが顕著です。) これは、特に割り当てにおいて、プログラマーが期待している結果ではありません。

10 進数の中間の精度を決定するときは、次の 2 つのステップを経て 行われます。

1. 結果の、希望の精度あるいは「自然の」精度が計算されます。

2. 自然の精度が 63 桁より大きい場合、精度は 63 桁に収まるように調整されます。 普通、この調整は、最初に小数点以下の桁数を減らし、次に必要に応じて 中間の合計桁数を減らす、という手順で行われます。

この作業はデフォルトのもので、モジュール全体 (制御仕様書キーワード の EXPROPTS(*MAXDIGITS) を使用) について指定することも、あるいは 単一の自由形式の式 (命令コード拡張 M を使用) について指定することもできます。

中間結果の精度

609 ページの表 124 では、デフォルトの精度規則について詳しく説明します。

表 *124.* 中間結果の精度

命令	結果の精度
注 : 次の演算では数値結果が作成されます。 L1 および L2 は、2 つのオペランドの桁数です。 Lr は結果の桁数です。 D1 および D2 は 2 つのオペランドの小数位の数です。 Dr は結果の小数位の数です。 T は一時的な値です。	
N1+N2	T=min (max (L1-D1, L2-D2)+1, 63) Dr=min (max (D1,D2), 63-t) Lr=t+Dr
N1-N2	T=min (max (L1-D1, L2-D2)+1, 63) Dr=min (max (D1,D2), 63-t) Lr=t+Dr
N1*N2	Lr=min (L1+L2, 63) Dr=min (D1+D2, 63-min ((L1-D1)+(L2-D2), 63))
N1/N2	Lr=63 Dr=max (63-((L1-D1)+D2), 0)
N1**N2	倍精度浮動小数点
注 : 次の演算では文字結果が作成されます。 Ln はオペランドの長さを文字数で表しています。	
C1+C2	Lr=min(L1+L2,16773104)
注 : 次の演算では DBCS 結果が作成されます。 Ln はオペランドの長さを DBCS 文字数で表しています。	
D1+D2	Lr=min(L1+L2,8386552)
注 : 次の演算では、サブタイプが標識である文字タイプの結果が作成されます。 結果は常に標識値 (1 文字) です。	
V1=V2	1 (標識)
V1>=V2	1 (標識)
V1>V2	1 (標識)
V1<=V2	1 (標識)
V1<V2	1 (標識)
V1<>V2	1 (標識)
V1 AND V2	1 (標識)

表 *124.* 中間結果の精度 *(続き)*	
命令	**結果の精度**
V1 OR V2	1 (標識)

デフォルトの精度規則の例

以下の例はデフォルトの精度の規則がどのように働くかを示しています。

```
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D FLD1            S             15P 4
D FLD2            S             15P 2
D FLD3            S              5P 2
D FLD4            S              9P 4
D FLD5            S              9P 4
CL0N01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++
C                   EVAL      FLD1 = FLD2/(((FLD3/100)*FLD4)+FLD5)
                                       (   1   )
                                     (       2   )
                                   (           3   )
                                 (   4               )
```

図 *196.* 中間結果の精度

上記の演算仕様書が処理された場合、FLD1 に割り当てられる結果の値の精度について、小数点以下の桁数は 3 桁 と予期されますが、小数点以下の桁はありません。 これは、最後の評価 (上記の例の 4) が行われたときに、演算項目が位取りされる数が負になるためです。 この理由を確認するために、式がどのように評価されるかを確認してください。

1

FLD3/100 の評価

規則は次のとおりです。

```
 Lr = 63
 Dr = max(63-((L1-D1)+D2),0)
    = max(63-((5-2)+0),0)
    = max(63-3,0)
    = 60
```

2

(1 の結果 * FLD4) の評価

規則は次のとおりです。

```
Lr = min(L1+L2,63)
   = min(63+9,63)
   = 63
Dr = min(D1+D2,63-min((L1-D1)+(L2-D2),63))
   = min(60+4,63-min((63-60)+(9-4),63))
   = min(64,63-min(4+5,63)
   = min(64,55)
   = 55
```

3

(2 の結果 + FLD5) の評価

規則は次のとおりです。

```
T  = min(max(L1-D1,L2-D2)+1,63)
   = min(max(63-55,9-4)+1,63)
   = min(max(8,5)+1,63)
   = min(9,63)
   = 9
Dr = min(max(D1,D2),31-T)
   = min(max(55,4),63-9)
```

```
   = min(55,54)
   = 54
Lr = T + Dr
   = 9 + 54 = 63
```

4

FLD2/3 の結果の評価

規則は次のとおりです。

```
Lr = 63
Dr = max(63-((L1-D1)+D2),0)
   = max(63-((15-2)+ 54),0)
   = max(63-(13+54),0)
   = max(-4,0)
****  NEGATIVE NUMBER TO WHICH FACTOR IS SCALED   ****    = 0
```

この問題を避けるためには、最初の評価が除算ではなく乗算、すなわち、FLD3 * 0.01 となるように上記の式を変更するか、%DEC 組み込み関数を 使用して、副次式 FLD3/100: %DEC(FLD3/100 : 15 : 4) を 設定するか 命令拡張 (R) を使用して、小数点以下の桁数が決して 4 桁より少なくならない ようにします。

「結果の小数点以下の桁数」精度規則の使用

「結果の小数点以下の桁数」精度規則は、10 進の中間の精度が、 小数位の数が割り当ての結果の小数点以下の桁数より少なくならないように 計算されることを意味します。 これは、以下によって指定されます。

1. 制御仕様書の EXPROPTS(*RESDECPOS)。 モジュール全体に対してこの動作を指定します。
2. 自由形式演算に指定された命令コード拡張 R。

「結果の小数点以下の桁数」規則は、次の状況において適用されます。

1. 「結果の小数点以下の桁数」規則は、パック 10 進中間結果にのみ適用 されます。 この作用は、整数、符号なし、または浮動結果を持つ演算の中間結果には 適用されません。
2. 「結果の小数点以下の桁数」規則は、10 進の目標 (パック、 ゾーン、または 2 進) への割り当て (明示的または暗黙) がある場合にのみ 適用されます。 これは、次の状況で行われます。
   a. EVAL ステートメントの場合、小数位の最小数は、割り当ての 目標の小数点以下の桁数によって指定され、この数は、その割り当ての右辺の 式に適用されます。 このステートメントに四捨五入も適用される場合、 小数点以下の最小桁数に 1 桁追加されます (ただし、最小桁数が 63 に満たない場合)。
   b. RETURN ステートメントの場合、小数位の最小数は、プロシージャー の PI 仕様に定義された戻り値の小数点以下の桁数によって指定 されます。 このステートメントに四捨五入も適用される場合、小数点以下の最小桁数に 1 桁追加されます (ただし、最小桁数が 63 に満たない場合)。
   c. VALUE または CONST パラメーターの場合、小数位の最小数は、形式パラメーター (プロシージャー・プロトタイプに 指定されている) の小数点以下の桁数によって指定され、渡された パラメーターとして指定された式に適用されます。
   d. 明示的な長さおよび小数点以下の桁数が指定されている 組み込み関数 %DEC および %DECH の場合、小数点以下の最小桁数 は、その組み込み関数の 3 番目のパラメーターによって指定され、その数は 1 番目の パラメーターとして指定された式に適用されます。

   小数点以下の最小桁数は、上記の演算の別の数によって一時変更 されていない限り、副次式全体に適用されます。 四捨五入が指定されている (H 命令コード拡張として、あるいは組み込み関数 %DECH によって) 場合、中間結果の 小数点以下の桁数は *N+1* より少なくなることは ありません。ここで、*N* は、その結果の小数点以下の桁数です。
3. 「結果の小数点以下の桁数」規則は、対応する結果がないために、 普通、条件式には適用されません。 (特定の精度との比較を行う必要がある場合、%DEC または %DECH をその 2 つの引数に使用しなければなりません。)

   それとは逆に、条件式が、小数点以下の最小桁数が指定される (上記の 技法のいずれかを用いて) 式の中に組み込まれる場合、「結果の小数点以下の桁数」規則が適用されます。

「結果の小数点以下の桁数」精度規則の例

次の例は、「結果の小数点以下の桁数」精度規則を示したものです。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
* This example shows the precision of the intermediate values
* using the two precision rules.

D p1              s             26p 2
D p2              s             26p 2
D p3              s             26p 2
D p4              s             26p 9
D s1              s             26s 2
D s2              s             26s 2
D i1              s             10i 0
D f1              s              8f
D proc            pr            15p 3
D   parm1                       20p 5 value

* In the following examples, for each sub-expression,
* two precisions are shown.  First, the natural precision,
* and then the adjusted precision.

 /FREE
      // Example 1:
    eval    p1 = p1 * p2 * p3;
      // p1*p2        -> P(52,4); P(52,4)
      // p1*p2*p3     -> P(78,6); P(63,0) (decimal positions are truncated)
    eval(r) p1 = p1 * p2 * p3;
      // p1*p2        -> P(52,4); P(52,4)
      // p1*p2*p3     -> P(78,6); P(63,2) (decimal positions do not drop
      //                 below target decimal positions)
    eval(rh)p1 = p1 * p2 * p3;
      // p1*p2        -> P(52,4); P(52,5)
      // p1*p2*p3     -> P(78,6); P(63,3) (decimal positions do not drop
      //                 below target decimals + 1)
      // Example 2:
    eval    p4 = p1 * p2 * proc (s1*s2*p4);
      // p1*p2        -> P(52,4); P(52,4)
      // s1*s2        -> P(52,4); P(52,4)
      // s1*s2*p4     -> P(78,13); P(63,0) (decimal positions are truncated)
      // p1*p2*proc() -> P(67,7); P(63,3)  (decimal positions are truncated)
    eval(r) p4 = p1 * p2 * proc (s1*s2*p4);
      // p1*p2        -> P(52,4); P(52,4)
      // s1*s2        -> P(52,4); P(52,4)
      // s1*s2*p4     -> P(78,13); P(63,5)
      // p1*p2*proc() -> P(67,7); P(63,7)  (we keep all decimals since we are
      //                  already below target decimals)
/END-FREE
```

図 *197.* 精度規則の例

短絡評価

関係演算の AND および OR は、左から右に評価されます。 しかし、その値がわかるとすぐに式の評価は停止し、値が戻されます。 結果として、式のオペランドをすべて評価する必要はありません。

演算 AND の場合、第 1 オペランドが偽であると第 2 オペランドは 評価されません。 同様に、演算 OR の場合、第 1 オペランドが真であると、第 2 オペランドは評価 されません。

この作用には 2 つの意味があります。 まず第 1 に、配列指標は、同じ式の中でテストされ、使用されます。 式

```
          I<=%ELEM(ARRAY) AND I>0 AND ARRAY(I)>10
```

によって、配列指標例外の結果にはなりません。

第 2 の意味として、第 2 オペランドがユーザー定義関数への呼び出しである 場合、その関数は呼び出されない、ということです。 これは、その関数がパラメーターまたはグローバル変数 の値を変更する場合には重要なことです。

評価の順序

式の中のオペランドの評価について保証された順序はありません。 したがって、ある変数が式の中の任意の位置で 2 回使用され、副次作用の可能性 がある場合には、予期した結果とならないことがあります。

例えば、613 ページの図 198 に示されているソースを考えてみます。ここで、A は変数、FN は A を変更するプロシージャーです。 2 番目の EVAL 命令の式部分に A が 2 回現れます。 **加算演算の左辺 (オペランド *1*) が最初に評価された場合には、**X に は値 17、つまり (5 + FN(5) = 5 + 12 = 17) が割り当てられます。 **加算演算の右辺 (オペランド *2*) が最初に評価された場合には、**X に は値 18、つまり (6 + FN(5) = 6 + 12 = 18) が割り当てられます。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
*   A is a variable.  FN is procedure that modifies A.
/free
    a = 5;
    x = a + fn(a);
/end-free

P fn              B
D fn              PI             5P 0
D   parm      5P 0
 /free
    parm = parm + 1;
    return  2 * parm;
 /end-free
P fn              E
```

図 *198.* 副次作用がある呼び出しのコーディング例

組み込み関数

この章では、それぞれの組み込み関数をアルファベット順に説明します。

%ABS (式の絶対値)

```
%ABS(numeric expression)
```

%ABS は、パラメーターとして指定されている数値式の絶対値を戻します。 数値式の値が負でない場合、その値は変更されずに戻されます。 その値が負である場合は、戻り値は式の値ですが負符号は外されて 戻されます。

%ABS は、式の中で、あるいはキーワードに対するパラメーターとして 使用されます。 キーワードと一緒に使用される場合、オペランドは、数値リテラル、 数値を示す定数名、あるいはコンパイル時に認識される数値を持つ 組み込み関数でなければなりません。

詳しくは、557 ページの『算術演算』または 551 ページの『組み込み関数』を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name +++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D f8          s              8f   inz (-1)
D i10         s             10i 0 inz (-123)
D p7          s              7p 3 inz (-1234.567)

 /FREE
    f8  = %abs (f8);         // "f8" is now 1.
    i10 = %abs (i10 - 321); // "i10" is now 444.
    p7  = %abs (p7);         // "p7" is now 1234.567.
 /END-FREE
```

図 *199. %ABS* の例

%ADDR (変数のアドレスの検索)

```
%ADDR(variable)
%ADDR(varying-length variable : *DATA)
```

%ADDR は基底ポインター・タイプの値を戻します。 この値は、指定された変数のアドレスです。 この値は基底ポインター・タイプの項目とだけ比較することができ、基底ポイン ター・タイプの項目にだけ割り当てることができます。

%ADDR は、2 番目のパラメーターとして *DATA が指定されている場合に、可変長フィールドのデータ部を戻します。

配列指標パラメーターを持つ %ADDR が定義仕様書のキーワード INZ または CONST のパラメーターとして指定された場合には、その配列指標がコンパイル時 に分かっていなければなりません。 指標は数値リテラルか数値定数のいずれかでなければなりません。

割り当ての結果が指標を持たない配列となる EVAL 命令では、割り当て演算子の右辺 の %ADDR は %ADDR の引数によって異なる意味を持ちます。 %ADDR の引数が指標 の指定がない配列名で、結果が配列名となる場合には、結果の配列の各要素には 引数の配列の先頭のアドレスが入れられることになります。
%ADDR の引数が (*) の指標を指定した配列名である場合には、結果の配列の各要 素には引数の配列の対応する要素のアドレスが入れられることになります。615 ページの図 200 を参照してください。

パラメーターとして指定された変数がテーブル、複数オカレンス・データ構造、または複数オカレンス・データ構造のサブフィールドである場合には、アドレスは現在のテーブル の指標または繰り返し番号になります。

変数が基底付きである場合には、%ADDR はその変数の基底ポインターの値を戻し てきます。 変数が基底付きデータ構造のサブフィールドである場合には、%ADDR の値は基底 ポインターにサブフィールドのオフセットを加えた値になります。

変数が *ENTRY PLIST の PARM として指定された場合には、%ADDR は呼び出し元に よってプログラムに渡されたアドレスを戻してきます。

%ADDR の引数を変更することができない場合、%ADDR は比較演算でのみ使用することができます。 変更できない引数の例として、読み取り専用参照パラメーター (プロシージャー・インターフェース で指定された CONST キーワード) があります。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
 *
 * The following set of definitions is valid since the array
 * index has a compile-time value
 *
D   ARRAY         S             20A   DIM (100)
 * Set the pointer to the address of the seventh element of the array.
D   PTR           S               *   INZ (%ADDR(ARRAY(SEVEN)))
D   SEVEN         C                   CONST (7)
 *
D DS1             DS                  OCCURS (100)D                            20A
D   SUBF                          10A
D                                 30A
D CHAR10          S               10A   BASED (P)
D PARRAY          S                 *   DIM(100)


  /FREE
    %OCCUR(DS1) = 23;
    SUBF = *ALL'abcd';
    P = %ADDR (SUBF);
    IF CHAR10 = SUBF;
       // This condition is true.
    ENDIF;
    IF %ADDR (CHAR10) = %ADDR (SUBF);
       // This condition is also true.
    ENDIF;
    // The following statement also changes the value of SUBF.
    CHAR10 = *ALL'efgh';
    IF CHAR10 = SUBF;
       // This condition is still true.
    ENDIF;
    //---------------------------------------------------------------
    %OCCUR(DS1) = 24;
    IF CHAR10 = SUBF;
      // This condition is no longer true.
    ENDIF;
    //---------------------------------------------------------------
    // The address of an array element is taken using an expression
    // as the array index.
    P = %ADDR (ARRAY (X + 10));
    //---------------------------------------------------------------
    // Each element of the array PARRAY contains the address of the
    // first element of the array ARRAY.
    PARRAY = %ADDR(ARRAY);
    // Each element of the array PARRAY contains the address of the
    // corresponding element of the array ARRAY.
    PARRAY = %ADDR(ARRAY(*));

    // The first three elements of the array PARRAY
    // contain the addresses of the first three elements
    // of the array ARRAY.
    %SUBARR(PARRAY : 1 : 3) = %ADDR(ARRAY(*));
  /END-FREE
```

図 *200. %ADDR の例*

```
1. Use %ADDR(fld:*DATA) to call a procedure with the
   address of the data portion of a varying field.

// Assume procedure "uppercaseData" requires a pointer and a length.
// %ADDR(fld:*DATA) returns the pointer to the data portion of
// the varying field, and %LEN(fld) returns the length.
uppercaseData (%ADDR(fld : *DATA) : %LEN(fld));

2. Use %ADDR(fld:*DATA) to determine the maximum size
   of the data portion of a varying field.

// The number of bytes used for the prefix is the
// offset between the address of the field and the
// address of the data.
prefix_size = %addr(fld : *data) - %addr(fld);

// The number of bytes used for the data is the
// difference between the total bytes and the
// bytes used for the prefix.
data_size = %size(fld) - prefix_size;

// If variable "fld" is UCS-2 or DBCS, the number
// of characters is half the number of bytes
max_dbcs_chars = data_size / 2;
```

図 *201. *DATA* が指定されている *%ADDR* の例

%ALLOC (記憶域の割り振り)

```
%ALLOC(num)
```

%ALLOC は、新たに割り振られる指定された長さのヒープ記憶域へのポインターを戻します。 新たに割り振られた記憶域は初期化されません。

パラメーターは、小数部ゼロの非浮動数値である必要があります。 指定される長さは、1 から最大許容サイズまでの範囲でなければなりません。

最大許容サイズは、制御仕様書の ALLOC キーワードによる、RPG メモリー管理命令に使用されるヒープ記憶域のタイプによって異なります。 モジュールがテラスペース・ヒープ記憶域を使用する場合、最大許容サイズは 4294967295 バイトです。 それ以外の場合、最大許容サイズは 16776704 バイトです。

実行時に使用可能な最大サイズは、RPG の許容最大サイズより小さい場合があります。

詳しくは、「581 ページの『メモリー管理命令』」を参照してください。

命令が正常に完了しなかった場合、例外 00425 または 00426 が出されます。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/FREE
  // Allocate an area of 200 bytes
  pointer = %ALLOC(200);
/END-FREE
```

図 *202. %ALLOC* の例

%BITAND (ビット単位の AND 演算)

```
%BITAND(expr:expr{:expr...})
```

%BITAND は、すべての引数のビットの論理積を、ビット単位で戻します。 つまり、引数内の対応するビットがすべてがオンの場合は、結果のビットはオンになり、 それ以外の場合は、結果のビットはオフになります。

この組み込み関数に対する引数は、文字でも数値でもかまいません。 数値の引数の場合は、引数が整数または符号なし数値ではないときには、引数はまず整数に変換されます。 値が 8 バイトの整数に収まらない場合は、 数値のオーバーフロー例外が発行されます。

%BITAND は複数の引数を持つことができます。 引数はすべて、同じタイプ (文字または数値) である必要があります。 結果のタイプは、引数のタイプと同じです。 数値の引数の場合は、すべての引数 が符号なし数値であるときには、結果の数値は符号なし数値になり、それ以外の場合は、結果の数値は整数になります。

長さは、最大オペランドの長さになります。 引数の長さがさまざまである場合、 数値の引数については、左側にゼロのビットが埋め込まれます。 文字の引数については、短い場合、右側に 1 のビットが埋め込まれます。

%BITAND は任意の式でコーディングすることができます。 また、すべての引数がコンパイル時に既知の場合は、 ファイルまたは定義仕様書キーワードに対する引数としてコーディングすることもできます。 この組み込み関数のすべての引数が 16 進数リテラルである場合、コンパイラーは 16 進数リテラルの定数結合された結果を作成します。

%BITAND の使用についての実例は、619 ページの図 203、619 ページの図 204、および 620 ページの図 205 を参照してください。

詳しくは、562 ページの『ビット操作』または 551 ページの『組み込み関数』を参照してください。

%BITNOT (ビットの反転)

```
%BITNOT(expr)
```

%BITNOT は、引数のビットの反転をビット単位で戻します。 つまり、引数内の対応するビットがオフの場合は、結果のビットはオンになり、それ以外の場 合は、結果のビットはオフになります。

この組み込み関数に対する引数は、文字でも数値でもかまいません。 数値の引数の場合は、引数が整数または符号なし数値ではないときには、引数はまず整数に変換されます。 値が 8 バイトの整数に収まらない場合は、 数値のオーバーフロー例外が発行されます。

%BITNOT は引数を 1 つのみ使用します。 結果のタイプは、引数のタイプと同じです。 数値の引数の場合は、すべての引数 が符号なし数値であるときには、結果の数値は符号なし数値になり、それ以外の場合は、結果の数値は整数になります。

長さは、最大オペランドの長さになります。 引数の長さがさまざまである場合、 数値の引数については、左側にゼロのビットが埋め込まれます。

%BITNOT は任意の式でコーディングすることができます。 また、すべての引数がコンパイル時に既知の場合は、 ファイルまたは定義仕様書キーワードに対する引数としてコーディングすることもできます。 この組み込み関数のすべての引数が 16 進数リテラルである場合、コンパイラーは 16 進数リテラルの定数結合された結果を作成します。

%BITNOT の使用についての実例は、619 ページの図 203 を参照してください。

詳しくは、562 ページの『ビット操作』または 551 ページの『組み込み関数』を参照してください。

%BITOR (ビット単位の OR 演算)

```
%BITOR(expr:expr{:expr...})
```

%BITOR は、すべての引数のビットの論理和をビット単位で戻します。 つまり、引数内の対応するビットのいずれかがオンの場合は、結果のビットはオンになり 、それ以外の場合は、結果のビットはオフになります。

この組み込み関数に対する引数は、文字でも数値でもかまいません。 数値の引数の場合は、引数が整数または符号なし数値ではないときには、引数はまず整数に変換されます。 値が 8 バイトの整数に収まらない場合は、 数値のオーバーフロー例外が発行されます。

%BITOR は複数の引数を持つことができます。 引数はすべて、同じタイプ (文字または数値) である必要があります。 しかし、 キーワード・パラメーターとしてコーディングされる場合、これら 2 つの BIF が持つ

ことのできる引数は 2 つのみとなります。 結果のタイプは、引数のタイプと同じです。 数値の引数の場合は、すべての引数 が符号なし数値であるときには、結果の数値は符号なし数値になり、それ以外の場合は、結果の数値は整数になります。

長さは、最大オペランドの長さになります。 引数の長さがさまざまである場合、 数値の引数については、左側にゼロのビットが埋め込まれます。 文字の 引数については、短い場合、右側にゼロのビットが埋め込まれます。

%BITOR は任意の式でコーディングすることができます。 また、すべての引数がコンパイル時に既知の場合は、 ファイルまたは定義仕様書キーワードに対する引数としてコーディングすることもできます。 この組み込み関数のすべての引数が 16 進数リテラルである場合、コンパイラーは 16 進数リテラルの定数結合された結果を作成します。

%BITOR の使用についての実例は、619 ページの図 203 を参照してください。

詳しくは、562 ページの『ビット操作』または 551 ページの『組み込み関数』を参照してください。

%BITXOR (ビット単位の排他 OR 演算)

```
%BITXOR(expr:expr)
```

%BITXOR は、2 つの引数のビットの排他論理和をビット単位で戻します。 つまり、引数内の対応するビットの 1 つのみがオンの場合は、結果のビットはオンになり、それ以外の場合は、結果のビ ットはオフになります。

この組み込み関数に対する引数は、文字でも数値でもかまいません。 数値の引数の場合は、引数が整数または符号なし数値ではないときには、引数はまず整数に変換されます。 値が 8 バイトの整数に収まらない場合は、 数値のオーバーフロー例外が発行されます。

%BITXOR は引数を厳密に 2 つ使用します。 結果のタイプは、引数のタイプと同じです。 数値の引数の場合は、すべての引数 が符号なし数値であるときには、結果の数値は符号なし数値になり、それ以外の場合は、結果の数値は整数になります。

長さは、最大オペランドの長さになります。 引数の長さがさまざまである場合、 数値の引数については、左側にゼロのビットが埋め込まれます。 文字の 引数については、短い場合、右側にゼロのビットが埋め込まれます。

%BITXOR は任意の式でコーディングすることができます。 また、すべての引数がコンパイル時に既知の場合は、 ファイルまたは定義仕様書キーワードに対する引数としてコーディングすることもできます。 この組み込み関数のすべての引数が 16 進数リテラルである場合、コンパイラーは 16 進数リテラルの定数結合された結果を作成します。

詳しくは、562 ページの『ビット操作』または 551 ページの『組み込み関数』を参照してください。

ビット演算の例

```
D const           c                   x'0007'

D ch1             s              4a   inz(%BITNOT(const))
 * ch1 is initialized to x'FFF84040'
D num1            s              5i 0 inz(%BITXOR(const:x'000F'))
 * num is initialized to x'0008', or 8

D char2a          s              2a
D char2b          s              2a
D uA              s              5u 0
D uB              s              3u 0
D uC              s              5u 0
D uD              s              5u 0

C                   eval      char2a = x'FE51'
C                   eval      char2b = %BITAND(char10a : x'0F0F')
 * operand1   = b'1111 1110 0101 0001'
 * operand2   = b'0000 1111 0000 1111'
 * bitwise AND:   0000 1110 0000 0001
 * char2b = x'0E01'
C                   eval      uA = x'0123'
C                   eval      uB = x'AB'
C                   eval      uc = x'8816'
C                   eval      uD = %BITOR(uA : uB : uC)
 * operand1   = b'0000 0001 0010 0011'
 * operand2   = b'0000 0000 1010 1011'   (fill with x'00')
 * operand3   = b'1000 1000 0001 0110'
 * bitwise OR:    1000 1001 1011 1111
 * uD = x'89BF'
```

図 *203.* ビット命令の使用

```
 * This example shows how to duplicate the function of TESTB using %BITAND
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D fld1            s              1a
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq
C                   testb     x'F1'         fld1                010203
    * Testing bits     1111 0001
    * If FLD1 = x'00' (0000 0000), the indicators have the values '1' '0' '0'
    *   (all tested bits are off)
    * If FLD1 = x'15' (0001 0101), the indicators have the values '0' '1' '0'
    *   (some tested bits are off and some are on)
    * If FLD1 = x'F1' (1111 0001), the indicators have the values '0' '0' '1'
    *   (all tested bits are on)
 /free
       // this code performs the equivalent of the TESTB operation above

       // test if all the "1" bits in x'F1' are off in FLD1
       *in01 = %bitand(fld1 : x'F1') = x'00';

       // test if some of the "1" bits in x'F1' are on
       // and some are off in FLD1
       *in02 = %bitand(fld1 : x'F1') <> x'00'
          and  %bitand(fld1 : x'F1') <> x'F1';

       // test if all the "1" bits in x'F1' are on in FLD1
       *in03 = %bitand(fld1 : x'F1') = x'F1';
 /end-free
```

図 *204. TESTB* 機能を *%BITAND* から派生

```
 * This example shows how to duplicate the function of
 * BITON and BITOFF using %BITAND, %BITNOT, and %BITOR
   DName++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++
   D fld1            s              1a   inz(x'01')
   D fld2            s              1a   inz(x'FF')
   CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq
   C                   biton     x'F4'         fld1
    * fld1 has an initial value of x'01' (0000 0001)
    * The 1 bits in x'F4' (1111 0100) are set on
    * fld1 has a final value of x'F5' (1111 0101)
   C                   bitoff    x'F1'         fld2
    * fld2 has an initial value of x'FF' (1111 1111)
    * The 1 bits in x'F1' (1111 0001) are set off
    * fld2 has a final value of x'0E' (0000 1110)
    /free
           // this code performs the equivalent of the
           // BITON and BITOFF operations above
           // Set on the "1" bits of x'F4' in FLD1
           fld1 = %bitor(fld1 : x'F4');
           // Set off the "1" bits of x'F1' in FLD2
           fld2 = %bitand(fld2 : %bitnot(x'F1'));

           /end-free
```

図 *205.* 組み込み関数を使用した *BITON/BITOFF* 機能

```
D c1              s              2a   inz(x'ABCD')
D c2hh            s              2a   inz(x'EF12')
D c2hl            s              2a   inz(x'EF12')
D c2lh            s              2a   inz(x'EF12')
D c2ll            s              2a   inz(x'EF12')
 /free
        // mhhzo     c1            c2hh
        // c2hh becomes x'AF12'
        %subst(c2hh:1:1)
               = %bitor(%bitand(x'0F'
                                : %subst(c2hh:1:1))
                        : %bitand(x'F0'
                                : %subst(c1:1:1)));
        // c2hl becomes x'EFA2'
        // mhlzo     c1            c2hl
        %subst(c2hl:%len(c2hl):1)
               = %bitor(%bitand(x'0F'
                                : %subst(c2hl:%len(c2hl):1))
                        : %bitand(x'F0'
                                : %subst(c1:1:1)));
        // mlhzo     c1            c2lh
        // c2lh becomes x'CF12'
        %subst(c2lh:1:1)
               = %bitor(%bitand(x'0F'
                                : %subst(c2lh:1:1))
                        : %bitand(x'F0'
                                : %subst(c1:%len(c1):1)));
        // mhllo     c1            c2ll
        // c2ll becomes x'EFC2'
        %subst(c2ll:%len(c2hl):1)
               = %bitor(%bitand(x'0F'
                                : %subst(c2ll:%len(c2ll):1))
                        : %bitand(x'F0'
                                : %subst(c1:%len(c1):1)));
```

図 *206. %BITOR* および *%BITAND* から *MxxZO* 機能を派生させる

%CHAR (文字データへの変換)

```
%CHAR(expression{: format | ccsid})
```

%CHAR は式の値を文字、図形、UCS-2、数値、日付、時刻、またはタイム・スタンプの各データから文字タイプに変換します。

%CHAR(日付|時刻|タイム・スタンプ {: 形式})

%CHAR は、日付、時刻、またはタイム・スタンプ式の値を文字に変換できます。

最初のパラメーターが定数の場合、変換はコンパイル時に行われます。

2 番目のパラメーターには、戻された文字データが変換される目標の日付、時刻、またはタイム・スタンプの形式が入ります。 戻り値には、指定された形式の後にゼロが続いていない限り、区切り記号を含みます。

戻り値の CCSID は *JOBRUN です。

詳しくは、569 ページの『変換命令』または 551 ページの『組み込み関数』を参照してください。

日付、時刻、タイム・スタンプがパラメーターに指定された %CHAR の例

```
DCL-S date DATE INZ(D'1997-02-03');
DCL-S time TIME INZ(T'12.23.34');
DCL-S timestamp6 TIMESTAMP INZ(Z'1997-02-03-12.45.59.123456');
DCL-S timestamp0 TIMESTAMP(0) INZ(Z'1997-02-03-12.45.59');
DCL-S timestamp1 TIMESTAMP(1) INZ(Z'1997-02-03-12.45.59.1');
DCL-S result VARCHAR(100);
```

次の例では、時刻および日付をデフォルト形式でフォーマットします。 デフォルト形式は両方とも *USA であると想定しています。

```
result = 'It is ' + %CHAR(time) + ' on ' + %CHAR(date);
// result = 'It is 12:23 PM on 02/03/1997'
```

次の例では、時刻および日付をジョブ形式でフォーマットします。 ジョブ日付形式は *MDY、ジョブ時刻区切り記号はピリオドであると想定しています。

```
result = 'It is ' + %CHAR(time : *jobrun)
       + ' on ' + %CHAR(date : *jobrun);
// result = 'It is 12.23.34 on 97-02-03'
```

次の例では、時刻および日付を特定の形式でフォーマットします。

```
result = 'It is ' + %CHAR(time : *hms:)
       + ' on ' + %CHAR(date : *iso);
// result = 'It is 12:23:34 on 1997-02-03'
```

次の例では、タイム・スタンプを区切り記号を使用してフォーマットします。

```
result = %CHAR(timestamp0);
// result = '1997-02-03-12.45.59'
result = %CHAR(timestamp1);
// result = '1997-02-03-12.45.59.1'
result = %CHAR(timestamp6);
// result = '1997-02-03-12.45.59.123456'
```

次の例では、区切り記号を指定しないことによって、区切り記号なしでタイム・スタンプをフォーマットします。

```
result = %CHAR(timestamp0 : *iso0);
// result = '19970203124559'
result = %CHAR(timestamp1 : *iso0);
// result = '199702031245591'
result = %CHAR(timestamp6 : *iso0);
// result = '19970203124559123456'
```

結果の一部のみが必要な場合は、%CHAR 結果と一緒に %SUBST を使用できます。

```
result = 'The time is now ' + %SUBST (%CHAR(time):1:5) + '.';
// result = 'The time is now 12:23.'
```

%CHAR(数値)

%CHAR は、数値式の値を文字型に変換できます。

式の値が浮動であれば、結果は浮動形式になります (たとえば、「+1.125000000000000E+020」)。 そうでない場合の結果は、値が負であれば先行負符号がついた 10 進数形式になります。 先行ゼロはありません。 小数点として使用される文字は、制御仕様書の DECEDIT キーワードで指定されている文字になります (デフォルトは「.」)。 たとえば、パック (7,3) の %CHAR 式は、値 「-1.234」を戻すことになります。

戻り値の CCSID は *JOBRUN です。

詳しくは、569 ページの『変換命令』または 551 ページの『組み込み関数』を参照してください。

数値がパラメーターに指定された %CHAR の例

```
D result          S            100A   VARYING
D points          S             10I 0 INZ(234)
D total           S              7P 2 INZ(523.45)
D adjust          S              7P 2 INZ(-123.45)
D variance        S              8F   INZ(.01234)
```

次の例では、整数値を文字形式に変換します。

```
result = 'You have ' + %char(points) + ' points.';
// result = 'You have 234 points.'
```

次の例では、浮動小数点値を文字形式に変換します。

```
result = 'The variance is ' + %char(variance);
// result = 'The variance is +1,234000000000000E-002'
```

次の例では、パック値を文字形式に変換します。

```
result = 'Total is ' + %char(total) + ' and '
       + 'Adjust is ' + %char(adjust);
// result = 'Total is 523.45 and Adjust is -123.45'
```

次の例では、%CHAR によって使用される小数点文字を制御仕様 DECEDIT 値で変更する方法を示します。DECEDIT(*JOBRUN) が制御ステートメントに指定されていると 想定します。したがって、ジョブからの DECFMT 値 が小数点値を決定します。 ジョブ DECFMT 値が「J」であると想定します。

```
result = 'Total is ' + %char(total) + ' and '
       + 'Adjust is ' + %char(adjust);
// result = 'Total is 523,45 and Adjust is -123,45'
```

%CHAR(文字 | グラフィック | UCS2 {: ccsid})

%CHAR は、文字、グラフィック、 または UCS-2 式の値を、特定の文字 CCSID の文字型に変換できます。CCSID オペランドは、%CHAR 組み込み関数の結果の CCSID を指定します。

CCSID オペランドが指定された場合は、*HEX、*JOBRUN、*JOBRUNMIX、*UTF8、EBCDIC または ASCII CCSID を表す数値、値 65535 (*HEX と同じ)、または値 1208 (*UTF8 と同じ) となることがあります。

CCSID オペランドが指定されていない場合、 制御キーワード CCSID(*CHAR) で指定されるように、モジュールのデフォルト文字 CCSID に デフォルト設定されます。

文字オペランドの CCSID が *HEX の場合、変換は行われません。 オペランドのデータは、CCSID オペランドで指定された CCSID であると想定されます。 CCSID *HEX の文字データについては、250 ページの『文字形式』を参照してください。

データから文字 CCSID に変換するときにデータを正常に変換できない場合があることについては、261 ページの『変換』を参照してください。

グラフィック・データで、%CHAR 戻り値の CCSID が EBCDIC CCSID の場合、 戻り値にはシフトイン文字およびシフトアウト文字が 含まれます。 たとえば、5 文字の図形フィールドが変換される場合、戻り値は 12 文字 (10 バイトの図形データと 2 つのシフト文字) になります。 式の値が可変長である場合、戻り値は可変長形式になります。

詳しくは、569 ページの『変換命令』または 551 ページの『組み込み関数』を参照してください。

文字、グラフィック、および UCS-2 オペランドが指定された %CHAR の例

注 : この例の GRAPHIC リテラルは、無効な GRAPHIC リテラルです。 詳しくは、252 ページの『グラフィック形式』 を参照してください。

```
D graphicName     S             20G   VARYING INZ(G'oXXYYZZi')
D greeting        S             20A   VARYING CCSID(*UTF8) INZ('Hello')
D message         S             30C   INZ('Successful operation')
D result          S            100A   VARYING
```

次の例では、グラフィック値を文字に変換します。 CCSID パラメーターは指定されていないため、%CHAR 組み込み関数の結果の CCSID はモジュールのデフォルト文字 CCSID です。

```
result = 'The customer''s name is ' + %CHAR(graphicName) + '.';
// result = 'The customer's name is oXXYYZZi.'
```

次の例では、UTF-8 値をジョブ CCSID の文字に変換します。

```
result = %CHAR(greeting : *JOBRUN);
// result = 'Hello'
```

次の例では、UCS-2 値をジョブ CCSID の文字に変換します。

```
result = %CHAR(message : *JOBRUN);
// result = 'Successful operation'
```

%CHECK (文字の検査)

```
%CHECK(comparator : base {: start})
```

%CHECK は、ストリング・コンパレーター の中に現れない文字を含む ストリング 基底 の、最初の位置を戻します。 基底 のすべての文字が コンパレーター にも現れていれば、関数は 0 を戻します。

検査は開始位置から始められ、コンパレーター・ストリングに含まれていない文字が見付かるまで、右方へと続けられます。 開始桁の省略時値の 1 が使用されます。

1 番目のパラメーターのタイプは、文字、図形、UCS-2、固定または可変長でなければなりません。 2 番目のパラメーターは、1 番目のパラメーターと同じタイプである必要があります。 3 番目のパラメーターが指定されている場合、それは、小数点以下の桁数がゼロである非浮動数値でなければなりません。

詳しくは、589 ページの『ストリング命令』または 551 ページの『組み込み関数』を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
 *----------------------------------------------------
 * A string contains a series of numbers separated
 * by blanks and/or commas.
 * Use %CHECK to extract the numbers
 *----------------------------------------------------
D string          s             50a   varying
D                                     inz('12, 233 17, 1, 234')
D delimiters      C                   ' ,'
D digits          C                   '0123456789'
D num             S             50a   varying
D pos             S             10i 0
D len             S             10i 0
D token           s             50a   varying

 /free

     // make sure the string ends with a delimiter
     string = string + delimiters;

     dou string = '';

       // Find the beginning of the group of digits
       pos = %check (delimiters : string);
       if (pos = 0);
          leave;
       endif;

       // skip past the delimiters
       string = %subst(string : pos);

       // Find the length of the group of digits
       len = %check (digits : string) - 1;

       // Extract the group of digits
       token = %subst(string : 1 : len);
       dsply ' ' ' ' token;

       // Skip past the digits
       if (len < %len(string));
         string = %subst (string : len + 1);
       endif;

     enddo;

 /end-free
```

図 *207. %CHECK* の例

626 ページの図 209 も参照してください。

%CHECKR (逆向きの検査)

```
%CHECKR(comparator : base {: start})
```

%CHECKR は、ストリング・コンパレーター の中に現れない文字を含むストリング 基底 の、末尾の位置を戻します。 基底 のすべての文字が コンパレーター にも現れていれば、関数は 0 を戻します。

検査は開始位置から始められ、コンパレーター・ストリングに含まれていない文字が見付かるまで、左方へと続けられます。 開始位置のデフォルトの値は、ストリングの末尾です。

1 番目のパラメーターのタイプは、文字、図形、UCS-2、固定または可変長でなければなりません。 2 番目のパラメーターは、1 番目のパラメーターと同じタイプである必要があります。 3 番目のパラメーターが指定されている場合、それは、小数点以下の桁数がゼロである非浮動数値でなければなりません。

詳しくは、589 ページの『ストリング命令』または 551 ページの『組み込み関数』を参照してください。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
*-------------------------------------------
* If a string is padded at the end with some
* character other than blanks, the characters
* cannot be removed using %TRIM.
* %CHECKR can be used for this by searching
* for the last character in the string that
* is not in the list of "pad characters".
*-------------------------------------------
D string1         s             50a   varying
D                                     inz('My *dog* Spot.* @ * @ *')
D string2         s             50a   varying
D                                     inz('someone@somewhere.com')
D padChars        C                   ' *@'

 /free

    %len(string1) = %checkr(padChars:string1);
    //  %len(string1) is set to 14 (the position of the last character
    //  that is not in "padChars").

    //  string1 = 'My *dog* Spot.'

    %len(string2) = %checkr(padChars:string2);
    //  %len(string2) is set to 21 (the position of the last character
    //  that is not in "padChars").

    //  string2 = 'someone@somewhere.com'  (the string is not changed)

 /end-free
```

図 *208. %CHECKR* の例

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
*-----------------------------------------------------
* A string contains a numeric value, but it might
* be surrounded by blanks and asterisks and might be
* preceded by a currency symbol.
*-----------------------------------------------------
D string          s             50a   varying inz('$****12.345***  ')

 /free
    // Find the position of the first character that is not one of ' $*'
    numStart = %CHECK (' $*' : string);
    //  = 6

    // Find the position of the last character that is not one of ' *'
    numEnd = %CHECKR (' *' : string);
    //  = 11

    // Extract the numeric string
    string = %SUBST(string : numStart : numEnd - numStart + 1);
    //  = '12.345'

 /end-free
```

図 *209. %CHECK* と *%CHECKR* の例

%DATA (文書 {:オプション})

%DATA は、DATA-INTO 命令コードおよび DATA-GEN 命令コードの第 2 オペランドとして使用されます。 %DATA は値を戻しません。また、DATA-INTO 命令コードおよび DATA-GEN 命令コード以外の場所に指定できません。

%DATA の第 1 オペランドは、文書を指定します。 これには、定数または可変文字、あるいは UCS-2 式を指定できます。

%DATA の第 2 オペランドは、命令を制御するオプションを指定します。 このオペランドは、定数または変数文字式の場合があります。 627 ページの『%DATA のオプションの指定』を参照してください。

DATA-INTO 命令の場合

- %DATA は解析する文書を指定し、文書からの情報をターゲット RPG 変数に入れる方法を制御するためのオプションを指定します。
- 第 1 オペランドには、定数または可変文字、あるいは文書または文書を含むファイル名のいずれかを持つ UCS-2 式を指定できます。
- %DATA の第 2 オペランドは、文書の構文解釈方法、および RPG 変数に文書データをどのように入れるかを制御するオプションを指定します。

  有効なオプションおよび値の完全なリストについては、747 ページの『DATA-INTO 命令コードの %DATA オプション』を参照してください。

DATA-GEN 命令の場合

- %DATA は、配置される文書の位置と、DATA-GEN の第 1 オペランドの RPG 変数から文書を生成する方法を制御するオプションを指定します。
- 第 1 オペランドは、定数または可変文字、あるいは UCS-2 式を指定できます。 「doc=file」オプションで指定する名前が、生成される文書を受け取るファイルの名前になります。 「doc=file」オプションが指定されていない場合は、生成される文書が変数に入れられます。
- %DATA の第 2 オペランドは、文書の生成方法を制御するオプション、および RPG 変数からのデータを生成プログラムに渡す方法を制御するオプションを指定します。

  有効なオプションおよび値の完全なリストについては、743 ページの『DATA-GEN 命令コードの %DATA オプション』を参照してください。

%DATA のオプションの指定

文字式の値は、次の形式で指定されたゼロ個以上のオプションのリストです。

```
optionname1=value1 optionname2=value2
```

オプション名および等号の間、または等号および値の間には、スペースは許可されていません。 ただし、オプションの前後、および間には、任意の数のスペースを指定できます。 オプションは、大/小文字で指定できます。 以下は、DATA-INTO の「doc=file」および「allowextra=yes」オプションを指定する有効なすべての方法です。

```
   'doc=file allowextra=yes'
   '        doc=file     allowextra=yes     '
   'ALLOWEXTRA=YES DOC=FILE     '
   'AllowExtra=Yes Doc=File     '
```

以下は**無効な**オプション・ストリングです。

オプション・ストリング	**オプション・ストリングの問題**
'doc = file'	等号の周辺のスペースは許可されていません。
'allowextra'	それぞれのオプションには、等号および値が必要です。
'badopt=yes'	有効なオプションのみ許可されています。
'allowextra=ok'	「allowextra」値は「yes」または「no」のみです。

1 つのオプションが複数回指定されていた場合は、最後に指定されている値が使用されます。 例えば、「options」オペランドの値が次のように指定されるとします。

```
   'doc=file doc=string'
```

この場合、パーサーは「doc」オプションに値「string」を使用します。

パーサーが無効なオプションまたは無効な値を発見した場合は、命令は状況コード 00352 で失敗します。

%DATA の例

「options」オペランドは省略されています。 すべてのオプションにデフォルト値が使用されます。 「doc」オプションのデフォルト値は必ず「string」であるために、パーサーは最初のオペランドに文書のテキストが含まれていると正しく想定します。 以下の例では、形式 「itemName=itemValue」で架空の言語を想定しています。

```
document = 'city=Toronto';
DATA-INTO city %DATA(document) %PARSER(p);
```

「options」オペランドは 2 つのオプション持つリテラルとして指定されています。

```
DATA-INTO myds %DATA(document : 'allowmissing=yes allowextra=yes')
          %PARSER(p);
```

「options」オペランドは 2 つのオプション持つ変数式として指定されています。

```
ccsidOpt = 'ccsid=ucs2';
DATA-INTO %HANDLER(mySaxHandler : myCommArea)
          %DATA('mydoc.txt' : 'doc=file ' + ccsidOpt)
          %PARSER(p);
```

%DATA の例、および DATA-INTO 命令と DATA-GEN 命令について詳しくは、743 ページの『DATA-INTO (文書の変数への構文解析)』および 740 ページの『DATA-GEN (変数からの文書の生成)』を参照してください。

%DATE (日付への変換)

```
%DATE{(expression{:date-format})}
```

%DATE は式の値を文字、数値、またはタイム・スタンプのデータから、日付タイプに変換します。 変換後の値は変更されないままですが、日付として戻されます。

最初のパラメーターは、変換される対象の値です。 値を指定しない場合、%DATE は現行システム日付を戻します。

2 番目のパラメーターは、文字または数値の入力データの日付の形式です。 入力データの形式にかかわらず、出力は *ISO 形式で戻されます。

使用できる入力形式については、273 ページの『日付データ・タイプ』を参照してください。 文字または数値の入力データの日付形式が指定されなかった場合、デフォルト形式は *ISO です。 詳しくは、「330 ページの『DATFMT(形式 {区切り記号 })』」を参照してください。

最初のパラメーターが、タイム・スタンプ、*DATE、または UDATE の場合は、2 番目のパラメーターは指定してはなりません。 システムは、これらの場合の入力データの形式については理解しています。

詳しくは、580 ページの『情報命令』または 551 ページの『組み込み関数』を参照してください。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/FREE

  string = '040596';
  date =  %date(string:*MDY0);
  // date now contains d'1996-04-05'
/END-FREE
```

図 *210. %DATE* の例

%DAYS (日数)

```
%DAYS(number)
```

%DAYS は、数を、日付またはタイム・スタンプ値に加算することができる期間に変換します。

%DAYS は、加算式または減算式の中でプラス符号またはマイナス符号の後にのみ置くことができます。 プラス符号またはマイナス符号の前の値は日付またはタイム・スタンプでなければなりません。 結果は、加算または減算がされた適切な日数を持つ、日付またはタイム・スタンプの値になります。 日付の場合、結果の値は *ISO 形式になります。

日時の算術演算の例は、668 ページの図 239 を参照してください。

詳しくは、572 ページの『日付命令』または 551 ページの『組み込み関数』を参照してください。

%DEC (パック 10 進数への変換)

```
%DEC(numeric or character expression{:precision:decimal places})
%DEC(date time or timestamp expression {:format})
```

%DEC は、最初のパラメーターの値を 10 進 (パック) 形式に変換します。

数値式または文字式

最初のパラメーターが数値式または文字式である場合、 その結果には精度 桁と小数位 桁が含まれます。 精度および小数位は、 数値リテラル、数値リテラルを表す名前付き定数、あるいはコンパイル時に 既知の数値を持つ組み込み関数でなければなりません。

注 : %LEN および %DECPOS は、その値が定数であっても、%DEC または %DECH の 2 番目 および 3 番目のパラメーターとして直接使用することはできません。 変数の長さおよび小数点以下の桁数を使用して %DEC および %DECH を制御する例については、654 ページの図 234 を参照してください。

精度 と小数位 のパラメーターは、 式のタイプが浮動でも文字でもない場合は省略してもかまいません。 これらのパラメーターが省略された場合は、精度および小数位は、数値式の 属性から取得されます。

パラメーターが文字式である場合は、次の規則が適用されます。

- %DEC の文字式の規則については、569 ページの『組み込み関数を使用して文字値を数値に変換するための規則』を参照してください。
- 浮動小数点データ (例えば、'1.2E6') は使用できません。
- 無効な数値データが検出された場合、例外が発生し、状況コード 105 が戻されます。

%DEC の使い方の例については、%DECH を参照してください。

日付、時刻、またはタイム・スタンプ式

最初のパラメーターが日付、時刻、またはタイム・スタンプ式である場合、 オプションの形式パラメーターで、戻り値の形式を指定します。 変換後の 10 進数値の桁数は、その形式の値が取り得る桁数になり、小数点以下の桁数はゼロになります。 例えば、最初のパラメーターが日付で、形式が *YMD である場合、

10 進数値は 6 桁になります。 最初のパラメーターがタイム・スタンプで、 秒の小数部が 3 桁ある場合、10 進値は 17 桁になります。

形式パラメーターを省略すると、最初のパラメーターの形式が使用されます。 330 ページの『DATFMT(形式 {区切り記号 })』および 347 ページの『TIMFMT(形式 {区切り記号 })』を参照してください。

形式 *USA は、時刻式では使用することができません。 最初のパラメーターが *USA という時刻形式の時刻値である場合には、%DEC の 2 番目の形式パラメーター を必ず指定する必要があります。

631 ページの図 212 は %DEC 組み込み関数の例を示しています。

詳しくは、569 ページの『変換命令』または 551 ページの『組み込み関数』を参照してください。

```
D   yyddd         S              5S 0
D   yyyymmdd      S              8P 0
D   hhmmss        S              6P 0
D   numeric       S             20S 0
D   date          S               D   inz(D'2003-06-27') DATFMT(*USA)
D   time          S               T   inz(T'09.25.59')
D   timestamp     S               Z   inz(Z'2003-06-27-09.25.59.123456')
D   timestamp3    S               Z 3 inz(Z'2003-06-27-09.25.59.123')
 /free

     // Using the format of the first parameter

     numeric = %dec(date);              //   numeric = 06272003
     numeric = %dec(time);             //   numeric = 092559
     numeric = %dec(timestamp);        //   numeric = 20030627092559123456
     numeric = %dec(timestamp3);       //   numeric = 20030627092559123

     // Using the second parameter to specify the result format

     yyddd = %dec(date : *jul);         //   yyddd = 03178
     yyyymmdd = %dec(date : *iso);     //   yyyymmdd = 20030627
```

図 *211. %DEC* を使用して日付、時刻、およびタイム・スタンプを数値に変換する

%DECH (四捨五入を伴うパック 10 進数形式への変換)

```
%DECH(numeric or character expression :precision:decimal places )
```

%DECH は %DEC と同じですが、式が 10 進数値または浮動値である場合には、希望の精度に変換する際に式の値に四捨五入が適用されるという点が異なります。 四捨五入が実行できない場合、メッセージは出されません。

%DEC とは異なり、3 つのパラメーターはすべて必要です。

詳しくは、569 ページの『変換命令』または 551 ページの『組み込み関数』を参照してください。

%DECH の例

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++
D p7              s              7p 3 inz (1234.567)
D s9              s              9s 5 inz (73.73442)
D f8              s              8f   inz (123.456789)
D c15a            s             15a   inz (' 123.456789 -')
D c15b            s             15a   inz (' + 9 , 8 7 6 ')
D result1         s             15p 5
D result2         s             15p 5
D result3         s             15p 5

    // using numeric parameters
    result1 = %dec (p7) + 0.011;  // "result1" is now 1234.57800
    result2 = %dec (s9 : 5: 0);   // "result2" is now   73.00000
    result3 = %dech (f8: 5: 2);   // "result3" is now  123.46000
    // using character parameters
    result1 = %dec (c15a: 5: 2);  // "result1" is now -123.45
    result2 = %dech(c15b: 5: 2);  // "result2" is now    9.88000
```

図 *212.* 数値パラメーターおよび文字パラメーターの使用

通貨記号と 3 桁ごとの区切りの処理

千単位の区切り文字 (例えば「1,234,567」) や先行アスタリスクや通貨記号などの非数値文字 (例えば「$***1,234,567.89」) などの非数値文字が文字データに含まれていることが分かっている場合は、これらの文字をデータから削除するために前処理がいくらか必要になる場合があります。

ただし、制御キーワード EXPROPTS(*USEDECEDIT) が指定されている場合、DECEDIT キーワードによって示される千単位の区切り文字は、数値データの一部と見なされます。

以下の例では、%XLATE 組み込み関数を使用して、シンボル、アスタリスク、または 3 桁の区切り文字をブランクで置き換えます。

```
D data            s             20a   inz('$1,234,567.89')
D num             s             21p 9
    num = %dech(%xlate('$*,' : '    ' : data)
              : 21 : 9);
```

以下の例では、制御キーワード EXPROPTS(*USEDECEDIT) が指定されているため、千単位の区切り文字をブランクで置き換える必要はありません。 前の例では、%XLATE の第 1 オペランドである '$*,' にコンマ (,) が含まれていますが、次の例では、%XLATE の第 1 オペランドは単に '$*' となっています。

```
H EXPROPTS(*USEDECEDIT)
D data            s             20a   inz('$1,234,567.89')
D num             s             21p 9
    num = %dech(%xlate('$*' : '    ' : data)
              : 21 : 9);
```

次の例では、通貨記号または千単位の区切り文字が実行時に異なる場合があるので、これらの値を保持するために変数が使用されています。

```
    num = %dech(%xlate(cursym + '*' + thousandsSep : '    ' : data)
              : 21 : 9);
```

%DECPOS (小数部の桁数の取得)

```
%DECPOS(numeric expression)
```

%DECPOS は、数値変数または数値式の小数点以下の桁数を戻します。 戻り値は定数であるので、その値は定数のフォールディングに 入る場合があります。

数値式は、浮動変数または浮動式であってはなりません。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D p7              s              7p 3 inz (8236.567)
D s9              s              9s 5 inz (23.73442)
D result1         s              5i 0
D result2         s              5i 0
D result3         s              5i 0

 /FREE
   result1 = %decpos (p7);     //  "result1" is now  3.
   result2 = %decpos (s9);     //  "result2" is now  5.
   result3 = %decpos (p7 * s9);//  "result3" is now  8.
 /END-FREE
```

図 *213. %DECPOS* の例

%DECPOS と %LEN を一緒に使用する例については、654 ページの図 234 を参照してください。

詳しくは、589 ページの『サイズ変更命令』または 551 ページの『組み込み関数』を参照してください。

%DIFF (2 つの日付、時刻、またはタイム・スタンプ値の差)

```
%DIFF(op1 : op2 : unit {: frac })
```

単位に指定できるのは、 *MSECONDS、*SECONDS、*MINUTES、*HOURS、*DAYS、*MONTHS、または *YEARS です。 単位の省略形を使用して、*MS、*S、*MN、*H、*D、*M、または *Y と指定することもできます。

%DIFF は、2 つの日付、または時刻の値の間の差 (期間) を生成します。 最初のパラメーターと 2 番目のパラメーターは、同じタイプかまたは互換性のあるタイプでなければなりません。 可能な組み合わせは以下のとおりです。

- 日付と日付
- 時刻と時刻
- タイム・スタンプとタイム・スタンプ
- 日付とタイム・スタンプ (タイム・スタンプの日付の部分だけが考慮されます)
- 時刻とタイム・スタンプ (タイム・スタンプの時刻の部分だけが考慮されます)

3 番目のパラメーターには単位を指定します。 次の単位が有効です。

- 2 つの日付または日付とタイム・スタンプの場合: *DAYS、*MONTHS、および *YEARS
- 2 つの時刻または時刻とタイム・スタンプの場合: *SECONDS、*MINUTES、および *HOURS
- 2 つのタイム・スタンプの場合: *MSECONDS、*SECONDS、*MINUTES、 *HOURS、*DAYS、*MONTHS、および *YEARS

*MONTHS と *YEARS の結果は意外なものになる可能性があります。574 ページの『予期しない結果』を参照してください。

3 番目のオペランドが *SECONDS または *S で、 両方のオペランドがタイム・スタンプの場合、戻される秒数の小数点以下の桁数を示す 4 番目のパラメーターを 指定できます。 指定できるのは 0 から 12 までの値です。 これは、戻される秒数の小数部の桁数を表します。

差異は、第 1 オペランドから第 2 オペランドを減算することによって計算されます。

結果は、すべての剰余を廃棄して切り捨てられます。 たとえば、61 分は 1 時間と等しく、59 分は 0 時間に等しくなります。

関数によって戻り値は、数字タイプと期間タイプの両方と互換性があります。 結果は、数 (数値タイプ) あるいは日付、時刻、またはタイム・スタンプ (期間タイプ) に加算することができます。

32 年 9 カ月よりも離れている 2 つのタイム・スタンプの間の差をマイクロ秒単位で得たい場合は、期間の値の限界である 15 桁を超えます。 この結果はエラーになるかまたは切り捨てが行なわれます。 ただし、小数点以下が 6 桁の秒単位で差を求めた後、結果として得た値 に 1000000 を掛けることにより、任意の 2 つの日付の差をマイクロ秒単位で得ることができます。 例えば、秒単位の差が 1041379205.123456 の場合、 マイクロ秒単位の差は 1041379205123456 です。

詳しくは、572 ページの『日付命令』または551 ページの『組み込み関数』を参照してください。

```
D due_date        S               D   INZ(D'2005-06-01')
D today           S               D   INZ(D'2004-09-23')
D num_days        S            15P 0

D start_time      S               Z

D time_taken      S            15P 0

 /FREE

     // Determine the number of days between two dates.

     // If due_date has the value 2005-06-01 and
     // today has the value 2004-09-23, then
     // num_days will have the value 251.

     num_days = %DIFF (due_date: today: *DAYS);

     // If the arguments are coded in the reverse order,
     // num_days will have the value -251.

     num_days = %DIFF (today: due_date: *DAYS);

     // Determine the number of seconds required to do a task:
     //  1. Get the starting timestamp
     //  2. Do the task
     //  3. Calculate the difference between the current
     //     timestamp and the starting timestamp

     start_time = %timestamp();
     process();
     time_taken = %DIFF (%timestamp() : start_time : *SECONDS);

 /END-FREE
```

図 *214. %DIFF* の結果を数値として使用する

```
D estimated_end...
D                 S               D
D prev_start      S               D   INZ(D'2003-06-21')
D prev_end        S               D   INZ(D'2003-06-24')

 /FREE

     // Add the number of days between two dates
     // to a third date

     // prev_start is the date a previous task began
     // prev_end is the date a previous task ended.

     // The following calculation will estimate the
     // date a similar task will end, if it begins
     // today.

     // If the current date, returned by %date(), is
     // 2003-08-15, then estimated_end will be
     // 2003-08-18.

     estimated_end = %date() + %DIFF(prev_end : prev_start : *days);

 /END-FREE
```

図 *215. %DIFF* の結果を期間として使用する

%DIV (商の戻り整数部分)

```
%DIV(n:m)
```

%DIV は、オペランド **n** を **m** で 除算した結果求められる商の整数部分を戻します。 これらの 2 つのオペランドは、小数点以下の桁数がない (ゼロの) 数値である必要があります。 いずれかのオペランドがパック数値、ゾーン数値、または 2 進数値である場合、結果 はパック数値になります。 いずれかのオペランドが整数値である場合、結果は整数になります。 これ以外の場合、結果は符号なし数値に なります。 浮動数値オペランドは使用できません。(679 ページの『%REM (戻り整数剰余)』 も参照してください。)

これらのオペランドが、8 バイト整数フィールド、または符号なしフィールド 内に収められる定数である場合、定数の折り畳みがこの組み込み関数に適用されます。 この場合、%DIV 組み込み関数は定義仕様書でコーディングできます。

詳しくは、557 ページの『算術演算』または 551 ページの『組み込み関数』を参照してください。

この関数は、679 ページの図 250 に示されています。

%EDITC (編集コードを使用する編集値)

```
%EDITC(numeric : editcode {: *ASTFILL | *CURSYM | currency-symbol})
```

この関数は、編集コードに従って編集された数値を表す文字結果を戻します。 一般に、数値および編集コードの規則は、出力仕様で数値を編集する 場合の規則と同一です。 3 番目のパラメーターはオプションですが、これを指定する場合は、次の いずれかにする必要があります。

***ASTFILL**
アスタリスク保護が使用されることを示します。 これは、 戻り値の中の先行ゼロは、アスタリスクによって置き換えられるという ことです。 たとえ ば、%EDITC(-0012.5 : 'K' : *ASTFILL) は '***12.5-' となって戻されます。

***CURSYM**
浮動通貨記号が使用されることを示します。 実際の記号は、CURSYM キーワードの中で、制御仕様書に指定された記号、あるいはデフォルトの '$' です。*CURSYM が指定されると、通貨記号は、結果の中の最初の有効数字の直前に入れられます。 たとえ ば、%EDITC(0012.5 : 'K' : *CURSYM) は ' $12.5 ' となって戻されます。

通貨記号
所定の通貨記号とともに浮動通貨が使用されることを示します。 これは、1 バイト文字定数 (コンパイル時に評価することができる リテラル、名前付き定数あるいは式) である必要があります。 たとえ ば、%EDITC(0012.5 : 'K' : 'X') は ' X12.5 ' と なって戻されます。

%EDITC の結果は常に同じ長さで、先行ブランクと後書きブランクを含む場合が あります。 たとえば、%EDITC(NUM : 'A' : '$') は、NUM のある値の 場合には '$1,234.56CR' を戻し、別の値の場合 には ' $4.56 ' を戻します。

浮動式は、1 番目のパラメーターでは使用できません (%DEC を 使用して浮動形式を編集可能形式に変換することができます)。 2 番目の パラメーターでは、編集コードは、文字定数として 指定されます。サポートされている編集コード は、'A' から 'D'、'J' から 'Q'、'X' から 'Z'、'1' から '9' です。 定数 は、コンパイル時に値を決めることができるリテラル、名前付き定数、 あるいは式の場合があります。

詳しくは、569 ページの『変換命令』または 551 ページの『組み込み関数』を参照してください。

```
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D msg             S             100A
D salary          S               9P 2 INZ(1000)
 * If the value of salary is 1000, then the value of salary * 12
 * is 12000.00. The edited version of salary * 12 using the A edit
 * code with floating currency is ' $12,000.00 '.
 * The value of msg is 'The annual salary is $12,000.00'
CL0N01Factor1+++++++Opcode&ExtExtended-factor2++++++++++++++++++++++++++++
C                   EVAL      msg = 'The annual salary is '
C                             + %trim(%editc(salary * 12
C                                            :'A': *CURSYM))
 * In the next example, the value of msg is 'The annual salary is &12,000.00'
C                   EVAL      msg = 'The annual salary is '
C                             + %trim(%editc(salary * 12
C                                            :'A': '&'))

 * In the next example, the value of msg is 'Salary is $*****12,000.00'
 * Note that the '$' comes from the text, not from the edit code.
C                   EVAL      msg = 'Salary is $'
C                             + %trim(%editc(salary * 12
C                                             :'B': *ASTFILL))

 * In the next example, the value of msg is 'The date is 1/14/1999'
C                   EVAL      msg = 'The date is '
C                             + %trim(%editc(*date : 'Y'))
```

図 *216. %EDITC* の例 *1*

共通の要求は、フィールドを次のように編集することです。

- 先行ゼロは消去される。
- 値が負の場合には、それが括弧で囲まれる。

次の例では、サブプロシージャー内で %EDITC を使用してこれを実行します。

```
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D neg             S              5P 2      inz(-12.3)
D pos             S              5P 2      inz(54.32)
D editparens      PR            50A
D    val                        30P 2      value
D editedVal       S             10A
CL0N01Factor1+++++++Opcode&ExtExtended-factor2++++++++++++++++++++++++++++
C                   EVAL      editedVal = editparens(neg)
 * Now editedVal has the value '(12.30)   '
C                   EVAL      editedVal = editparens(pos)
 * Now editedVal has the value ' 54.32    '
 *---------------------------------------------------------------
 * Subprocedure EDITPARENS
 *---------------------------------------------------------------
P editparens      B
D editparens      PI            50A
D    val                        30P 2      value
D lparen          S              1A        inz(' ')
D rparen          S              1A        inz(' ')
D res             S             50A
 * Use parentheses if the value is negative
C                   IF        val < 0
C                   EVAL      lparen = '('
C                   EVAL      rparen = ')'
C                   ENDIF
 * Return the edited value
 * Note that the '1' edit code does not include a sign so we
 * don't have to calculate the absolute value.
C                   RETURN    lparen             +
C                             %editc(val : '1')  +
C                             rparen
P editparens      E
```

図 *217. %EDITC の例 2*

%EDITFLT (浮動外部表現への変換)

```
%EDITFLT(numeric expression)
```

%EDITFLT は、数値式の値を、浮動の文字外部表示表現に変換します。 結果は 14 文字または 23 文字です。 引数が 4 バイト浮動フィールドである 場合、結果は 14 文字となります。 それ以外の場合、結果は 23 文字です。

定義仕様書キーワードに対するパラメーターとして指定された場合、この パラメーターは、数値リテラル、浮動リテラル、あるいは数値の定数名 もしくは組み込み関数でなければなりません。 式の中で指定された場合、数値式に定数値がある場合には、 定数結合が適用されます。

詳しくは、569 ページの『変換命令』または 551 ページの『組み込み関数』を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D f8              s               8f   inz (50000)
D string          s              40a   varying

 /FREE
    string = 'Float value is ' + %editflt (f8 - 4E4) + '.';
    // Value of "string" is 'Float value is +1.000000000000000E+004. '
 /END-FREE
```

図 *218. %EDITFLT の例*

%EDITW (編集語を使用する編集値)

```
%EDITW(numeric : editword)
```

この関数は、編集語に従って編集された数値を表す 文字結果を戻します。 数値および編集語の規則は、出力仕様で数値を編集する場合の規則と同一 です。

浮動式は 1 番目のパラメーターでは使用できません。 %DEC を使用して、浮動形式を編集可能形式に変換します。

編集語は文字定数でなければなりません。

詳しくは、569 ページの『変換命令』または 551 ページの『組み込み関数』を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D amount          S             30A
D salary          S              9P 2
D editwd          C                    '$ ,   ,  **Dollars&  &Cents'

 * If the value of salary is 2451.53, then the edited version of
 * (salary * 12) is '$***29,418*Dollars 36 Cents'. The value of
 * amount is 'The annual salary is $***29,418*Dollars 36 Cents'.

 /FREE
   amount = 'The annual salary is '
            + %editw(salary * 12 : editwd);
 /END-FREE
```

図 *219. %EDITW* の例

%ELEM (要素数の検索)

```
%ELEM(table_name)
%ELEM(array_name)
%ELEM(multiple_occurrence_data_structure_name)
%ELEM(array_name:*ALLOC)
%ELEM(array_name:*KEEP)
%ELEM(array_name:*MAX)
```

%ELEM は、指定された配列、テーブル、または複数オカレンス・データ構造の中の要素の 数を戻します。戻り値は符号なし整数形式 (タイプ U) です。 これは、数値定数が使用できる定義仕様書または拡張演算項目 2 フィールドの 式のどこにでも指定することができます。

パラメーターは、配列、テーブル、または複数オカレンス・データ構造の名前としなけれ ばなりません。

配列に変数ディメンションが含まれている場合 (配列が DIM(*AUTO) または DIM(*VAR) で定義済み)、%ELEM をいくつか追加の方法で使用することができます。

- %ELEM を割り当てステートメントのターゲットとして使用して、可変次元配列の現在の要素数を変更できます。
- %ELEM がターゲットの値に使用される場合、%ELEM には 2 番目のパラメーターを指定できます。

  ***ALLOC**
  配列に割り振られる要素数が返されます。

  ***MAX**
  配列の最大要素数が返されます。
- %ELEM が割り当てステートメントのターゲットである場合、%ELEM に 2 番目のパラメーターを指定できます。

  ***ALLOC**
  割り当てステートメントの右側の値が配列の現在の要素数より大きい場合、配列に割り振られる要素数が増えます。 値が現在の要素数より小さい場合、要素数は変更されません。 配列に割り振られる要素数は、指定された値より大きくなる可能性があります。 現在の要素数は変更されません。

  ***KEEP**
  配列の新規要素の値が変更されません。

422 ページの『可変次元配列』を参照してください。

パラメーターが複合修飾名であり、必要なサブフィールドを含むデータ構造が配列の場合、パラメーターは以下の 2 つの方法のいずれかで指定できます。

- 複合修飾名でデータ構造配列すべてに指標を指定。
- 複合修飾名でどのデータ構造配列にも指標を指定しない。

638 ページの『複合データ構造を使用した %ELEM の例』を参照してください。

詳しくは、561 ページの『配列命令』または 551 ページの『組み込み関数』を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D arr1d           S             20    DIM(10)
D table           S             10    DIM(20) ctdata
D mds             DS            20    occurs(30)
D num             S              5p 0

 * like_array will be defined with a dimension of 10.
 * array_dims will be defined with a value of 10.
D like_array      S                   like(arr1d) dim(%elem(arr1d))
D array_dims      C                   const (%elem (arr1d))

 /FREE
    num = %elem (arr1d);  // num is now 10
    num = %elem (table);  // num is now 20
    num = %elem (mds);    // num is now 30
 /END-FREE
```

図 *220. %ELEM* の例

複合データ構造を使用した %ELEM の例

以下の例では、複合修飾サブフィールド *PET* を参照するための標準的な方法が次のようになっています。

```
family(x).child(y).pet
```

%ELEM のパラメーターとして指定する場合、ステートメント 1 のようにデータ構造配列にどの指標も指定しない場合、1 または、ステートメント 2 のようにデータ構造配列にすべての指標を指定した場合のいずれかで指定できます。2.

```
   DCL-DS family QUALIFIED DIM(3);
      name VARCHAR(25);
      numChildren INT(10);
      DCL-DS child DIM(10);
         name VARCHAR(25);
         numPets INT(10);
         pet VARCHAR(100) DIM(3);
      END-DS;
   END-DS;
   DCL-S x INT(10);

   x = %ELEM(family.child.pet);        // 1

   x = %ELEM(family(1).child(1).pet);  // 2
```

%EOF (ファイルの終わりまたは先頭条件の戻し)

```
%EOF{(file_name)}
```

%EOF は、サブファイルに対して実行された最後の読み取り操作または書き出しが、 ファイルの終わり条件またはファイルの先頭条件で終了した場合に '1' を戻します。 他の場合には '0' を戻します。

%EOF を設定する命令を以下に示します。

- 851 ページの『READ (レコードの読み取り)』
- 852 ページの『READC (次の変更レコードの読み取り)』
- 853 ページの『READE (等しいキーのレコードの読み取り)』
- 856 ページの『READP (前のレコードの読み取り)』
- 857 ページの『READPE (等しいキーの前のレコードの読み取り)』
- 905 ページの『WRITE (新しいレコードの作成)』 (サブファイルのみ)。

次の命令は、成功した場合は %EOF(ファイル名) をオフに設定します。 命令が成功しなかった場合、%EOF(ファイル名) は変更されません。パラメーターなしの %EOF が、これらの命令によって変更されることはありません。

- 728 ページの『CHAIN (ファイルからのランダム検索)』
- 842 ページの『OPEN (処理のためのファイルのオープン)』
- 873 ページの『SETGT (より大きい設定)』
- 876 ページの『SETLL (下限の設定)』

全手順ファイルを指定した場合、指定のファイルに対する上記のリストに示した 直前の命令の結果が、ファイルの終わり条件またはファイルの先頭条件であれば、この関数は '1' を戻します。 1 次ファイルおよび 2 次ファイル の場合、%EOF は、ファイル名を指定した場合のみ使用することができます。 この関数は、*GETIN 処理中に実行された最後の入力操作の結果が、ファイルの終わり条件 またはファイルの先頭条件である場合には、'1' に設定されます。 他の場合には、'0' を戻します。

この関数は、入力ファイル、更新ファイル、レコード・アドレス・ファイルに使用 することができ、表示装置ファイルの場合にはサブファイル・レコードへの WRITE に使用することができます。

詳しくは、576 ページの『ファイル操作』または 551 ページの『組み込み関数』を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
F*Filename+IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++
 * File INFILE has record format INREC
FINFILE    IF   E             DISK

 /FREE
    READ INREC;  // read a record
    IF  %EOF;
                 // handle end of file
    ENDIF;
 /END-FREE
```

図 *221.* ファイル名パラメーターを指定しない *%EOF*

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
 * This program is comparing two files

F*Filename+IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++
FFILE1     IF   E             DISK
FFILE2     IF   E             DISK

 * Loop until either FILE1 or FILE2 has reached end-of-file
 /FREE
    DOU %EOF(FILE1) OR %EOF(FILE2);
       // Read a record from each file and compare the records

       READ REC1;
       READ REC2;
       IF %EOF(FILE1) AND %EOF(FILE2);
          // Both files have reached end-of-file
          EXSR EndCompare;

       ELSEIF %EOF(FILE1);
          // FILE1 is shorter than FILE2
          EXSR F1Short;

       ELSEIF %EOF(FILE2);
          // FILE2 is shorter than FILE1
          EXSR F2Short;

       ELSE;
          // Both files still have records to be compared
          EXSR CompareRecs;
       ENDIF;
    ENDDO;
  // ...
 /END-FREE
```

図 *222.* ファイル名パラメーターを指定した *%EOF*

%EQUAL (完全な一致条件の戻し)

```
%EQUAL{(file_name)}
```

%EQUAL は、最後に実行された関係のある命令が正確な一致を見付けた場合には '1' を 戻します。他の場合には '0' を戻します。

%EQUAL を設定する命令を以下に示します。

- 876 ページの『SETLL (下限の設定)』
- 798 ページの『LOOKUP (テーブルまたは配列要素の検索)』

任意指定のファイル名パラメーターを指定しないで %EQUAL を使用した場合、この 関数は、最後に実行された関係のある命令について設定された値を戻します。

SETLL 命令の場合、この関数は、そのキーまたは相対レコード番号が検索引数に 等しいレコードが存在する場合に '1' を戻します。

EQ 標識を指定した LOOKUP 命令の場合、この関数は、検索引数に正確に一致する 要素が見付かった場合に '1' を戻します。

ファイル名を指定した場合、この関数は、指定したファイルに対して最後に実行 された SETLL 命令に適用されます。 この関数は、SETLL 命令コードを使用できる ファイルに対してのみ使用することができます。

詳しくは、800 ページの図 338 および 879 ページの図 384 を参照してください。

詳しくは、576 ページの『ファイル操作』、589 ページの『結果命令』、または 551 ページの『組み込み関数』 を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
F*Filename+IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++
 * File CUSTS has record format CUSTREC
FCUSTSIF   E           K DISK

 /FREE
    // Check if the file contains a record with a key matching Cust
    setll Cust CustRec;
    if %equal;
    // an exact match was found in the file
    endif;
 /END-FREE
```

図 *223. SETLL* を使用した場合の *%EQUAL* の例

```
     DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
     D TabNames        S             10A   DIM(5) CTDATA ASCEND
     D SearchName      S             10A
      * Position the table at or near SearchName
      * Here are the results of this program for different values
      * of SearchName:
      *   SearchName   |   DSPLY
      *   -------------+------------------------------
      *   'Catherine ' |   'Next greater     Martha'
      *   'Andrea    ' |   'Exact            Andrea'
      *   'Thomas    ' |   'Not found        Thomas'
     CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
     C     SearchName    LOOKUP    TabNames                        10  10
     C                   SELECT
     C                   WHEN      %EQUAL
      * An exact match was found
     C     'Exact       'DSPLY                   TabNames
     C                   WHEN      %FOUND
      * A name was found greater than SearchName
     C     'Next greater'DSPLY                   TabNames
     C                   OTHER
      * Not found.  SearchName is greater than all the names in the table
     C     'Not found   'DSPLY                   SearchName
     C                   ENDSL
     C                   RETURN
**CTDATA TabNames
Alexander
Andrea
Bohdan
Martha
Samuel
```

図 *224. LOOKUP* を使用した場合の *%EQUAL* および *%FOUND* の例

%ERROR (エラー条件の戻し)

%ERROR は、最後に実行された、拡張 'E' が指定された命令の結果がエラー条件で ある場合に '1' を戻します。 これは、命令に関してエラー標識が設定された場合 と同じになります。 拡張 'E' が指定された命令が開始される前に、%ERROR は '0' を 戻すように設定され、エラーが発生しない場合には、命令後に変更されないまま 残ります。 エラー標識を指定できるすべての命令は、%ERROR 組み込み関数も設定することができます。 CALLP 命令も %ERROR を設定することができます。

%ERROR 組み込み関数の例については、692 ページの図 256 および 693 ページの図 257 を参照してください。

詳しくは、589 ページの『結果命令』 または 551 ページの『組み込み関数』 を参照してください。

%FIELDS

```
%FIELDS(name{:name...})
```

- フィールドのリストは、自由形式グループでコーディングされる入出力操作 UPDATE に対する最後の引数として指定することができます。 指定されたフィールドのみが更新され、入出力バッファーに入れられます。 642 ページの『%FIELDS (更新するフィールド)』を参照してください。
- 配列データ構造の SORTA 命令の最後の引数として、サブフィールドのリストを指定することができます。 指定されたサブフィールドの値は、配列要素を比較するときに使用されます。 642 ページの『%FIELDS (ソートするためのサブフィールド)』を参照してください。

%FIELDS (更新するフィールド)

```
%FIELDS(name{:name...})
```

フィールドのリストは、自由形式グループでコーディングされる入出力操作 UPDATE に対する最後の引数として指定することができます。 指定されたフィールドのみが更新され、入出力バッファーに入れられます。

注 :

1. それぞれの名前は、そのレコードの入力バッファー内のフィールド名になっている必要があります。 フィールドの名前が変更された場合、 内部名が使用されます。
2. この名前は、更新対象のレコードのファイル名または形式名を EXTNAME/LIKEREC キーワードに指定して定義されているデータ構造のサブフィールドになります。使用するキーワードと共に、*INPUT を指定する必要があります。*NULL を指定してはなりません。 指定する名前には、入力フィールドに対応するサブフィールド名が含まれていなければなりません。 修飾データ構造の場合には、サブフィールドの簡易修飾名を使用します。
3. この名前は、前述のように定義されたデータ構造の LIKEDS キーワードを使用して定義されるデータ構造のサブフィールドになります。

%FIELDS は更新するフィールドのリストを指定します。 例:

図 *225.* フィールドの更新

```
/free
  chain empno record;
  salary = salary + 2000;
  status = STATEXEMPT;
  update record %fields(salary:status);
/end-free
```

%FIELDS (ソートするためのサブフィールド)

```
%FIELDS(name{:name...})
```

配列データ構造の SORTA 命令の最後の引数として、サブフィールドのリストを指定することができます。 サブフィールドは、配列要素の順序を決定するために使用されます。

2 つの配列要素が比較される場合、%FIELDS リストの最初のサブフィールドが比較されます。サブフィールドが等しい場合は、%FIELDS リストの次のサブフィールドが比較されます。 これは、2 つの配列要素が異なるサブフィールドが %FIELDS リストで見つかるか、%FIELDS リストのすべてのサブフィールドが等しくなるまで続きます。

注 : SORTA を使用する %FIELDS には、スカラー・サブフィールド名のみが許可されます。 修飾名、配列、および配列要素は許可されません。

%FIELDS を使用する SORTA の例

SORTA 命令で *ARRAY* の 2 つの要素が比較される場合、*NAME* サブフィールドが最初に比較されます。 *NAME* サブフィールドの値が同じ場合には、*ID* サブフィールドが比較されます。

- *ARRAY* の要素 1 および要素 2 には、*NAME* サブフィールドに同じ値がありません。 要素 1 の *NAME* サブフィールドは「Alice」で、要素 2 の *NAME* サブフィールドは「Tom」であるため、*ARRAY* の要素 2 は要素 1 より大きいと見なされます。

2 番目のサブフィールドである *ID* を比較する必要はありません。

- 要素 1 と要素 3 が比較されると、*NAME* サブフィールドの値は「Alice」と同じになります。 それらが等しいため、2 番目のサブフィールドの *ID* が比較されます。 要素 1 の *ID* サブフィールドは 34567 で、要素 3 の *ID* サブフィールドは 12345 であるため、*ARRAY* の要素 1 は要素 3 より大きいと見なされます。

```
DCL-DS array QUALIFIED DIM(3);
   id PACKED(5);
   name VARCHAR(10);
   type CHAR(2);
END-DS;

array(1).name = 'Alice';
array(1).id = 34567;
array(1).type = 'AB';

array(2).name = 'Tom';
array(2).id = 65432;
array(2).type = 'CD';

array(3).name = 'Alice';
array(3).id = 12345;
array(3).type = 'AB';

SORTA array %FIELDS(name : id);

// array(1).name = 'Alice'
// array(1).id = 12345
// array(1).type = 'AB'

// array(2).name = 'Alice'
// array(2).id = 34567
// array(2).type = 'AB'

// array(3).name = 'Tom'
// array(3).id = 65432
// array(3).type = 'CD'
```

%FLOAT (浮動形式への変換)

```
%FLOAT(numeric or character expression)
```

%FLOAT は、式の値を浮動形式に変換します。 この組み込み関数は式の中でのみ 使用することができます。

パラメーターが文字式である場合は、次の規則が適用されます。

- %DEC の文字式の規則については、569 ページの『組み込み関数を使用して文字値を数値に変換するための規則』を参照してください。
- 無効な数値データが検出された場合、例外が発生し、状況コード 105 が戻されます。

詳しくは、569 ページの『変換命令』または 551 ページの『組み込み関数』を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++
D p1              s             15p 0 inz (1)
D p2              s             25p13 inz (3)
D c15a            s             15a   inz('-5.2e-1')
D c15b            s             15a   inz(' + 5 . 2 ')
D result1         s             15p 5
D result2         s             15p 5
D result3         s             15p 5
D result4         s              8f
 /FREE
 // using numeric parameters
    result1 = p1 / p2;          //  "result1" is now  0.33000.
    result2 = %float (p1) / p2; //  "result2" is now  0.33333.
    result3 = %float (p1 / p2); //  "result3" is now  0.33333.
   result4 = %float (12345);    //  "result4" is now 1.2345E4
 // using character parameters
   result1 = %float (c15a);     //  "result1" is now -0.52000.
   result2 = %float (c15b);     //  "result2" is now  5.20000.
   result4 = %float (c15b);     //  "result4" is now  5.2E0
 /END-FREE
```

図 *226. %FLOAT* の例

%FOUND (検出条件の戻し)

```
%FOUND{(file_name)}
```

%FOUND は、最後に実行された関係のあるファイル命令がレコードを見付けたり、ストリング命令が等しい項目を見付けたり、検索命令が要素を見付けたりした 場合、'1' を戻します。 他の場合、この関数は '0' を戻します。

%FOUND を設定する命令を以下に示します。

- ファイル命令
  - 728 ページの『CHAIN (ファイルからのランダム検索)』
  - 757 ページの『DELETE (レコードの削除)』
  - 873 ページの『SETGT (より大きい設定)』
  - 876 ページの『SETLL (下限の設定)』
- ストリング命令
  - 730 ページの『CHECK (文字の検査)』
  - 732 ページの『CHECKR (逆向きの検査)』
  - 870 ページの『SCAN (ストリングの走査)』

    注 : 組み込み関数 %SCAN は %FOUND の値を 変更しません。
- 検索命令
  - 798 ページの『LOOKUP (テーブルまたは配列要素の検索)』

任意指定のファイル名パラメーターを指定しないで %FOUND を使用した場合、この 関数は、最後に実行された関係のある命令について設定された値を戻します。 ファイル名を指定すると、この関数は、そのファイルで最後に実行された関係のある命令に適用されます。

ファイル命令の場合、%FOUND の機能 は "レコードが見付からない NR" 標識の逆です。

ストリング命令の場合、%FOUND の機能は "検出 FD" 標識と 同じです。

LOOKUP 命令の場合、%FOUND は、検索条件を満たす要素を命令が見付けると、'1' を 戻します。LOOKUP を伴う %FOUND の例については、641 ページの図 224 を参照してください。

詳しくは、576 ページの『ファイル操作』、589 ページの『結果命令』、または 551 ページの『組み込み関数』 を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
F*Filename+IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++
 * File CUSTS has record format CUSTREC
FCUSTS     IF   E           K DISK

 /FREE
    // Check if the customer is in the file
    chain Cust CustRec;
    if %found;
       exsr HandleCustomer;
    endif;
 /END-FREE
```

図 *227.* パラメーターを指定しないでファイル命令のテストに使用する *%FOUND*

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
F*Filename+IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++
 * File MASTER has all the customers
 * File GOLD has only the "privileged" customers
FMASTER    IF   E           K DISK
FGOLD      IF   E           K DISK

 /FREE
   // Check if the customer exists, but is not a privileged customer
   chain Cust MastRec;
   chain Cust GoldRec;

   // Note that the file name is used for %FOUND, not the record name
   if  %found (Master)  and not  %found (Gold);
   //
   endif;
 /END-FREE
```

図 *228.* パラメーターを指定してファイル命令のテストに使用する *%FOUND*

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D Numbers         C                   '0123456789'
D Position        S              5I 0
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 * If the actual position of the name is not required, just use
 * %FOUND to test the results of the SCAN operation.
 * If Name has the value 'Barbara' and Line has the value
 * 'in the city of Toronto.      ', then %FOUND will return '0'.
 * If Line has the value 'the city of Toronto where Barbara lives, '
 * then %FOUND will return '1'.
C     Name          SCAN      Line
C                   IF        %FOUND
C                   EXSR      PutLine
C                   ENDIF
 * If Value contains the value '12345.67', Position would be set
 * to 6 and %FOUND would return the value '1'.
 * If Value contains the value '10203040', Position would be set
 * to 0 and %FOUND would return the value '0'.
C     Numbers       CHECK     Value         Position
C                   IF        %FOUND
C                   EXSR      HandleNonNum
C                   ENDIF
```

図 *229.* ストリング命令をテストするために使用する *%FOUND*

%GEN (生成プログラム {: オプション})

%GEN は、DATA-GEN 命令コードの第 3 オペランドとして使用され、文書を生成するプログラムまたはプロシージャー、および生成プログラムによってサポートされるオプションを指定します。%GEN は値を返しません。また、DATA-GEN 命令コード以外の場所に指定することはできません。

第 1 オペランドは、文書を生成するプログラムまたはプロシージャーを指定します。 以下のいずれかになります。

- プロシージャー・ポインター式
- %PADDR 組み込み関数
- プログラムを特定する文字式です。 これは，次の形式のうちの 1 つを使用しなければなりません
  - MYPGM
  - MYLIB/MYPGM
  - *LIBL/MYPGM
- サービス・プログラム内のプロシージャーを特定する文字式です。 これは，次の形式のうちの 1 つを使用しなければなりません
  - MYSRVPGM(myProcedure)
  - MYLIB/MYSRVPGM(myProcedure)
  - *LIBL/MYSRVPGM(myProcedure)

注 : プロトタイプ名が指定された場合、それはプロシージャー・ポインター値または文字値のいずれかを返すプロシージャーであると想定されます。 生成プログラムがプロトタイプ・プロシージャーである場合は、%PADDR 組み込み関数を使用します。

第 2 オペランドは、生成プログラムに直接渡されるオプションを指定します。 生成プログラムは、サポートするオプションの性質を決定します。

第 2 オペランドが変更可能な変数の名前である場合、変数のアドレスは直接生成プログラムに渡されます。

第 2 オペランドが変更できない文字変数の名前を含む文字式である場合は、式の内容を含むヌル終了ストリングを指すポインターが生成プログラムに渡されます。

生成プログラムは、ポインターがヌル終了ストリングであるかどうかを示す情報を受け取ります。

%GEN の例

第 1 オペランドにはプログラム名が指定されています。 第 2 オペランドは省略されています。 生成プログラム「MYGEN」は、どのオプションも受け取りません。

```
DATA-GEN myfld %DATA(document) %GEN('MYGEN');
```

第 1 オペランドには、サービス・プログラムのプロシージャーが指定されています。 第 2 オペランドには変更可能な変数が指定されています。 プロシージャー「myProcedure」は、「genOptions」データ構造を指すポインターを受け取ります。

```
DCL-DS genOptions LIKEDS(myGenOpts_T);
DATA-GEN myDs %DATA('myData.txt' : 'doc=file')
              %GEN('MYGENPROCS(myProcedure)' : genOptions);
```

第 1 オペランドにはプロシージャー・ポインターが指定されています。 第 2 オペランドには文字式が指定されています。 プロシージャー・ポインターに指定されたプロシージャーは、値「sep=comma」を含むヌル文字終了ストリングを指すポインターを受け取ります。

```
DCL-S p POINTER(*PROC);
DCL-S sep CHAR(1) INZ(',');
DATA-GEN myds %DATA('myData.txt' : 'doc=file')
              %GEN(p : 'sep=' + sep);
```

第 1 オペランドには組み込み関数 %PADDR が指定されています。 第 2 オペランドには変更不可能な文字変数「constParm」が指定されています。 「constParm」の値は「boolean=indicator」です。 プロトタイプ「myProc」によって指定されたプロシージャーは、第 2 オペランドとして指定された値「boolean=indicator」を含むヌル文字終了ストリングを指すポインターを受け取ります。

```
DCL-PI *N;
   constParm VARCHAR(20) CONST;
END-PI;
DATA-GEN myds %DATA('myData.txt' : 'doc=file')
              %GEN(%PADDR(myProc) : constParm);
```

%GEN の例、および DATA-GEN 命令について詳しくは、740 ページの『DATA-GEN (変数からの文書の生成)』 を参照してください。

生成プログラムの作成について詳しくは、Rational Open Access: RPG 版トピックを参照してください。

%GRAPH (図形値への変換)

```
%GRAPH(char-expr | graph-expr | UCS-2-expr { : ccsid })
```

%GRAPH は、文字、図形、または UCS-2 から式の値を変換し、図形値を 戻します。 結果は、パラメーターが可変長であれば可変長になります。

2 番目のパラメーターの *ccsid* はオプション で、結果の式の CCSID を示します。 CCSID は、制御キーワード CCSID(*GRAPH) で指定されるように、モジュールのデフォルト図形 CCSID に デフォルト設定されます。制御仕様書で CCSID(*GRAPH : *IGNORE) が指定されている場合、または モジュールについて想定されている場合、%GRAPH 組み込み関数は使用できません。

パラメーターが定数の場合、変換はコンパイル時に行われます。 この 場合、CCSID は、ソース・ファイルの CCSID に関連した図形 CCSID です。

パラメーターが EBCDIC CCSID の文字データである場合、文字データは次の 形式でなければなりません。

```
shift-out graphic-data shift-in
```

例えば、'oAABBCCi' のように指定します。

データからグラフィックに変換するときにデータを正常に変換できない場合があることについては、261 ページの『変換』を参照してください。

詳しくは、252 ページの『グラフィック形式』、569 ページの『変換命令』、または 551 ページの『組み込み関数』を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
H*Keywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H ccsid (*graph: 300)

D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++
D  char           S              8A   inz('oXXYYZZi')
 * The %GRAPH built-in function is used to initialize a graphic field
D  graph          S             10G   inz (%graph ('oAABBCCDDEEi'))
D  ufield         S              2C   inz (%ucs2 ('oFFGGi'))
D  graph2         S              2G   ccsid (4396) inz (*hival)
D  isEqual        S              1N
D  proc           PR
D    gparm                       2G   ccsid (4396) value

 /FREE
    graph = %graph (char) + %graph (ufield);
    // graph now has the value XXYYZZFFGG.
    // %graph(char) removes the shift characters from the
    // character data, and treats the non-shift data as
    // graphic data.

    isEqual = graph = %graph (graph2 : 300);
    // The result of the %GRAPH built-in function is the value of
    // graph2, converted from CCSID 4396 to CCSID 300.

    graph2 = graph;
    // The value of graph is converted from CCSID 300 to CCSID 4396
    // and stored in graph2.
    // This conversion is performed implicitly by the compiler.

    proc (graph);
    // The value of graph is converted from CCSID 300 to CCSID 4396
    // implicitly, as part of passing the parameter by value.
 /END-FREE
```

図 *230. %GRAPH* の例

注 : この例で、例えば 'oFFGGi' のような、DBCS データ を含んでいるリテラルは、有効な DBCS データまたは有効なシフト文字を含んでいません。 詳しくは、252 ページの『グラフィック形式』 を参照してください。

%HANDLER (handlingProcedure : communicationArea)

%HANDLER は、イベントまたは一連のイベントを処理するプロシージャーを識別するために使用します。 %HANDLER は値を戻しません。また、XML-SAX、XML-INTO、および DATA-INTO の第 1 オペランドとしてのみ指定できます。

第 1 オペランド *handlingProcedure* は、処理プロシージャーのプロトタイプを指定します。 プロトタイプによって指定された、またはプロトタイプが明示的に指定されていない場合にはプロシージャー・インターフェースによって指定された戻り値およびパラメーターは、 処理プロシージャーが必要とするパラメーターと一致している必要があります。 要件は、%HANDLER が指定されている命令によって決まります。 処理プロシージャーの定義に関する特定の要件については、947 ページの『XML-SAX (XML 文書の構文解析)』、908 ページの『XML-INTO (XML 文書の変数への構文解析)』、および 743 ページの『DATA-INTO (文書の変数への構文解析)』 を参照してください。

第 2 オペランド *communicationArea* は、処理プロシージャーのすべての呼び出しでパラメーターとして渡される変数を指定します。 オペランドは、参照によって渡されるプロトタイプ・パラメーターの検査で使用する規則と同じ規則に従って、 処理プロシージャーの最初のプロトタイプ・パラメーターに完全に一致してい る必要があります。 通信域パラメーターには、配列やデータ構造など、任意のタイプを使用できます。

命令コードが %HANDLER 組み込み関数を使用している場合は、以下の一連のイベントが発生します。

1. %HANDLER 組み込み関数を使用する命令が開始します。
2. 処理プロシージャーが処理する必要がある命令の最中にイベントが発生した場合は、RPG ランタイムは、%HANDLER の第 1 オペランドとして指定されている処理プロシージャーを呼び出します。 処理プ

ロシージャーに渡される最初のパラメーターは、%HANDLER の第 2 オペランドとして指定された通信域です。 他のパラメーターは、命令および発生したイベントの性質によって異なります。

3. 処理プロシージャーは、パラメーターを処理する際に通信域パラメーターを更新することがあります。
4. 処理プロシージャーは、正常終了した場合にはゼロを戻し、正常に終了しなかった場合はゼロ以外の値を戻します。
5. 戻り値がゼロの場合、RPG ランタイムは、命令が完了するか、他のイベントが発生するまで処理を継続します。 戻り値がゼロではなかった場合、命令は終了します。
6. 他のイベントが発生した場合は、処理プロシージャーが再び呼び出されます。 処理プロシージャーを呼び出した直前の呼び出しが通信域を変更した場合、 その変更は後続の呼び出しで確認できます。
7. 命令完了時に、制御は %HANDLER 組み込み関数を使用したその命令の次のステートメントに渡されます。 処理プロシージャーが通信域を変更した場合は、その変更は、%HANDLER 組み込み関数を使用したプロシージャーで確認できます。

通信域は、複数の目的で使用できます。

1. %HANDLER 組み込み関数をコード化するプロシージャーから処理プロシージャーに情報を通知します。
2. 処理プロシージャーから %HANDLER 組み込み関数をコード化するプロシージャーに、情報を通知します。
3. 処理プロシージャーの連続的な呼び出しの間で、状況情報を保持します。 状況情報は、処理プロシージャー内の静的変数にも保持できます。 ただし、静的変数が使用されている場合に、処理プロシージャーが、複数の %HANDLER 命令によって使用可能になっている場合、 誤った結果になる場合があります。 処理プロシージャーは、それぞれ独立して使用できるようになります。

```
 * Data structure used as a parameter between
 * the XML-SAX operation and the handling
 * procedure.
 *   - "attrName" is set by the procedure doing the
 *     XML-SAX operation and used by the handling procedure
 *   - "attrValue" is set by the handling procedure
 *     and used by the procedure doing the XML-SAX
 *     operation
 *   - "haveAttr" is used internally by the handling
 *     procedure
D info            DS
D   attrName                    20A    VARYING
D   haveAttr                      N
D   attrValue                   20A    VARYING

 * Prototype for procedure "myHandler" defining
 * the communication-area parameter as being
 * like data structure "info"
D myHandler       PR            10I 0
D   commArea                            LIKEDS(info)
D   event                       10I 0  VALUE
D   string                        *    VALUE
D   stringLen                   20I 0  VALUE
D   exceptionId                 10I 0  VALUE
 /free
   // The purpose of the following XML-SAX operation
   // is to obtain the value of the first "companyname"
   // attribute found in the XML document.

   // The communication area "info" is initialized with
   // the name of the attribute whose value is
   // to be obtained from the XML document.
   attrName = 'companyname';

   // Start SAX processing.  The procedure "myHandler"
   // will be called for every SAX event; the first
   // parameter will be the data structure "info".
   xml-sax(e) %handler(myHandler : info) %xml(xmldoc);
   // The XML-SAX operation is complete.  The
   // communication area can be checked to get the
   // value of the attribute.
   if not %error() and attrValue <> '';
     dsply (attrName + '=' + attrValue);
   endif;
:
:
 * The SAX handling procedure "myHandler"
P myHandler       B
D                 PI            10I 0
D   comm                                LIKEDS(info)
D   event                       10I 0  VALUE
D   string                        *    VALUE
D   stringLen                   20I 0  VALUE
D   exceptionId                 10I 0  VALUE
D value           S          65535A    VARYING
D                                       BASED(string)
D ucs2value       S          16383C    VARYING
D                                       BASED(string)
D rc              S             10I 0  INZ(0)
 /free

     select;
```

図 *231. %HANDLER* を使用した通信域の使用

```
     // When the event is a "start document" event,
     // the handler can initialize any internal
     // subfields in the communication area.
     when event = *XML_START_DOCUMENT;
        comm.haveAttr = *OFF;

     // When the event is an "attribute name" event,
     // and the value of the event is the required
     // name, the internal subfield "haveAttr" is
     // set to *ON.  If the next event is an
     // attribute-value event, the value will be
     // saved in the "attrValue" subfield.
     when event = *XML_ATTR_NAME
     and  %subst(value : 1 : stringLen) = comm.attrName;
        comm.haveAttr = *ON;
        comm.attrValue = '';

     // When "haveAttr" is on, the data from any
     // attribute-value should be saved in the "attrValue"
     // string until the *XML_END_ATTR event occurs
     when comm.haveAttr;
        select;
        when event = *XML_ATTR_CHARS
        or   event = *XML_ATTR_PREDEF_REF;
           comm.attrValue +=
               %subst(value : 1 : stringLen);
        when event = *XML_ATTR_UCS2_REF;
           stringLen = stringLen / 2;
           comm.attrValue +=
               %char(%subst(ucs2value : 1 : stringLen));
        when event = *XML_END_ATTR;
           // We have the entire attribute value
           // so no further parsing is necessary.
           // A non-zero return value tells the
           // RPG runtime that the handler does
           // not want to continue the operation
           rc = -1;
       endsl;

     endsl;

   return rc;
 /end-free
P                 E
```

その他の %HANDLER の例については、947 ページの『XML-SAX (XML 文書の構文解析)』および 908 ページの『XML-INTO (XML 文書の変数への構文解析)』を参照してください。

詳しくは、596 ページの『XML 命令』または 551 ページの『組み込み関数』を参照してください。

%HOURS (時間数)

```
%HOURS(number)
```

%HOURS は、数を、時刻またはタイム・スタンプ値に加算することができる期間に変換します。

%HOURS は、加算式または減算式の中でプラス符号またはマイナス符号の後にのみ置くことができます。 プラス符号またはマイナス符号の前の値は時刻またはタイム・スタンプでなければなりません。 結果は、加算または減算がされた適切な時間数を持つ、時刻またはタイム・スタンプの値になります。 時刻の場合、結果の値は *ISO 形式になります。

日時の算術演算の例は、668 ページの図 239 を参照してください。

詳しくは、572 ページの『日付命令』または 551 ページの『組み込み関数』を参照してください。

%INT (整数形式への変換)

```
%INT(numeric or character expression)
```

%INT は、式の値を整数に変換します。 10 進数はすべて切り捨てられます。 この組み込み関数は式の中でのみ使用できます。%INT を使用すると、浮動値または 10 進数値から小数点以下の桁を切り捨てて、その値を配列指標として使用することができます。

パラメーターが文字式である場合は、次の規則が適用されます。

- %DEC の文字式の規則については、569 ページの『組み込み関数を使用して文字値を数値に変換するための規則』を参照してください。
- 浮動小数点データ (例えば、'1.2E6') は使用できません。
- 浮動小数点データは使用できません。 つまり、数値の後に E と指数 ('1.2E6' など) が存在するものは使用できません。
- 無効な数値データが検出された場合、例外が発生し、状況コード 105 が戻されます。

詳しくは、569 ページの『変換命令』または 551 ページの『組み込み関数』を参照してください。

652 ページの図 232 は %INT 組み込み関数の例を示しています。

%INTH (四捨五入を伴う整数形式への変換)

```
%INTH(numeric or character expression)
```

%INTH は %INT と同じですが、式が 10 進数値、浮動値、または文字値の場合には、整数タイプへの変換時に、式の値に四捨五入が適用されるという点 が異なります。 四捨五入が実行できない場合、メッセージは出されません。

詳しくは、569 ページの『変換命令』または 551 ページの『組み込み関数』を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D p7              s              7p 3 inz (1234.567)
D s9              s              9s 5 inz (73.73442)
D f8              s              8f   inz (123.789)
D c15a            s             15a   inz (' 12345.6789 -')
D c15b            s             15a   inz (' + 9 8 7 . 6 5 4 ')
D result1         s             15p 5
D result2         s             15p 5
D result3         s             15p 5
D array           s              1a   dim (200)
D a               s              1a

 /FREE
 // using numeric parameters
    result1 = %int (p7) + 0.011;  // "result1" is now 1234.01100.
    result2 = %int (s9);          // "result2" is now   73.00000
    result3 = %inth (f8);         // "result3" is now  124.00000.
 // using character parameters
    result1 = %int (c15a);        // "result1" is now  -12345.00000
    result2 = %inth (c15b);       // "result2" is now     988.00000

    // %INT and %INTH can be used as array indexes
    a = array (%inth (f8));
 /END-FREE
```

図 *232. %INT および %INTH の例*

%KDS (データ構造の検索引数)

```
%KDS(data-structure-name{:num-keys})
```

%KDS は、自由形式グループでコーディングされているキー順入出力命令 (CHAIN、DELETE、READE、READPE、SETGT、SETLL) の検索引数として 使用することができます。 検索引数は、組み込み関数の最初の引数としてコーディングされているデータ構造名のサブフィールドによって 指定されます。 キーのデータ構造は、キーワード EXTNAME(...:*KEY) または LIKEREC(...:*KEY) を指定した外部記述データ構造にすることができます (ただし 、これに 限定されません)。

注:

1. 最初の引数は、データ構造名にする必要があります。 これには、キーワード LIKEDS または LIKEREC を定義したサブフィールドが含まれます。
2. 2 番目の引数は、検索引数として使用するサブフィールド数を指定します。

   これには、定数、変数、または式を指定できます。
3. 複合キーの個々のキー値は、データ構造内の最上位のサブフィールドから取得されます。 LIKEDS が定義されたサブフィールドは、 文字データと見なされます。
4. 複合キーの構成に用いられるサブフィールドには、配列は使用できません。
5. すべてのサブフィールドのタイプ ("数値キー" によって指定されている数まで) は実際のキーのタイプと一致する必要があります。 長さ、形式、および CCSID が異なっている場合、 値は変換されます。

   制御キーワード EXPROPTS(*STRICTKEYS) が、%KDS を使用したキーの指定に関する規則に与える影響について詳しくは、335 ページの『*STRICTKEYS』を参照してください。
6. データ構造が、キーワード DIM を使用して配列データ構造として定義されている場合、データ構造に指標を提供する必要があります。
7. キー順入出力命令コードに指定されている命令コードの拡張 H、M、または R は、キー作成域内の対応する位置への検索 引数の転送に影響を与えます。

例:

```
A..........T.Name++++++RLen++TDpB......Functions++++++++++++++++++
A          R CUSTR
A            NAME         100A
A            ZIP           10A
A            ADDR         100A
A          K NAME
A          K ZIP
FFilename++IPEASF.....L.....A.Device+.Keywords++++++++++++++++++++++++++
Fcustfile  if   e           k disk    rename(CUSTR:custRec)
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D custRecKeys     ds                  likerec(custRec : *key)
D numKeys         s             10i 0
 ...
 /free
        // custRecKeys is a qualified data structure
        custRecKeys.name = customer;
        custRecKeys.zip = zipcode;
        // The *KEY data structure is used as the search argument for CHAIN
        chain %kds(custRecKeys) custRec;
        // The number of keys can be a constant
        chain %kds(custRecKeys : 2) custRec;
        // The number of keys can be a variable or an expression
        numKeys = 1;
        chain %kds(custRecKeys : numKeys) custRec;
        chain %kds(custRecKeys : numKeys + 1) custRec;
 /end-free
```

図 *233.* キー順入出力命令における検索の例

%LEN (長さの入手または設定)

```
%LEN(expression)
```

```
%LEN(varying-length expression : *MAX)
```

%LEN を使用して、変数式の長さの入手、可変長フィールドの現在の長さの設定、 または可変長式の最大長の入手が可能です。

このパラメーターは形象定数であってはなりません。

詳しくは、589 ページの『サイズ変更命令』または 551 ページの『組み込み関数』を参照してください。

値として使用される %LEN

この関数を式の右側で使用すると、変数式の桁数または文字数が 戻されます。

数値式の場合、戻り値は、式の精度を表しますが、これは必ずしも、 実際の有効数字の桁数である必要はありません。 浮動変数または浮動式の場合、戻り値は 4 または 8 です。 パラメーターが数値リテラルである場合、戻される長さは、リテラルの桁数です。

英数字の式の場合、戻り値は、式の値の中のバイト数です。

グラフィックまたは UCS-2 の式の場合、戻り値は、式の値の中の 2 バイトの数です。

可変長の値 (組み込み関数または可変長フィールドから戻り値など) の場合、%LEN によって戻り値は、文字、図形、または UCS-2 の値の現在の長さです。 ただし、%LEN が、定義ステートメント内で名前付き定数の値として、またはキーワードのパラメーター の値として使用される場合、%LEN が戻す値は、その可変長フィールドの最大長です。

警告 : CCSID 内の文字の長さがすべて同じではない、UTF-8、UTF-16、または SBCS/DBCS が混在する CCSID などの一部の CCSID では、文字数が %LEN によって戻される値より少なくなる可能性があります。 249 ページの『文字データ・タイプ』を参照してください。

それ以外のデータ・タイプの場合はすべて、戻り値は、その値の バイト数です。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D num1            S                7P 2
D NUM1_LEN        C                   %len(num1)
D NUM1_DECPOS     C                   %decpos(num1)
D num2            S                5S 1
D num3            S                5I 0 inz(2)
D chr1            S               10A   inz('Toronto   ')
D chr2            S               10A   inz('Munich    ')
D ptr             S                 *

 * Numeric expressions:
 /FREE
   num1 = %len(num1);                  //  7
   num1 = %decpos(num2);               //  1
   num1 = %len(num1*num2);             // 12
   num1 = %decpos(num1*num2);          //  3
   // Character expressions:
   num1 = %len(chr1);                  // 10
   num1 = %len(chr1+chr2);             // 20
   num1 = %len(%trim(chr1));           //  7
   num1 = %len(%subst(chr1:1:num3) + ' ' + %trim(chr2));//  9
   // %len and %decpos can be useful with other built-in functions:
   // Although this division is performed in float, the result is
   // converted to the same precision as the result of the eval:
   // Note: %LEN and %DECPOS cannot be used directly with %DEC
   //       and %DECH, but they can be used as named constants
   num1 = 27 + %dec (%float(num1)/num3 : NUM1_LEN : NUM1_DECPOS);
   // Allocate sufficient space to hold the result of the catenation
   // (plus an extra byte for a trailing null character):
   num3 = %len (chr1 + chr2) + 1;
   ptr = %alloc (num3);
   %str (ptr: num3) = chr1 + chr2;
 /END-FREE
```

図 *234. %DECPOS および %LEN の例*

可変長フィールドの長さを設定するために使用する %LEN

この関数を式の左側で使用すると、可変長フィールドの現在の長さが設定されます。 設定する長さが現在の長さより大きい場合、 古い長さと新しい長さの間のフィールド内の文字数は ブランクに設定されます。

注 : パラメーターが可変長である場合、および *MAX が指定されていない場合、%LEN は式の左側でしか 使用できません。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
 *
D city            S             40A   varying inz('North York')
D n1              S              5i 0

 * %LEN used to get the current length of a variable-length field:
 /FREE
    n1 = %len(city);
    // Current length, n1 = 10

    // %LEN used to set the current length of a variable-length field:
    %len (city) = 5;
    // city = 'North' (length is 5)

    %len (city) = 15;
    // city = 'North          '  (length is 15)
 /END-FREE
```

図 *235.* 可変長フィールドに *%LEN* を使用する例

可変長式の最大長を入手するために使用する %LEN

%LEN の 2 番目のパラメーターが *MAX である場合、この関数は、可変長式の最大文字数を戻します。%LEN の最初のパラメーターがフィールド名であれば、この値は、フィールドの定義長と同じになります。例えば、可変長 UCS-2 フィールドが 25C と定義されている場合、%LEN(fld:*MAX) は 25 を戻します。

```
D char_varying    s            100a   varying
D ucs2_varying    s           5000c   varying
D graph_varying   s           7000g   varying(4)
D graph_fld10     s             10g
D char_fld10      s             10a
 /free
     // Calculate several length and size values
     // - The maximum length, %LEN(*MAX), measured in characters
     // - The current length, %LEN, measured in characters
     // - The size, %SIZE, measured in bytes, including the
     //   2- or 4-byte length prefix

     // Each alphanumeric character has one byte
     char_varying = 'abc';
     // Length is 3
     max_len = %len(char_varying : *MAX);
     len = %len(char_varying);
     size = %size(char_varying);
     // max_len = 100
     // len     = 3
     // size    = 102    (100 + 2)

     // Each UCS-2 character has two bytes
     ucs2_varying = 'abc';
     // Length is 3
     max_len = %len(ucs2_varying : *MAX);
     len = %len(ucs2_varying);
     size = %size(ucs2_varying);
     // max_len = 5000
     // len     = 3
     // size    = 10002    (5000 * 2 + 2)

     // Each graphic character has two bytes.
     // For field graph_varying, VARYING(4) was specified,
     // so the length prefix has four bytes
     graph_varying = graph_fld10;
     // Length is 10
     max_len = %len(graph_varying : *MAX);
     len = %len(graph_varying);
     size = %size(graph_varying);
     // max_len = 7000
     // len     = 10
     // size    = 14004    (7000 * 2 + 4)

     // Calculate %LEN(*MAX) of a concatenation
     graph_varying = %subst(graph_fld10:1:5);
     // Length is 5
     max_len = %len(graph_varying + graph_fld10 : *MAX);
     len = %len(graph_varying + graph_fld10);
     // max_len = 7010 (7000 + 10)
     // len     = 15   (5 + 10)

     // Calculate %LEN(*MAX) of a %TRIM expression
     char_fld10 = '1234';
     // Trimmed length is 4
     max_len = %len(%trim(char_fld10) : *MAX);
     len = %len(%trim(char_fld10));
     // max_len = 10 (maximum trimmed length)
     // len = 4      (actual trimmed length)
```

図 *236. *MAX を指定した %LEN の例*

%LIST (項目 { : 項目 { : 項目 ... } })

%LIST は、要素がそのオペランドにリストされている項目の値を持つ一時配列を戻します。

%LIST は、配列を使用できる演算式で使用できます。ただし、以下は例外です。

- SORTA
- %ELEM
- %LOOKUP
- %SUBARR

%LIST の規則:

- オペランドに、配列およびデータ構造を指定することはできません。
- オペランドはすべて、互換性のあるタイプである必要があります。
- 少なくとも 1 つのオペランドが必要です。
- 事実上、オペランドの数に制限はありません。
- オペランドのタイプがオブジェクトである場合、それらはすべて同じクラスで定義されている必要があります。
- %LIST によって戻される配列のデータ・タイプは、オペランドのデータ・タイプによって異なります。666 ページの『複数オペランドの共通タイプの判別』を参照してください。

%LIST の例

- 以下の例では、プログラマーは配列にデータを代入しています。

```
DCL-S colors VARCHAR(20) DIM(5);

colors = %LIST('red' : 'blue' : 'yellow' : 'green': 'orange');
```

- 以下の例では、プログラマーはリスト内の項目のいずれかに値 'Y' があるかどうかを検査しています。

```
IF 'Y' IN %LIST(hadError : notFound : alwaysReport);
   report (hadError : notFound : alwaysReport);
ENDIF;
```

- 以下の例では、プログラマーはリスト内の要素を一度に 1 つずつ処理しています。

```
DCL-S type VARCHAR(20);

FOR-EACH type in %LIST(OVERDUE : PENDING : CANCELLED);
   printReport (type);
ENDFOR;
```

%LOOKUPxx (配列要素の検索)

```
%LOOKUP(arg : array | keyed_array_DS {: start_index {: number_of_elements}})
%LOOKUPLT(arg : array  {: start_index {: number_of_elements}})
%LOOKUPGE(arg : array  {: start_index {: number_of_elements}})
%LOOKUPGT(arg : array  {: start_index {: number_of_elements}})
%LOOKUPLE(arg : array  {: start_index {: number_of_elements}})
```

以下の関数は、配列内の項目のうち *arg* と一致する項目の配列指標を戻します。%LOOKUP は、キー付き配列データ構造内の項目の配列指標を戻すために使用することもできます。

%LOOKUP
正確に一致するもの。

%LOOKUPLT
引数 に最も近いが、引数 よりも小さいもの。

%LOOKUPLE
正確に一致しているか、または引数 に最も近いが引数 よりは小さい値。

%LOOKUPGT
引数 に最も近いが、引数 より大きいもの。

%LOOKUPGE
正確に一致しているか、または引数 に最も近いが引数 より大きい値。

指定された条件に一致する値がない場合は、ゼロが戻されます。 戻り値は符号なし整数形式 (タイプ U) です。

指定された条件に一致する要素が複数ある場合が 考えられます。 以下の説明では、昇順配列と降順配列に次のような値が含まれていると 想定します。

```
           +---+---+---+---+---+---+---+
           | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
-----------+---+---+---+---+---+---+---+
ASCEND     | A | C | C | C | E | E | G |
-----------+---+---+---+---+---+---+---+
DESCEND    | G | E | E | E | C | C | A |
-----------+---+---+---+---+---+---+---+
```

- %LOOKUP、%LOOKUPLE、または %LOOKUPGE では、 検索引数と等しい要素が複数ある場合、最初に一致した要素の 指標が戻されます。 例えば、検索引数が C の場合、これらの組み込み関数 は、昇順配列では 2 を戻し、降順配列では 5 を 戻します。
- 昇順配列に対する %LOOKUPLE 組み込み関数では、検索引数と等しい要素がない 場合、検索引数よりも小さい最後の要素の指標が戻されます。 例えば、検索引数が D の場合、%LOOKUPLE は 4 を戻します。
- 降順配列に対する %LOOKUPLE 組み込み関数では、検索引数と等しい要素がない 場合、検索引数よりも小さい最初の要素の指標が戻されます。 例えば、検索引数が D の場合、%LOOKUPLE は 5 を戻します。
- 昇順配列に対する %LOOKUPGE 組み込み関数では、検索引数と等しい要素がない場合、 検索引数より大きい最初の要素の指標が戻されます。 例えば、検索引数が D の場合、%LOOKUPGE は 5 を戻します。
- 降順配列に対する %LOOKUPGE 組み込み関数では、検索引数と等しい要素がない場合、 検索引数より大きい最後の要素の指標が戻されます。 例えば、検索引数が D の場合、%LOOKUPGE は 4 を戻します。
- 昇順配列に対する %LOOKUPLT 組み込み関数では、検索引数より小さい最後の要素の指標が戻されます。例えば、検索引数が D の場合、%LOOKUPLT は 4 を戻します。
- 降順配列に対する %LOOKUPLT 組み込み関数では、検索引数より小さい最初の要素の指標が戻されます。例えば、検索引数が D の場合、%LOOKUPLT は 5 を戻します。
- 昇順配列に対する %LOOKUPGT 組み込み関数では、 検索引数より大きい最初の要素の指標が戻されます。 例えば、検索引数が D の場合、%LOOKUPGT は 5 を戻します。
- 降順配列に対する %LOOKUPGT 組み込み関数では、 検索引数より大きい最後の要素の指標が戻されます。 例えば、検索引数が D の場合、%LOOKUPGT は 4 を戻します。

検索は指標開始指標 から始まり、要素数 個の要素について続けられます。 デフォルトでは、配列全体が検索されます。

2 番目のパラメーターには ARRAY_NAME 形式の スカラー配列を指定できます。 %LOOKUP の場合は、ARRAY_DS_NAME(*).SUBFIELD_NAME 形式のキー付き配列データ構造 を指定することも可能です。

配列データ構造を検索するには、データ構造名と指標 (*) を指定してから、検索のキーとして使用するサブフィールドを指定します。 例えば､配列データ構造 INFO のサブフィールド CODE で値 'XP2' を検索する場合、 最初のパラメーターとして 'XP2' を指定し、2 番目のパラメーターとして INFO(*).CODE を指定します。 修飾名で指標 (*) までの部分が配列を表し、 (*) の後の部分が、スカラー・サブフィールド、またはスカラーの指標付き配列を表す必要があります。

最初の 2 つのパラメーターはどんなタイプであっても構いませんが、同じタイプでなければなりません。キー付きデータ構造配列の場合、最初のパラメーターは、キーとタイプが同じでなければなりません。 これらの長さまたは小数点以下の桁数は、同じである必要はありません。 3 番目と 4 番目のパラメーターは、小数点以下の桁数がゼロの非浮動数値である必要があります。

%LOOKUPLT、%LOOKUPLE、%LOOKUPGT、および %LOOKUPGE の場合、配列がキーワード ASCEND または DESCEND を使用して定義されている必要があります。 引数 または配列 が ALTSEQ(*NONE) を指定して定義されている場合を除 き、ALTSEQ テーブルが使用されます。

組み込み関数 %FOUND と %EQUAL は %LOOKUP 命令に続けて設定されません。

%LOOKUPxx 組み込み関数は、順序配列 (ASCEND または DESCEND キーワードが指定された配列) を検索するために二分探索を使用します。

注 : LOOKUP 命令コードとは異なり、%LOOKUP は配列にのみ適用されます。 テーブル内の値を検索するには %TLOOKUP 組み込み関数を使用します。

詳細については、以下の資料を参照してください。

- 561 ページの『配列命令』
- 551 ページの『組み込み関数』
- 211 ページの『配列データ構造』

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/FREE
  arr(1) = 'Cornwall';
  arr(2) = 'Kingston';
  arr(3) = 'London';
  arr(4) = 'Paris';
  arr(5) = 'Scarborough';
  arr(6) = 'York';

  n = %LOOKUP('Paris':arr);
  // n = 4

  n = %LOOKUP('Thunder Bay':arr);
  // n = 0 (not found)

  n = %LOOKUP('Kingston':arr:3);
  // n = 0 (not found after start index)

  n = %LOOKUPLE('Paris':arr);
  // n = 4

  n = %LOOKUPLE('Milton':arr);
  // n = 3

  n = %LOOKUPGT('Sudbury':arr);
  // n = 6

  n = %LOOKUPGT('Yorks':arr:2:4);
  // n = 0 (not found between elements 2 and 5)
/END-FREE
```

図 *237.* スカラー配列を使用した *%LOOKUPxx*

```
D emps            DS                  QUALIFIED DIM(20)
D    name                        25A   VARYING
D    id                           9S 0
D numEmps         S              10I 0
 /FREE
   emps(1).name = 'Mary';
   emps(1).id = 00138;
   emps(2).name = 'Patrick';
   emps(2).id = 10379;
   emps(3).name = 'Juan';
   emps(3).id = 06254;
   numEmps = 3;

   // Search for employee 'Patrick'
   n = %lookup('Patrick' : emps(*).name : 1 : numEmps);
   // n = 2

   // Search for the employee with id 06254
   n = %lookup(06254 : emps(*).id : 1 : numEmps);
   // n = 3

   // Search for employee 'Bill' (not found)
   n = %lookup('Bill' : emps(*).name : 1 : numEmps);
   // n = 0
```

図 *238.* 配列データ構造を使用した *%LOOKUP*

正しい順序になっていない順序配列

順序配列の場合にデータが正しい順序になっていないときには、%LOOKUPxx 組み込み関数を使用した場合と LOOKUP 命令コードを使用した場合とで、検出される値が異なることがあります。 %LOOKUPxx 組み込み関数は、配列内にデータ値が存在していても、それを見付けられない場合があります。

%LOOKUPxx 組み込み関数では、順序配列の場合に二分探索が使用され、また、 二分探索が正しく機能するためにはデータが正しい順序になっている必要があるため、 配列の一部の要素だけしか検索で調べられないことがあります。 配列の順序が正しくなっていない場合、二分探索の結果は予測できません。

注 : 順序配列から完全一致項目を見付けるために LOOKUP 命令コードを使用した場合、 検索は指定された要素から開始され、その値が検出されるか、配列の最後の要素に達するかするまで、 一度に 1 つずつ要素が検索されていきます。

%LOWER (小文字に変換)

```
%LOWER(string {: start { : length } })
```

%LOWER は、ストリング・オペランドを小文字に変換します。

%LOWER および %UPPER について詳しくは、660 ページの『%LOWER および %UPPER (小文字または大文字に変換)』 を参照してください。

%LOWER および %UPPER (小文字または大文字に変換)

```
%LOWER(string {: start { : length } })
```

```
%UPPER(string {: start { : length } })
```

%LOWER は、オペランドのすべてまたは一部を小文字に変換して、ストリング・オペランドを戻します。

%UPPER は、オペランドのすべてまたは一部を大文字に変換して、ストリング・オペランドを戻します。

それ以外の場合、組み込み関数は同一です。

第 1 オペランドは、小文字または大文字に変換されるストリングです。 これには英数字タイプまたは UCS-2 タイプを指定できます。

第 2 オペランドは、変換の開始位置です。 これは、小数点以下の桁数がゼロの数値式である必要があります。また、1 からストリングの長さまでの値である必要があります。 これはオプションです。 指定されない場合、変換はストリングの最初の位置から開始されます。

第 3 オペランドは、変換のための長さです。 これは、小数点以下の桁数がゼロの数値式である必要があります。また、開始位置から始まるストリングの長さ以下である必要があります。 ゼロの場合もあります。 これはオプションです。 指定されない場合、変換はストリングの終わりまで続きます。

%LOWER および %UPPER に関する注意事項:

- 既に必要な大/小文字になっている文字は、変更されません。 例えば、%LOWER('AbC') の結果は「abc」です。 「b」は既に小文字になっているため、変更されません。
- 開始位置および長さのオペランドによって表されるサブストリング内のすべての英字が変換されます。 例えば、%LOWER('ÅNGSTRÖM') の結果は 「ångström」です。 *start* の値が 1 で、*length* の値が 1 の場合、%UPPER('a1234bc':start:length) の結果は「A1234bc」です。
- 開始位置および長さのオペランドは、英数字データの場合はバイト数、UCS-2 データの場合は 2 バイト数を表します。 開始位置および長さのオペランドによって表されるサブストリングが完全な文字を表すことを確認するのは、RPG プログラマーの責任です。 一部の CCSID のサブストリングに関する警告を参照してください。
  - 開始位置および長さによって表されるサブストリングが開始位置が 1 より小さいために無効であったり、長さがストリング内に残っているデータの量より大きいために無効であったりする場合、組み込み関数は状況コード 100 で失敗します。
  - 開始位置と長さによって表されるサブストリングが原因で最後の文字が不完全になるために、変換されるデータが無効な場合、組み込み関数は状況コード 125 で失敗します。

%LOWER および %UPPER の例

1. 1 つのパラメーターを持つ %LOWER および %UPPER

```
DCL-S result VARCHAR(10);

result = %LOWER ('HELLO');
// result = "hello"

result = %UPPER ('world');
// result = "WORLD"
```

2. 2 つのパラメーターを持つ %LOWER および %UPPER

```
DCL-S result VARUCS2(10);
DCL-S string UCS2(5);

string = 'HELLO';
result = %LOWER (string : 2);
// result = "Hello"

string = 'world';
result = %UPPER (string : 2);
// result = "wORLD"
```

3. 3 つのパラメーターを持つ %LOWER および %UPPER

```
DCL-S result VARCHAR(10);

result = %LOWER ('HELLO' : 2 : 3);
// result = "HellO"

result = %UPPER ('world' : 1 : 1);
// result = "World"
```

%MAX (最大値)

```
%MAX(item1 : item2 {: item3 { item4 ... } })
```

%MAX は、オペランドの最大値を返します。

%MAX および %MIN について詳しくは、662 ページの『%MAX および %MIN (最大値または最小値)』 を参照してください。

%MAXARR (配列内の最大要素)

```
%MAXARR(array {: start-index {:number-of-elements}})
```

%MAXARR は、配列内の最大値の指標、または *start-element* オペランドおよび *number-of-elements* オペランドによって識別される配列のサブセクションの指標を戻します。

%MAXARR 組み込み関数と %MINARR 組み込み関数について詳しくは、663 ページの『%MAXARR および %MINARR (配列内の最大要素および最小要素)』 を参照してください。

%MIN (最小値)

```
%MIN(item1 : item2 {: item3 { item4 ... } })
```

%MIN は、オペランドの最小値を返します。

%MAX および %MIN について詳しくは、662 ページの『%MAX および %MIN (最大値または最小値)』 を参照してください。

%MINARR (配列内の最小要素)

```
%MINARR(array {: start-index {:number-of-elements}})
```

%MINARR は、配列内の最小値、または *start-element* オペランドおよび *number-of-elements* オペランドによって識別される配列のサブセクションの指標を戻します。

%MAXARR 組み込み関数と %MINARR 組み込み関数について詳しくは、663 ページの『%MAXARR および %MINARR (配列内の最大要素および最小要素)』 を参照してください。

%MAX および %MIN (最大値または最小値)

```
%MAX(item1 : item2 {: item3 { item4 ... } })
```

```
%MIN(item1 : item2 {: item3 { item4 ... } })
```

%MAX はオペランドの最大値を返し、%MIN はオペランドの最小値を返します。 その他について、これらの組み込み関数の規則と動作は同一です。

オペランドはすべて、相互比較用に互換性のあるデータ・タイプを持っていなければなりません。 例えば、リスト内のある項目が英数字の場合、その他の項目には英数字、UCS-2、またはグラフィックを指定可能で

す。 ある項目がパック形式の数値の場合、その他の項目にはパック形式の数値、ゾーン形式の数値、整数、符号なし整数、2 進化 10 進数、または浮動を指定することができます。

タイプが「プロシージャー・ポインター」または「オブジェクト」の項目はオペランドとして使用できません。

2 つ以上のオペランドが必要です。 オペランドの数には事実上、上限はありません。

この組み込み関数を宣言ステートメントで使用する場合、オペランドは確実に 2 つでなければなりません。どちらのオペランドも数値でなければならず、浮動や 16 進数は使用できません。

10 進数オペランドに命令の結果以上の小数点以下の桁数が含まれている場合、四捨五入が使用されます。

この組み込み関数を宣言ステートメントで使用する場合、その結果は、より高い (%MAX) 値またはより低い (%MIN) 値が指定されたオペランドになります。

計算で組み込み関数により返される値のデータ・タイプは、オペランドのデータ・タイプに依存します。666 ページの『複数オペランドの共通タイプの判別』を参照してください。

配列内の最大値または最小値を検索するには、%MAXARR または %MINARR を使用します。

%MAX および %MIN の例

1. %MAX が宣言ステートメントで使用されています。 *arr3* の次元は 5 で、*arr1* および *arr1* の次元の最大値になります。

```
DCL-S arr1 CHAR(10) DIM(3);
DCL-S arr2 CHAR(10) DIM(5);
DCL-S arr3 CHAR(10) DIM(%MAX(%ELEM(arr1) : %ELEM(arr2)));
```

2. %MIN が演算ステートメントで使用されています。

```
DCL-S triangleArea PACKED(7 : 2);
DCL-S squareArea PACKED(7 : 2);
DCL-S circleArea PACKED(5 : 2);
DCL-S size PACKED(7 : 2);

size = %MIN (triangleArea : squareArea : circleArea);
```

%MAXARR および %MINARR (配列内の最大要素および最小要素)

```
%MAXARR(array {: start-index {:number-of-elements}})
```

```
%MINARR(array {: start-index {:number-of-elements}})
```

%MAXARR は、配列の最大値の指標を戻します。%MINARR は、配列の最小値の指標を戻します。

%MAXARR および %MINARR の第 1 オペランドには、*ARRAY_DS_NAME(*).SUBFIELD_NAME* 形式のスカラー配列またはキー付き配列データ構造を指定できます。

配列データ構造内のサブフィールドの最大値を検索するには、データ構造名に指標 (*) を指定してから、サブフィールドを指定します。 例えば、サブフィールド *CODE* に最大値を持つ配列データ構造 *INFO* を検索するには、第 1 オペランドとして *INFO(*).CODE* を指定します。 (*) 指標までの修飾名の部分は配列を表す必要があり、(*) の後の修飾名の部分は、スカラー・サブフィールドまたは指標付きスカラー配列を表す必要があります。

start-element オペランドを指定する場合、最大要素または最小要素の検索は、指定された要素から開始されます。 それ以外の場合、最大要素または最小要素の検索は、最初の要素から開始されます。

number-of-elements オペランドが指定される場合、最大要素および最小要素の検索は、指定された数の要素に制限されます。 それ以外の場合、最大要素または最小要素の検索には、*start-element* から始まるすべ

ての要素が含まれます。 *number-of-elements* オペランドの値がゼロの場合、要素は検索されません。 %MAXARR および %MINARR は、値ゼロを戻します。

DIM(*AUTO) または DIM(*VAR) によって指定される変動次元配列では、配列の要素が現時点でゼロの場合、%MAXARR および %MINARR は値ゼロを戻します。

英数字配列に対して代替照合順序が有効な場合、配列に ALTSEQ(*NONE) が指定されていない限り、代替照合順序が使用されます。 例えば、制御キーワード ALTSEQ(*EXT) および SRTSEQ(*LANGIDSHR) が指定されている場合、値「ABC」と値「abc」は等しいと見なされます。 配列にこれら 2 つの値を持つ要素が含まれている場合、これらの値のいずれかを持つ最初の要素の指標が %MAXARR または %MINARR によって戻されます。

シーケンス配列の %MAXARR および %MINARR

配列が昇順の場合 (配列にキーワード ASCEND が指定されている場合)、%MAXARR の検索は最後の検索対象要素から始まり、最後の要素と等しくない値が見つかるまで後方に戻り続けます。 例えば、昇順配列 *array1* に値 [1,2,3,3,3] が含まれている場合、%MAXARR は、値 3 を持つ最初の要素の指標を戻します。

配列が降順の場合 (配列にキーワード DESCEND が指定されている場合)、%MAXARR は配列内の最初の要素の指標を戻します。

配列が昇順の場合 (配列にキーワード ASCEND が指定されている場合)、%MINARR は配列内の最初の要素の指標を戻します。

配列が降順の場合 (配列にキーワード DESCEND が指定されている場合)、%MINARR の検索は最後の検索対象要素から始まり、最後の要素と等しくない値が見つかるまで後方に戻り続けます。 例えば、降順配列 *array2* に値 [5,4,3,2,2] が含まれている場合、%MAXARR は、値 2 を持つ最初の要素の指標を戻します。

警告 : %MAXARR および %MINARR は、配列が正しい順序になっていると想定します。 シーケンス配列が現時点で正しい順序でソートされていない場合、結果は、ソートされていない配列の最大値または最小値の指標を表さない可能性があります。

%MAXARR および %MINARR の例

- 式で使用されている %MAXARR および %MINARR
  1. 3 番目の要素は配列内の最大値を持つため、%MAXARR は値 3 を戻します。 %MAXARR の結果は配列の指標として使用されるため、*value* には要素 3 の値「Saturn」が入ります。
  2. 4 番目の要素は配列内の最小値を持つため、%MINARR は値 4 を戻します。 %MINARR の結果は配列の指標として使用されるため、*value* には要素 4 の値「Jupiter」が入ります。

```
DCL-S array CHAR(10) DIM(5);
DCL-S value LIKE(array);

array = %LIST('Mercury' : 'Mars' : 'Saturn' : 'Jupiter' : 'Neptune');

value = array(%MAXARR(array)); // 1

value = array(%MINARR(array)); // 2
```

- キー付き配列データ構造の %MAXARR および %MINARR
  1. *array* の 2 番目の要素は *NAME* サブフィールドに最大値「Tom」を持つので、%MAXARR は値 2 を戻します。
  2. *array* の最初の要素は *ID* サブフィールドで最大値「12345」を持つので、%MAXARR は値 1 を戻します。

```
DCL-DS array QUALIFIED DIM(3);
   name VARCHAR(10);
   id PACKED(5);
END-DS;
DCL-S p INT(10);

array(1).name = 'Jack';
array(1).id = 12345;
array(2).name = 'Tom';
array(2).id = 65432;
array(3).name = 'Alice';
array(3).id = 34567;

p = %MAXARR(array(*).name); // 1

p = %MINARR(array(*).id);   // 2
```

- シーケンス化されていない配列 (ASCEND および DESCEND が配列に指定されていない) の %MAXARR
  1. 最大要素を求めて配列全体が検索されます。 配列内の最大値は「g」であるため、*maxElem* の値は 2 です。
  2. *start-element* オペランドには値 3 が指定されているため、*maxElem* には要素 3 から始まる要素の最大値の指標が設定されます。 要素 3 から 5 の中で、最大値は「f」です。 2 つの要素に値「f」がありますが、%MAXARR は最大値を持つ最初の要素の指標を戻します。そのため、*maxElem* には値 3 が入ります。
  3. *start-element* オペランドには値 4 が指定されているので、*maxElem* には要素 4 から 5 の最大値の指標が設定されます。 要素 3 と 4 の両方に値「f」がありますが、要素 3 は検索対象外であるため、*maxElem* には値 4 が入ります。

```
DCL-S array CHAR(10) DIM(5);
DCL-S maxElem INT(10);

array = %LIST('a' : 'g' : 'f' : 'f' : 'c');

maxElem = %MAXARR (array); // 1

maxElem = %MAXARR (array : 3); // 2

maxElem = %MAXARR (array : 4); // 3
```

- シーケンス化されていない配列 (ASCEND および DESCEND が配列に指定されていない) の %MINARR
  1. 最小要素を求めて配列全体が検索されます。 配列内の最大値は「g」であるため、*maxElem* には値 2 が入ります。
  2. *start-element* オペランドには値 3 が指定されているため、*minElem* には、要素 3 から始まる要素の最小値の指標が設定されます。 要素 3 から 5 の中で、最小値は「c」です。 2 つの要素に値「c」が含まれていますが、%MINARR は最大値を持つ最初の要素の指標を戻します。したがって、*minElem* には値 3 が入ります。
  3. *start-element* オペランドには値 4 が指定されているので、*minElem* には要素 4 から 5 の最小値の指標が設定されます。 要素 3 と 4 の両方に値「c」が含まれていますが、要素 3 は検索対象外であるため、*minElem* には値 4 が入ります。
  4. *start-element* オペランドに値 2 が指定され、*number-of-elements* オペランドに値 3 が指定されているため、*minElem* には要素 2 から 4 の最小値の指標が設定されます。 これらの要素の最小値は「b」であるため、*minElem* には値 2 が入ります。

```
DCL-S array CHAR(10) DIM(5);
DCL-S minElem INT(10);

array = %LIST('k' : 'b' : 'c' : 'c' : 'x');

minElem = %MINARR (array); // 1

minElem = %MINARR (array : 3); // 2

minElem = %MINARR (array : 4); // 3

minElem = %MINARR (array : 2 : 3); // 4
```

- 昇順配列 (ASCEND が配列に指定されている) の %MAXARR および %MINARR
  1. 最大要素の検索は、配列の最後から開始されます。 配列内の最後の 2 つの要素の値は同じであるため、*maxElem* には値 4 が入ります。
  2. 昇順の配列の最初の要素は配列の最小値であるため、*minElem* には値 1 が入ります。

```
DCL-S array CHAR(10) DIM(5) ASCEND;
DCL-S maxElem INT(10);
DCL-S minElem INT(10);

array = %LIST('a' : 'a' : 'b' : 'c' : 'c');

maxElem = %MAXARR (array); // 1

minElem = %MINARR (array); // 2
```

複数オペランドの共通タイプの判別

%MAX、%MIN、および %LIST などの組み込み関数の場合、組み込み関数によって戻される値のデータ・タイプは、そのオペランドのデータ・タイプによって決まります。

- すべてのオペランドが 16 進リテラルの場合、結果は CCSID(*HEX) の英数字になります。 そうでない場合、16 進数リテラルは、組み込み関数のその他のオペランドに応じて数値または文字として処理されます。
- すべてのオペランドのデータ・タイプが日付である場合、その結果には形式 *ISO を使用した日付のデータ・タイプが使用されます。
- すべてのオペランドのデータ・タイプが時刻である場合、その結果には形式 *ISO を使用した時刻のデータ・タイプが使用されます。
- すべてのオペランドのデータ・タイプがタイム・スタンプである場合、その結果にはタイム・スタンプのデータ・タイプが使用され、いずれかのオペランドの最大小数桁数が戻り値の小数桁数になります。
- すべてのオペランドのデータ・タイプがポインターであった場合、その結果にはポインターのデータ・タイプが使用されます。
- すべてのオペランドの日付タイプがプロシージャー・ポインターである場合、その結果にはプロシージャー・ポインターのデータ・タイプが使用されます。
- すべてのオペランドのデータ・タイプが数値であった場合、形式、長さおよび小数点以下の桁数を判別するために、以下の考慮事項が適用されます。
  - いずれかのオペランドが浮動の場合、戻り値は長さが 8 の浮動になります。
  - それ以外で、いずれかのオペランドがパック 10 進数、ゾーン 10 進数、または 2 進化 10 進数の場合は、戻り値はパック 10 進数になります。 オペランドの整数桁の最大数が戻り値の整数の桁数になります。 オペランドの小数点以下の最大桁数が戻り値の小数点以下の桁数になります。 整数の桁数に小数点以下の桁数を加えた数が 63 を超えた場合、合計長が 63 になるように小数点以下の桁数が減らされます。

  - それ以外で、いずれかのオペランドが整数であり、もう 1 つのオペランドが符号なし整数の場合は、戻り値は長さが 20 桁で小数点以下の桁数が 0 のパック 10 進数になります。
  - それ以外で、いずれかのオペランドが整数の場合は、戻り値は長さが 20 の整数になります。
  - それ以外の場合、戻り値は長さが 20 の符号なし整数になります。
- すべてのオペランドに英数字、UCS-2、またはグラフィックのデータ・タイプが使用されている場合、データ・タイプ、長さ、CCSID を判別するために、以下の考慮事項が適用されます。
  - いずれかのオペランドが CCSID *UTF-8 を使用した英数字である場合、戻り値は CCSID *UTF-8 を使用した英数字になります。
  - それ以外で、いずれかのオペランドが UCS-2 の場合は、戻り値が UCS-2 になります。 すべての UCS-2 オペランドに同じ CCSID が使用されている場合、戻り値にもその CCSID が使用されます。 それ以外の場合、戻り値にはモジュールのデフォルトの UCS-2 CCSID が使用されます。
  - それ以外で、英数字とグラフィックのオペランドが混在している場合は、戻り値はモジュールのデフォルトの UCS-2 CCSID を使用した UCS-2 になります。
  - それ以外で、すべてのオペランドがグラフィックであり、すべてのグラフィック・オペランドに同じ CCSID が使用されている場合、戻り値はオペランドと同じ CCSID を使用したグラフィックになります。
  - それ以外で、すべてのオペランドがグラフィックであるが異なる CCSID が使用されている場合は、戻り値はモジュールのデフォルトの UCS-2 CCSID を使用した UCS-2 になります。
  - それ以外で、すべてのオペランドが英数字であり、すべての英数字オペランドに同じ CCSID が使用されている場合、戻り値はオペランドと同じ CCSID を使用した英数字になります。 これを判別する際、ジョブ CCSID とそのジョブ CCSID に関連付けられた混合 CCSID は同一のものであるとみなされます。 少なくとも 1 つの CCSID が、ジョブ CCSID に関連付けられた混合 CCSID である場合、戻り値にはその CCSID が使用されます。
  - それ以外で、すべてのオペランドが英数字であるものの異なる CCSID が使用されている場合は、戻り値は CCSID *UTF-8 を使用した英数字になります。 戻り値の長さは、オペランドの最大長になります。または、CCSID の変換が必要な場合には、戻り値の CCSID に変換されるオペランドの可能な最大長になります。 この長さが許容最大長を超えた場合、その長さは最大長に短縮されます。 英数字またはグラフィックのオペランドに CCSID *HEX が使用されている場合、すべてのオペランドは同じデータ・タイプ (英数字かグラフィック) でなければなりません。 戻り値に CCSID *HEX が使用されるのは、すべてのオペランドに CCSID *HEX が使用されている場合のみです。

    警告 : Unicode または ASCII での比較は EBCDIC での比較とは異なります。 249 ページの『異なるデータ・タイプまたは CCSID とデータを比較するときに生じる予期しない結果』を参照してください。

%MINUTES (分数)

```
%MINUTES(number)
```

%MINUTES は、数を、時刻またはタイム・スタンプ値に加算することができる期間に変換します。

%MINUTES は、加算式または減算式の中でプラス符号またはマイナス符号の後にのみ置くことができます。 プラス符号またはマイナス符号の前の値は時刻またはタイム・スタンプでなければなりません。 結果は、加算または減算がされた適切な分数を持つ、時刻またはタイム・スタンプの値になります。 時刻の場合、結果の値は *ISO 形式になります。

日時の算術演算の例は、668 ページの図 239 を参照してください。

詳しくは、572 ページの『日付命令』または 551 ページの『組み込み関数』を参照してください。

%MONTHS (月数)

```
%MONTHS(number)
```

%MONTHS は、数を、日付またはタイム・スタンプ値に加算することができる期間に変換します。

%MONTHS は、加算式または減算式の中でプラス符号またはマイナス符号の後にのみ置くことができます。 プラス符号またはマイナス符号の前の値は日付またはタイム・スタンプでなければなりません。 結果は、加算または減算がされた適切な月数を持つ、日付またはタイム・スタンプの値になります。 日付の場合、結果の値は *ISO 形式になります。

ほとんどの場合、所与の月の数値を加算したり減算したりした結果は明白です。 例えば、2000-03-15 + %MONTHS(1) は 2000-04-15 となります。 加算または減算によって実在しない日付が生成された場合 (たとえば 2 月 30 日) は、代わりにその月の最後の日付が使用されます。

ある月の 29 日、30 日、31 日に対してある月数を加算または減算することは、元に戻すことができない場合があります。 たとえば、2000-03-31 + %MONTHS(1) - %MONTHS(1) は 2000-03-30 になります。

詳しくは、572 ページの『日付命令』、551 ページの『組み込み関数』、および 574 ページの『予期しない結果』 を参照してください。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/FREE

    // Determine the date in 3 years
    newdate = date + %YEARS(3);

    // Determine the date in 6 months prior
    loandate = duedate - %MONTHS(6);

    // Construct a timestamp from a date and time
    duestamp = duedate + t'12.00.00';
/END-FREE
```

図 *239. %MONTHS と %YEARS の例*

%MSECONDS (マイクロ秒数)

```
%MSECONDS(number)
```

%MSECONDS は、数を、時刻またはタイム・スタンプ値に加算することができる期間に変換します。

%MSECONDS は、加算式または減算式の中でプラス符号またはマイナス符号の後にのみ置くことができます。 プラス符号またはマイナス符号の前の値は時刻またはタイム・スタンプでなければなりません。 結果は、加算または減算がされた適切なマイクロ秒数を持つ、時刻またはタイム・スタンプの値になります。 時刻の場合、結果の値は *ISO 形式になります。

小数点以下の桁を持つ値を指定して %SECONDS を使用することによっても、マイクロ秒または他の任意の 1 秒未満のサイズ (ミリ秒など) をタイム・スタンプに加算したり、タイム・スタンプから減算したりすることができます。%SECONDS(.000005) と %MSECONDS(5) はどちらも 5 マイクロ秒を表します。 %SECONDS(.123) は 123 ミリ秒を表します。 詳しくは、686 ページの『%SECONDS (秒数)』 を参照してください。

日時の算術演算の例は、668 ページの図 239 を参照してください。

詳しくは、572 ページの『日付命令』 または 551 ページの『組み込み関数』 を参照してください。

%NULLIND (ヌル標識の照会または設定)

```
%NULLIND(fieldname)
```

%NULLIND 組み込み関数は、ヌル値可能フィールドのヌル標識を照会または設定するために使用することができます。 この組み込み関数 は、ALWNULL(*USRCTL) キーワードが 制御仕様書に指定されたか、またはコマンド・パラメーターとして指定された場合のみ使用 することができます。 フィールド名は、ヌル値可能配列要素、データ構造、 独立フィールド、サブフィールド、または複数オカレンス・データ構造の 場合があります。

%NULLIND は、拡張演算項目 2 の中の式でのみ使用することができます。

この関数を式の右側で使用すると、ヌル値可能フィールドのヌル標識の設定値が戻されます。 この設定値は *ON または *OFF です。

この関数を式の左側で使用すると、ヌル値可能フィールドのヌル標識 を *ON または *OFF に設定することができます。 ヌル値可能フィールドの内容は変わりません。

ヌル値可能フィールドおよびキーを持つレコードの処理については、286 ページの『データベースのヌル値サポート』を参照してください。

詳しくは、580 ページの『標識設定命令』または 551 ページの『組み込み関数』を参照してください。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
* Test the null indicator for a null-capable field.
/FREE
   if  %nullind (fieldname1);
       // field is null
   endif;

   // Set the null indicator for a null-capable field.
   %nullind(fieldname1) = *ON;
   %nullind (fieldname2) = *OFF;
/END-FREE
```

図 *240. %NULLIND* の例

%OCCUR (データ構造のオカレンスの設定/取り出し)

```
%OCCUR(dsn-name)
```

%OCCUR は、複数オカレンス・データ構造の現在位置の設定や入手を行ないます。

この関数は、その値について評価を受けた場合、指定されたデータ構造の現在のオカレンス番号を戻します。 これは符号なしの数値になります。

この関数が EVAL ステートメントの左側に指定された時は、指定されている数値が現在のオカレンス番号になります。 これは、小数部ゼロの非浮動数値である必要があります。 例外 00122 は、この値が 1 より小さいか、またはオカレンスの合計数よりも大きい場合に出されます。

複数オカレンス・データ構造および OCCUR 命令コードについては、835 ページの『OCCUR (データ構造のオカレンスの設定/取り出し)』を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D mds             DS                  OCCURS(10)

 /FREE
   n = %OCCUR(mds);
   // n = 1

   %OCCUR(mds) = 7;

   n = %OCCUR(mds);
   // n = 7
 /END-FREE
```

図 *241. %OCCUR* の例

%OPEN (ファイル・オープン条件の戻し)

```
%OPEN(file_name)
```

%OPEN は、指定されたファイルがオープンされている場合に '1' を戻します。 ファイルが「オープンしている」と考えられるのは、初期化時に RPG モジュール によって、あるいは OPEN 命令によってオープンさ

れ、その後クローズされていない場合です。 ファイルが外部標識によって条件付けられ、 外部標識がモジュール初期化時にオフであった場合、ファイルはクローズしていると見なされ、%OPEN は '0' を戻します。

詳しくは、576 ページの『ファイル操作』または 551 ページの『組み込み関数』を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
F*Filename+IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++
 * The printer file is opened in the calculation specifications
FQSYSPRT   O    F  132        PRINTER USROPN

 /FREE
    // Open the file if it is not already open
    if not %open (QSYSPRT);
       open QSYSPRT;
    endif;
 /END-FREE
```

図 *242. %OPEN* の例

%PADDR (プロシージャー・アドレスの検索)

```
%PADDR(string|prototype)
```

%PADDR はプロシージャー・ポインター・タイプの値を戻します。 この値は、引数として指定された入り口点のアドレスです。

%PADDR はプロシージャー・ポインター・タイプの項目とだけ比較することが でき、プロシージャー・ポインター・タイプの項目にだけ割り当てることができ ます。

%PADDR に対するパラメーターは、文字定数またはプロトタイプ名でなければなりません。 プロシージャーのプロトタイプが、そのプロシージャー・インターフェースから暗黙的に定義されている場合、 プロトタイプ名はプロシージャー名と同じです。

文字定数は、文字リテラルまたは 16 進数リテラル、あるいは文字リテラル または 16 進数リテラルを表す定数名でなければなりません。 文字定数が使用される場合、これは名前によって入り口点を示しています。

プロトタイプは、結合呼び出し用のプロトタイプでなければなりません。 EXTPGM キーワードは使用できません。 プロトタイプによって示される入り口点は、プロトタイプの EXTPROC キーワードに指定されているプロシージャーになります。 EXTPROC キーワードが指定されていない場合は、入り口点はプロトタイプ名 (英大文字) と同じになります。

```
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D
D PROC            S               *   PROCPTR
D                                     INZ (%PADDR ('FIRSTPROG'))
D PROC1           S               *   PROCPTR
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
CL0N01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++
 *
 *  The following statement calls procedure 'FIRSTPROG'.
 *
C                   CALLB     PROC
 *---------------------------------------------------------------------
 * The following statements call procedure 'NextProg'.
 * This a C procedure and is in mixed case.  Note that
 * the procedure name is case sensitive.
 *
C                   EVAL      PROC1 = %PADDR ('NextProg')
C                   CALLB     PROC1
```

図 *243.* 入り口点のある *%PADDR* の例

プロトタイプとともに使用される %PADDR

%PADDR の引数はプロトタイプ名であっても構いませんが、次の制約事項があります。

- これは、Java メソッドのプロトタイプであってはなりません。
- EXTPGM キーワードがあってはなりません。
- EXTPROC キーワードに引数へのプロシージャー・ポインターがある場合、%PADDR を定義仕様書で使用することはできません。

```
 *-------------------------------------------------------------
 * Several prototypes
 *-------------------------------------------------------------
D proc1           PR
D proto2          PR                  EXTPROC('proc2')
D proc3           PR                  EXTPROC(procptr3)
D pgm1            PR                  EXTPGM('PGM3')
D meth            PR                  EXTPROC(*JAVA : 'myClass'
D                                           : 'meth1')

D procptr3        S               *

 *-------------------------------------------------------------
 * Valid examples of %PADDR with prototype names as the argument
 *-------------------------------------------------------------

 * constant1 is the same as %PADDR('PROC1') since 'PROC1' is the
 * procedure called by the prototype proc1
D constant1       C                   %PADDR(proc1)

 * constant2 is the same as %PADDR('proc2') since 'proc2' is the
 * procedure called by the prototype proto2
D constant2       C                   %PADDR(proto2)

 * %paddr(proc3) is the same as procedure pointer procptr3 since
 * procptr3 points to the procedure called by prototype proc3
C                   eval      procptr = %paddr(proc3)

 *-------------------------------------------------------------
 * Examples of %PADDR with prototype names as the argument
 * that are not valid
 *-------------------------------------------------------------
 * %PADDR(pgm1) is not valid because it is a prototype for a program
 * %PADDR(meth) is not valid because it is a prototype for a Java method
```

図 *244.* プロトタイプとともに使用される *%PADDR* の例

```
 * constant1 is the same as %PADDR('myProc1').  Prototype
 * proc1 is implicitly defined from the procedure interface
 * of procedure proc1.  The external name 'myProc1' is
 * defined by the EXTPROC keyword of the implicitly defined
 * prototype.
D constant1       C                   %PADDR(proc1)

 * constant2 is the same as %PADDR('PROC2').  Prototype
 * proc2 has no prototype or procedure interface, so it has
 * a default prototype with the external name the same as
 * the internal procedure name.
D constant2       C                   %PADDR(proc2)


P proc1           B
 * The prototype for proc1 is implicitly defined from the
 * procedure interface.
 * - The name of the implicit prototype is proc1, the name
 *   of the procedure
 * - The external procedure name is 'myProc1' taken from the
 *   EXTPROC keyword of the procedure interface
D                 PI                  EXTPROC('myProc1')
 ...
P                 E

P proc2           B
 * No procedure interface is specified.
 * A default prototype is implicitly defined.
 * - The name of the implicit prototype is proc2, the name
 *   of the procedure
 * - The external procedure name is 'PROC2' taken from the
 *   uppercased form of the name of the procedure.
...P                E
```

図 *245.* プロシージャー・インターフェースから暗黙的に定義されたプロトタイプのプロシージャーを使用する *%PADDR*

%PARMS (パラメーター数の戻り)

%PARMS は、%PARMS が使用されたプロシージャーに渡されているパラメーターの 数を戻します。 サイクル・メイン・プロシージャーの %PARMS は、プログラム状況データ構造の *PARMS と同じです。

バインド呼び出しによって呼び出されたプロシージャーで %PARMS が使用される場合、呼び出し側のプログラムまたはプロシージャーが最小操作記述子を渡さないと、%PARMS によって戻される値は使用可能になりません。 他の言語とは異なり、ILE RPG コンパイラーは常に 1 つの操作記述子を渡します。 したがって、呼び出し元が別の ILE 言語で作成されていた場合には、呼び出し時に操 作記述子を渡すことが必要になります。 操作記述子が渡されない場合、%PARMS によって戻された値は信頼できません。操作記述子が渡されなかったとシステムによって判断された場合、%PARMS によって戻される値は -1 になります。しかし、場合によっては、システムがそれを検知できず、%PARMS によって戻される値が正しくない値 (0 以上) になることがあります。

%PARMS で戻される値には、RTNPARM キーワードが指定されたときに戻り値の処理に使用される追加の最初のパラメーターが含まれます。 詳しくは、「477 ページの『RTNPARM』」を参照してください。

詳しくは、563 ページの『呼び出し命令』または 551 ページの『組み込み関数』を参照してください。

```
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
 * Prototype for procedure MaxInt which calculates the maximum
 * value of its parameters (at least 2 parameters must be passed)
D MaxInt          PR            10I 0
D  p1                           10I 0 VALUE
D  p2                           10I 0 VALUE
D  p3                           10I 0 VALUE OPTIONS(*NOPASS)
D  p4                           10I 0 VALUE OPTIONS(*NOPASS)
D  p5                           10I 0 VALUE OPTIONS(*NOPASS)
D Fld1            S             10A   DIM(40)
D Fld2            S             20A
D Fld3            S            100A
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
CL0N01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++
C     *ENTRY        PLIST
C                   PARM                    MaxSize          10 0
 * Make sure the main procedure was passed a parameter
C                   IF        %PARMS < 1
C     'No parms'    DSPLY
C                   RETURN
C                   ENDIF
 * Determine the maximum size of Fld1, Fld2 and Fld3
C                   EVAL      MaxSize = MaxInt(%size(Fld1:*ALL) :
C                                              %size(Fld2) :
C                                              %size(Fld3))
C     'MaxSize is'  DSPLY                   MaxSize
C                   RETURN
```

図 *246. %PARMS の例*

```
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
 *---------------------------------------------------------------
 * MaxInt - return the maximum value of the passed parameters
 *---------------------------------------------------------------
P MaxInt          B
D MaxInt          PI            10I 0
D  p1                           10I 0 VALUE
D  p2                           10I 0 VALUE
D  p3                           10I 0 VALUE OPTIONS(*NOPASS)
D  p4                           10I 0 VALUE OPTIONS(*NOPASS)
D  p5                           10I 0 VALUE OPTIONS(*NOPASS)
D Max             S             10I 0 INZ(*LOVAL)
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
CL0N01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++
 * Branch to the point in the calculations where we will never
 * access unpassed parameters.
C                   SELECT
C                   WHEN      %PARMS = 2
C                   GOTO      PARMS2
C                   WHEN      %PARMS = 3
C                   GOTO      PARMS3
C                   WHEN      %PARMS = 4
C                   GOTO      PARMS4
C                   WHEN      %PARMS = 5
C                   GOTO      PARMS5
C                   ENDSL
 * Determine the maximum value.  Max was initialized to *LOVAL.
C     PARMS5        TAG
C                   IF        p5 > Max
C                   EVAL      Max = p5
C                   ENDIF
 *
C     PARMS4        TAG
C                   IF        p4 > Max
C                   EVAL      Max = p4
C                   ENDIF
 *
C     PARMS3        TAG
C                   IF        p3 > Max
C                   EVAL      Max = p3
C                   ENDIF
 *
C     PARMS2        TAG
C                   IF        p2 > Max
C                   EVAL      Max = p2
C                   ENDIF
C                   IF        p1 > Max
C                   EVAL      Max = p1
C                   ENDIF
C                   RETURN    Max
P MaxInt          E
```

%PARMNUM (パラメーター番号を戻す)

%PARMNUM は、パラメーター・リストにおけるパラメーターの番号を戻します。 %PARMNUM のオペランドは、プロシージャー・インターフェースの中で定義されたパラメーターの名前です。

注 :

1. *ENTRY PLIST を使用して定義されたパラメーターは、%PARMNUM のオペランドとして指定できません。
2. パラメーターは、プロシージャー・インターフェース・パラメーター・リスト内と同じように指定してください。 パラメーターが 配列の場合，指標は指定できません。 パラメーターがデータ構造の場合，サブフィールドは指定できません。 パラメーターがファイルの場合， レコード様式は指定できません。
3. プロシージャーに対して RTNPARM キーワードがコーディングされた場合、戻り値は、追加の最初のパラメーターとして処理されます。 それ以外のパラメーターの番号は、見かけ上の番号より 1 つ大きくなります。 例えば、RTNPARM を指定して定義されたプロシージャーに P1 と P2 という 2 つのパラメーターがある場合、%PARMNUM(P1) は 2 を戻し、%PARMNUM(P2) は 3 を戻します。

詳しくは、「551 ページの『組み込み関数』」を参照してください。

```
D  myProc         pi            10A   RTNPARM OPDESC
D   companyName                 25A   OPTIONS(*VARSIZE)
D   errorCode                    1A   OPTIONS(*OMIT)
D   cityName                    25A   OPTIONS(*NOPASS)
 /free

    // test the length of companyName
    callp CEEDOD(%parmnum(companyName) : more parameters ...
                 : parmlen : *omit);
    if parmlen < 25;
      // the full parameter was not passed
    endif;

    // test the presence of the omissible errorCode parameter
    callp CEETSTA(isPresent : %parmnum(errorCode) : *omit);
    if isPresent = 1;
       // errorCode was not omitted
    endif;

    // test the presence of the optional city parameter
    if %parms >= %parmnum(cityName);
       // cityName was passed
    endif;
```

図 *247. %PARMNUM* の例

%PARSER(パーサー {: オプション})

%PARSER は、DATA-INTO 命令コードの第 3 オペランドとして使用され、構文解析を行うプログラムとプロシージャー、およびパーサーによりサポートされるオプションを指定します。%PARSER は値を戻しません。また、DATA-INTO 命令コード以外の場所に指定できません。

第 1 オペランドは構文解析を行うプログラムまたはプロシージャーを指定します。 以下のいずれかになります。

- プロシージャー・ポインター式
- %PADDR 組み込み関数
- プログラムを特定する文字式です。 これは，次の形式のうちの 1 つを使用しなければなりません
  - MYPGM
  - MYLIB/MYPGM
  - *LIBL/MYPGM
- サービス・プログラム内のプロシージャーを特定する文字式です。 これは，次の形式のうちの 1 つを使用しなければなりません
  - MYSRVPGM(myProcedure)
  - MYLIB/MYSRVPGM(myProcedure)
  - *LIBL/MYSRVPGM(myProcedure)

注 : プロトタイプ名が指定された場合、それはプロシージャー・ポインター値または文字値のいずれかを返すプロシージャーであると想定されます。 パーサーがプロトタイプ・プロシージャーの場合は %PADDR 組み込み関数を使用します。

第 2 オペランドは、パーサーに直接渡されるオプションを指定します。 パーサーは、サポートするオプションの性質を判別します。

第 2 オペランドが、変更可能な変数の名前である場合、その変数のアドレスがパーサーに直接渡されます。

第 2 オペランドが、変更できない文字変数の名前を含む文字式である場合、 その式の内容を含むヌル文字終了ストリングを指すポインターがパーサーに渡されます。

パーサーは、ポインターがヌル文字終了ストリングであるかどうかを示す情報を受け取ります。

%PARSER の例

第 1 オペランドにはプログラム名が指定されています。 第 2 オペランドは省略されています。 パーサー・プログラム「MYPARSER」はオプションを受け取りません。

```
DATA-INTO myfld %DATA(document) %PARSER('MYPARSER');
```

第 1 オペランドには、サービス・プログラムのプロシージャーが指定されています。 第 2 オペランドには変更可能な変数が指定されています。 プロシージャー「myProcedure」は、「parserOptions」データ構造を指すポインターを受け取ります。

```
DCL-DS parserOptions LIKEDS(myParserOpts_T);
DATA-INTO myDs %DATA('myData.txt' : 'doc=file')
               %PARSER('MYPARSERS(myProcedure)' : parserOptions);
```

第 1 オペランドにはプロシージャー・ポインターが指定されています。 第 2 オペランドには文字式が指定されています。 プロシージャー・ポインターに指定されたプロシージャーは、値「sep=comma」を含むヌル文字終了ストリングを指すポインターを受け取ります。

```
DCL-S p POINTER(*PROC);
DCL-S sep CHAR(1) INZ(',');
DATA-INTO myds %DATA('myData.txt' : 'doc=file')
               %PARSER(p : 'sep=' + sep);
```

第 1 オペランドには組み込み関数 %PADDR が指定されています。 第 2 オペランドには変更不可能な文字変数「constParm」が指定されています。 「constParm」の値は「boolean=indicator」です。 プロトタイプ「myProc」によって指定されたプロシージャーは、第 2 オペランドとして指定された値「boolean=indicator」を含むヌル文字終了ストリングを指すポインターを受け取ります。

```
DCL-PI *N;
   constParm VARCHAR(20) CONST;
END-PI;
DATA-INTO myds %DATA('myData.txt' : 'doc=file')
               %PARSER(%PADDR(myProc) : constParm);
```

%PARSER の例および DATA-INTO 命令について詳しくは、743 ページの『DATA-INTO (文書の変数への構文解析)』 を参照してください。

パーサーの作成については、トピック 『Rational Open Access: RPG 版』を参照してください。

%PROC (現行プロシージャーの戻り値の名前 Current®)

```
%PROC()
```

%PROC は、現行プロシージャーの外部名を返します。

%PROC は、演算ステートメントでのみ指定できます。

- サイクル・メイン・プロシージャーの場合、プロシージャーの外部名は、コンパイルされたときのモジュールの名前になります。
- リニア・メイン・プロシージャーの場合、プロシージャーの外部名は、メイン・プロシージャーの大文字形式の名前になります。 参照 1 以下の例を参照してください。
- EXTPROC が指定されていないサブプロシージャーの場合、プロシージャーの外部名は、プロシージャーの大文字形式の名前になります。 参照 2 以下の例を参照してください。

- EXTPROC が指定されているサブプロシージャーの場合、プロシージャーの外部名は、EXTPROC により指定された値になります。 参照3以下の例を参照してください。
- Java プロシージャーの場合、プロシージャーの外部名は、「Java_classname_methodname」形式で指定されます。 参照4以下の例を参照してください。

```
CTL-OPT MAIN(myPgm);
DCL-S x VARCHAR(100);

DCL-PR myPgm EXTPGM('MYLIB/MYPGM345') END-PR;
DCL-PR proc1 END-PR;
DCL-PR proc2 EXTPROC('myProc2') END-PR;
DCL-PR proc3 EXTPROC(*JAVA:'MyClass':'proc3') END-PR;

DCL-PROC myPgm;  // 1
   x = %PROC();
   // x = "MYPGM"
END-PROC;

DCL-PROC proc1;  // 2
   x = %PROC();
   // x = "PROC1"
END-PROC;

DCL-PROC proc2;  // 3
   x = %PROC();
   // x = "myProc2"
END-PROC;

DCL-PROC proc3 EXPORT; // 4
   x = %PROC();
   // x = "Java_MyClass_proc3"
END-PROC;
```

図 *248. %PROC* の例

%RANGE (下限 : 上限)

%RANGE は、IN 演算子と一緒に使用されます。%RANGE は値を戻しません。また、IN 演算子の後ろ以外には指定できません。

IN 演算子を %RANGE と一緒に使用すると、第 1 オペランドが %RANGE で指定された範囲内にあるかどうかが判別されます。

%RANGE を指定した IN 演算子を使用する式は、IN 演算子の第 1 オペランドが %RANGE の第 1 オペランド以上であり、かつ %RANGE の第 2 オペランド以下の場合に真になります。

IN 演算子の第 1 オペランドに配列を指定することはできません。

%RANGE のオペランドは相互に比較可能であり、IN 演算子の第 1 オペランドに対しても比較可能である必要があります。 例えば、IN 演算子の第 1 オペランドの形式が日付である場合、%RANGE のオペランドの形式も日付である必要があります。

以下の 2 つのステートメントは同等です。

```
IF x IN %RANGE(y1 : y2);

IF x >= y1 AND x <= y2;
```

IN 演算子の第 1 オペランドの形式が英数字、グラフィック、または UCS-2 である場合は、%RANGE のオペランドも、英数字、グラフィック、または UCS-2 のいずれかの形式である必要があります。 IN 演算子の第 1 オペランドの形式が日付である場合は、%RANGE のオペランドの形式も日付である必要があります。

警告 : Unicode または ASCII での比較は EBCDIC での比較とは異なります。 249 ページの『異なるデータ・タイプまたは CCSID とデータを比較するときに生じる予期しない結果』を参照してください。

IN %RANGE のオペランドがポインターである場合は、すべてのポインターが関連する記憶域を指していない限り、結果は意味を持ちません。 以下の例では、*P1*、*P2*、および *P3* はすべて、変数 *STRING* のセクションを指すポインターです。 これらのポインターに対して %RANGE と IN 演算子を使用することには意味があります。

```
DCL-S string CHAR(1000);
DCL-S p1 POINTER;
DCL-S p2 POINTER;
DCL-S p3 POINTER;

p1 = %ADDR(string) + 10;
p2 = %ADDR(string) + 50;
p3 = %ADDR(string) + 75;

IF p2 IN %RANGE(p1 : p3);
   DSPLY ('true');
ENDIF;
```

%REALLOC (記憶域の再割り振り)

```
%REALLOC(ptr:num)
```

%REALLOC は、最初のパラメーターが指しているヒープ記憶域を変更して、2 番目のパラメーターに指定されている長さにします。 戻されたポインターによって指し示されているヒープ・ストレージには、*ptr* によって指し示されているヒープ・ストレージと同じ値があります。新しい長さが以前のものより長い場合には、追加記憶域は初期化されません。

最初のパラメーターは基底ポインター値である必要があります。 2 番目のパラメーターは、小数点以下の桁数がゼロである非浮動数値でなければなりません。 指定される長さは、1 から最大許容サイズまでの範囲でなければなりません。

最大許容サイズは、制御仕様書の ALLOC キーワードによる、RPG メモリー管理命令に使用されるヒープ記憶域のタイプによって異なります。 モジュールがテラスペース・ヒープ記憶域を使用する場合、最大許容サイズは 4294967295 バイトです。 それ以外の場合、最大許容サイズは 16776704 バイトです。

実行時に使用可能な最大サイズは、RPG の許容最大サイズより小さい場合があります。

関数は割り振られる記憶域へのポインターを戻します。 これはポインター と同じである場合も異なる場合もあります。%REALLOC 関数が正常に終了した場合には、第 1 オペランドで指定されていた元のポインター値は使用できません。

モジュールの RPG メモリー管理命令が、制御仕様書の ALLOC キーワードにより、単一レベル・ヒープ記憶域を使用している場合、 %REALLOC 組み込み関数は、単一レベル・ヒープ記憶域へのポインターのみを処理できます。 モジュールの RPG メモリー管理命令が、テラスペース・ヒープ記憶域を使用している場合、%REALLOC 組み込み関数命令は、単一レベルとテラスペースの両方のヒープ記憶域へのポインターを処理できます。

詳しくは、「581 ページの『メモリー管理命令』」を参照してください。

命令が正常に完了しなかった場合、例外 00425 または 00426 が出されます。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/FREE
  // Allocate an area of 200 bytes
  pointer = %ALLOC(200);
  // Change the size of the area to 500 bytes
  pointer = %REALLOC(pointer:500);
  // Using two different pointers:
  pointer2 = %REALLOC(pointer1:500);
  pointer1 = *NULL;;
  // The returned value was assigned to
  // "pointer2", a different variable
  // from the input pointer "pointer1".
  // In this case, the value of "pointer1"
  // is no longer valid, so "pointer1" must
  // be set to *NULL to avoid using the
  // old value.
/END-FREE
```

図 *249. %REALLOC* の例

%REM (戻り整数剰余)

```
%REM(n:m)
```

%REM は、オペランド **n** を **m** で除算した結果求められる剰余を戻します。 これらの 2 つのオペランドは、小数点以下の桁数がない (ゼロの) 数値である必要があります。 いずれかのオペランドがパック数値、ゾーン数値、または 2 進数値である場合、結果 はパック数値になります。 いずれかのオペランドが整数値である場合、結果は整数になります。 これ以外の場合、結果は符号なし数値に なります。 浮動数値オペランドは使用できません。 結果には、被除数と同じ符号が付きます。(634 ページの『%DIV (商の戻り整数部分)』も参照してください。)

%REM と %DIV には、次の関係があります。

```
%REM(A:B) = A - (%DIV(A:B) * B)
```

これらのオペランドが、8 バイト整数フィールド、または符号なしフィールド 内に収められる定数である場合、定数の折り畳みがこの組み込み関数に適用されます。 この場合、%REM 組み込み関数は定義仕様書でコーディングできます。

詳しくは、557 ページの『算術演算』または 551 ページの『組み込み関数』を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D A               S             10I 0 INZ(123)
D B               S             10I 0 INZ(27)
D DIV             S             10I 0
D REM             S             10I 0
D E               S             10I 0

 /FREE
    DIV = %DIV(A:B);  // DIV is now 4
    REM = %REM(A:B);  // REM is now 15
    E = DIV*B + REM;  // E is now 123
 /END-FREE
```

図 *250. %DIV* と *%REM* の例

%REPLACE (文字ストリングの置換)

```
%REPLACE(replacement string: source string{:start position {:source
length to replace}})
```

%REPLACE は、開始位置から開始し、指定された文字数を置換して、 置換ストリングをソース・ストリングに挿入することによって 生成される文字ストリングを戻します。

最初と 2 番目のパラメーターのタイプは文字、図形、また は UCS-2 でなければなりません。これらのパラメーターは 固定長または可変長のいずれの形式でもかまいません。 2 番目のパラメーターは、1 番目のパラメーターと同じタイプでなければなりません。

3 番目のパラメーターは置換ストリングの開始位置を、文字数で示します。 開始位置を指定しない場合、開始位置はソース・ストリングの先頭になります。 この値の範囲は、1 から ソース・ストリングの現在の長さ + 1 までです。

4 番目のパラメーターは、置換するソース・ストリング内の文字数を示します。 ゼロを指定した場合、置換ストリングは、指定した開始位置の前に挿入されます。 このパラメーターを指定しない場合、置換される文字数は、置換ストリングの長さと 同じになります。 このパラメーターの値は、ゼロ以上で、ソース・ストリングの 現在の長さ以下でなければなりません。

開始位置および長さは、小数点以下の桁数のない任意の数値または数値式とする ことができます。

ソース・ストリングまたは置換ストリングが可変長である場合、あるいは開始位置 または置換するソースの長さが変数である場合、戻り値は可変長になります。 他の場合、結果は固定長になります。

詳しくは、589 ページの『ストリング命令』または 551 ページの『組み込み関数』を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D var1            S             30A   INZ('Windsor') VARYING
D var2            S             30A   INZ('Ontario') VARYING
D var3            S             30A   INZ('Canada') VARYING
D fixed1          S             15A   INZ('California')
D date            S               D   INZ(D'1997-02-03')
D result          S            100A   VARYING

 /FREE
     result = var1 + ', ' + 'ON';
  // result = 'Windsor, ON'

  // %REPLACE with 2 parameters to replace text at begining of string:
     result = %replace ('Toronto': result);
  // result = 'Toronto, ON'

  // %REPLACE with 3 parameters to replace text at specified position:
     result = %replace (var3: result: %scan(',': result) + 2);
  // result = 'Toronto, Canada'

  // %REPLACE with 4 parameters to insert text:
     result = %replace (', ' + var2: result: %scan (',': result): 0);
  // result = 'Toronto, Ontario, Canada'

  // %REPLACE with 4 parameters to replace strings with different length
     result = %replace ('Scarborough': result:
    1: %scan (',': result) - 1);
  // result = 'Scarborough, Ontario, Canada'

  // %REPLACE with 4 parameters to delete text:
     result = %replace (': result: 1: %scan (',': result) + 1);
  // result = 'Ontario, Canada'

  // %REPLACE with 4 parameters to add text to the end of the string:
     result = %replace (', ' + %char(date): result:
   %len (result) + 1: 0);
  // result = 'Ontario, Canada, 1997-02-03'

  // %REPLACE with 3 parameters to replace fixed-length text at
  // specified position:  (fixed1 has fixed-length of 15 chars)
     result = %replace (fixed1: result: %scan (',': result) + 2);
  // result = 'Ontario, California     -03'

  // %REPLACE with 4 parameters to prefix text at beginning:
     result = %replace ('Somewhere else: ': result: 1: 0);
  // result = 'Somewhere else: Ontario, California     -03'
 /END-FREE
```

図 *251. %REPLACE* の例

%SCAN (文字の走査)

```
%SCAN(search argument : source string {: start position {: length}})
```

%SCAN は、ソース・ストリングの中の検索引数の 1 桁目を戻し、または それが見付からない場合には 0 を戻します。

開始位置と長さは、検索するソース・ストリングのサブストリングを指定します。 開始位置のデフォルトは 1 であり、長さのデフォルトはソース・ストリングの残り部分です。 結果は、開始位置が指定されている場合でも、常にソース・ストリング内の 位置です。

1 番目のパラメーターのタイプは、文字、図形、または UCS-2 でなければなりません。 2 番目 のパラメーターは、1 番目のパラメーターと同じタイプである必要があります。 3 番目と 4 番目のパラメーター (指定された場合) は、小数点以下の桁数がゼロの数値でなければなりません。

いずれかのパラメーターが可変長である場合、他のパラメーターの値は、 最大長ではなく、現在の長さに対して検査されます。

戻り値のタイプは数値です。 この組み込み関数は、数値式が有効であればどこででも使用できます。

検索引数に後書きブランクが含まれている場合、走査はそれらの後書きブランクを含めて行われます。 例えば、「b」がブランクを表すとして、%SCAN('12b':'12312b') は 4 を戻します。 後書きブランクを走査の対象から除外する場合は、検索引数で %TRIMR を使用します。 例えば、%SCAN(%TRIMR('12b'):'12312b') と指定すると 1 が戻されます。

詳しくは、589 ページの『ストリング命令』または 551 ページの『組み込み関数』を参照してください。

注 : SCAN 命令コードとは異なり、%SCAN は検索ストリングの全オカレンスを 含む配列を戻すことはできません。また、その結果は %FOUND 組み込み関数を 使用してテストすることはできません。

%SCAN の例

以下の定義について考えてみます。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D source          S             15A   inz ('Dr. Doolittle')
D pos             S              5U 0
D posTrim         S              5U 0
D posVar          S              5U 0
D srchFld         S             10A
D srchFldVar      S             10A   varying
```

```
     pos = %scan ('oo' : source);
```

前の代入後、'oo' の開始位置は 'Dr. Doolittle' 内の 6 桁目であるため、*pos* の値は 6 です。

```
     pos = %scan ('D' : source : 2);
```

前の代入後、2 桁目以降で最初に見つかった 'D' は 5 桁目にあるため、*pos* の値は 5 です。

```
     pos = %scan ('D' : source : 2 : 3);
```

前の代入後、2 桁目以降の長さ 3 の部分では 'D' が見つからないため、*pos* の値は 0 です。

```
     pos = %scan ('D' : source : 2 : 4);
```

前の代入後、2 桁目以降の長さ 4 の部分を検索すると 'D' が見つかるため、*pos* の値は 5 です。

```
     pos = %scan ('abc' : source);
```

前の代入後、'Dr. Doolittle' 内に 'abc' は見つからないため、*pos* の値は 0 です。

```
     pos = %scan ('Dr.' : source : 2);
```

前の代入後、検索が 2 桁目で開始された場合は、'Dr. Doolittle' 内で 'Dr.' は見つからないため、*pos* の値は 0 です。

```
     srchFld = 'Dr.';
     srchFldVar = 'Dr.';
     pos = %scan (srchFld : source);
     posTrim = %scan (%trimr(srchFld) : source);
     posVar = %scan (srchFldVar : source);
```

前の代入後、srchFld は 10 バイトのフィールドであり、検索引数は 'Dr.' の後に 7 個のブランクが続いたものであるため、*pos* の値は 0 です。 ただし、%TRIMR と srchFldVar のどちらの走査にも、後にブランクを伴わない 3 バイトの検索引数 'Dr.' が使用されるため、*posTrim* と *posVar* の値はいずれも 1 です。

%SCAN と %SCANR を比較する例については、684 ページの『%SCAN と %SCANR を一緒に使用する例』を参照してください。

%SCANR (文字の逆方向走査)

```
%SCANR(search argument : source string {: start position {: length}})
```

%SCANR は、ソース・ストリングの中で見つかった検索引数の最後の位置を戻します。検索引数が見つからなければ 0 を戻します。

開始位置と長さは、検索するソース・ストリングのサブストリングを指定します。 開始位置のデフォルトは 1 であり、長さのデフォルトはソース・ストリングの残り部分です。 結果は、開始位置が指定されている場合でも、常にソース・ストリング内の 位置です。

1 番目のパラメーターのタイプは、文字、図形、または UCS-2 でなければなりません。 2 番目 のパラメーターは、1 番目のパラメーターと同じタイプである必要があります。 3 番目と 4 番目のパラメーター (指定された場合) は、小数点以下の桁数がゼロの数値でなければなりません。

いずれかのパラメーターが可変長である場合、他のパラメーターの値は、 最大長ではなく、現在の長さに対して検査されます。

戻り値のタイプは数値です。 この組み込み関数は、数値式が有効であればどこででも使用できます。

検索引数に後書きブランクが含まれている場合、走査はそれらの後書きブランクを含めて行われます。 例えば、「b」がブランクを表すとして、%SCANR('12b':'12b312') は 1 を戻します。 後書きブランクを走査の対象から除外する場合は、検索引数で %TRIMR を使用します。 例えば、
%SCAN(%TRIMR('12b'):'12b312') は 5 を戻します。

詳しくは、589 ページの『ストリング命令』または 551 ページの『組み込み関数』を参照してください。

%SCANR の例

- 最も簡単な %SCANR の形式 (オペランドが 2 つの場合)

  次の例はパスの中で最後にある名前を見つけるものです。

  1. *lastSlash* の値は、これに %SCANR 組み込み関数の結果が代入されると、27 になります。
  2. *lastSlash* はゼロでないため、ファイル名を取得するために %SUBST 組み込み関数が使用されます。 コードが完了すると、*filename* の値が "a.txt" になります。

```
path = '/home/locations/manchester/a.txt';
lastSlash = %SCANR('/' : path); 1
if lastSlash = 0;
   filename = path;
else;
   filename = %SUBST(path : lastSlash + 1); 2
endif;
```

- %SCANR の第 3 オペランドを使用してソース・ストリング中の検索対象部分の開始位置を指定する

  次の例はパス内のファイル名の接尾部を見つけるものです (接尾部がある場合)。

  1. ファイル名の先頭を示す最後のスラッシュの位置を見つけるために %SCANR が使用されます。
  2. ファイル名の先頭から始めて最後のピリオドの位置を見つけるために %SCANR が使用されます。 %SCANR 組み込み関数に開始位置を指定することによって、プログラムはパス内の無関係なピリオド ("st.johns" ディレクトリー内のピリオドなど) の検索を回避します。
  3. この例では、*path* 内の最後のスラッシュ以降にはピリオドがないため、*suffix* の値は空ストリングになります。
  4. しかし、*path* の値が "/home/locations/st.johns/report.txt" であった場合、*suffix* の値は "txt" になります。

```
path = '/home/locations/st.johns/report';
p = %SCANR('/' : path); 1
if p = 0;
   p = 1; // start the next search at the beginning
endif;
dot = %SCANR('.' : path : p + 1); 2
if dot = 0;
   suffix = ''; 3
else;
     suffix = %SUBST(path : dot + 1);  4
endif;
```

- %SCANR の第 4 オペランドを使用して検索対象部分の長さを指定する

  次の例はパス内の最後にあるディレクトリーの名前を見つけるものです。

  1. 最終スラッシュがパスの 27 桁目に見つかります。
  2. 2 番目の %SCANR 組み込み関数は、スラッシュの検索を 1 桁目と 26 桁目の間でのみ行うことを示します。
  3. この例では *p* がゼロではないため、ディレクトリー名を取得するために %SUBST 組み込み関数が使用されます。 コードが終了すると、*dir* の値が "manchester" になります。

```
path = '/home/locations/manchester/a.txt';
dir = '';
p = %SCANR('/' : path);  1
if p > 0;
   p2 = %SCANR('/' : path : 1 : p - 1);
   if p2 > 0;
      dir = %SUBST(path : p2 + 1 : (p - p2 - 1));
   endif;
endif;
```

%SCAN と %SCANR を一緒に使用する例

次の例では、%SCAN と %SCANR が、同じオペランドを指定して使用されます。 %SCAN は先頭から検索し、%SCANR は末尾から検索するので、検索ストリング中に検索引数のオカレンスが複数ある場合は、これらの組み込み関数はそれぞれ異なる結果を戻します。

1. %SCAN 組み込み関数によって戻される値は 1 です。
2. %SCANR 組み込み関数によって戻される値は 24 です。
3. これらの値は同じではありません。 つまり、ストリング中に "VALUE" のオカレンスが少なくとも 2 つあるということです。

```
string = 'VALUE 9.56, VALUE 7.3, VALUE 4.71';
p1 = %SCAN ('VALUE' : string); 1
p2 = %SCANR ('VALUE' : string); 2
if p1 <> p2;
   moreThanOne = *ON; 3
endif;
```

%SCANRPL (文字の走査と置換)

```
%SCANRPL(scan string : replacement : source { : scan start  { : scan length  } )
```

%SCANRPL は、ソース・ストリングにおける走査ストリングのすべての出現を置換ストリングに置換して生成されたストリングを戻します。 走査ストリングの検索は、走査開始位置から走査の長さの分について

続けられます。 ソース・ストリングの中で、走査開始位置と走査の長さで指定された範囲以外の部分も、結果に含まれます。

最初、2 番目、3 番目のパラメーターのタイプは文字、図形、または UCS-2 でなければなりません。 これらには、固定長または可変長の形式が可能です。 これらのパラメーターはすべて、タイプと CCSID が同じでなければなりません。

4 番目のパラメーターは、走査ストリングの検索の開始位置を文字数で示します。 開始位置は、指定しないと、デフォルトの 1 になります。 この値の範囲は、1 からソース・ストリングの現在の長さまでです。

5 番目のパラメーターは、走査するソース・ストリング内の文字数を示します。 このパラメーターを指定しない場合、長さはデフォルトとして、 開始位置からの残りのソース・ストリングになります。 この値は、ゼロ以上で、開始位置からの残りのソース・ストリングの長さ以下でなければなりません。

開始位置および長さは、小数点以下の桁数のない任意の数値または数値式とする ことができます。

戻り値は、ソース・ストリングと比べて大きい場合も、等しい場合も、小さい場合もあります。 結果の長さは、走査ストリングと置換ストリングの長さや、また、置換の実行回数によっても異なります。 例えば、走査ストリングが 'a' で、置換ストリングが 'bc' であるとします。 ソース・ストリングが 'ada' の場合、戻り値の長さは 5 になります ('bcdbc')。 ソース・ストリングが 'ddd' の場合、戻り値の長さは 3 になります ('ddd')。

ソース・ストリングと置換ストリングの長さが異なる場合、またはストリングのどれかが可変長である場合、戻り値は可変長になります。 それ以外の場合は、戻り値は固定長になります。 戻り値のタイプは、ソース・ストリングと同じです。

ソース・ストリング内の各位置は、一度だけ走査されます。 例えば、走査ストリングが 'aa'、ソース・ストリングが 'baaaaac' であるとすると、2 桁目と 3 桁目が最初の一致です。 次の走査は 4 桁目から始まり、4 桁目と 5 桁目で一致が見つかります。 その次の操作は 6 桁目から始まり、それ以上一致は見つかりません。 置換ストリングが 'xy' であれば、戻り値は 'bxyxyac' になります。

ヒント: 空の置換ストリングを指定することによって、%SCANRPL を使用して、 ソース・ストリングから走査ストリングの出現を完全に除去することができます。

詳しくは、589 ページの『ストリング命令』または 551 ページの『組み込み関数』を参照してください。

```
//         ....+....1....+....2....+....3....+...string1 = 'See NAME. See NAME run. Run NAME
run.';

// 1. All occurrences of "NAME" are replaced by the
//    replacement value.  In the first case,
//    the resulting string is shorter than the source
//    string, since the replacment string is shorter
//    than the scan string. In the second case, the
//    resulting string is longer.string2 = %ScanRpl('NAME' : 'Tom' : string1);
// string2 = 'See Tom. See Tom run. Run Tom run.'string2 = %ScanRpl('NAME' : 'Jenny' :
string1);
// string2 = 'See Jenny. See Jenny run. Run Jenny run.'

// 2. All occurrences of ** are removed from the string.
//    The replacement string, '', has zero length.string3 = '*Hello**There**Everyone*';
string2 = %ScanRpl('**' : '' : string3);
// string2 = '*HelloThereEveryone*'

// 3. All occurrences of "NAME" are replaced by "Tom"
//    starting at position 6.  Since the first "N" of
//    the first "NAME" in the string is not part of the
//    source string that is scanned, the first "NAME"
//    is not considered replaceable.string2 = %ScanRpl('NAME' : 'Tom' : string1 : 6);
// string2 = 'See NAME. See Tom run. Run Tom run.'

// 4. All occurrences of "NAME" are replaced by "Tom"
//    up to length 31.  Since the final "E" of
//    the last "NAME" in the string is not part of the
//    source string that is scanned, , the final "NAME"
//    is not considered replaceable.string2 = %ScanRpl('NAME' : 'Tom' : string1 : 1 : 31);
// string2 = 'See Tom. See Tom run. Run NAME run.'

// 5. All occurrences of "NAME" are replaced by "Tom"
//    from position 10 for length 10.  Only the second
//    "NAME" value falls in that range.string2 = %ScanRpl('NAME' : 'Tom' : string1 : 10 :
10);
// string2 = 'See NAME. See Tom run. Run NAME run.'
```

図 *252. %SCANRPL* の例

%SECONDS (秒数)

```
%SECONDS(number)
```

%SECONDS は、数を、時刻またはタイム・スタンプ値に加算することができる期間に変換します。

%SECONDS は、加算式または減算式の中でプラス符号またはマイナス符号の後にのみ置くことができます。 プラス符号またはマイナス符号の前の値は時刻またはタイム・スタンプでなければなりません。 結果は、加算または減算がされた適切な秒数を持つ、時刻またはタイム・スタンプの値になります。 時刻の場合、結果の値は *ISO 形式になります。

%SECONDS をタイム・スタンプ値に加算またはタイム・スタンプ値から減算 する場合、加算または減算する秒数の小数部として、小数点以下の桁もパラメーターに指定できます。 例えば、次の例では 5.72 秒がタイム・スタンプに加算されます。

```
   timestamp2 = timestamp1 + %SECONDS(5.72);
```

日時の算術演算の例は、668 ページの図 239 を参照してください。

詳しくは、572 ページの『日付命令』または551 ページの『組み込み関数』を参照してください。

%SHTDN (シャットダウン)

```
%SHTDN
```

%SHTDN は、システム操作員がシャットダウンを要求している場合に '1' を戻します。 そうでない場合には '0' を戻します。 詳しくは、880 ページの『SHTDN (シャットダウン)』 を参照してください。

詳しくは、580 ページの『情報命令』または 551 ページの『組み込み関数』を参照してください。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/FREE
  // If the operator has requested shutdown, quit the
  // program.

  IF %SHTDN;
     QuitProgram();
  ENDIF;
/END-FREE
```

図 *253. %SHTDN* の例

%SIZE (サイズ (バイト数) の検索)

```
%SIZE(variable)
%SIZE(literal)
%SIZE(array{:*ALL})
%SIZE(table{:*ALL})
%SIZE(multiple-occurrence data structure{:*ALL})
```

%SIZE は定数またはフィールドが占めているバイト数を戻します。 この引数は、リテラル、名前付き定数、データ構造、データ構造サブフィー ルド、フィールド、配列、またはテーブル名とすることができます。 式を含めることはできませんが、一部の定数値組み込み関数および定数式は受け入れられます。 戻り値は符号なし整数形式 (タイプ U) です。

図形リテラルの場合は、サイズは、先行および後書きのシフト文字を含まない図形 文字によって占められるバイト数です。16 進数リテラルまたは UCS-2 リテラルの場合、戻されるサイズは、そのリテラル内の 16 進数の桁数の半分です。

可変長フィールドの場合、%SIZE は フィールドによって占有されている 合計バイト数 (宣言された最大長より 2 バイトまたは 4 バイト長い) を戻します。

ヌル値可能フィールドで戻される長さ (%SIZE) は、そのヌル標識の設定値とは 無関係に、常に全体の長さです。

引数が配列名、テーブル名、または複数オカレンス・データ構造名であった場合には、%SIZE に以下の考慮事項が適用されることにご注意ください。

- 戻り値は 1 つの要素またはオカレンスのサイズです。
- %SIZE の 2 番目のパラメーターとして *ALL が指定された場合には、戻り値 はすべての要素またはオカレンスが占める記憶域となります。
- データ構造のサブフィールドに必要な最大の位置合わせは、そのデータ構造の位置合わせです。 ALIGN(*FULL) が指定された場合、データ構造の各要素のサイズは、その位置合わせの倍数になります。 ALIGN がパラメーターなしで指定されている場合、または ALIGN キーワードが指定されておらず、データ構造に少なくとも 1 つのポインターが含まれている場合、データ構造に占有されるサイズは、その位置合わせの倍数よりも小さくなる可能性があります。 詳しくは、414 ページの『ALIGN{(*FULL)}』 を参照してください。
- ポインター・サブフィールドを含む複数オカレンス・データ構造またはデータ構造配列の場合には、データ構造全体に占有されるサイズ が 1 つのオカレンスにオカレンス数を乗じたサイズより大きくなることがあります。 システムではポインターを 16 バイトで位置合わせすることが必要です。すなわち、ポインターは 16 で割り切れるアドレスで記憶域に入れる必要があります。 結果として、すべてのオカレンスについてポインター・サブフィールドを記憶域内で正 しく位置決めできるように、各オカレンスの長さが正確に 16 バイトの倍数となるよう に増やさなければならないことがあります。 同様に、ALIGN キーワードが指定された場合、浮動、整数、符号なし整数の各サブフィールドは、そのサブフィールドのサイズで割り切れるアドレスでデータ構造内に位置付けられます。 データ構造全体のサイズが 1 つのオカレンスにオカレンス数を乗じたサイズと同じであるようにするには、データ構造にポインターが含まれている

場合や浮動、符号なし整数、および整数のサブフィールドを位置合わせする必要がある場合、ALIGN(*FULL) を指定します。

- より大きな配列にオーバーレイされたために、配列が不連続になっている場合、戻される値は、配列が連続している場合に戻される値と同じになります。 その値には、不連続配列の要素間の記憶域は含まれません。 不連続配列の要素間の距離を取得するには、2 番目の要素のアドレスから最初の要素のアドレスを減算します。

```
distance_between = %ADDR(elem(2)) - %ADDR(elem(1));
```

引数が複合修飾名であり、必要なサブフィールドを含むデータ構造が配列の場合、パラメーターは以下の 2 とおりの方法のいずれかで指定できます。

- 複合修飾名でデータ構造配列すべてに指標を指定。
- 複合修飾名でどのデータ構造配列にも指標を指定しない。

688 ページの『複合データ構造を使用した %SIZE の例』を参照してください。

%SIZE は、定義仕様書で使用できる数値定数および演算仕様書の拡張演算項目 2 フィールドの式のどこにでも指定することができます。

詳しくは、589 ページの『サイズ変更命令』または551 ページの『組み込み関数』を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7....+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++
D arr1            S              10    DIM(4)
D table1          S               5    DIM(20)
D field1          S              10
D field2          S               9B 0
D field3          S               5P 2
D num             S               5P 0
D mds             DS             20    occurs(10)
D mds_size        C                    const (%size (mds: *all))
D mds_ptr         DS             20    OCCURS(10)
D   pointer                        *
D vCity           S              40A   VARYING INZ('North York')
D fCity           S              40A           INZ('North York')

 /FREE
     num = %SIZE(field1);        //  10
     num = %SIZE('HH');          //   2
     num = %SIZE(123.4);         //   4
     num = %SIZE(-03.00);        //   4
     num = %SIZE(arr1);          //  10
     num = %SIZE(arr1:*ALL);     //  40
     num = %SIZE(table1);        //   5
     num = %SIZE(table1:*ALL);   // 100
     num = %SIZE(mds);           //  20
     num = %SIZE(mds:*ALL);      // 200
     num = %SIZE(mds_ptr);       //  20
     num = %SIZE(mds_ptr:*ALL);  // 320
     num = %SIZE(field2);        //   4
     num = %SIZE(field3);        //   3
     n1 = %SIZE(vCity);          //  42
     n2 = %SIZE(fCity);          //  40
 /END-FREE
```

図 *254. %SIZE* の例

ALIGN(*FULL) を指定した場合と指定しない場合のデータ構造の %SIZE の例については、415 ページの『ALIGN キーワードの影響を示す例』を参照してください。

複合データ構造を使用した %SIZE の例

以下の例では、複合修飾サブフィールド *PET* を参照する標準的な方法が次のようになっています。

```
family(x).child(y).pet
```

%SIZE のパラメーターとして指定する場合、ステートメント 1 のようにデータ構造配列にどの指標も指定しない場合、 1 または、ステートメント 2 のようにデータ構造配列にすべての指標を指定した場合のいずれかで指定できます。 2 .

```
DCL-DS family QUALIFIED DIM(3);
   name VARCHAR(25);
   numChildren INT(10);
   DCL-DS child DIM(10);
      name VARCHAR(25);
      numPets INT(10);
      pet VARCHAR(100) DIM(3);
   END-DS;
END-DS;
DCL-S x INT(10);

x = %SIZE(family.child.pet);        // 1

x = %SIZE(family(1).child(1).pet);  // 2
```

%SPLIT (ストリングをサブストリングに分割)

```
%SPLIT(string {: separators })
```

%SPLIT は、ストリングをサブストリングの配列に分割します。 これは、サブストリングの一時配列を返します。

%SPLIT は、配列を使用できる演算ステートメントで使用できます。ただし、以下は例外です。

- SORTA
- %ELEM
- %LOOKUP
- %SUBARR

第 1 オペランドは、分割されるストリングです。 これには英数字、グラフィック、または UCS-2 を指定できます。

第 2 オペランドは、各サブストリングの終わりを示す文字のリストです。 これはオプションです。 そのタイプと CCSID が第 1 オペランドと同じである必要があります。 これを指定しない場合、%SPLIT はデフォルトのブランクでの分割になります。 第 2 オペランドの長さが 1 より大きい場合、第 2 オペランドの中のいずれかの文字が、各サブストリングの終わりを示します。 例えば、%SPLIT('abc.def-ghi' : '.-') には '.' と '-' の 2 つの区切り文字があるため、('abc','def','ghi') という 3 つの要素を持つ配列を返します。

ストリングの先頭と末尾の区切り文字は無視されます。 例えば、%SPLIT('..abc.def..' : '.') は、('abc','def') という 2 つの要素を持つ配列を返します。

区切り文字の後に別の区切り文字が続く場合、それは無視されます。 例えば、%SPLIT('abc..def' : '.') は、('abc','def') という 2 つの要素を持つ配列を返します。

区切り文字オペランドの長さがゼロの場合、結果はストリング・オペランドの値を持つ単一の要素になります。 例えば、*sep* の長さがゼロの場合、%SPLIT('a.b.c' : sep) は ('a.b.c') という 1 つの要素を持つ配列を返します。

ストリングの長さがゼロの場合、またはストリング内のすべての文字が区切り文字の 1 つである場合、%SPLIT はゼロ要素を返します。

%SPLIT の例

- 以下の例では、%SPLIT に含まれるパラメーターは 1 つのみです。そのため、ストリングはブランクで分割されます。

```
DCL-S array VARCHAR(10) DIM(10);

array = %SPLIT('Monday Tuesday Wednesday');
// array(1) = "Monday"
// array(2) = "Tuesday"
// array(3) = "Wednesday"
```

- 以下の例では、%SPLIT に 2 つのパラメーターがあります。 2 番目のパラメーターには、ピリオド (.) とブランクの 2 文字があります。 2 番目のパラメーター内のいずれかの文字が検出されると、ストリングは分割されます。

```
DCL-S array VARCHAR(10) DIM(10);

array = %SPLIT('Today is Monday. Tomorrow is Tuesday.' : '. ');
// array(1) = "Today"
// array(2) = "is"
// array(3) = "Monday"
// array(4) = "Tomorrow"
// array(5) = "is"
// array(6) = "Tuesday"
```

- 以下の例では、%SPLIT によって返される一時配列の処理に、FOR-EACH 命令が使用されます。

  1. このストリングはまず、文に分割されます。この分割は、文を終了するために使用される文字で行われます。
  2. 次に、ストリングは句に分割されます。この分割は、コンマ、コロン、およびセミコロンで行われます。
  3. 次に、それぞれの句が単語に分割されます。この分割は、ブランクで行われます。

```
DCL-S sentence VARCHAR(10000);
DCL-S phrase VARCHAR(10000);
DCL-S word VARCHAR(10000);
DCL-S string VARCHAR(10000);

FOR-EACH sentence in %SPLIT(string : '.!?');    // 1
   ...
   FOR-EACH phrase in %SPLIT(sentence : ',;:'); // 2
      ...
      FOR-EACH word in %SPLIT(phrase);          // 3
         ...
      ENDFOR;
   ENDFOR;
ENDFOR;
```

- 以下の例で RPG プログラマーは、%SPLIT で余分な区切り文字が無視されないようにします。
  - ストリングには通常、「Mary,Jane,Smith」のように、コンマで区切られた 3 つの名前が含まれます。
  - ミドルネームが存在しない場合は、「Mary,,Smith」のように、2 つのコンマが一緒になります。

  1. すべてのコンマを区切り文字として使用するには、%SCANRPL 組み込み関数を使用して、「Mary*,*Jane*,*Smith」や 「Mary*,**,*Smith」のように、コンマを「*,*」に変更します。(完全なプログラムでは、「*」文字はストリング内に表示されません。)
  2. コンマは区切り文字として指定されます。
  3. %TRIM は、先頭と末尾の「*」文字を名前から削除するために使用されます。

  この例では、各ステートメントの後に続く注記に、2 つの値のステートメントの結果が示されています。

  -「Mary,Jane,Smith」

– 「Mary,,Smith」

```
DCL-S string VARCHAR(100);
DCL-S string2 VARCHAR(100);
DCL-S names VARCHAR(100) DIM(3);

string2 = %SCANRPL (',' : '*,*' : string); // 1
// string2 = "Mary*,*Jane*,*Smith"
//           "Mary*,**,*Smith"
names = %SPLIT (string2 : ',');            // 2
// names =
//     Mary* | *Jane* | *Smith
//     Mary* | **     | *Smith
names = %TRIM (names : '*');               // 3
// names =
//     Mary  | Jane   | Smith
//     Mary  |        | Smith
```

%SQRT (式の平方根)

```
%SQRT(numeric expression)
```

%SQRT 指定された数値式の平方根を戻します。 オペランドが浮動タイプの場合、結果は浮動タイプになります。 それ以外の場合は、結果はパック 10 進数値になります。 パラメーターがゼロより小さい値の場合は、例外 00101 が出されます。

詳しくは、557 ページの『算術演算』または 551 ページの『組み込み関数』を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D n               S             10I 0
D p               S              9P 2
D f               S              4F

 /FREE

   n = %SQRT(239874);
   // n = 489

   p = %SQRT(239874);
   // p = 489.76

   f = %SQRT(239874);
   // f = 489.7693
 /END-FREE
```

図 *255. %SQRT の例*

%STATUS (ファイルまたはプログラム状況の戻し)

```
%STATUS{(file_name)}
```

%STATUS は、プログラムまたはファイル状況に関して設定された最新の値を戻します。%STATUS は、プログラム状況またはいずれかのファイルの状況が変更された場合 (通常はエラーが発生した場合) に設定されます。

任意指定のファイル名パラメーターを指定しないで %STATUS を使用した場合、この 関数は、最後に変更されたプログラムまたはファイルの状況を戻します。 ファイルを 指定した場合、指定したファイルに関する INFDS *STATUS フィールド に入っている値が戻されます。 このファイルに関して INFDS を指定する必要はありません。

%STATUS は戻り値 00000 で開始し、'E' 拡張の指定がある命令が開始される 前に 00000 にリセットされます。

%STATUS の検査に最適な時期は、'E' 拡張またはエラー標識が指定されている命令の 直後あるいは INFSR または *PSSR サブルーチンの先頭です。

詳しくは、576 ページの『ファイル操作』、589 ページの『結果命令』、または 551 ページの『組み込み関数』を参照してください。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
* The 'E' extender indicates that if an error occurs, the error
* is to be handled as though an error indicator were coded.
* The success of the operation can then be checked using the
* %ERROR built-in function.  The status associated with the error
* can be checked using the %STATUS built-in function.
/FREE
   exfmt(e) InFile;
   if %error;
      exsr CheckError;
   endif;

 //----------------------------------------------------------------
 // CheckError: Subroutine to process a file I/O error
 //----------------------------------------------------------------
   begsr CheckError;
      select;
      when  %status < 01000;

         // No error occurred
      when  %status = 01211;
         // Attempted to read a file that was not open
         exsr InternalError;

      when  %status = 01331;
         // The wait time was exceeded for a READ operation
         exsr TimeOut;

      when  %status = 01261;
         // Operation to unacquired device
         exsr DeviceError;

      when  %status = 01251;
         // Permanent I/O error
         exsr PermError;

      other;
         // Some other error occurred
         exsr FileError;
      endsl;
   endsr;
/END-FREE
```

図 *256. 'E'* 拡張を使用する場合の *%STATUS* と *%ERROR*

```
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D Zero            S                 5P 0 INZ(0)
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 * %STATUS starts with a value of 0
 *
 * The following SCAN operation will cause a branch to the *PSSR
 * because the start position has a value of 0.
C     'A'           SCAN      'ABC':Zero    Pos
C     BAD_SCAN      TAG
 * The following EXFMT operation has an 'E' extender, so %STATUS will
 * be set to 0 before the operation begins.  Therefore, it is
 * valid to check %STATUS after the operation.
 * Since the 'E' extender was coded, %ERROR can also be used to
 * check if an error occurred.
C                   EXFMT(E)  REC1
C                   IF        %ERROR
C                   SELECT
C                   WHEN      %STATUS = 01255
C ...
C                   WHEN      %STATUS = 01299
C ...
 * The following scan operation has an error indicator.  %STATUS will
 * not be set to 0 before the operation begins, but %STATUS can be
 * reasonably checked if the error indicator is on.
C     'A'           SCAN      'ABC':Zero    Pos                   10
C                   IF        *IN10 AND %STATUS = 00100
C ...

 * The following scan operation does not produce an error.
 * Since there is no 'E' extender %STATUS will not be set to 0,
 * so it would return a value of 00100 from the previous error.
 * Therefore, it is unwise to use %STATUS after an operation that
 * does not have an error indicator or the 'E' extender coded since
 * you cannot be sure that the value pertains to the previous
 * operation.
C     'A'           SCAN      'ABC'         Pos
C ...
C     *PSSR         BEGSR
 * %STATUS can be used in the *PSSR since an error must have occurred.
C                   IF        %STATUS = 00100
C                   GOTO      BAD_SCAN
C ...
```

図 *257. 'E'* 拡張、エラー標識および **PSSR* を使用する場合 の *%STATUS* と *%ERROR*

%STR (ヌル文字で終了するストリングの入手または保管)

```
%STR(basing pointer{: max-length})(right-hand-side)
%STR(basing pointer : max-length)(left-hand-side)
```

%STR は、ヌル文字で終了する文字ストリングを作成または 使用するために使用します。このストリングは、C および C++ アプリケーションで 非常に一般的に使用されています。

最初のパラメーターは基底ポインター値でなければなりません。 ("%ADDR(DATA)" や "P+1" など、どのような基底ポインター式でも有効です。) 2 番目 のパラメーターが指定されている場合、小数点以下の桁数がゼロである 数値でなければなりません。 指定されない場合、文字変数定義の最大許容長がデフォルト値になります。

1 番目のパラメーターは、少なくとも 2 番目のパラメーターによって指定された 長さの記憶域を指し示している必要があります。

エラー条件:

1. 長さパラメーターが 1 未満である場合、または最大許容長よりも大きい場合には、エラーとなります。
2. ポインターが設定されていない場合、エラーとなります。
3. ポインターによってアドレスされた記憶域が、長さパラメーターによって 示された長さより短いと、次のいずれかの状況になります。
   a. エラーとなります。

b. データ破壊が起こります。

詳しくは、589 ページの『ストリング命令』または551 ページの『組み込み関数』を参照してください。

ヌル文字で終了するストリングの入手に使用する %STR

この関数は、式の右側で使用された場合、指定された長さの中の 最初のヌル文字 (x'00') までで (ただし、その文字は含まない)、最初の パラメーターによって指し示されたデータを戻します。 この組み込み関数は、文字式が有効であればどこでも使用する ことができます。 ヌル文字終了記号が指定された長さの中で見付からない場合、 実行時にエラーとはなりません。 この場合、結果の値の長さは指定された長さと同じになります。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D String1         S               *
D Fld1            S             10A

 /FREE
     Fld1 = '<' + %str(String1) + '>';
  // Assuming that String1 points to '123¬' where '¬' represents the
  // null character, after the EVAL, Fld1 = '<123>     '.
 /END-FREE
```

図 *258. %STR (右側) 例 1*

次は、2 番目のパラメーターを指定した %STR の例です。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D String1         S               *
D Fld1            S             10A

 /FREE
     Fld1 = '<' + %str(String1 : 2) + '>';
  // Assuming that String1 points to '123¬' where '¬' represents the
  // null character, after the EVAL, Fld1 = '<12>      '.
  // Since the maximum length read by the operation was 2, the '3' and
  // the '¬' were not considered.
 /END-FREE
```

図 *259. %STR (右側) 例 2*

この例で、ヌル文字終了記号は、指定された最大長の中で見付かります。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D String1         S               *
D Fld1            S             10A

 /FREE
    Fld1 = '<' + %str(String1 : 5) + '>';
    // Assuming that String1 points to '123¬' where '¬' represents the
    // null character, after the EVAL, Fld1 = '<123>     '.
    // Since the maximum length read by the operation was 5, the
    // null-terminator in position 4 was found so all the data up to
    // the null-terminator was used.
 /END-FREE
```

図 *260. %STR (右側) 例 3*

ヌル文字で終了するストリングの保管に使用する %STR

%STR(ポインター:長さ) は、式の左側で使用されると、式の右側の値を、 ポインターによって指し示されている記憶域に割り当て、最後にヌル文字 終了記号のバイトを追加します。 %STR の 2 番目のパラメータ

ーとして指定されている長さが N である場合、最後のヌル文字終了記号用に 1 バイトを予約しておく必要があるため、右側で使用可能な最大値は N-1 バイトになります。

指定可能な最大長は 65535 です。 これは、最後に ヌル文字終了記号用に 1 バイトを予約しておく必要があるので、最高でも、右側の 65534 バイトが使用可能であるということです

この長さは、ポインターが指し示す記憶域の量を示します。 また、この長さは、右側にある最大長より長くなければなりません。 ポインターは、少なくとも長さパラメーターと同じ長さの記憶域を 指し示すように設定する必要があります。 式の右側の長さが指定された長さより長い場合、その右側の値は切り捨てられます。

注 : 次の条件がいずれも真となった場合、データ破壊が起こります。

1. 長さパラメーターが、ポインターによってアドレスされたデータの実際の 長さより長い。
2. 右側の長さが、ポインターによってアドレスされたデータの実際の 長さと同じかそれより長い。

%STR が使用するための記憶域を動的に割り振っている場合、 割り振った長さを記録しておく必要があります。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D String1         S               *
D Fld1            S             10A

 /FREE
     %str(String1: 25)= 'abcdef';
     // The storage pointed at by String1 now contains 'abcdef¬'
     // Bytes 8-25 following the null-terminator are unchanged.

     %str (String1: 4) = 'abcdef';
     // The storage pointed at by String1 now contains 'abc¬'
 /END-FREE
```

図 *261. %STR (左側) の例*

%SUBARR (配列の部分の設定/入手)

```
%SUBARR(array:start-index{:number-of-elements})
```

組み込み関数 %SUBARR は、指定された配列の、開始指標 から始まるセクションを戻します。 戻される要素の数は、オプションの要素の数 パラメーターで指定します。 このパラメーターを指定しない場合、要素の数 のデフォルト値として、 配列のそれ以降の部分の要素の数が使用されます。

%SUBARR の最初のパラメーターは配列でなければなりません。 つまり、配列として定義された独立のフィールド、データ構造、またはサブフィールドを指定する必要があります。 この最初のパラメーターは、テーブル名またはプロシージャー呼び出しであってはなりません。

開始指標 パラメーターは、小数点以下の桁数がゼロである 数値でなければなりません。 浮動小数点数値は使用できません。 この値は、1 以上で、かつ配列の要素数以下でなければなりません。

オプションの要素の数 パラメーターは、小数点以下の桁数がゼロである 数値でなければなりません。 浮動小数点数値は使用できません。 この値は、1 以上で、かつ開始指標 値を適用した後で 配列内に残っている要素の数以下でなければなりません。

一般に、%SUBARR は、索引のない配列が使用できる式であれば、どのような式でも有効です。 ただし、%SUBARR を以下の用途で使用することはできません。

- 組み込み関数 %LOOKUPxx の配列引数として
- 参照によって受け渡されるパラメーターとして

%SUBARR は、以下の方法で使用することができます。

- EVAL または EVALR が使用されている代入の左側で使用する。 これにより、指定された配列内の指定された要素が変更されます。

- 代入のターゲットである配列を含む、EVAL または EVALR が使用されている代入の右側の式で使用する。これにより、配列の指定された要素の値が使用されるようになります。 配列要素が直接使用され、副配列の一時コピーは作成されません。
- SORTA 命令の拡張演算項目 2 で使用する。
- RETURN 命令の拡張演算項目 2 で使用する。
- 対応するパラメーターが配列として定義されている場合に、VALUE または読み取り専用参照 (CONST キーワード) によって受け渡される。
- %XFOOT 組み込み関数のパラメーターとして使用する。

詳しくは、561 ページの『配列命令』または 551 ページの『組み込み関数』を参照してください。

```
D a               s             10i 0 dim(5)
D b               s             10i 0 dim(15)
D resultArr       s             10i 0 dim(20)
D sum             s             20i 0
 /free
     a(1)=9;
     a(2)=5;
     a(3)=16;
     a(4)=13;
     a(5)=3;
     // Copy part of an array to another array:
     resultArr = %subarr(a:4:n);
          // this is equivalent to:
          //   resultArr(1) = a(4)
          //   resultArr(2) = a(5)
          //   ...
          //   resultArr(n) = a(4 + n - 1)

     // Copy part of an array to part of another array:
     %subarr(b:3:n) = %subarr(a:m:n);
     // Specifying the array from the start element to the end of the array
     // B has 15 elements and A has 5 elements.  Starting from element 2
     // in array A means that only 4 elements will be copied to array B.
     // The remaining elements in B will not be changed.
       b = %subarr(a : 2);

     // Sort a subset of an array:
     sorta %subarr(a:1:4);
          // Now, A=(5 9 13 16 3);
          // Since only 4 elements were sorted, the fifth element
          // is out of order.
          // Using %SUBARR in an implicit array indexing assignment
     resultArr = b + %subarr(a:2:3)
     // this is equivalent to:
     //   resultArr(1) = b(1) + a(2)
     //   resultArr(2) = b(2) + a(3)
     //   resultArr(3) = b(3) + a(4)

     // Using %SUBARR nested within an expression
     resultArr = %trim(%subst(%subarr(stringArr:i):j));
          // this is equivalent to:
          //   resultArr(1) = %trim(%subst(stringArr(i+0):j))
          //    resultArr(2) = %trim(%subst(stringArr(i+1):j))
          //   resultArr(3) = %trim(%subst(stringArr(i+2):j))

     // Sum a subset of an array
     sum = %xfoot (%subarr(a:2:3));
          // Now sum = 9 + 13 + 16 = 38
```

図 *262. %SUBARR* を使用する

```
// Using %SUBARR with dynamically allocated arrays
D dynArrInfo      ds                  qualified
D   numAlloc                    10i 0 inz(0)
D   current                     10i 0 inz(0)
D   p                             *
D dynArr          s              5a   dim(32767) based(dynArrInfo.p)
D otherArray      s              3a   dim(10) inz('xy')
 /free
     // Start the array with an allocation of five elements,
     // and with two current elements
     dynArrInfo.numAlloc = 5;
     dynArrInfo.p = %alloc(%size(dynArr) *
                    dynarrInfo.numAlloc);
     dynArrInfo.current = 2;
     // Initialize to blanks
     %subarr(dynArr : 1 : dynarrInfo.current) = *blank;

     // Set the two elements to some values
     dynArr(1) = 'Dog';
                  dynArr(2) = 'Cat';

     // Sort the two elements
     sorta %subarr(dynArr : 1 : dynarrInfo.current);
          // dynArr(1) = 'Cat'
          // dynArr(2) = 'Dog'

     // Assign another array to the two elements
     otherArray(1) = 'ab';
     otherArray(2) = 'cd';
     otherArray(3) = 'ef';
     %subarr(dynArr : 1 : dynarrInfo.current) = otherArray;
          // dynArr(1) = 'ab'
          // dynArr(2) = 'cd'

     // Changing the size of the array
     oldElems = dynArrInfo.current;
     dynArrInfo.current = 7;
     if (dynArrInfo.current > dynArrInfo.numAlloc);
          dynArrInfo.p = %realloc (dynArrInfo.p : dynArrInfo.current);
          dynArrInfo.numAlloc = dynArrInfo.current;
     endif;
     if (oldElems < dynArrInfo.current);
          // Initialize new elements to blanks
         clear %subarr(dynArr : oldElems + 1 : dynArrInfo.current - oldElems);
     endif;
```

図 *263.* 動的に割り振られる配列で *%SUBARR* を使用する

注意 : 配列の一部を同じ配列内の別の部分に割り当てるために %SUBARR を使用することができます。 ただし、その配列のソース部分が配列のターゲット部分とオーバーラップしている場合には、予測不能な結果が生じる可能性があります。

詳しくは、「551 ページの『組み込み関数』」を参照してください。

%SUBDT (日付、時刻、またはタイム・スタンプの一部の取り出し)

```
%SUBDT(value : unit { : digits { : decpos } })
```

単位に指定できるのは、 *MSECONDS、*SECONDS、*MINUTES、*HOURS、*DAYS、*MONTHS、または *YEARS です。 単位の省略形を使用して、*MS、*S、*MN、*H、*D、*M、または *Y と指定することもできます。

%SUBDT は、日付、時刻、またはタイム・スタンプの情報の一部を取り出します。 これは符号なしの数値を戻します。

最初のパラメーターは、日付、時刻、またはタイム・スタンプ値です。

2 番目のパラメーターは、ユーザーが取り出したい部分です。 有効な値は，次の通りです。

- 日付の場合: *DAYS、*MONTHS、および *YEARS
- 時刻の場合: *SECONDS、*MINUTES、および *HOURS

- タイム・スタンプの場合: *MSECONDS、*SECONDS、*MINUTES、*HOURS、 *DAYS、*MONTHS、および *YEARS
- 3 番目のパラメーターは任意指定です。 これは戻り値の桁数を表します。
- 4 番目のパラメーターは任意指定です。 これは、戻り値の小数点以下 (つまり、秒の小数部) の桁数を表します。 これを指定できるのは、最初のパラメーターがタイム・スタンプであり、2 番目 のパラメーターが *SECONDS または *S の場合です。 例えば、戻り値の小数点以下の桁数を 7 桁にしたい場合、 *digits* パラメーターに 9 を、 *decpos* パラメーターに 7 を指定します。

この関数では、*DAYS は必ず年間通算日ではなく月の中での日を参照します (ユーザーが年間通算日を使用していても)。 たとえば、2 月 10 日の日の部分とは、10 であって 41 ではありません。

この関数は必ず 4 桁の年を戻します (日付の形式が 2 桁の年の場合も)。

詳しくは、572 ページの『日付命令』または 551 ページの『組み込み関数』を参照してください。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....

     date = d'1999-02-17';
     time = t'01.23.45';
     timestamp = z'1999-02-17-01.23.45.98765';

     num = %subdt(date:*YEARS);
     // num = 1999

     num = %subdt(time:*MN);
     // num = 23

     seconds = %subdt(timestamp:*S:5:3);
     // seconds = 45.987
```

図 *264. %SUBDT* の例

%SUBST (サブストリングの検索)

```
%SUBST(string:start{:length})
```

%SUBST は引数ストリングの部分を戻します。EVAL 命令コードによる代入の結果として使用することもできます。

開始位置パラメーターはサブストリングの開始桁を表します。

長さパラメーターはサブストリングの長さを表します。 これが指定されない場合には、長さはストリング・パラメーターの長さ - 開始位 置の値 + 1 の長さになります。

このストリングは、文字データ、図形データ、また は UCS-2 データでなければなりません。 開始位置および長さは、小数点以下の桁数のない任意の数値または数値式とする ことができます。 開始位置はゼロより大きくなければなりません。 長さはゼロより大きいかまたは等しくすることができます。

ストリング・パラメーターが可変長である場合、 他のパラメーターの値は、 最大長ではなく、現在の長さに対して検査されます。

定義仕様書のキーワードのパラメーターとして指定する場合には、パラメーター はリテラルまたはリテラルを表す名前付き定数でなければなりません。 自由形式の演算仕様書に指定する場合には、パラメーターを任意の式とすることが できます。

詳しくは、589 ページの『ストリング命令』または 551 ページの『組み込み関数』を参照してください。

値として使用される %SUBST

%SUBST は、指定されたストリングの内容からのサブストリングを戻します。 この ストリングは、任意の文字、図形、または UCS-2 の フィールドまたは式でかまいません。 ストリング、開始位置、および長さとして指標のない配列を使用することができ ます。 サブストリングは、ストリング中の指定された開始位置

から始まり、指定された 長さだけ続きます。 長さが指定されていない場合には、サブストリングはストリングの終わりまで続き ます。 例えば、次のとおりです。

```
The value of  %subst('Hello World': 5+2) is  'World'
The value of  %subst('Hello World':5+2:10-7) is 'Wor'
The value of  %subst('abcd' + 'efgh':4:3) is 'def'
```

英数字データの場合、開始位置および長さはバイト数で測定されます (3 桁目は 3 番目のバイト文字であり、長さ 3 は 3 バイトが操作対象になることを示します)。

グラフィックまたは UCS-2 データの場合、開始位置および長さは 2 バイト文字長に適応する値です (3 桁目は 3 番目の 2 バイト文字であり、長さ 3 は 3 個の 2 バイト文字が操作対象になることを示します)。

CCSID 内の文字の長さがすべて同じではない、UTF-8、UTF-16、または SBCS/DBCS 混在の CCSID などの一部の CCSID では、開始位置と長さは、値の単一バイトまたは 2 バイトの数を表します。249 ページの『文字データ・タイプ』を参照してください。

699 ページの図 265 は、その値について使用される %SUBST 組み込み関数の例を示しています。

割り当ての結果として使用される %SUBST

割り当ての結果として使用された場合に、この組み込み関数は引数のストリングの一 定の部分を参照します。 開始位置および長さとして指標のない配列を使用することはできません。

結果は、変数で指定された開始位置から始まり、指定された長さだけ続きます。 長さが指定されていない場合には、ストリングの終わりまで参照されます。 長さがストリングの終わりを超えた文字を参照している場合には、実行時エラーが 出されます。

%SUBST を割り当ての結果として使用する場合には、最初のパラメーターが記憶位置 を参照していなければなりません。 すなわち、%SUBST 操作の最初のパラメーターは次の 1 つでなければなりません。

- フィールド
- データ構造
- データ構造サブフィールド
- 配列名
- 配列要素
- テーブル要素

EVAL 命令による割り当ての結果として表す場合には、%SUBST の 2 番目および 3 番目のパラメーターとして有効な任意の式を使用することができます。

```
CL0N01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++++
 *
 * In this example, CITY contains 'Toronto, Ontario'
 * %SUBST returns the value 'Ontario'.
 *
C     ' '           SCAN      CITY          C
C                   IF        %SUBST(CITY:C+1) = 'Ontario'
C                   EVAL      CITYCNT = CITYCNT+1
C                   ENDIF
 *
 * Before the EVAL, A has the value 'abcdefghijklmno'.
 * After the EVAL A has the value 'ab****ghijklmno'
 *
C                   EVAL      %SUBST(A:3:4) = '****'
```

図 *265. %SUBST* の例

%THIS (ネイティブ・メソッド用のクラス・インスタンスの戻し)

```
%THIS
```

%THIS は、ネイティブ・メソッドが呼び出されるように、クラス・インスタンスへの参照を含むオブジェクト値を戻します。%THIS は、非静的ネイティブ・メソッドでのみ有効です。 この組み込み関数は、非静的ネイティブ・メソッドからクラス・インスタンスへのアクセスを提供します。

非静的ネイティブ・メソッドは、自分のクラスの特定のインスタンスを処理します。 このオブジェクトは、Java によってパラメーターとしてネイティブ・メソッドに渡されますが、ネイティブ・メソッド用のプロトタイプやプロシージャー・インターフェースには現れません。 Java メソッドでは、このオブジェクト・インスタンスは Java の予約語 ***this*** によって参照されます。 RPG ネイティブ・メソッドにおいては、このオブジェクト・インスタンスは %THIS 組み込み関数によって参照されます。

```
 * Method "vacationDays" is a method in the class 'Employee'
D vacationDays    PR            10I 0 EXTPROC(*JAVA
D                                     : 'Employee'
D                                     : 'vacationDays')

 * Method "getId" is another method in the class 'Employee'
D getId           PR            10I 0 EXTPROC(*JAVA
D                                     : 'Employee'
D                                     : 'getId')
...
 * "vacationDays" is an RPG native method.  Since the STATIC keyword
 * is not used, it is an instance method.
P vacationDays    B                   EXPORT
D vacationDays    PI            10I 0

D id_num          S             10I 0

 * Another Employee method must be called to get the Employee's
 * id-number.  This method requires an Object of class Employee.
 * We use %THIS as the Object parameter, to get the id-number for
 * the object that our native method "vacationDays" is working on.
C                   eval      id_num = getId(%THIS)
C     id_num        chain     EMPFILE
C                   if        %found
C                   return    VACDAYS
C                   else
C                   return    -1
C                   endif

P vacationDays    E
```

図 *266. %THIS* の例

%TIME (時刻への変換)

```
%TIME{(expression{:time-format})}
```

%TIME は式の値を文字、数字、またはタイム・スタンプのデータから、時刻タイプに変換します。 変換後の値は変更されないままですが、時刻として戻されます。

最初のパラメーターは、変換される対象の値です。 値を指定しない場合、%TIME は現行システム時刻を戻します。

2 番目のパラメーターは、数値または文字の入力データの時刻の形式です。 入力データの形式にかかわらず、出力は *ISO 形式で戻されます。

使用できる入力形式については、275 ページの『時刻データ・タイプ』を参照してください。 数値または文字の入力データの時刻形式が指定されなかった場合、デフォルト形式は *ISO です。 詳しくは、「347 ページの『TIMFMT(形式 {区切り記号 })』」を参照してください。

最初のパラメーターがタイム・スタンプの場合は、2 番目のパラメーターは指定してはなりません。 システムは、この場合の入力データの形式については理解しています。

詳しくは、580 ページの『情報命令』または551 ページの『組み込み関数』を参照してください。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/FREE

   string = '12:34 PM';
   time =  %time(string:*USA);
   // time = t'12.34.00'
/END-FREE
```

図 *267. %TIME* の例

%TIMESTAMP (タイム・スタンプへの変換)

```
%TIMESTAMP{(char-num-expression { : *ISO|*ISO0 : {fractional-seconds}})}
%TIMESTAMP{(date-timestamp-expression { : fractional-seconds})}
%TIMESTAMP{(*SYS { : fractional-seconds})}
%TIMESTAMP{(*UNIQUE)}
```

%TIMESTAMP を使用して現在のシステム・タイム・スタンプを戻す

パラメーターが指定されていないか、または最初のパラメーターとして *SYS または *UNIQUE が指定されている場合、%TIMESTAMP は現在のシステム・タイム・スタンプをマイクロ秒の精度で戻します。

最初のパラメーターが *SYS の場合、2 番目のパラメーターは任意指定であり、タイム・スタンプ内の秒の小数部の桁数が戻されます。 秒の小数部の桁数は 0 から 12 までの間にすることができます。 秒の小数部のデフォルトの桁数は 6 です。

最初のパラメーターが *UNIQUE の場合、%TIMESTAMP は現在のシステム・タイム・スタンプをマイクロ秒単位の精度で戻します。 タイム・スタンプの小数部の最初の 6 桁には、タイム・スタンプのマイクロ秒の部分が設定されます。 残りの 6 桁の小数秒に値に設定され、この値によって、結果として生成されるタイム・スタンプが固有になります。 ただし、その残りの 6 桁の小数秒によって、タイム・スタンプの精度が上がることはありません。

ヒント: 固有のタイム・スタンプが 2 つの固有のタイム・スタンプの間の経過時間を判別するために使用される場合、結果をマイクロ秒の精度で計算する必要があります。

%TIMESTAMP を使用して式をタイム・スタンプに変換する

- 最初のパラメーターが文字式または数値式の場合、2 番目のパラメーターは文字データまたは数値データの形式になります。 入力データの形式にかかわらず、出力は *ISO 形式で戻されます。

  文字入力の場合、*ISO (デフォルト) または *ISO0 のいずれかを指定できます。 詳しくは、「277 ページの『タイム・スタンプ・データ・タイプ』」を参照してください。

  最初のパラメーターが数値の場合は、2 番目のパラメーターを指定する必要はありません。 唯一許される値は *ISO (デフォルトの値) のみです。

  3 番目のパラメーターは、任意指定であり、タイム・スタンプ内の秒の小数部の桁数 です。 秒の小数部の桁数は 0 から 12 までの間にすることができます。 秒の小数部のデフォルトの桁数は 6 です。
- 最初のパラメーターが日付またはタイム・スタンプ式である場合、2 番目のパラメーターは任意指定であり、戻されるタイム・スタンプ内の秒の小数部の桁数です。

  最初のオペランドが日付の場合、システムはその日付を現在の形式 から *ISO 形式に変換し、時刻 00.00.00 とゼロの小数点以下の秒を加算 します。 秒の小数部の桁数は 0 から 12 までの間にすることができます。 秒の小数部のデフォルトの桁数は 6 です。

詳しくは、580 ページの『情報命令』または551 ページの『組み込み関数』を参照してください。

%TIMESTAMP の例

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....

  string = '1960-09-29-12.34.56.000000';
  timest =  %timestamp(string);
   // timest now contains z'1960-09-29-12.34.56.000000'

  date = '2001-03-05';
  timest =  %timestamp(date);
   // timest now contains z'2001-03-05-00.00.00.000000'

  dsply (%timestamp(*SYS));
   // It displays 2014-06-27-01.02.03.421345

  dsply (%timestamp(*SYS : 1));
   // It displays 2014-06-27-01.02.03.4

  dsply (%timestamp(*SYS : 0));
   // It displays 2014-06-27-01.02.03

  dsply (%timestamp(*UNIQUE);
   // It displays 2014-06-27-01.02.03.923481000244
```

%TLOOKUPxx (テーブル要素の検索)

```
%TLOOKUP(arg : search-table {: alt-table})
%TLOOKUPLT(arg : search-table {: alt-table})
%TLOOKUPGE(arg : search-table {: alt-table})
%TLOOKUPGT(arg : search-table {: alt-table})
%TLOOKUPLE(arg : search-table {: alt-table})
```

次の関数は、次のように引数 と一致する値を求めて検索テーブル を検索します。

%TLOOKUP
正確に一致するもの。

%TLOOKUPLT
引数 に最も近いが、引数 よりも小さいもの。

%TLOOKUPLE
正確に一致しているか、または引数 に最も近いが引数 よりは小さい値。

%TLOOKUPGT
引数 に最も近いが、引数 より大きいもの。

%TLOOKUPGE
正確に一致しているか、または引数 に最も近いが引数 より大きい値。

値が指定された条件に一致する場合、検索テーブルの現在のテーブルの要素は条件を満たす要素に設定され、代替テーブルに対する現行テーブル要素が同じ要素に設定され、そして関数が値 *ON を戻します。

指定された条件に一致する値がない場合は、*OFF が戻されます。

最初の 2 つのパラメーターはどんなタイプであっても構いませんが、同じタイプでなければなりません。これらの長さまたは小数点以下の桁数は、同じである必要はありません。

引数 または検索テーブル が ALTSEQ(*NONE) を指定して定義されている場合を除き、ALTSEQ テーブルが使用されます。

組み込み関数 %FOUND と %EQUAL は %LOOKUP 命令に続けて設定されません。

注 : LOOKUP 命令コードとは異なり、%TLOOKUP はテーブルにのみ適用されます。 配列内の値を検索するには %LOOKUP 組み込み関数を使用します。

%TLOOKUPxx 組み込み関数は、 順序テーブル (ASCEND または DESCEND キーワードが指定されたテーブル) を検索するために二分探索を使用します。660 ページの『正しい順序になっていない順序配列』を参照してください。

詳しくは、561 ページの『配列命令』または 551 ページの『組み込み関数』を参照してください。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/FREE
  *IN01 = %TLOOKUP('Paris':tab1);
  IF %TLOOKUP('Thunder Bay':tab1:tab2);
  // code to handle Thunder Bay
  ENDIF;
/END-FREE
```

図 *268. %TLOOKUPxx* の例

%TRIM (両端の文字のトリミング)

```
%TRIM(string {: characters to trim})
```

1 つのパラメーターのみが指定された %TRIM は、すべての先行および後書きブランクを除去したうえで、指定されたストリングを戻します。

2 つのパラメーターが指定された %TRIM は、トリミング対象文字パラメーター で 先行および末尾の文字を除去したうえで、指定されたストリングを戻します。

このストリングは、文字、図形、または UCS-2 のどのデータでもかまいません。

トリミング対象文字 パラメーターを指定する場合には、 ストリング・パラメーターと同じタイプを指定する必要があります。

定義仕様書のキーワードのパラメーターとして指定する場合には、ストリング・ パラメーターは定数でなければなりません。

注 : 定義キーワードのパラメーターでは、2 つのパラメーターを使用して %TRIM を指定することはサポートされません。

詳しくは、589 ページの『ストリング命令』または 551 ページの『組み込み関数』を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D Location        S             16A
D FirstName       S             10A   inz ('   Chris ')
D LastName        S             10A   inz ('   Smith ')

D Name            S             20A

 * LOCATION will have the value 'Toronto, Ontario'.
 /FREE
     Location = %trim ('  Toronto, Ontario  ');

  // Name will have the value 'Chris Smith!        '.
   Name = %trim (FirstName) + ' ' + %trim (LastName) + '!';
 /END-FREE
```

図 *269. %TRIM* の例

```
D edited          S             20A   INZ('$******5.27***      ')
D trimmed         S             20A   varying
D numeric         S             15P 3
 /FREE

     // Trim '$' and '*' from the edited numeric value
     // Note: blanks will not be trimmed, since a blank
     // is not specified in the 'characters to trim' parameter

     trimmed = %trim(edited : '$*');     // trimmed is now '5.27***      '

     // Trim '$' and '*' and blank from the edited numeric value

     trimmed = %trim(edited : '$* ');     // trimmed is now '5.27'

     // Get the numeric value from the edited value

     numeric = %dec(%trim(edited : '$* ') : 31 : 9);     // numeric is now 5.27
```

図 *270.* ブランク以外の文字のトリミング

%TRIML (先行文字のトリミング)

```
%TRIML(string {: characters to trim})
```

1 つのパラメーターのみが指定された %TRIML は、すべての先行ブランクを除去したうえで、 指定されたストリングを戻します。

2 つのパラメーターが指定された %TRIML は、トリミング対象文字パラメーター で 先行文字を除去したうえで、指定されたストリングを戻します。

このストリングは、文字、図形、または UCS-2 のどのデータでもかまいません。

トリミング対象文字 パラメーターを指定する場合には、 ストリング・パラメーターと同じタイプを指定する必要があります。

定義仕様書のキーワードのパラメーターとして指定する場合には、ストリング・ パラメーターは定数でなければなりません。

注 : 定義キーワードのパラメーターでは、2 つのパラメーターを使用して %TRIML を指定することはサポートされません。

詳しくは、589 ページの『ストリング命令』または 551 ページの『組み込み関数』を参照してください。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
* LOCATION will have the value 'Toronto, Ontario  '.

/FREE
    // Trimming blanks
    Location = %triml('  Toronto, Ontario  ');
    // LOCATION now has the value 'Toronto, Ontario  '.

    // Trimming other characters

    trimmed = %triml('$******5.27***        ' : '$* ');
    // trimmed is now '5.27***        '
```

図 *271. %TRIML* の例

%TRIMR (末尾の文字のトリミング)

```
%TRIMR(string {: characters to trim})
```

1 つのパラメーターのみが指定された %TRIMR は、すべての後書きブランクを除去したうえで、 指定されたストリングを戻します。

2 つのパラメーターが指定された %TRIMR は、トリミング対象文字パラメーター で 末尾の文字を除去したうえで、指定されたストリングを戻します。

このストリングは、文字、図形、または UCS-2 のどのデータでもかまいません。

トリミング対象文字 パラメーターを指定する場合には、 ストリング・パラメーターと同じタイプを指定する必要があります。

定義仕様書のキーワードのパラメーターとして指定する場合には、ストリング・ パラメーターは定数でなければなりません。

注 : 定義キーワードのパラメーターでは、2 つのパラメーターを使用して %TRIMR を指定することはサポートされません。

詳しくは、589 ページの『ストリング命令』または 551 ページの『組み込み関数』を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D Location        S             16A   varying
D FirstName       S             10A   inz ('Chris')
D LastName        S             10A   inz ('Smith')
D Name            S             20A   varying

     // LOCATION will have the value '  Toronto, Ontario'.
     Location = %trimr ('  Toronto, Ontario  ');
     // Name will have the value 'Chris Smith:'.
     Name = %trimr (FirstName) + ' ' + %trimr (LastName) + ':';
```

図 *272. %TRIMR* の例

```
     string = '(' + %trimr('$******5.27***      ' : '$*') + ')';
     // string is now '($******5.27***      )'
     //
     // Nothing has been trimmed from the right-hand side because
     // the right-most character is a blank, and a blank does not
     // appear in the 'characters to trim' parameter

     string = '(' + %trimr('$******5.27***      ' : '$ *') + ')';
     // string is now '($******5.27)'
```

図 *273.* ブランク以外の文字のトリミング

%UCS2 (UCS-2 値への変換)

%UCS2 は、文字、図形、または UCS-2 から式の値を変換し、UCS-2 値を 戻します。 結果は、パラメーターが可変長である場合か、パラメーター が 1 バイト文字である場合は、可変長になります。

2 番目のパラメーターの *ccsid* はオプション で、結果の式の CCSID を示します。 CCSID は、制御キーワード CCSID(*UCS2) で 指定されるように、モジュールのデフォルト UCS-2 CCSID にデフォルト設定されます。

パラメーターが定数の場合、変換はコンパイル時に行われます。

変換の結果、置換文字になる場合、コンパイル時に警告メッセージ が出されます。 実行時には状況 00050 が設定され、エラー・メッセージは出されません。

詳しくは、569 ページの『変換命令』または 551 ページの『組み込み関数』を参照してください。

```
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H CCSID(*UCS2 : 13488)
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D  char           S              5A   INZ('abcde')
D  graph          S              2G   INZ(G'oAABBi')
 * The %UCS2 built-in function is used to initialize a UCS-2 field.
D  ufield         S             10C   INZ(%UCS2('abcdefghij'))
D  ufield2        S              1C   CCSID(61952) INZ(*LOVAL)
D  isLess                        1N
D  proc           PR
D    uparm                       2G   CCSID(13488) CONST
CL0N01Factor1+++++++Opcode&ExtExtended-factor2++++++++++++++++++++++++++
C                   EVAL      ufield = %UCS2(char) +  %UCS2(graph)
 * ufield now has 7 UCS-2 characters representing
 * 'a.b.c.d.e.AABB' where 'x.' represents the UCS-2 form of 'x'
C                   EVAL      isLess = ufield < %UCS2(ufield2:13488)
 * The result of the %UCS2 built-in function is the value of
 * ufield2, converted from CCSID 61952 to CCSID 13488
 * for the comparison.

C                   EVAL      ufield = ufield2
 * The value of ufield2 is converted from CCSID 61952 to
 * CCSID 13488 and stored in ufield.
 * This conversion is handled implicitly by the compiler.

C                   CALLP     proc(ufield2)
 * The value of ufield2 is converted to CCSID 13488
 * implicitly, as part of passing the parameter by constant reference.
```

注 : この例の GRAPHIC リテラルは、無効な GRAPHIC リテラルです。 詳しくは、252 ページの『グラフィック形式』 を参照してください。

図 *274. %UCS2* の例

%UNS (符号なし形式への変換)

```
%UNS(numeric or character expression)
```

%UNS は式の値を符号なし形式に変換します。 10 進数はすべて切り捨てられます。%UNS を使用すると、浮動値または 10 進数値から小数点以下の桁を切り捨てて、その値を配列指標として使用することができます。

パラメーターが文字式である場合は、次の規則が適用されます。

- %DEC の文字式の規則については、569 ページの『組み込み関数を使用して文字値を数値に変換するための規則』 を参照してください。
- 浮動小数点データ (例えば、'1.2E6') は使用できません。
- 浮動小数点データは使用できません。 つまり、数値の後に E と指数 ('1.2E6' など) が存在するものは使用できません。
- 無効な数値データが検出された場合、例外が発生し、状況コード 105 が戻されます。

詳しくは、569 ページの『変換命令』または 551 ページの『組み込み関数』を参照してください。

707 ページの図 275 は %UNS 組み込み関数の例を示しています。

%UNSH (四捨五入を伴う符号なし形式への変換)

```
%UNSH(numeric or character expression)
```

%UNSH は %UNS と同じですが、式が 10 進数値、浮動値、または文字値の場合には、整数タイプへの変換時に、 式の値に四捨五入が適用されるという点が異なります。 四捨五入が実行できない場合、メッセージは出されません。

詳しくは、569 ページの『変換命令』または 551 ページの『組み込み関数』を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++
D p7              s              7p 3 inz (8236.567)
D s9              s              9s 5 inz (23.73442)
D f8              s              8f   inz (173.789)
D c15a            s             15a   inz (' 12345.6789 +')
D c15b            s             15a   inz (' + 5 , 6 7 ')
D result1         s             15p 5
D result2         s             15p 5
D result3         s             15p 5
D array           s              1a   dim (200)
D a               s              1a

 /FREE
 // using numeric parameters
    result1 = %uns (p7) + 0.1234; // "result1" is now 8236.12340
    result2 = %uns (s9);          // "result2" is now   23.00000
    result3 = %unsh (f8);         // "result3" is now  174.00000
 // using character parameters
     result1 = %uns (c15a);        // "result1" is now 12345.0000
     result2 = %unsh (c15b);       // "result2" is now     6.00000
    // %UNS and %UNSH can be used as array indexes
    a = array (%unsh (f8));
 /END-FREE
```

図 *275. %UNS および %UNSH の例*

%UPPER (大文字に変換)

```
%UPPER(string {: start { : length } })
```

%UPPER は、ストリング・オペランドを大文字に変換します。

%LOWER および %UPPER について詳しくは、660 ページの『%LOWER および %UPPER (小文字または大文字に変換)』 を参照してください。

%XFOOT (配列式要素の合計)

```
%XFOOT(array-expression)
```

%XFOOT によって、指定された数値配列式のすべての要素の合計が求められます。

結果の精度は、全配列要素を合計した結果を保持できる最小のものであり、最高で 63 桁です。 結果の中の小数部の数は、常に配列式の小数部と同じです。

たとえば、ARR が精度 (17,4) の 500 の要素の配列である 場合、%XFOOT(ARR) の結果は (20,4) です。

%XFOOT(X) の X の精度が (m,n) である場合、次の表は、X の要素の数に 基づく結果の精度を示しています。

```
Elements of X      Precision of %XFOOT(X)
1                  (m,n)
2-10               (m+1,n)
11-100             (m+2,n)
101-1000           (m+3,n)
1001-10000         (m+4,n)
10001-32767        (m+5,n)
```

規則は、配列式の通常の規則が適用されます。 たとえ ば、ARR1 に 10 個の要素があり、ARR2 に 20 個の要素がある 場合、%XFOOT(ARR1+ARR2) の結果は、ARR1+ARR2 の最初の 10 個の要素の合計です。

組み込み関数は XFOOT 命令と同様ですが、浮動配列 が、他のすべてのタイプのように、索引 1 から始まって合計される点は異なります。

詳しくは、561 ページの『配列命令』または 551 ページの『組み込み関数』を参照してください。

%XLATE (変換)

```
%XLATE(from:to:string{:startpos})
```

%XLATE は、ストリング を変換元、変換先、および開始位置 の値にしたがって変換します。

最初のパラメーターには置換される対象の文字のリストが入り、2 番目のパラメーターには置き換わる文字が入ります。 たとえば、ストリングが変換元 の 3 番目の文字を含んでいる場合、この文字のオカレンスごとに 変換先 の 3 番目の文字で置き換えられます。

3 番目のパラメーターは、変換の対象となるストリングです。 4 番目のパラメーターは、変換が開始される位置です。 デフォルトでは、変換は 1 桁目から開始されます。

最初のパラメーターが 2 番目のパラメーターより長い場合、 最初のパラメーターにある余分な文字は無視されます。

最初の 3 つのパラメーターのタイプは、文字、図形、または UCS-2 が可能です。 3 つすべてが同じタイプでなければなりません。 戻り値は、ストリング と同じタイプ、同じ長さになります。

4 番目のパラメーターは、小数点以下の桁数がゼロである非浮動数値でなければなりません。

詳しくは、589 ページの『ストリング命令』または 551 ページの『組み込み関数』を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D up              C                   'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
D lo              C                   'abcdefghijklmnopqrstuvwxyz'
D string          S             10A   inz('rpg dept')

 /FREE

   string = %XLATE(lo:up:'rpg dept');
   // string now contains 'RPG DEPT'

   string = %XLATE(up:lo:'RPG DEPT':6);
   // string now contains 'RPG Dept'
 /END-FREE
```

図 *276. %XLATE* の例

%XML (xmlDocument {:options})

%XML は、構文解析する XML 文書、および文書の構文解析方法を制御するオプションを指定する XML-SAX 命令コードおよび XML-INTO 命令コードの第 2 オペランドとして使用します。%XML は値を戻しません。また、XML-SAX 命令コードおよび XML-INTO 命令コード以外の場所に指定できません。

第 1 オペランドは、構文解析する文書を指定します。 このオペランドは、定数あるいは変数文字、または XML 文書あるいは XML 文書を含むファイル名のいずれかを持つ UCS-2 表記の場合があります。

第 2 オペランドは、XML 文書が解釈され、構文解析される方法を制御するオプションを指定します。 このオペランドは、定数または変数文字式の場合があります。 文字式の値は、次の形式で指定されたゼロ個以上のオプションのリストです。

```
    optionname1=value1 optionname2=value2
```

オプション名および等号の間、または等号および値の間には、スペースは許可されていません。 ただし、オプションの前後、および間には、任意の数のスペースを指定できます。 オプションは、大/小文字で指定できます。 以下は、XML-INTO の「doc=file」および「allowextra=yes」オプションを指定する有効なすべての方法です。

```
   'doc=file allowextra=yes'
   '        doc=file     allowextra=yes     '
   'ALLOWEXTRA=YES DOC=FILE     '
   'AllowExtra=Yes Doc=File     '
```

以下は**無効な**オプション・ストリングです。

オプション・ストリング	オプション・ストリングの問題
'doc = file'	等号の周辺のスペースは許可されていません。
'allowextra'	それぞれのオプションには、等号および値が必要です。
'badopt=yes'	有効なオプションのみ許可されています。
'allowextra=ok'	「allowextra」値は「yes」または「no」のみです。

有効なオプションおよび値は、%XML 組み込み関数の内容によって異なります。 有効なオプションおよび値の完全なリストについては、947 ページの『XML-SAX (XML 文書の構文解析)』 および 908 ページの『XML-INTO (XML 文書の変数への構文解析)』 を参照してください。

1 つのオプションが複数回指定されていた場合は、最後に指定されている値が使用されます。 例えば、オプション・パラメーターの値が次のように指定されているとします。

```
    'doc=file doc=string'
```

この場合、パーサーは「doc」オプションに値「string」を使用します。

パーサーが無効なオプションまたは無効な値を発見した場合は、命令は状況コード 00352 で失敗します。

```
// The "options" parameter is omitted.  Default values are used for
// all options.  Since the default value for the "doc" option is
// always "string", the parser will correctly assume that the first
// parameter contains an XML document.
xmldocument = '<myfld>new value</myfld>';
XML-INTO myfld %XML(xmldocument);

// The "options" parameter is specified as a literal with two options.
XML-INTO myds %XML(xmldocument : 'allowmissing=yes allowextra=yes');

// The "options" parameter is specified as a variable expression
// with two options.
ccsidOpt = 'ccsid=' + %char(ccsid);
XML-SAX %HANDLER(mySaxHandler : myCommArea)
        %XML('myinfo.xml' : 'doc=file ' + ccsidOpt);
```

図 *277. %XML の例*

その他の %XML の例については、947 ページの『XML-SAX (XML 文書の構文解析)』 および 908 ページの『XML-INTO (XML 文書の変数への構文解析)』 を参照してください。

詳しくは、596 ページの『XML 命令』 または 551 ページの『組み込み関数』 を参照してください。

%YEARS (年数)

```
%YEARS(number)
```

%YEARS は、数を、日付またはタイム・スタンプ値に加算することができる期間に変換します。

%YEARS は、加算式または減算式の中でプラス符号またはマイナス符号の後にのみ置くことができます。 プラス符号またはマイナス符号の前の値は日付またはタイム・スタンプでなければなりません。 結果は、加算または減算がされた適切な年数を持つ、日付またはタイム・スタンプの値になります。 日付の場合、結果の値は *ISO 形式になります。

日付値またはタイム・スタンプ値によって表された日付が 2 月 29 日で、結果の年が閏年でない場合は、代わりに 2 月 28 日が使用されます。 2 月 29 日の日付に対する年数の加算または減算は、元に戻せない場合があります。 たとえば、2000-02-29 + %YEARS(1) - %YEARS(1) は 2000-02-28 になります。

%YEARS 組み込み関数の例については、668 ページの図 239 を参照してください。

詳しくは、572 ページの『日付命令』、551 ページの『組み込み関数』、および 574 ページの『予期しない結果』を参照してください。

命令コード

この章では、それぞれの命令コードをアルファベット順に説明します。

ACQ (獲得)

自由形式構文	ACQ{(E)} 装置名 ワークステーション・ファイル

コード	演算項目 1	演算項目 2	結果フィールド	標識		
ACQ (E)	装置名	ワークステーション・ファイル		–	ER	–

ACQ 命令は、ワークステーション・ファイル に指定されたワークステーション・ ファイルに対して、装置名 に指定されたプログラム装置を獲得します。 この装置が使用可能な場合には、ACQ は装置をファイルに接続します。 使用可能でないかまたはすでにファイルに接続されている場合には、エラーが 起こります。

ACQ 例外 (ファイル状況コードが 1000 より大きい) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、両方を指定することはできません。 エラー標識または 'E' 拡張が指定されないで、INFSR サブルーチンが指定されている場合には、エラー/例外が起こる と、INFSR に制御が渡されます。 エラー標識、'E' 拡張、INFSR サブルーチンの いずれも指定されていない場合には、エラー/例外が起こると デフォルトのエラー/例外処理プログラムが制御を受け取ります。エラー処理について詳しくは、146 ページの『ファイル例外/エラー』を参照してください。

ACQ 命令の処理時には入出力操作は行われません。 ACQ は複数の装置ファイルによって使用されるか、またはエラー回復のために、 単一の装置ファイルによって使用することができます。 1 つのプログラムが装置を獲得しますが、その装置をファイルを 共用する呼び出されたすべてのプログラムで使用できるようにします。またそのプログラムは、呼び出されたプログラムが その装置を解放することを許可します。「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」のワークステーション・ファイルの使用法についての章の「複数の装置ファイル」のセクションを参照してください。

詳しくは、「576 ページの『ファイル操作』」を参照してください。

ADD (加算)

自由形式構文	(許可されていない - + または += 演算子を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
ADD (H)	加数	加数	和	+	-	Z

演算項目 1 が指定されている場合には、ADD 命令は、演算項目 1 と演算項目 2 を加算して、その合計を結果フィールドに入れます。 演算項目 1 が指定さ れていない場合には、演算項目 2 の内容が結果フィールドに加えられてその 合計が結果フィールドに入れられます。 演算項目 1 と演算項目 2 は数値でなければならず、配列、配列要素、定数、フ ィールド名、リテラル、サブフィールド、またはテーブル名の 1 つを入れるこ とができます。ADD 命令を指定する場合の規則については、557 ページの『算術演算』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 * The value 1 is added to RECNO.
C                   ADD       1             RECNO
 * The contents of EHWRK are added to CURHRS.
C                   ADD       EHWRK         CURHRS
 * The contents of OVRTM and REGHRS are added together and
 * placed in TOTPAY.
C     OVRTM         ADD       REGHRS        TOTPAY
```

図 *278. ADD* 命令

ADDDUR (期間の加算)

自由形式構文	(許可されていない - %YEARS および %MONTHS などの 期間関数では + または += 演算子を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
ADDDUR (E)	日付/時刻	期間: 期間コード	日付/時刻	_	ER	_

ADDDUR 命令は演算項目 2 に指定された期間を日付または時刻に加算し、結 果の日付、時刻、またはタイム・スタンプを結果フィールドに入れます。

演算項目 1 は任意指定で、日付、時刻、またはタイム・スタンプ・フィールド、 サブフィールド、配列、配列要素、リテラル、または定数を入れることが できます。 演算項目 1 にフィールド名、配列、または配列要素が入っている場合には、 そのデータ・タイプは結果フィールドに指定されたフィールドと同じデータ ・タイプでなければなりません。 演算項目 1 が指定されていない場合には、期間は結果フィールドに指定され たフィールドに加えられます。

演算項目 2 は必須で、2 つの副演算項目が入ります。 1 番目は期間で、数値フィールド、配列要素、または小数点以下の桁数がゼロ の定数になります。 期間が負の場合には、その値が日付から引かれます。 2 番目の副演算項目は、期間のタイプを示す有効な期間コードでなければなりません。 期間コードは、結果フィールドのデータ・タイプと同じでなければなりません。 年、 月、または日の期間は日付フィールド加算できますが、時間の分の期間は 加算できません。期間コードとその短縮形のリストについては、572 ページの『日付命令』を参照してください。

結果フィールドは、日付、時刻、またはタイム・スタンプ・データ・タイプ・フ ィールド、配列、または配列要素でなければなりません。 演算項目 1 がブランクの場合には、期間は結果フィールドの値に加えられま す。 結果フィールドが配列の場合には、演算項目 2 の値が配列のそれぞれの要素 に加算されます。 結果フィールドが時刻フィールドの場合には、結果は常に有効な時刻となりま す。 例えば、23:59:59 に 59 分を加えると 24:58:59 になります。 この時刻は有効でないので、コンパイラーはこれを 00:58:59 に調整します。

月の期間を日付に加算する場合の一般的な規則では、月の部分が期間の月数だ け増やされて、日の部分は変わりません。 この例外は、結果の日の部分が結果の月の実際の日数を超える場合です。 この場合には、結果の日の部分が実際の月の最終日付に合わせて調整されます。 次の例 (*YMD 形式と想定している) はこの点について示しています。

- `'98/05/30' ADDDUR 1:*MONTHS` は '98/06/30' になる。

  結果の月の部分が 1 だけ大きくなって、日の部分は変わりません。
- `'98/05/31' ADDDUR 1:*MONTHS` は '98/06/30' になる。

  結果の月の部分が 1 大きくなって (6 月は 30 日しかないので) 結果の 日の部分が調整されています。

年の期間を加算する場合にも同様の結果になります。 例えば、'92/02/29' に 1 年を加えると (結果の年は閏年ではないので) 調 整された値の '93/02/28' になります。

月の期間と年の期間の加算について詳しくは、574 ページの『予期しない結果』を参照してください。

次のいずれかの場合にはエラー状態になります。

- 演算項目 1 の日付、時刻、またはタイム・スタンプ・フィールドの値が正しく ない
- 演算項目 1 がブランクで、 演算の前の結果フィールドの値が正しくない
- オーバーフローまたはアンダーフローが起こった (すなわち、結果 の値が *HIVAL より大きいか、または *LOVAL より小さい)

エラー状態では次のことが行われます。

- エラー (状況コード 112 または 113) が出されます。
- エラー標識 (73-74 桁目) が指定されている場合には、この標識がオンに 設定され、'E' 拡張が指定されている場合には、%ERROR 組み込み関数が '1' を 戻すように設定されます。
- 結果フィールドの値は変わりません。

プログラム状況コードが 112 または 113 である 例外を処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、163 ページの『プログラム例外/エラー』を参照してください。

注 : システムは期間を 15 桁に制限します。 有効数字が 15 桁を超える期間を加え るとエラーまたは切り捨ての原因になります。 これらの問題は、演算項目 2 の最初の副演算項目を 15 桁に制限することによって回避することができます。

日付/時刻フィールドの処理について詳しくは、572 ページの『日付命令』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
H TIMFMT(*USA) DATFMT(*MDY&)
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
 *
DDateconst        C                   CONST(D'12 31 92')
 *
 * Define a Date field and initialize
 *
DLoandate         S               D   DATFMT(*EUR) INZ(D'12 31 92')
DDuedate          S               D   DATFMT(*ISO)
Dtimestamp        S               Z
Danswer           S               T
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 * Determine a DUEDATE which is xx years, yy months, zz days later
 * than LOANDATE.
C     LOANDATE      ADDDUR    XX:*YEARS     DUEDATE
C                   ADDDUR    YY:*MONTHS    DUEDATE
C                   ADDDUR    ZZ:*DAYS      DUEDATE
 * Determine the date 23 days later
 *
C                   ADDDUR    23:*D         DUEDATE
 * Add a 1234 microseconds to a timestamp
 *
C                   ADDDUR    1234:*MS      timestamp
 * Add 12 HRS and 16 minutes to midnight
 *
C     T'00:00 am'   ADDDUR    12:*Hours     answer
C                   ADDDUR    16:*Minutes   answer
 * Subtract 30 days from a loan due date
 *
C                   ADDDUR    -30:*D        LOANDUE
```

図 *279. ADDDUR* 命令

ALLOC (記憶域の割り振り)

自由形式構文	(許可されていない - %ALLOC 組み込み関数を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
ALLOC (E)		長さ	ポインター	_	ER	_

ALLOC 命令は、演算項目 2 に指定された長さのデフォルト・ヒープの記憶域を割り振ります。 結果のフィールド・ポインターは、新しいヒープ記憶域を指し示すように設定されます。 この記憶域は初期化されません。

演算項目 2 は小数点以下の桁がない数値でなければなりません。 これは、リテラル、定数、独立フィールド、サブフィールド、 テーブル名または配列要素のいずれかにすることができます。 値は、1 から最大サポート・サイズまでの範囲でなければなりません。 実行時に この値が範囲外になると、エラーとなり、状況 425 が出されます。 その 記憶域を割り振ることができない場合、エラーとなり、状況 426 が 出されます。 このようなエラーが発生した場合、結果フィールド・ポインターは 変更されません。

最大許容サイズは、制御仕様書の ALLOC キーワードによる、メモリー管理命令に使用されるヒープ記憶域のタイプによって異なります。 モジュールがメモリー管理命令にテラスペース記憶域モデルを使用することがコンパイル時に分かっている場合、 許容最大サイズは 4294967295 バイトです。 それ以外の場合、最大許容サイズは 16776704 バイトです。

実行時に使用可能な最大サイズは、RPG の許容最大サイズより小さい場合があります。

結果フィールドは基底ポインター・スカラー変数 (独立フィールド、データ構造サブフィールド、テーブル名、または配列要素) でなければなりません。

プログラム状況コードが 425 または 426 である 例外を処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、163 ページの『プログラム例外/エラー』を参照してください。

詳しくは、「581 ページの『メモリー管理命令』」を参照してください。

```
D Ptr1            S               *
D Ptr2            S               *
C                   ALLOC     7                Ptr1
 * Now Ptr1 points to 7 bytes of storage
 *
C                   ALLOC (E) 12345678         Ptr2
 * This is a large amount of storage, and sometimes it may
 * be unavailable.  If the storage could not be allocated,
 * %ERROR will return '1', the status is set to 00426, and
 * %STATUS will return 00426.
```

図 *280. ALLOC* 命令

ANDxx (かつ)

自由形式構文	(許可されていない - AND 演算子を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
ANDxx	被比較値	被比較値				

この命令は、ANDxx、DOUxx、DOWxx、IFxx、ORxx、 または WHENxx 命令の直後に 続けられなければなりません。 ANDxx によって、DOUxx、DOWxx、IFxx、および WHENxx 命令と複合条件を指 定することができます。 ANDxx 命令の方が ORxx 命令よりも優先順位が高くなります。

制御レベル項目 (7 から 8 桁目) はブランクにするか、 あるいは L1 から L9 標識、LR 標識、または L0 項目を入れてプログラムの該当する セクション内のステートメントをグループにまとめることができます。 制御レベルの指定は、対応する DOUxx、DOWxx、IFxx、または WHENxx 命令の制 御レベルの指定と同じでなければなりません。 条件付け標識の指定 (9 から 11 桁目) は使用できません。

演算項目 1 と演算項目 2 には、リテラル、名前のついた定数、形象定数、 テーブル名、配列要素、データ構造名、またはフィールド名を入れなければな りません。 演算項目 1 と 2 は同じタイプでなければなりません。 例えば、 文字フィールドを数値と比較することはできません。 演算項目 1 と 2 の比較は、 比較命令の場合と同じ規則に従って行われます。567 ページの『比較命令』を参照してください。

詳しくは、「591 ページの『構造化プログラミング命令』」を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 * If ACODE is equal to A and indicator 50 is on, the MOVE
 * and WRITE operations are processed.
C     ACODE         IFEQ      'A'
C     *IN50         ANDEQ     *ON
C                   MOVE      'A'           ACREC
C                   WRITE     RCRSN
 * If the previous conditions were not met but ACODE is equal
 * to A, indicator 50 is off, and ACREC is equal to D, the
 * following MOVE operation is processed.
C                   ELSE
C     ACODE         IFEQ      'A'
C     *IN50         ANDEQ     *OFF
C     ACREC         ANDEQ     'D'
C                   MOVE      'A'           ACREC
C                   ENDIF
C                   ENDIF
```

図 *281. ANDxx* 命令

BEGSR (サブルーチンの開始)

自由形式構文	BEGSR サブルーチン名

コード	演算項目 1	演算項目 2	結果フィールド	標識		
BEGSR	サブルーチン名					

BEGSR 命令は RPG IV サブルーチンの開始を識別します。サブルーチン名 にはサブルーチン名が入ります。 サブルーチン を参照している EXSR 命令のサブルーチン名 と同じ名前を、サブルーチンを参照している CASxx 命令の結果フィールドに、または、 (サブルーチンがファイル・エラー処理サブルーチンの場合は) INFSR ファイル 仕様キーワードの記入項目に指定できます。 制御レベル項目 (7 から 8 桁目) は SR またはブランクにすることができます。 条件付け標識は指定することができません。

すべてのサブルーチンには固有の記号名が必要です。 演算項目 1 に使用されたキーワード *PSSR は、それがプログラム検出の例外/エラーを処理するプログラム例外/エラー処理サブルーチンであることを示します。 このキーワードで定義できるサブルーチンは 1 つのみです。演算項目 1 の *INZSR は、初期化ステップで実行するサブルーチンを指定します。 *INZSR に定義できるサブルーチンは 1 つだけです。

サブルーチンのコーディングの例については 595 ページの図 192 を、サブルーチン命令の概要については 594 ページの『サブルーチン命令』を参照してください。

BITOFF (ビットをオフに設定)

自由形式構文	(許可されていない - %BITAND および %BITNOT 組み込み関数を使用。620 ページの図 205 を参照。)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
BITOFF		ビット数	文字フィールド			

BITOFF 命令では、演算項目 2 で識別されたビットが結果フィールドでオ フに設定 (0 に設定) されます。 演算項目 2 で識別されないビットは変わりま せん。 したがって、BITOFF を使用して文字を形式設定する場合には、BITON と BITOFF の両方を使用しなければなりません。ビットをオン (=1) に設定する ように指定するためには BITON、ビットをオフ (=0) に設定するように指定 するためには BITOFF を使用します。 文字のすべてのビットをオンまたはオフに明示的に設定しない限り、希望す る文字を取り出すことはできません。

文字フィールドに特定のビット・パターンを割り当てる場合には、演算項目 2 に 16 進数リテラルを指定して 803 ページの『MOVE (転送)』 命令を使用します。

演算項目 2 には以下のものを入れることができます。

- ビット番号 *0* から *7:* 1 つの命令で 1-8 ビットをオフに 設定することができます。 これらは 0 から 7 の番号で識別されます (0 が左端 ビットです)。 ビット番号をアポストロフィで囲みます。 例えば、ビット 0、2、および 5 をオフに設定するためには、 演算項目 2 に '025' と指定します。
- フィールド名: 演算項目 2 に、1 桁の文字フィールドの名前、テーブル要素、または配列要素を指定することができます。 このフィールド、テーブル要素、または配列要素でオンのビットは結果フィールドでオフに設定されます。オフのビットは結果に影響を与えません。
- *16* 進数リテラルまたは名前のついた定数*:* 1 バイトの 16 進数リテラルまたは 16 進数の名前のついた定数を指定する ことができます。 演算項目 2 のオンのビットは結果フィールドでオフに設定され、オフのビ ットは影響を受けません。
- 名前のついた定数*:* オフに設定するビット番号 が入った 8 桁までの長さの文字の名前のついた定数。

結果フィールドには、1 桁の文字フィールドを指定します。 配列のそれぞれの要素が 1 桁の文字フィールドの場合には、配列要素にするこ とができます。

詳しくは、「562 ページの『ビット操作』」を参照してください。

```
 *    Set off bits 0,4,6 in  FieldG.  Leave bits 1,2,3,5,7 unchanged.
 *    Setting off bit 0, which is already off, results in bit 0 remaining off.
 *       Factor 2 = 10001010
 *       FieldG   = 01001111  (before)
 *       FieldG   = 01000101  (after)
C                   BITOFF    '046'          FieldG
 *    Set off bits 0,2,4,6 in FieldI.  Leave bits 1,3,5,7 unchanged.
 *    Setting off bit 2, which is already off, results in bit 2 remaining off.
 *       Factor 2 = 10101010
 *       FieldI   = 11001110  (before)
 *       FieldI   = 01000100  (after)
C                   BITOFF    BITNC          FieldI
 *    HEXNC is equivalent to literal '4567', bit pattern 00001111.
 *    Set off bits 4,5,6,7 in  FieldK.  Leave bits 0,1,2,3 unchanged.
 *       Factor 2 = 11110000
 *       FieldK   = 10000000  (before)
 *       FieldK   = 00000000  (after)
C                   BITOFF    HEXNC2         FieldK
C                   RETURN
```

図 *282. BITOFF* の例

BITON (ビットをオンに設定)

自由形式構文	(許可されていない - %BITOR 組み込み関数を使用。620 ページの図 205 を参照。)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
BITON		ビット数	文字フィールド			

BITON 命令では、演算項目 2 で識別されたビットが結果フィールドでオンに 設定 (1 に設定) されます。 演算項目 2 で識別されないビットは変わりま せん。 したがって、BITON を使用して文字を形式設定する場合には、BITON と BITOFF の両方を使用しなければなりません。ビットをオン (=1) に設定する ように指定するためには BITON、ビットをオフ (=0) に設定するように指定 するためには BITOFF を使用します。 文字のすべてのビットをオンまたはオフに明示的に設定しない限り、希望す る文字を取り出すことはできません。

文字フィールドに特定のビット・パターンを割り当てる場合には、演算項目 2 に 16 進数リテラルを指定して 803 ページの『MOVE (転送)』 命令を使用します。

演算項目 2 には以下のものを入れることができます。

- ビット番号 *0* から *7:* 1 つの命令で 1 から 8 ビットをオンに 設定することができます。 これらは 0 から 7 の番号で識別されます (0 が左端 ビットです)。 ビット番号をアポストロフィで囲みます。 例えば、ビット 0、2、および 5 をオンに設定するためには、演算項目 2 に '025' と指定します。
- フィールド名: 演算項目 2 に、1 桁の文字フィールドの名前、テーブル要素、または配列要素を指定することができます。 このフィールド、テーブル要素、または配列要素でオンのビットは結果フィールドでオンに設定されます。オフのビットは影響を受けません。
- *16* 進数リテラルまたは名前のついた定数*:* 1 バイトの 16 進数リテラルを指定することができます。 演算項目 2 でオンのビット が、結果フィールドでオンに設定されます。オフのビットは結果に影響を 与えません。
- 名前のついた定数*:* オンに設定するビット番号 を含む 8 桁までの長さの文字の名前のついた定数を指定することができます。

結果フィールドには、1 桁の文字フィールドを指定します。 配列のそれぞれの要素が 1 桁の文字フィールドの場合には、配列要素にするこ とができます。

詳しくは、「562 ページの『ビット操作』」を参照してください。

```
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D FieldA          S              1A   INZ(X'00')
D FieldB          S              1A   INZ(X'00')
D FieldC          S              1A   INZ(X'FF')
D FieldD          S              1A   INZ(X'C0')
D FieldE          S              1A   INZ(X'C0')
D FieldF          S              1A   INZ(X'81')
D FieldG          S              1A   INZ(X'4F')
D FieldH          S              1A   INZ(X'08')
D FieldI          S              1A   INZ(X'CE')
D FieldJ          S              1A   INZ(X'80')
D FieldK          S              1A   INZ(X'80')
D BITNC           C                   CONST('0246')
D HEXNC           C                   CONST(X'0F')
D HEXNC2          C                   CONST(X'F0')
C*0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *    Set on bits 0,4,5,6,7 in  FieldA.  Leave bits 1,2,3 unchanged.
 *       Factor 2 = 10001111
 *       FieldA   = 00000000  (before)
 *       FieldA   = 10001111  (after)
C                   BITON     '04567'       FieldA
 *    Set on bit 3 in FieldB.  Leave bits 0,1,2,4,5,6,7 unchanged.
 *       Factor 2 = 00010000
 *       FieldB   = 00000000  (before)
 *       FieldB   = 00010000  (after)
C                   BITON     '3'           FieldB
 *    Set on bit 3 in FieldC.  Leave bits 0,1,2,4,5,6,7 unchanged.
 *    Setting on bit 3, which is already on, results in bit 3 remaining on.
 *       Factor 2 = 00010000
 *       FieldC   = 11111111  (before)
 *       FieldC   = 11111111  (after)
C                   BITON     '3'           FieldC
 *    Set on bit 3 in FieldD.  Leave bits 0,1,2,4,5,6,7 unchanged.
 *       Factor 2 = 00010000
 *       FieldD   = 11000000  (before)
 *       FieldD   = 11010000  (after)
C                   BITON     '3'           FieldD
 *    Set on bits 0 and 1 in FieldF.  Leave bits 2,3,4,5,6,7 unchanged.
 *    (Setting on bit 0, which is already on, results in bit 0 remaining on.)
 *       Factor 2 = 11000000
 *       FieldF   = 10000001  (before)
 *       FieldF   = 11000001  (after)
C                   BITON     FieldE        FieldF
 *    X'C1' is equivalent to literal '017', bit pattern 11000001.
 *    Set on bits 0,1,7 in  FieldH.  Leave bits 2,3,4,5,6 unchanged.
 *       Factor 2 = 11000001
 *       FieldH   = 00001000  (before)
 *       FieldH   = 11001001  (after)
C                   BITON     X'C1'         FieldH
 *    HEXNC is equivalent to literal '4567', bit pattern 00001111.
 *    Set on bits 4,5,6,7 in  FieldJ.  Leave bits 0,1,2,3 unchanged.
 *       Factor 2 = 00001111
 *       FieldJ   = 10000000  (before)
 *       FieldJ   = 10001111  (after)
C                   BITON     HEXNC         FieldJ
C                   RETURN
```

図 *283. BITON* の例

CABxx (比較および分岐)

自由形式構文	(許可されていない - LEAVE、ITER、 および RETURN などの他の命令コードを使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
CABxx	被比較値	被比較値	Label	HI	LO	EQ

CABxx 命令は、演算項目 1 を演算項目 2 と比較します。 xx に よって指定された条件が真の場合には、プログラムは、結果フィールドに指 定されたラベルに対応する TAG または ENDSR 命令に分岐します。 そうでない場合には、プログラムは次の命令から順番に続行します。 結果フィールドが指定されていない場合には、それに応じて結果の 標識 (71-76 桁目) が設定され、 プログラムは次の命令から順番に続行します。

条件付け標識を指定することができます。 演算項目 1 と演算項目 2 には、リテラル、名前のついた定数、形象定数、 テーブル名、配列要素、データ構造名、またはフィールド名を入れなければな りません。 演算項目 1 と 2 は同じタイプでなければなりません。 結果フィールドに指定されたラベルは固有の TAG 命令と関連付けられていなけ ればならず、また固有の記号名でなければなりません。

サイクル・メイン・プロシージャーの CABxx 命令では、次の分岐を指定することができます。

- 前の仕様行または後続の仕様行への分岐
- 明細演算行から別の明細演算行への分岐
- 合計演算行から別の合計演算行への分岐
- 明細演算行から合計演算行への分岐
- サブルーチンから明細演算行または合計演算行への分岐

サブプロシージャーの CABxx 命令では次の分岐を指定することができます。

- サブプロシージャーの本体の行からサブプロシージャーの本体の別の行へ の分岐
- サブルーチンの行から同じサブルーチンの別の行への分岐
- サブルーチンの行からサブプロシージャーの本体の行への分岐

CABxx 命令では、サブルーチンの外からそのサブルーチン内の TAG または ENDSR 命令への分岐を指定することはできません。

注意!

論理内の一部から別の部分へ分岐すると、無限ループに入ることがあります。 プログラムまたはプロシージャーの論理によって好ましくない結果が起こるこ とのないようにしてください。

結果の標識は任意指定です。 指定すると、比較操作の結果を反映するように設定されます。 例えば、F1>F2 の場合には HI 標識が設定され、F1<F2 の場合には LO、F1=F2 の場合には EQ が設定されます。

演算項目 1 と演算項目 2 を比較する場合の規則については、567 ページの『比較命令』を参照してください。

詳しくは、「562 ページの『分岐命令』」を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *         The field values are:
 *         FieldA = 100.00
 *         FieldB = 105.00
 *         FieldC = ABC
 *         FieldD = ABCDE
 *
 *          Branch to TAGX.
C     FieldA        CABLT     FieldB        TAGX
 *
 *          Branch to TAGX.
C     FieldA        CABLE     FieldB        TAGX
 *
 *          Branch to TAGX; indicator 16 is off.
C     FieldA        CABLE     FieldB        TAGX                    16
 *
 *          Branch to TAGX; indicator 17 is off, indicator 18 is on.
C     FieldA        CAB       FieldB        TAGX                  1718
 *
 *          Branch to TAGX; indicator 19 is on.
C     FieldA        CAB       FieldA        TAGX                    19
 *
 *          No branch occurs.
C     FieldA        CABEQ     FieldB        TAGX
 *
 *          No branch occurs; indicator 20 is on.
C     FieldA        CABEQ     FieldB        TAGX                  20
 *
 *          No branch occurs; indicator 21 is off.
C     FieldC        CABEQ     FieldD        TAGX                21
C                   :
C     TAGX          TAG
```

図 *284. CABxx* 命令

CALL (プログラムの呼び出し)

自由形式構文	(許可されていない - CALLP 命令コードを使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
CALL (E)		プログラム名	PLIST 名	_	ER	LR

CALL 命令では、演算項目 2 に指定されたプログラムに制御が渡されます。

演算項目 2 には、呼び出されるプログラムの名前を指定する文字項目を入れな ければなりません。

結果フィールドには、次のいずれかによってパラメーターを指定します。

- PLIST の名前を入れます。
- 結果フィールドをブランクのままにします。 これは、呼び出されたプログラムがパラメーターにアクセスしない場合、または PARM ステートメントが CALL 命令の直後に続く場合に有効です。

71 桁目と 72 桁目はブランクでなければなりません。

CALL 例外 (プログラム状況コードが 202、211、 または 231) を処理するために、命令コード拡張 'E' またはエラー標識 ER を指定 できますが、両方を指定することはできません。エラー処理について詳しくは、163 ページの『プログラム例外/エラー』を参照してください。

呼び出し先プログラムが LR 標識をオンにして戻された RPG プログラムまたはサイクル・メイン・プロシージャーである場合、有効な結果の標識 をオンに設定して、75 桁目と 76 桁目に指定することができます。

注 : スレッド・セーフ環境内では、LR 標識は使用できません。

呼び出し命令について詳しくは、563 ページの『呼び出し命令』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *  The CALL operation calls PROGA and allows PROGA to access
 *  FieldA and FieldB, defined elsewhere. PROGA is run using the content
 *  of FieldA and FieldB.  When PROGA has completed, control
 *  returns to the statement following the last PARM statement.
 *
 *
C                   CALL      'PROGA'
C                   PARM                    FieldA
C                   PARM                    FieldB
```

図 *285. CALL* 命令

CALLB (バインド・プロシージャーの呼び出し)

自由形式構文	(許可されていない - CALLP 命令コードを使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
CALLB (D E)		プロシージャー名またはプロシージャー・ポインター	PLIST 名	_	ER	LR

CALLB 命令は、ILE 言語で書かれたバインド・プロシージャーの呼び出しに使 用されます。

操作記述子を組み込むために命令拡張 D を使用することができます。 これは、キーワード OPDESC でパラメーターが定義されている場合に CALLP でプロトタイプ・プロシージャーを呼び出すことに似ています。(操作記述子は、プログラマーに、渡された文字またはグラフィック・ストリングの正確な属性 (つまり、ストリングの長さおよびタイプ) の実行時解析を提供します。) 詳しくは、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」のプログラムおよびプロシージャーの呼び出しに関する章を参照してください。

演算項目 2 は必須で、呼び出されるプロシージャーの名前が入っているリテラ ルまたは定数、あるいは呼び出されるプロシージャーのアドレスが入って いるプロシージャー・ポインターでなければなりません。すべての参照は、バインド時に解析できるものでなければなりません。 指定するプロシージャー名は大文字と小文字が区別され、10 桁より多くの文字を含むことができますが、255 桁を超えることはできません。 名前が 255 桁を超えた場合には、255 桁で切り捨て られます。 結果フィールドは任意指定で、PLIST 名を入れることができます。

CALLB 例外 (プログラム状況コードが 202、211、 または 231) を処理するために、命令コード拡張 'E' またはエラー標識 ER を指定 できますが、両方を指定することはできません。エラー処理について詳しくは、163 ページの『プログラム例外/エラー』を参照してください。

75-76 桁目に指定された標識は、呼び出しが LR をオンに設定して終わった時 にオンに設定されます。

注 : スレッド・セーフ環境内では、LR 標識は使用できません。

呼び出し命令について詳しくは、563 ページの『呼び出し命令』を参照してください。

```
DName++++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
 * Define a procedure pointer
D
D ProcPtr         S               *   PROCPTR INZ(%PADDR('Create_Space'))
D Extern          S             10
D
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 * The following call linkage would be STATIC
C                   CALLB     'BOUNDPROC'
 * The following call linkage would be DYNAMIC
C                   CALL      Extern
 * The following call linkage would be STATIC, using a procedure pointer
C                   CALLB     ProcPtr
```

図 *286. CALLB* 命令

CALLP (プロトタイプ・プロシージャーまたはプログラムの呼び出し)

自由形式構文	{CALLP{(EMR) } } 名前({パラメーター *1*{:パラメーター *2*... } })

コード	演算項目 1	拡張演算項目 2
CALLP (E M/R)		名前{ (パラメーター 1 {:パラメーター 2 ...}) }

CALLP 命令は、プロトタイプ・プロシージャーまたはプログラムの呼び出しに使用されます。

他の呼び出し命令とは違って、CALLP では自由形式の構文が使用されます。 呼び出されるプログラムまたはプロシージャーのプロトタイプの名前および渡されるパラメーターを指定するためには、名前 オペランドを使用します。 (これは組み込み関数の呼び出しに似ています。) プログラム呼び出しには最大 255、プロシージャー呼び出しには最大 399 の パラメーターを使用することができます。

自由形式演算仕様においては、拡張が不要である場合、およびプロトタイプに命令コードと同じ名前がない場合は、命令コード名を省略できます。

次に、コンパイラーはこのプロトタイプ名を使用して、 呼び出しに必要な外部名を取り出します。 プロトタイプにキーワード EXTPGM が指定されている場合には、呼び出しは動的外部 呼び出しになります。そうでない場合には、これはバインド・プロシージャー呼び出しです。

呼び出し先のプログラムまたはプロシージャーが別のモジュールで定義されている場合、 呼び出されるプログラムまたはプロシージャーのプロトタイプは、CALLP の前 の定義仕様書に含まれていなければなりません。 呼び出し先のプログラムまたはプロシージャーが、呼び出しと同じモジュールに定義されている場合、明示的なプロトタイプは必要ありません。 プロトタイプは、呼び出し先のプログラムまたはプロシージャーのプロシージャー・インターフェースから暗黙的に定義できます。

値を戻すプロシージャーの呼び出しに CALLP を使用した場合には、呼び出し元はそ の値を使用できないことに注意してください。 値が必要な場合は、式の中でプロトタイプ・プロシージャーを呼び出します。

CALLP 例外 (プログラム状況コードが 202、211、 または 231) を処理するために、命令コード拡張 'E' を指定することができます。エラー処理について詳しくは、163 ページの『プログラム例外/エラー』を参照してください。

注 : E 拡張は、CALLP の最終呼び出し時のみアクティブになります。 パラメーター処理の一部として実行された呼び出し時にエラーが発生した場合、 制御は次の命令に渡されません。 例えば、FileRecs が数値を戻すプロシージャーであるときに、FileRecs が 以下のステートメントに呼び出されたときにエラーが起こった場合、E 拡張は何の効果も与えません。

```
CALLP(E) PROGNAME(FileRecs(Fld) + 1)
```

呼び出し命令について詳しくは、563 ページの『呼び出し命令』を参照してください。プロトタイプの定義について詳しくは、224 ページの『プロトタイプおよびパラメーター』を参照してください。命令拡張 M および R の使用法については、608 ページの『数値演算の精度の規則』を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
 *---------------------------------------------------------------
 *  This prototype for QCMDEXC defines two parameters:
 *   1- a character field that may be shorter in length
 *      than expected
 *   2- any numeric field
 *---------------------------------------------------------------
D qcmdexc         PR                  extpgm('QCMDEXC')
D   cmd                         200A   options(*varsize) const
D   cmdlen                       15P 5 const

 /FREE
      qcmdexc ('WRKSPLF' : %size ('WRKSPLF'));
 /END-FREE
```

図 *287. CALLP* を使用したプロトタイプ・プログラムの呼び出し

```
 * The prototype for the procedure has an array parameter.
D proc            pr
D   parm                        10a   dim(5)

* An array to pass to the procedure
D array           s             10a   dim(5)

* Call the procedure, passing the array
C                   callp     proc (array)
```

図 *288. CALLP* を使用した配列パラメーターの受け渡し

CALLP の次の例は「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」にあるサービス・プログラムの例から取られたものです。 CvtToHex は、変換ルーチンを入れるために作成されたサービス・プログラムの プロシージャーです。 CvtToHex は入力ストリングをその 16 進数形式に変換します。 プロトタイプ呼び出しは ILE CEE API, CEEDOD (操作記述子の検索) に対する ものです。 これは入力ストリングの長さを確認するために使用されます。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
 *===================================================================*
 * CvtToHex - convert input string to hex output string              *
 *===================================================================*
D/COPY MYLIB/QRPGLESRC,CVTHEXPR

 *-------------------------------------------------------------------*
 * Main entry parameters                                             *
 * 1. Input:   string                 character(n)                   *
 * 2. Output:  hex string             character(2 * n)               *
 *-------------------------------------------------------------------*
D CvtToHex        PI                    OPDESC
D   InString                 16383      CONST OPTIONS(*VARSIZE)
D   HexString                32766      OPTIONS(*VARSIZE)

 *-------------------------------------------------------------------*
 * Prototype for CEEDOD (Retrieve operational descriptor)            *
 *-------------------------------------------------------------------*
D CEEDOD          PR
D                                10I 0 CONST
D                                10I 0
D                                10I 0
D                                10I 0
D                                10I 0
D                                10I 0
D                                12A   OPTIONS(*OMIT)

 * Parameters passed to CEEDOD
D ParmNum         S              10I 0
D DescType        S              10I 0
D DataType        S              10I 0
D DescInfo1       S              10I 0
D DescInfo2       S              10I 0
D InLen           S              10I 0
D HexLen          S              10I 0
```

図 *289. CALLP* を使用したプロトタイプ・プロシージャーの呼び出し

```
 *---------------------------------------------------------------*
 * Other fields used by the program                              *
 *---------------------------------------------------------------*
D HexDigits       C                   CONST('0123456789ABCDEF')
D IntDs           DS
D   IntNum                       5I 0 INZ(0)
D   IntChar                      1    OVERLAY(IntNum:2)
D HexDs           DS
D   HexC1                        1
D   HexC2                        1
D InChar          S              1
D Pos             S              5P 0
D HexPos          S              5P 0

 /FREE
    //-------------------------------------------------------------//
    // Use the operational descriptors to determine the lengths of //
    // the parameters that were passed.                            //
    //-------------------------------------------------------------//
    CEEDOD (1 : DescType : DataType :
                DescInfo1 : DescInfo2 : Inlen : *OMIT);
    CEEDOD (2 : DescType : DataType :
                DescInfo1 : DescInfo2 : HexLen : *OMIT);

    //-------------------------------------------------------------//
    // Determine the length to handle (minimum of the input length //
    // and half of the hex length)                                 //
    //-------------------------------------------------------------//
    if InLen > HexLen / 2;
       InLen = HexLen / 2;
    endif;

    //-------------------------------------------------------------//
    // For each character in the input string, convert to a 2-byte //
    // hexadecimal representation (for example, '5' --> 'F5')      //
    //-------------------------------------------------------------//
    HexPos = 1;
    for Pos = 1 to InLen;
        InChar = %SUBST(InString : Pos :1);
        exsr GetHex;
        %subst (HexString: HexPos: 2) = HexDs;
        HexPos = HexPos + 2;
    endfor;

    return;

    //=============================================================//
    // GetHex - subroutine to convert 'InChar' to 'HexDs'          //
    // Use division by 16 to separate the two hexadecimal digits.  //
    // The quotient is the first digit, the remainder is the second.  //
    //=============================================================//
    begsr GetHex;
       IntChar = InChar;

       //-----------------------------------------------------//
       // Use the hexadecimal digit (plus 1) to substring the //
       // list of hexadecimal characters '012...CDEF'.        //
       //-----------------------------------------------------//
       HexC1 = %subst (HexDigits: %div(IntNum:16) + 1: 1);
       HexC2 = %subst (HexDigits: %rem(IntNum:16) + 1: 1);
    endsr;  // GetHex
```

CASxx (サブルーチンの条件付き呼び出し)

自由形式構文	(許可されていない - IF 命令コードおよび EXSR 命令コードを使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
CASxx	被比較値	被比較値	サブルーチン名	HI	LO	EQ

CASxx 命令では、サブルーチンを条件付きで選択して処理することができます。 この選択は、xx で指定された演算項目 1 と 演算項目 2 の関係に基づいています。 演算項目 1 と演算項目 2 の間に xx で示された関係が存在する場合には、結果 フィールドに指定されたサブルーチンが処理されます。

条件付け標識を指定することができます。 演算項目 1 と演算項目 2 には、 リテラル、名前のついた定数、形象定数、フィールド名、テーブル名、配列要素、 またはデータ構造名を入れるか、あるいはブランクにすることができます (ブランクを 使用できるのは、xx がブランクで、71 から 76 桁目に結果の 標識が指定されていない場合だけです)。 演算項目 1 と演算項目 2 がブランク以外の場合には、両方とも同じデータ・タ イプでなければなりません。 CAS?? 命令で 演算項目 1 と演算項目 2 が 必要になるのは、71 から 76 桁目に演算結果標識が指定されている場合だけです。

結果フィールドには、*PSSR (プログラム例外/エラー処理サブルーチン)、および *INZSR (プログラム初期化サブルーチン) を含む有効な RPG IV サブルーチンの名前が入る必要があります。 演算項目 1 と演算項目 2 の間に xx で示された関係が存在する場合には、結果 フィールドに指定されたサブルーチンが処理されます。 xx で示された関係が 存在しない場合には、プログラムは CAS グループの次の CASxx 命令から続行します。 CAS グループには CASxx 命令しか入れることができません。 CAS グループの終わりを示すために、最後の CASxx 命令の後には ENDCS 命令を入れなけ ればなりません。 このサブルーチンが処理されると、サブルーチンによって別 の命令に制御が渡されない限り、プログラムは ENDCS 命令の後の次の命令から 処理を続行します。

71 から 76 桁目に演算結果標識が指定されていない CAS?? 命令 は、CAS?? 命令の結果フィールドに指定されたサブルーチンを 無条件に実行するので、機能的には EXSR と同じです。 同じ CAS グループの中で無条件の CAS?? 命令に続く CASxx 命令が テストされることはありません。 したがって、無条件の CAS?? 命令は、通常は CAS グループの中の 他のすべての CASxx 命令の後に入れます。

CAS グループの ENDCS 命令に条件付け標識を使用することはできません。

CASxx 命令に関する規則について詳しくは、567 ページの『比較命令』または 594 ページの『サブルーチン命令』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  The CASGE operation compares FieldA with FieldB.  If FieldA is
 *  greater than or equal to FieldB, Subr01 is processed and the
 *  program continues with the operation after the ENDCS operation.
 *
C     FieldA        CASGE     FieldB        Subr01
 *
 *  If FieldA is not greater than or equal to FieldB, the program
 *  next compares FieldA with FieldC.  If FieldA is equal to FieldC,
 *  SUBR02 is processed and the program continues with the operation
 *  after the ENDCS operation.
 *
C     FieldA        CASEQ     FieldC        Subr02
 *
 *  If FieldA is not equal to FieldC, the CAS operation causes Subr03
 *  to be processed before the program continues with the operation
 *  after the ENDCS operation.
 *  The CAS statement is used to provide a subroutine if none of
 *  the previous CASxx operations have been met.
 *
C                   CAS                     Subr03
 *
 *  The ENDCS operation denotes the end of the CAS group.
 *
C                   ENDCS
```

図 *290. CASxx* 命令

CAT (2 つの文字ストリングの連結)

自由形式構文	(許可されていない - + 演算子を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
CAT (P)	ソース・ストリング 1	ソース・ストリング 2: ブランクの数	ターゲット・ストリング			

CAT 命令は、演算項目 2 に指定されたストリングを演算項目 1 に指定されたス トリングの終わりに連結して、これを結果フィールドに入れます。 ソース・ストリングとターゲット・ストリングはすべて同じタイプで、すべて文字か、すべて図形か、すべて UCS-2 のいずれかでなければなりません。 演算項目 1 が指定されていない場合には、演算項目 2 は結果フィールドのスト リングの終わりに連結されます。

演算項目 1 にはストリングを入れることができ、このストリングはフィールド 名、配列要素、名前のついた定数、データ構造名、テーブル名、またはリ テラルのいずれかとすることができます。 演算項目 1 が指定されていない場合には、結果フィールドが使用されます。 次の説明では、演算項目 1 が指定されていない場合に、演算項目 1 が結果フィ ールドに適用されることを示します。

演算項目 2 にはストリングを入れなければならず、連結するストリング間に挿 入するブランクの数を入れることができます。 その形式は、ストリングの後にコロンを付けブランクの数が続いた形にな ります。 これらのブランクはデータの形式です。 例えば、文字データの場合、ブランクは x'40' で、UCS-2 データの場合、ブランクは x'0020' です。 ストリング部分には、フィールド名、配列要素、名前のついた定数、デー タ構造名、テーブル名、リテラル、または構造サブフィールド名のいずれかを 入れることができます。 ブランクの数の部分は小数点以下の桁数がゼロの数値でなければならず、名前 のついた定数、配列要素、リテラル、テーブル名、またはフィールド名の いずれかを入れることができます。

コロンを指定した場合には、ブランクの数を指定しなければなりません。 コロンを指定しない場合には、演算項目 1 (演算項目 1 が指定されていない場 合には結果フィールド) の後書きブランク (もしあれば) に連結が行われま す。

ブランクの数 N が指定されている場合には、演算項目 1 は結果フィールドに左 寄せでコピーされます。 演算項目 1 が指定されていない場合には、結果フィールドのストリングが使用 されます。 その後で、最後の非ブランク文字に続けて N 個のブランクが追加されます。 次に、この結果に演算項目 2 が付加されます。 結果に N 個のブランクを追加するときには演算項目 2 の先行ブランクはカウントされません。演算項目 2 の一部と見なされるだけです。 ブランクの数が指定されていない場合、演算項目 1 と演算項目 2 の後書きブランクと先行ブランクは結果に含められます。

結果フィールドはストリングでなければならず、フィールド名、配列要素、デ ータ構造名、またはテーブル名のいずれかを入れることができます。 その長さは、演算項目 1 と演算項目 2 に中間のブランクを加えた長さでなけれ ばならず、そうでない場合には右側で切り捨てが行われます。 結果のフィールドが可変長の場合、その長さは変わりません。

命令拡張 P は、連結が行われた後で結果フィールドが命令の結果より長い場 合に、結果フィールドの右にブランクを埋め込む必要があることを示します。 埋め込みが指定されていない場合には、このフィールドの左端部分が影響を受け るだけです。

実行時にブランクの数がゼロより小さい場合には、コンパイラーはデフォルトの値 としてブランクの数をゼロとします。

詳しくは、「589 ページの『ストリング命令』」を参照してください。

注 : 演算項目 1、演算項目 2、または結果フィールドに形象定数を使用することは できません。 演算項目 1 と結果フィールド、または演算項目 2 と結果フィールドのデータ構 造にオーバーラップがあってはなりません。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 * The following example shows leading blanks in factor 2.  After
 * the CAT, the RESULT contains 'MR.bSMITH'.
 *
C                   MOVE      'MR.'         NAME              3
C                   MOVE      ' SMITH'      FIRST             6
C     NAME          CAT       FIRST         RESULT            9
 *
 *  The following example shows the use of CAT without factor 1.
 *  FLD2 is a 9 character string.  Prior to the concatenation, it
 *  contains 'ABCbbbbbb'; FLD1 contains 'XYZ
 *  After the concatenation, FLD2 contains 'ABCbbXYZb'.
 *
C                   MOVEL(P) 'ABC'         FLD2              9
C                   MOVE     'XYZ'         FLD1              3
C                   CAT      FLD1:2        FLD2
```

図 *291. CAT* 命令

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 * CAT concatenates LAST to NAME and inserts one blank as specified
 * in factor 2.  TEMP contains 'Mr.bSmith'.
C                   MOVE      'Mr.   '      NAME              6
C                   MOVE      'Smith '      LAST              6
C     NAME          CAT       LAST:1        TEMP              9
 *
 * CAT concatenates 'RPG' to STRING and places 'RPG/400' in TEMP.
C                   MOVE     '/400'         STRING            4
C     'RPG'         CAT      STRING         TEMP              7
 *
 * The following example is the same as the previous example except
 * that TEMP is defined as a 10 byte field.  P operation extender
 * specifies that blanks will be used in the rightmost positions
 * of the result field that the concatenation result, 'RPG/400',
 * does not fill.  As a result, TEMP contains 'RPG/400bbb'
 * after concatenation.
C                   MOVE      *ALL'*'       TEMP             10
C                   MOVE      '/400'        STRING            4
C     'RPG'         CAT(P)    STRING        TEMP
 *
 * After this CAT operation, the field TEMP contains 'RPG/4'.
 * Because the field TEMP was not large enough, truncation occurred.
C                   MOVE      '/400'        STRING            4
C     'RPG'         CAT       STRING        TEMP              5
 *
 * Note that the trailing blanks of NAME are not included because
 * NUM=0.  The field TEMP contains 'RPGIVbbbbb'.
C                   MOVE      'RPG  '       NAME              5
C                   MOVE      'IV   '       LAST              5
C                   Z-ADD     0             NUM               1 0
C     NAME          CAT(P)    LAST:NUM      TEMP             10
```

図 *292.* 先行ブランクのある *CAT* 命令

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
 *
 * The following example shows the use of graphic strings
 *
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++++
 *       Value of Graffld  is 'AACCBBGG'.
 *       Value of Graffld2 after CAT 'aa    AACCBBGG      '
 *       Value of Graffld3 after CAT 'AABBCCDDEEFFGGHHAACC'
 *
D Graffld                        4G   INZ(G'oAACCBBGGi')
D Graffld2                      10G   INZ
D Graffld3                      10G   INZ(G'oAABBCCDDEEFFGGHHi')
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq.
 * The value 2 represents 2 graphic blanks as separators
C     G'oaai'       cat       Graffld:2     Graffld2
C                   cat       Graffld       Graffld3
```

図 *293.* 図形データによる *CAT* 命令

CHAIN (ファイルからのランダム検索)

自由形式構文	CHAIN{(ENHMR) } 検索引数 名前 {データ構造 }

コード	演算項目 1	演算項目 2	結果フィールド	標識		
CHAIN (E N)	検索引数	名前 (ファイルまたはレコード様式)	データ構造	NR	ER	_

CHAIN 命令は、全手順ファイルからレコードを取り出し、レコード識別標識を オンに設定し (入力仕様に指定された場合)、レコードのデータを入力フィールドに 入れます。

検索引数 (検索引数) は、レコードの検索に使用するキーまたは相対レコード番号でなければなりません。アクセスがキーによる場合には、検索引数 はフィールド名、名前の付いた固定情報、 形象定数、またはリテラルの形式の単一キーにすることができます。

ファイルが外部記述ファイルの場合、検索引数 は KLIST 名、値のリスト、または %KDS の形式の複合キーにすることもできます。 KLIST を 使用して指定されたキーの場合、キー・フィールドはファイル内のキーと同じ CCSID である必要があります。 %KDS の例については、652 ページの『%KDS (データ構造の検索引数)』の終わりにある例を参照してください。 アクセスが相対レコード番号による場合には、検索引数 に整数のリテラルまたは小数点以下の桁数がゼロの数値フィールドを入れなければなりません。

制御キーワード EXPROPTS(*STRICTKEYS) が値のリストまたは %KDS でキーを指定する規則に与える影響について詳しくは、335 ページの『*STRICTKEYS』を参照してください。

名前 オペランドには、読み取るファイルまたはレコード様式の名前を指定します。 レコード様式名を使用できるのは、外部記述ファイルの場合だけのみです。 ファイル名が 名前 に指定されていてアクセスがキーによる場合には、CHAIN 命令によって検索引数と一致する最初のレコードが検索されます。

名前 がレコード様式名でアクセスがキーによる場合には、CHAIN 命令によってそのキーが検索引数と一致する指定されたレコード・タイプの最初のレコードが検索されます。 検索引数と一致する指定されたレコード・タイプの レコードが見付からない場合には、該当レコードなしの状況になります。

データ構造 オペランドが指定されている場合、レコードはデータ構造に直接読み込まれます。 名前 がプログラム記述ファイルを参照する場合、データ構造は、宣言されたファイルのレコード長と同じ長さの任意のデータ構造にできます。 名前 が外部記述ファイルまたは外部記述ファイルのレコード様式を参照する場合、データ構造は EXTNAME(...:*INPUT or *ALL) または LIKEREC(...:*INPUT or *ALL) で定義されているデータ構造にする必要があります。 データ構造の定義方法、およびファイルとデータ 構造の間でどのようにデータが転送されるかについては、576 ページの『ファイル操作』を参照してください。

ワークステーション・ファイルの場合には、CHAIN 命令によってサブファイル・レコードが 検索されます。

複数装置ファイルの場合には、名前 オペランドにレコード様式を指定しなければなりません。 データは、369 ページの『DEVID(フィールド名)』装置ファイルのファイル仕様書の キーワードに指定されたフィールド名で識別されるプログラム装置から読み取られます。 このキーワードが指定されていない場合には、データは、ファイルに対して最 後に正常な入力操作が行われた装置から読み取られます。

ファイルが入力 DISK ファイルとして指定されている場合には、すべての レコードはロックなしで読み取られ、そのために命令拡張を指定する ことはできません。 ファイルが更新として指定されている場合には、命令拡張 N が指定されていな ければ、すべてのレコードがロックされます。

更新ディスク・ファイルから読み取っている場合には、命令拡張 N を指定して 読み取り時にレコードをロックしないように (例えば、CHAIN (N)) 指示するこ とができます。 詳細については、*Rational Development Studio for i: ILE RPG* プログラマーの手引き を参照してください。

ファイルに検索引数と一致するレコードがない場合にオンに設定される標識 を 71 から 72 桁目に指定することができます。 この情報は %FOUND 組み込み関数 からも入手することができます。この関数は、レコードが見付からない場合は '0' を 戻し、レコードが見付かった場合は '1' を戻します。

75 桁目と 76 桁目はブランクでなければなりません。

CHAIN 例外 (ファイル状況コードが 1000 より大きい) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。 エラー処理の詳細については、146 ページの『ファイル例外/エラー』を 参照してください。

CHAIN 命令が正常に実行された場合には、名前 に指定されたファイルは、以後の読み取り操作で論理的に検索済みレコードの後または前にあるレコードが検索されるように位置付けられます。 CHAIN 命令が正常に完了しなかった場合 (例えば、エラーが起こったりレコードが見付からなかった場合) には、名前 に指定されたファイルは、そのファイルに対しての次の読み取り操作が実行される前に (例えば、CHAIN または SETLL 命令によって) 再度位置決めされなければなりません。

名前 に指定されたファイルに、そのファイルへの CHAIN 命令が正常に実行された直後に (演算仕様書または出力仕様で) 更新が実行されると、最後に検索されたレコードが更新されます。

ヌル値可能フィールドおよびキーを持つレコードの処理については、286 ページの『データベースのヌル値サポート』を参照してください。

詳細については、576 ページの『ファイル操作』を参照してください。

注 : 命令コード拡張 H、M、および R は、検索引数がリストまたは %KDS() である場合にのみ使用できます。579 ページの『ファイル命令のキー』および 558 ページの『精度の確認』を参照してください。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
*
*  The CHAIN operation retrieves the first record from the file,
*  FILEX, that has a key field with the same value as the search
*  argument KEY (factor 1).

/FREE
    CHAIN  KEY  FILEX;

 //  If a record with a key value equal to the search argument is
 //  not found, %FOUND returns '0' and the EXSR operation is
 //  processed. If a record is found with a key value equal
 //  to the search argument, the program continues with
 //  the calculations after the EXSR operation.

    IF  NOT %FOUND;
       EXSR  Not_Found;
    ENDIF;
/END-FREE
```

図 *294.* ファイル名を指定した *CHAIN* 命令

```
FFilename++IPEASF.....L.....A.Device+.Keywords++++++++++++++++++++++++++
FCUSTFILE   IF   E           K DISK
 /free
        // Specify the search keys directly in a list
        chain ('abc' : 'AB') custrec;
        // Expressions can be used in the list of keys
        chain (%xlate(custname : LO : UP) : companyCode + partCode)
              custrec;
        return;
```

図 *295.* キー・フィールドのリストを使用した *CHAIN* 命令

```
FFilename++IPEASF.....L.....A.Device+.Keywords++++++++++++++++++++++++++
FCUSTFILE   IF   E           K DISK
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D custRecDs       ds                  likerec(custRec)

 /free
        // Read the record directly into the data structure
        chain ('abc' : 'AB') custRec custRecDs;
        // Use the data structure fields
        if (custRecDs.code = *BLANKS);
           custRecDs.code = getCompanyCode (custRecDs);
           update custRec custRecDs;
        endif;
```

図 *296.* 外部記述ファイルのデータ構造を使用した *CHAIN* 命令

CHECK (文字の検査)

自由形式構文	(許可されていない - %CHECK 組み込み関数を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
CHECK (E)	比較ストリング	基本ストリング: 開始	左端の配置	–	ER	FD

CHECK 命令は、基本ストリング (演算項目 2) のそれぞれの文字が比較ストリ ング (演算項目 1) に示された文字の中に含まれているかどうかを検査します。 基本ストリングと比較ストリングは同じタイプで、 両方とも文字か、両方とも図形か、あるいは両方とも UCS-2 のいずれかでなければなりません。 (図形タイプと UCS-2 タイプの CCSID 値は同じでなければなりません。) 検査は演算項目 2 の左端の文字から開始されて、左から右へ 1 文字ずつ続けら れます。 基本ストリングのそれぞれの文字が演算項目 1 の文字と比較されます。 演算項目 2 の文字に演算項目 1 の文字と一致するものがあれば、次の基本ストリング文字が検査されます。 一致するものが見付からない場合には、一致する文字 が見付からない文字の位置を示すために、結果フィールドに整数値が入れられ ます。

演算項目 2 の開始位置は、基本ストリングとコロンで区切って指定することが できます。 開始は任意指定で、デフォルトの値は 1 です。開始位置が 1 より大きい場合には、結果フィールドの値は、開始位置に関係なく基本ストリングの左端からの相対位置になります。

この命令は、一致しない最初の文字が見付かったか、あるいは基本ストリング の終わりになった時に検査を停止します。 一致する文字が見付からなかった場合には、結果フィールドはゼロに設定さ れます。

結果フィールドが配列の場合には、この命令は、配列内に要素がある限り、 一致しない最初の文字が見付かった後でも検査を続行します。 配列の要素が一致しない文字より多い場合には、残りのすべての要素がゼロに 設定されます。

演算項目 1 はストリングでなければならず、フィールド名、配列要素、名前の ついた定数、データ構造名、データ構造サブフィールド、リテラル、また はテーブル名のいずれかを入れることができます。

演算項目 2 には、基本ストリングまたは基本ストリングにコロンを付けて 開始位置を指定しなければなりません。 演算項目 2 の基本ストリング部分には、フィールド名、配列要素、名前のつい た定数、データ構造名、リテラル、またはテーブル名を入れることができ ます。 演算項目 2 の開始位置部分は小数点以下 の桁数がゼロの数値でなければならず、名前のついた定数、配列要素、フ ィールド名、リテラル、またはテーブル名とすることができます。 開始位置が指定されていない場合には、値 1 が使用されます。

結果フィールドは、数値変数、数値配列要素、数値テーブル名、または数値配 列とすることができます。 フィールドまたは配列は、小数点以下の桁数にゼロを指定して定義します。 図形データまたは UCS-2 データを使用する場合には、結果フィールドには、2 バイト文字位置 (すなわち、 位置 3 である、3 番目の 2 バイト文字が文字位置 5 になる) が入ります。

注 : 演算項目 1、演算項目 2、または結果フィールドに形象定数を使用することは できません。 演算項目 1 と結果フィールド、または演算項目 2 と結果フィールドのデータ構 造にオーバーラップがあってはなりません。

7 から 11 桁目には、有効な標識を指定することができます。

CHECK 例外 (プログラム状況コード 100) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、163 ページの『プログラム例外/エラー』を参照してください。

正しくない文字が見付かった場合にオンに設定される標識を 75-76 桁目に 指定することができます。 この情報は %FOUND 組み込み関数からも入手することができます。 この関数は、 正しくない文字が見付かった場合に '1' を戻します。

詳しくは、「589 ページの『ストリング命令』」を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords+++++++++++++++++++++++++++++
 * In this example, the result will be N=6, because the start
 * position is 2 and the first nonnumeric character found is the '.'.
 * The %FOUND built-in function is set to return '1', because some
 * nonnumeric characters were found.
 *
D
D Digits          C                   '0123456789'
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
C
C                   MOVE      '$2000.'      Salary
C     Digits        CHECK     Salary:2      N
C                   IF        %FOUND
C                   EXSR      NonNumeric
C                   ENDIF
 *
 * Because factor 1 is a blank, CHECK indicates the position
 * of the first nonblank character.  If STRING contains 'bbbth
 * NUM will contain the value 4.
 *
C
C           ' '     CHECK     String        Num               2 0
```

図 *297. CHECK* 命令

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
 * The following example checks that FIELD contains only the letters
 * A to J.  As a result, ARRAY=(136000) after the CHECK operation.
 * Indicator 90 turns on.
 *
D
D Letter          C                   'ABCDEFGHIJ'
D
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
C
C                   MOVE      '1A=BC*'      Field             6
C     Letter        CHECK     Field         Array                   90
C
 *
 * In the following example, because FIELD contains only the
 * letters A to J, ARRAY=(000000).  Indicator 90 turns off.
 *
C
C                   MOVE      'FGFGFG'      Field             6
C     Letter        CHECK     Field         Array                   90
C
C
```

図 *298. CHECK* 命令

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D
 * The following example checks a DBCS field for valid graphic
 * characters starting at graphic position 2 in the field.
D
 *       Value of Graffld  is 'DDBBCCDD'.
 *       The value of num after the CHECK is 4, since this is the
 *       first character 'DD' which is not contained in the string.
D
D Graffld                       4G   INZ(G'oDDBBCCDDi')
D Num                           5 0
D
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq.
C
C
C     G'oAABBCCi'   check     Graffld:2     Num
```

図 *299.* 図形データによる *CHECK* 命令

CHECKR (逆向きの検査)

自由形式構文	(許可されていない - %CHECKR 組み込み関数を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
CHECKR (E)	比較ストリング	基本ストリング: 開始	右端の配置	–	ER	FD

CHECKR 命令は、基本ストリング (演算項目 2) のそれぞれの文字が比較ストリング (演算項目 1) に示された文字の中に含まれているかどうかを検査し ます。 基本ストリングと比較ストリングは同じタイプで、両方とも文字か、両方とも図形か、あるいは両方とも UCS-2 のいずれかでなければなりません。 (図形タイプと UCS-2 タイプの CCSID 値は同じでなければなりません。) 検査は演算項目 2 の右端の文字から開始されて、右から左へ 1 文字ずつ続けら れます。 基本ストリングのそれぞれの文字が演算項目 1 の文字と比較

されます。 演算項目 2 の文字に演算項目 1 の文字と一致するものがあれば、次の比較元の文字が検査されます。 一致するものが見付からない場合には、一致する文字 が見付からない文字の位置を示すために、結果フィールドに整数値が入れられ ます。 検査は右から行われますが、結果フィールドに入れられる位置は、左からの 相対位置になります。

演算項目 2 の開始位置は、基本ストリングとコロンで区切って指定することが できます。 開始位置は任意指定で、デフォルトの値はストリングの長さになります。 結果フィールドの値は、開始位置に関係なく、基本ストリングの左端位置から の相対値になります。

結果フィールドが配列でない場合には、この命令は、一致しない最初の文字 が見付かったか、または基本ストリングの終わりになった時に停止します。 一致する文字が見付からなかった場合には、結果フィールドはゼロに設定さ れます。

結果フィールドが配列の場合には、この命令は、配列内に要素がある限り、 一致しない最初の文字が見付かった後でも検査を続行します。 配列の要素が一致しない文字より多い場合には、残りのすべての要素がゼロに 設定されます。

演算項目 1 はストリングでなければならず、フィールド名、配列要素、名前 のついた定数、データ構造名、データ構造サブフィールド、リテラル、ま たはテーブル名のいずれかを入れることができます。

演算項目 2 には、基本ストリングまたは基本ストリングにコロンを付けて 開始位置を指定しなければなりません。 演算項目 2 の基本ストリング部分には、フィールド名、配列要素、名前のつい た定数、データ構造名、データ構造サブフィールド名、リテラル、または テーブル名を入れることができます。 演算項目 2 の開始位置部分は小数点以下 の桁数がゼロの数値でなければならず、名前のついた定数、配列要素、フィールド名、リテラル、またはテーブル名とすることができます。 開始位置が指定されていない場合には、ストリングの長さが使用されます。

結果フィールドは、数値変数、数値配列要素、数値テーブル名、または数値配 列とすることができます。 フィールドまたは配列は、小数点以下の桁数にゼロを指定して定義します。 図形データまたは UCS-2 データを使用する場合には、結果フィールドには、2 バイト文字位置 (すなわち、 位置 3 である、3 番目の 2 バイト文字が文字位置 5 になる) が入ります。

注 : 演算項目 1、演算項目 2、または結果フィールドに形象定数を使用することは できません。 演算項目 1 と結果フィールド、または演算項目 2 と結果フィールドのデータ構 造にオーバーラップがあってはなりません。

7 から 11 桁目には、有効な標識を指定することができます。

CHECKR 例外 (プログラム状況コード 100) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、163 ページの『プログラム例外/エラー』 を参照してください。

正しくない文字が見付かった場合にオンに設定される標識を 75-76 桁目に 指定することができます。 この情報は %FOUND 組み込み関数からも入手することができます。 この関数は、 正しくない文字が見付かった場合に '1' を戻します。

詳しくは、「589 ページの『ストリング命令』」を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 * Because factor 1 is a blank character, CHECKR indicates the
 * position of the first nonblank character.  This use of CHECKR
 * allows you to determine the length of a string.  If STRING
 * contains 'ABCDEF   ', NUM will contain the value 6.
 * If an error occurs, %ERROR is set to return '1' and
 * %STATUS is set to return status code 00100.
 *
C
C     ' '           CHECKR(E) String        Num
C
C                   SELECT
C                   WHEN      %ERROR
C ... an error occurred
C                   WHEN      %FOUND
C ... NUM is less than the full length of the string
C                   ENDIF
```

図 *300. CHECKR* 命令

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
 *
 * After the following example, N=1 and the found indicator 90
 * is on. Because the start position is 5, the operation begins
 * with the rightmost 0 and the first nonnumeric found is the '$'.
 *
D Digits          C                   '0123456789'
D
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C
C                   MOVE      '$2000.'      Salary          6
C     Digits        CHECKR    Salary:5      N                       90
C
```

図 *301. CHECKR* 命令

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
 *
 * The following example checks that FIELD contains only the letters
 * A to J.  As a result, ARRAY=(876310) after the CHECKR operation.
 * Indicator 90 turns on. %FOUND would return '1'.
D
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D Array           S              1    DIM(6)
D Letter          C                   'ABCDEFGHIJ'
D
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C
C                   MOVE      '1A=BC***'    Field           8
C     Letter        CHECKR    Field         Array                   90
C
```

図 *302. CHECKR* 命令

CLEAR (消去)

自由形式構文	CLEAR {*NOKEY } {*ALL } 名前

コード	演算項目 1	演算項目 2	結果フィールド	標識		
CLEAR	*NOKEY	*ALL	名前 (変数またはレコード様式)			

CLEAR 命令は、構造 (レコード様式、データ構造、配列、またはテーブル) 内の要素、または変数 (フィールド、サブフィールド、配列要素、または標識) をフィールドのタイプ (数値、文字、図形、UCS-2、標識、ポインター、または日付/時刻/タイム・スタンプ) によって、それぞれのデフォルトの初期値に設定します。 データ・タイプのデフォルトの初期値については、246 ページの『データ・タイプおよびデータ形式』を参照してください。

自由形式演算仕様書でコーディングされる場合、完全修飾名は CLEAR の結果フィールド・オペランドとして指定することができます。 消去中の構造または変数が可変長の場合、その長さは 0 に変わります。 CLEAR 命令では、実行時に要素ごとに消去するだけでなく、グローバルな基準で構造を消去することができます。

580 ページの『初期化命令』を参照してください。

変数の消去

*NOKEY を指定することはできません。

*ALL は任意指定です。 *ALL が指定されて、名前 オペランドに 複数オカレンス・データ構造またはテーブル名が入っている場合には、 すべてのオカレンスまたはテーブル要素が消去されて、オカレンス・レベルまたは テーブル指標は 1 に設定されます。

名前 オペランドには、消去される変数を指定します。 名前 オペランドへの特定の入力により、次のような消去処理を判別します。

単一オカレンス・データ構造
すべてのフィールドが構造内で宣言されている順序で消去されます。

複数回繰り返しデータ構造
*ALL が指定されていない場合は、現在の オカレンスのすべてのフィールドだけが消去されます。 *ALL が指定されている場合は、すべての オカレンスのすべてのフィールドが消去されます。

テーブル名
*ALL が指定されていない場合は、現在の テーブル要素が消去されます。 *ALL が指定されている場合は、すべてのテーブル要素が消去されます。

配列名
配列全体が消去されます。

配列要素 (標識を含む)
指定された要素だけが消去されます。

レコード様式の消去

*NOKEY は任意指定です。 *NOKEY が指定されている場合には、キー・フィールドはそれらの初期値で消去されることはありません。

*ALL は任意指定です。 *ALL が指定され、*NOKEY が指定されない場合は、レコード様式内のすべてのフィールドが消去されます。 *ALL が指定されない場合は、そのレコード様式で出力されるフィールドだけが影響を受けます。 *NOKEY が指定されている場合には、*ALL が指定されていても、キー・フィールドが消去されることはありません。

名前 オペランドは、消去されるレコード様式です。 ワークステーション・ファイル・レコード様式 (ファイル仕様書の 36 から 42 桁目) で、*ALL が指定されていない場合には、用途が出力または入出力共用のフィールドだけが影響を受けます。 この命令によってすべてのフィールドの条件づけ標識が影響を受けます。 CLEAR 命令がレコード様式名に適用され、DDS に INDARA が指定されていると、そのレコード様式内の標識は消去されません。

DISK、SEQ、または PRINTER ファイル・レコード様式内のフィールドが影響を受けるのは、そのレコード様式がプログラム内で出力される場合か、プログラム内にサブプロシージャーが定義されている場合だけです。 *ALL が指定された場合を除き、入力専用フィールドは CLEAR 命令の影響を受けません。

*ALL が指定されたレコード様式の CLEAR 命令は、次の場合には無効です。

- フィールドが入力専用として外部で定義されて、レコードが入力用に使用 されていない場合。
- フィールドが出力専用として外部で定義されて、レコードが出力用に使用 されていない場合。
- フィールドが入出力共用として外部で定義されて、レコードが入力または 出力用に使用されていない場合。

詳しくは、「580 ページの『初期化命令』」を参照してください。

注 : 論理ファイルの入力専用フィールドは、実際にそのファイルに書き出されない 場合でも、出力仕様に現れます。 これらのフィールドが入っているレコードに、*NOKEY が指定されていない CLEAR または RESET が実行されると、 これらのフィールドは、出力仕様に出ているために消去またはリセットされます。

CLEAR の例

- 736 ページの図 303 に、CLEAR 命令の例を示します。
- 737 ページの図 304 に、CLEAR レコード様式のフィールドの初期化の例を示します。
- 864 ページの『RESET の例』 の例は、フィールドに実際に命令が実行される場合を除き、CLEAR にも適用されます。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D DS1             DS
D  Num                     2      5  0
D  Char                   20     30A
D
D MODS            DS                   OCCURS(2)
D  Fld1                    1      5
D  Fld2                    6     10  0

 *  In the following example, CLEAR sets all subfields in the data
 *  structure DS1 to their defaults, CHAR to blank, NUM to zero.
 /FREE
   CLEAR DS1;

  //  In the following example, CLEAR sets all occurrences for the
  //  multiple occurrence data structure MODS to their default values
  //  Fld1 to blank, Fld2 to zero.
   CLEAR *ALL MODS;
 /END-FREE
```

図 *303. CLEAR* 命令

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
A* Field2 and Field3 are defined as output capable fields and can be
A* affected by the CLEAR operation.  Indicator 10 can also be
A* changed by the CLEAR operation even though it conditions an
A* input only field because field indicators are all treated
A* as output fields. The reason for this is that *ALL was not specifie
A* on the CLEAR operation
A*
A*N01N02N03T.Name++++++RLen++TDpBLinPosFunctions++++++++++++++++++++
A          R FMT01
A  10       Field1        10A  I  2 30
A           Field2        10A  O  3 30
A           Field3        10A  B  4 30
A*
A*  End of DDS source
A*

F*Flename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++
FWORKSTN   CF   E             WORKSTN INCLUDE(FMT01)
F
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++
D IN              C                   'INPUT DATA'

 /FREE
   CLEAR FMT01;
   WRITE FMT01;

   // Loop until PF03 is pressed
   DOW NOT *IN03;
      READ FMT01;
      *INLR = %EOF;

      // PF04 will transfer input fields to output fields.
      IF *IN04;
         Field2 = Field3;
         Field3 = Field1;
         CLEAR *IN04;
      ENDIF;
      Field1 = IN;

      // When PF11 is pressed, all the fields in the record format
      // defined as output or both will be reset to the values they
      // held after the initialization step.
      IF *IN11;
         RESET FMT01;
         CLEAR *IN11;
      ENDIF;

      // When PF12 is pressed, all the fields in the record
      // format defined as output or both will be cleared.
      IF *IN12;
         CLEAR FMT01;
         CLEAR *IN12;
      ENDIF;

      IF NOT *IN03;
         WRITE FMT01;
      ENDIF;
   ENDDO;

   *INLR = *ON;
 /END-FREE
```

図 *304. CLEAR* レコード様式のフィールドの初期化

CLOSE (ファイルのクローズ)

自由形式構文	CLOSE{(E) } ファイル名|*ALL

コード	演算項目 1	演算項目 2	結果フィールド	標識		
CLOSE (E)		ファイル名 または *ALL		_	ER	_

明示的な CLOSE 命令では、1 つまたは複数のファイルあるいは装置がクローズされ、それらがモジュールから切り離されます。 そのファイルに対して明示的に OPEN を指定しないかぎり、モジュールの中で再度そのファイルを使用することはできません。 すでにクローズされたファイルに CLOSE 命令を出してもエラーにはなりません。

ファイル名 にはクローズされるファイルを指定します。

キーワード *ALL を指定すると、グローバル・ファイル仕様書で定義されているすべてのファイルを、一度にクローズすることができます。 サブプロシージャーで CLOSE *ALL を指定しても、そのサブプロシージャー内のローカル・ファイルに対しては何の効果もありません。 サブプロシージャー内のすべてのローカル・ファイルをクローズするには、ファイルごとに別々の CLOSE 命令をコーディングする必要があります。 配列またはテーブル・ファイル (ファイル仕様書の 18 桁目の T で識別される) を指定することはできません。

CLOSE 例外 (ファイル状況コードが 1000 より大きい) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、146 ページの『ファイル例外/エラー』を参照してください。

71 桁目、72 桁目、75 桁目、および 76 桁目はブランクでなければなりません。

配列またはテーブルを出力ファイル (TOFILE キーワードを使用して指定) に書き出す場合には、CLOSE 命令によってファイルがクローズされていれば、 LR 時にその配列またはテーブルのダンプは行われません。 ファイルがクロー ズされている場合には、ダンプを行うために再度ファイルをオープンしなけ ればなりません。

詳しくは、「576 ページの『ファイル操作』」を参照してください。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
*  The explicit CLOSE operation closes FILEB.

/FREE
   CLOSE FILEB;

     //  The CLOSE *ALL operation closes all files in the
     //  module. You must specify an explicit OPEN for any file that
     //  you wish to use again.  If the CLOSE operation is not
     //  completed successfully, %ERROR returns '1'.

   CLOSE(E) *ALL;

/END-FREE
```

図 *305. CLOSE* 命令

COMMIT (コミット)

自由形式構文	COMMIT{(E) }{境界 }

コード	演算項目 1	演算項目 2	結果フィールド	標識		
COMMIT (E)	境界			_	ER	_

COMMIT 命令では以下のことが行われます。

- 前のコミットまたはロールバック 869 ページの『ROLBK (ロールバック)』 命令以降 (あるいは前のコミットまたはロールバック命令がなかった場合には、コミットメント制御のもとにある命令の開始以降) の出力命令で指定されたすべての変更が、コミットメント制御用に開かれたファイルに対して行われます。

コミット用にオープンするファイルは、ファイル仕様書に COMMIT キーワードを指定して指定することができます。

- コミットメント制御のもとにあるファイルのすべてのレコード・ロックを 解除します。

ファイルの変更およびレコード・ロックの解除はコミットメント制御のもと にあるすべてのファイルに適用され、その変更が COMMIT 命令を出しているプ ログラムによって要求されているか、または STRCMTCTL コマンドに指定された コミット範囲によって異なる同じ活動化グループまたはジョブ内の別のプログ ラムによって要求されているかには関係がありません。 COMMIT 命令を出しているプログラムでは、ファイルをコミットメント制御のも とに置く必要はありません。 COMMIT 命令では、ファイルの位置は変更されません。

コミットメント制御は、CL コマンド STRCMTCTL が実行された時点で開始され ます。 詳しくは、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」の『コミットメント制御』のセクションを参照してください。

境界 オペランドには、この COMMIT 命令で行われた変更と以後の変更との間の境界を識別するために、定数または変数 (ポインター以外の任意のタイプの) を指定することができます。 境界 が指定されないと、この識別子はヌルになります。

COMMIT 例外 (プログラム状況コード 802 から 805) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。 例えば、コミットメント制御が活動状態で ない場合には、エラーが発生します。エラー処理について詳しくは、163 ページの『プログラム例外/エラー』を参照してください。

詳しくは、「576 ページの『ファイル操作』」を参照してください。

COMP (比較)

自由形式構文	(許可されていない - =、<、<=、>、>=、または <> 演算子を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
COMP	被比較値	被比較値		HI	LO	EQ

COMP 命令は、演算項目 1 を演算項目 2 と比較します。演算項目 1 と演算項目 2 には、リテラル、名前のついた定数、フィールド名、テーブル名、配列要素、データ構造、または形象定数を入れることができます。 演算項目 1 と演算項目 2 は同じデータ・タイプでなければなりません。 比較の結果として、次のように標識が設定されます。

- 高 *(HI): (71* から *72)* 演算項目 1 が演算項目 2 より大きい。
- 低 *(LO): (73* から *74)* 演算項目 1 が演算項目 2 より小さい。
- 等 *(EQ): (75* から *76)* 演算項目 1 が演算項目 2 と等しい。

71 から 76 桁目には、少なくとも 1 つの結果の標識を指定しな ければなりません。 この 3 つの条件全部に同じ標識を指定してはいけません。 指定した場合に、結果の標識は、比較の結果を反映して (それぞれのサイクル に対して) オンまたはオフに設定されます。

COMP 命令に関する規則について詳しくは、567 ページの『比較命令』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  Initial field values are:
 *                    FLDA = 100.00
 *                    FLDB = 105.00
 *                    FLDC = 100.00
 *                    FLDD = ABC
 *                    FLDE = ABCDE
 *
 *  Indicator 12 is set on; indicators 11 and 13 are set off.
C     FLDA          COMP      FLDB                                 111213
 *
 *  Indicator 15 is set on; indicator 14 is set off.
C     FLDA          COMP      FLDB                                 141515
 *
 *  Indicator 18 is set on; indicator 17 is set off.
C     FLDA          COMP      FLDC                                 171718
 *
 *  Indicator 21 is set on; indicators 20 and 22 are set off
C     FLDD          COMP      FLDE                                 202122
```

図 *306. COMP* 命令

DATA-GEN (変数からの文書の生成)

自由形式構文	DATA-GEN{(E)} 受信側 %DATA(文書 {: オプション *1* }) %GEN(生成プログラム {: オプション *2* }); DATA-GEN{(E)} *START %DATA(文書 : オプション *1*) %GEN(生成プログラム {: オプション *2* }); DATA-GEN{(E)} *END %DATA(文書 : オプション *1*) %GEN(生成プログラム {: オプション *2* });

コード	演算項目 1	拡張演算項目 2
DATA-GEN		*receiver* %DATA(文書 {: オプション *1* }) %GEN(生成プログラム {: オプション *2* })
DATA-GEN		*START %DATA(文書 : オプション *1*) %GEN(生成プログラム {: オプション *2* })
DATA-GEN		*END %DATA(文書 : オプション *1*) %GEN(生成プログラム {: オプション *2* })

DATA-GEN 命令は、RPG 変数から構造化文書を生成します。 DATA-GEN は、文書のテキストの生成に生成プログラムまたはプロシージャーを必要とします。

DATA-GEN 命令では、変数の名前と値が生成プログラムに渡されます。この生成プログラムは、コールバック関数を使用して、DATA-GEN 命令に文書のテキストを徐々に渡します。 DATA-GEN 命令は、情報をターゲット RPG 変数またはターゲットの統合ファイル・システム・ファイルに入れます。

DATA-GEN 命令用の生成プログラムの作成について詳しくは、Rational Open Access: RPG 版 トピックを参照してください。 DATA-GEN 命令の生成プログラムの例については、743 ページの『DATA-GEN 生成プログラムの例』を参照してください。

第 1 オペランドは、文書の変数を指定します。 第 1 オペランドが配列の場合、%SUBARR を使用して、DATA-GEN に使用する要素の数を制限することができます。 文書の生成にいくつかの DATA-GEN 命令が必要な場合には、第 1 オペランドを *START または *END に設定することもできます。

第 2 オペランドは %DATA 組み込み関数である必要があり、文書の出力場所と、RPG 変数の情報を生成する方法を制御するオプションを識別します。 %DATA 組み込み関数には、いくつかのオプションがあります。 743 ページの『DATA-GEN 命令コードの %DATA オプション』を参照してください。

%DATA について詳しくは、626 ページの『%DATA (文書 {:オプション})』を参照してください。

第 3 オペランドは %GEN 組み込み関数である必要があり、文書を生成するプログラムまたはプロシージャー、および生成プログラム固有のオプションを識別します。 %GEN について詳しくは、645 ページの『%GEN (生成プログラム {: オプション})』を参照してください。

第 1 オペランドが変数名である場合は、以下のようになります。

- 文書は、変数の名前と値から生成されます。 生成プログラムに渡される名前の大/小文字は、変数またはサブフィールドの定義に使用される大/小文字と同じになります。 変数が外部記述データ構造の場合、サブフィールドが EXTFLD ステートメントで指定されない限り、その名前は大文字になります。 429 ページの『EXTFLD{(フィールド名) }』を参照してください。
- 生成プログラムに渡される変数の最上位の名前は、name オプションを使用して変更できます。

  例えば、変数名が *myDs* で、オプション「name=info」が指定されている場合、文書の最上位レベルの生成プログラムに渡される名前は「info」です。
- 変数がデータ構造の場合は、命令がエラーで終了したときに、一部のサブフィールドの情報が文書に出力されることがあります。 生成プログラムによって DATA-GEN が報告されるとすぐに情報が文書に追加されます。

データ構造のサブフィールドの名前と値は、データ構造定義に現れる順序で生成プログラムに渡されます。

第 1 オペランドが *START の場合、DATA-GEN 命令は統合ファイル・システム内の同じファイルに出力される DATA-GEN 命令のシーケンスを開始します。 第 1 オペランドが *END の場合、DATA-GEN 命令は DATA-GEN 命令のシーケンスを終了します。 742 ページの『複数の DATA-GEN 命令を使用した単一文書の生成』を参照してください。

命令拡張 E を指定すると、以下の状況コードを処理できます。

00352
DATA-GEN オプションが無効です。

00355
生成プログラムまたはプロシージャーを使用できません。

00359
生成プログラムまたはプロシージャーの実行中にエラーが発生しました。

00361
RPG 変数からデータを準備しているときにエラーが発生しました。

00362
生成プログラムによって提供された情報にエラーがありました。

00363
生成プログラムがエラーを検出しました。

00364
出力ファイルの処理中にエラーが発生しました。

00365
DATA-GEN 操作の順序でエラーが発生しました。

注 : 命令拡張は、自由形式構文が使用された場合にのみ指定できます。

状況 00363 の場合、生成プログラムからのエラー・コードは、PSDS の 368 から 371 桁目にあるサブフィールド「外部戻りコード」に入ります。 このサブフィールドは、命令の開始時にゼロに設定され、命令の終了時に生成プログラムによって返される値に設定されます。 エラー・コードの意味は、生成プログラムによって判別されます。

%DATA 組み込み関数のオプション・パラメーターに不明、無効、または無関係のオプションが見つかった場合、状況コード 00352 (DATA-GEN オプションのエラー) で命令は失敗します。 PSDS の外部戻りコード・サブフィールドは、命令の開始時に設定された初期値のゼロから更新されません。

複数の DATA-GEN 命令を使用した単一文書の生成

DATA-GEN 命令のシーケンスを使用して、単一文書を統合ファイル・システム・ファイルに生成することができます。 このファイルは、シーケンスの終わりが検出されるまで開かれたままになります。

DATA-GEN *START
シーケンスは、DATA-GEN *START 命令を使用して開始します。 オプション「doc=file」は、%DATA 組み込み関数の第 2 オペランドに指定する必要があります。

DATA-GEN 変数
*START 命令と *END 命令の間に DATA-GEN 命令を指定します。その際、変数名を第 1 オペランドとして使用し、DATA-GEN *START 命令の %DATA 組み込み関数の第 1 オペランドとして指定されたファイルと同じファイルを指定します。 *START 命令と *END 命令の間の DATA-GEN 命令の %DATA 組み込み関数の 2 番目のパラメーターに「output=continue」を指定する必要があります。

DATA-GEN *END
DATA-GEN *START 命令に対する %DATA 組み込み関数の第 1 オペランドとして指定されたファイルと同じファイルを使用して、DATA-GEN *END 命令でシーケンスを終了します。

シーケンスの DATA-GEN 命令は、さまざまなプロシージャーで実行できます。 ただし、そのすべてを同じ活動化グループおよび同じスレッドで実行する必要があります。

DATA-GEN *START 命令を指定したプロシージャーの呼び出しが、同じ出力ファイルに対して DATA-GEN *END を指定せずに終了した場合、出力ファイルは閉じられます。

DATA-GEN のトレースの取得

生成プログラムとの間で受け渡しされる情報のトレースを取得するには、QIBM_RPG_DATA_GEN_TRACE 環境変数に値「*STDOUT」を設定します。

```
   ADDENVVAR QIBM_RPG_DATA_GEN_TRACE VALUE('*STDOUT')
```

注 : トレース出力がすぐに表示されない場合、または頻繁に明滅して見えづらい場合、以下の ILE RPG プログラムを呼び出して標準出力を表示することができます。 CRTBNDRPG を使用してプログラムをコンパイルします。

```
**FREE
CTL-OPT ACTGRP(*NEW);
DCL-PR printf EXTPROC(*DCLCASE);
  p POINTER VALUE OPTIONS(*STRING : *NOPASS);
END-PR;
DCL-PR getchar INT(10) EXTPROC(*DCLCASE) END-PR;
DCL-C EOL x'15';

printf (EOL);
getchar ();
return;
```

以下の C プログラムは同じ内容を実施します。 CRTBNDC を使用してプログラムをコンパイルします。

```
#include <stdio.h>
main()
{
   printf("\n");
   getchar();
}
```

DATA-GEN 命令コードの %DATA オプション

DATA-GEN 命令をカスタマイズするために使用できるオプションがいくつかあります。 オプションは、%DATA 組み込み関数の第 2 パラメーターとして指定します。 パラメーターには定数または変数の式を使用できます。 オプションは、「opt1=val1 opt2=val2」の形式で指定します。

オプションの指定方法について詳しくは、626 ページの『%DATA (文書 {:オプション})』を参照してください。

- *doc* オプションは、%DATA 組み込み関数の最初のパラメーターが文書を受け取る変数であるか、または文書を受け取るファイルの名前であるかを指定します。
- *countprefix* オプションは、カウントされる配列サブフィールドの要素をどのくらい生成するかを示すサブフィールドの接頭部、または非配列サブフィールドを生成するかどうかを指定します。
- *fileccsid* オプションは、出力ファイルが存在しない場合に、出力ファイル作成時に使用される CCSID を指定します。
- *name* オプションは、文書の最上位に使用される名前を指定します。
- *output* オプションは、命令が開始される前に出力変数またはファイルをクリアするかどうかどうか、または変数あるいはファイル内の既存のデータに新規データを付加するかどうかを指定します。
- *renameprefix* オプションはサブフィールドの接頭部を指定します。この接頭部は、サブフィールド自体の名前ではなく、サブフィールドに対して生成される名前を指定します。
- *trim* オプションは、文字、UCS-2、およびグラフィック・データからブランクを削除してから生成プログラムに渡すかどうかを指定します。

DATA-GEN 生成プログラムの例

DATA-INTO 生成プログラムの詳細な例については、Rational Open Access: RPG 版 のトピックを参照してください。

一部のサンプル生成プログラムは、ライブラリー QOAR の SAMPLE ファイルにも提供されています。 これらは、そのまま使用することも、別のライブラリーのソース・ファイルにコピーし、必要に応じて変更することもできます。

- GENHTMLTAB は、HTML テーブルの生成プログラムです。
- GENPROP は、プロパティー・ファイルの生成プログラムです。

DATA-INTO (文書の変数への構文解析)

自由形式構文	DATA-INTO{(EH)} 受信側 %DATA(文書 {: オプション *1* }) %PARSER(パーサー {: オプション *2* }); DATA-INTO{(EH)} %HANDLER(*handlerProc* : 共通域) %DATA(文書 {: オプション *1* }) %PARSER(パーサー {: オプション *2* });

コード	演算項目 1	拡張演算項目 2
DATA-INTO		*receiver* %DATA(文書 {: オプション *1* }) %PARSER(パーサー {: オプション *2* })
DATA-INTO		%HANDLER(*handlerProc* : 共通域) %DATA(文書 {: オプション *1* }) %PARSER(パーサー {: オプション *2* })

DATA-INTO 命令は、構造化文書から RPG 変数にデータをインポートします。 DATA-INTO 命令は XML-INTO と似ていますが、XML-INTO が処理できるのは XML 文書のみである一方、 DATA-INTO はどの構造化文書も処理できる点が異なります。 また DATA-INTO は、文書のデータを構文解析できるパーサーを必要とする点も XML-INTO とは異なります。

DATA-INTO 命令は文書テキストをパーサーに渡します。パーサーはコールバック関数を使用して文書内のデータの名前と値を段階的に DATA-INTO 命令に渡します。 DATA-INTO 命令はその情報をターゲット RPG 変数に入れます。

DATA-GEN 命令用のパーサーの作成について詳しくは、Rational Open Access: RPG 版 トピックを参照してください。

DATA-INTO 命令は、以下の 2 つの方法で作動します。

- RPG 変数に直接データを読み取る
- %HANDLER(handlerProc) により指定されたプロシージャーに渡される配列パラメーターに対して段階的にデータを読み取る

第 1 オペランドには、構文解析されるデータのターゲットを指定します。 変数名または %HANDLER 組み込み関数を使用できます。

第 2 オペランドは %DATA 組み込み関数にする必要があり、構文解析される文書、および RPG 変数を設定するための情報の使用方法を制御するオプションを識別します。 %DATA について詳しくは、626 ページの『%DATA (文書 {:オプション})』を参照してください。

第 3 オペランドは %PARSER 組み込み関数にする必要があり、この関数が構文解析を行うプログラムとプロシージャー、およびパーサー固有のオプションを識別します。 %PARSER について詳しくは、675 ページの『%PARSER(パーサー {: オプション})』を参照してください。

916 ページの『XML-INTO および DATA-INTO の RPG 変数にデータを転送する場合の規則』を参照してください。

第 1 オペランドが変数名である場合は、以下のようになります。

- 構文解析は、変数に対して直接実行される。
- パーサーが最初に見つけた項目の名前を報告するかどうかはオプションである。 ただし、パーサーが最初に見つけた項目の名前を報告する場合、その名前は変数の名前と同じであると期待されます。 これは、path オプションを使用してオーバーライドすることができます。

  例えば、変数名が *MYDS* であり、パーサーが最外部構造の名前を「order」と報告した場合、 %DATA のオプションで「path=order」を指定する必要があります。

  必要な情報が文書内に深くネストされている場合、「path」オプションを使用して情報を見つける必要があります。 例えば、文書の最外部構造の名前が「info」であり、 「info」構造内にネストされた「order」という名前の構造の中に必要な情報が存在する場合、%DATA のオプションには「path=info/order」を指定する必要があります。
- 変数がデータ構造である場合は、その命令がエラーで終了した場合でも、一部のサブフィールドはその命令により設定される。
- 変数が配列である場合、構文解析は配列に適合する量のデータに対してのみ検索を行う。 PSDS の 372 から 379 桁目にある「XML-INTO または DATA-INTO に設定される要素の数」サブフィールドには、命令により正常に設定された要素の数が設定されます。 データ構造の配列の場合、その要素のサブフィールドのデータの構文解析中に構文解析エラーが発生したときは、この値には設定中の要素が含まれません。ただし、この配列要素には、命令により設定されたそのサブフィールドの一部が含まれています。

第 1 オペランドが %HANDLER 組み込み関数である場合は、以下のようになります。

- %HANDLER の第 1 オペランドとして指定されたプロシージャーは、そ のプロシージャーにより処理される RPG 配列要素の指定された数を充てんす るのに十分な数のデータがパーサーにより構文解析されると呼び出される。 ハンドラーから戻ると、配列要素の指定された数が再度充てんされ、処理プロシージャーを呼び出すために十分な数のデータが構文解析されるまで、パーサーはデータの構文解析を継続します。 これは、その文書が完全に構文解析されるまで、または構文解析が停止されたことを示す戻りコードをそのプロシージャーが戻すまで継続されます。

  処理プロシージャーの最後の呼び出しでは、RPG 配列要素数が処理プロシージャーが処理する数より少ない場合があります。 処理プロシージャーは常に「要素の数」パラメーターを参照し、データがない配列要素にアクセスしないようにする必要があります。

  %HANDLER の第 2 オペランドとして指定される通信域変数は、処理プロシ ージャーへの最初のパラメーターとしてパーサーにより受け渡されます。 これにより、DATA-INTO 命令をコーディングするプロシー

ジャーは処理プロ シージャーと通信でき、さらに処理プロシージャーはある呼び出しから次の呼び出しに情報を保管できます。

- プロシージャーの配列パラメーターの保持に使用される一時変数の各要素は、そのデフォルト値に初期化された後にデータからロードされる。
- 検索対象の文書内の項目の名前を指定するには、*path* オプションを使用する必要がある。 *path* オプションについて詳しくは、747 ページの『DATA-INTO 命令コードの %DATA オプション』および 747 ページの『期待される DATA-INTO の形式』を参照してください。
- 配列処理プロシージャーは、DATA-INTO 命令の実行中に複数回呼び出される場合がある。 パーサーが第 2 パラメーターの DIM キーワードにより指定された要素の数を検出すると、そのプロシージャーが呼び出されます。 そのプロシージャーが呼び出される最後の回では、要素の数が DIM キーワードで指定された数より少ない場合があります。 要素がない場合、そのプロシージャーは呼び出されません。

  処理プロシージャーでは、以下のパラメーター数および戻り値の型を指定 する必要があります。

パラメーター数または戻り値	データ型および引き渡しモード	記述
戻り値	4 バイト整数 (10I 0)	戻り値がゼロの場合、構文解析が継続することを示します。 戻り値がその他の値の場合、構文解析が終了することを示します。
1	任意の型、参照による受け渡し	DATA-INTO 命令とハンドラーの間、およびハンドラーの連続呼び出しの間の通信に使用される。
2	配列またはデータ構造の配列、読み取り専用の参照による受け渡し (CONST キーワード)	配列要素には、*path* オプションにより指定された項目からのデータが含まれています。
3	4 バイト符号なし整数 (10U 0)、値による受け渡し	データを表す第 2 パラメーター内の配列要素の数。

- %HANDLER について詳しくは、648 ページの『%HANDLER (handlingProcedure : communicationArea)』を参照してください。

データ構造のサブフィールドは、文書内にある順序で設定されます。 その順序は、データ構造内でサブフィールドのオーバーラップがある場合に重要になることがあります。

%NULLIND は、DATA-INTO 命令の実行中にはフィールドまたはサブフィールドに対して更新されません。

命令拡張 H を指定すると、数値データを四捨五入して代入できます。 命令拡張 E を指定すると、以下の状況コードを処理できます。

00352
無効な DATA-INTO オプション。

00354
構文解析の準備時エラー。

00355
パーサー・プログラムおよびプロシージャーは使用可能ではありません。

00356
この文書は RPG 変数に一致しません。

00357
パーサーが文書内にエラーを検出しました。

00358
これは、パーサーにより提供された情報内のエラーでした。

00359
パーサー・プログラムまたはプロシージャーを実行中にエラーが発生しました。

注 : 命令拡張は、自由形式構文が使用された場合にのみ指定できます。

状況 00357 の場合、パーサーからのエラー・コードは、PSDS の 368 から 371 桁目のサブフィールド「外部戻りコード」に配置されます。 このサブフィールドは、命令の開始時にゼロに設定され、命令の終了時にはパーサーにより戻された値に設定されます。 エラー・コードの意味はパーサーによって判別されます。

%DATA 組み込み関数のオプション・パラメーターにおいて不明、無効、または無関係のオプションが検出された場合、その命令は状況コード 00352 (DATA-INTO オプション・エラー) で失敗します。 PSDS の外部戻りコード・サブフィールドは、命令の開始時に設定された初期値のゼロから更新されません。

文書では、文書内の項目の各名前に関して RPG 変数と一致することが必要です。

- RPG データ構造のデータには、そのデータ構造と同じ名前の項目、および RPG サブフィールドと同じ名前のネストされた項目が含まれている必要があります。
- RPG 配列のデータは、通常はパーサーによって配列として報告されます。 ただし、RPG 配列と同じ名前の一連の項目がある場合も有効です。

path オプションを使用すると、指定された変数の名前と一致する項目の名前を設定できます。 ただし、指定された変数のサブフィールドに一致する項目の名前の設定には使用できません。 例えば、変数 DS1 にサブフィールド SF1 がある場合、DS1 の項目に は任意の名前を使用できますが、SF1 の項目には名前「sf1」(または *case* オプションに応じて「SF1」、「Sf1」など) を使用する必要があります。

文書が RPG 変数に一致しない場合、例えば文書にデフォルト のパスまたは指定されたパスがない場合、または RPG データ構造のサブフィ ールドに一致する一部の項目が欠落している場合、DATA-INTO 命令は状況 00356 で失敗します。 *allowextra*、*allowmissing*、および *countprefix* オプションを使用すると、項目のデータを RPG 変数を完 全に設定するために必要なデータよりも多くするかまたは少なくするかを指定できます。

項目のデータがそれに一致する RPG 変数の型に対して無効である場合は、その命令は状況コード 0356 で失敗します。 代入エラーに固有の状況コードがメッセージ RNX0356 の置換テキストに表示されます。

ヒント: 日付や数値などの型の RPG フィール ドにデータを正常に代入できなかったことで失敗する DATA-INTO 命令を防止するには、文字または UCS-2 の型のすべてのサブフィールドに受信側変数を定義します。それにより、データは、変換組み込み関数 %DATE や %INTT などを使用して RPG プログラムにより別のデータ型に変換されます。

DATA-INTO 命令のパーサーの例については、752 ページの『DATA-INTO パーサーの例』を参照してください。

パーサーのトレースを取得するには、値「*STDOUT」を使用して QIBM_RPG_DATA_INTO_TRACE_PARSER 環境変数を設定してください。

```
ADDENVVAR QIBM_RPG_DATA_INTO_TRACE_PARSER VALUE('*STDOUT')
```

注 : トレース出力がすぐに表示されない場合、または頻繁に明滅して見えづらい場合、以下の ILE RPG プログラムを呼び出して標準出力を表示することができます。 CRTBNDRPG を使用してプログラムをコンパイルします。

```
**FREE
CTL-OPT ACTGRP(*NEW);
DCL-PR printf EXTPROC(*DCLCASE);
  p POINTER VALUE OPTIONS(*STRING : *NOPASS);
END-PR;
DCL-PR getchar INT(10) EXTPROC(*DCLCASE) END-PR;
DCL-C EOL x'15';

printf (EOL);
getchar ();
return;
```

以下の C プログラムは同じ内容を実施します。 CRTBNDC を使用してプログラムをコンパイルします。

```
#include <stdio.h>
main()
{
   printf("\n");
   getchar();
}
```

DATA-INTO 命令コードの %DATA オプション

オプションを使用して、DATA-INTO 命令をカスタマイズできます。 オプションは、%DATA 組み込み関数の第 2 パラメーターとして指定します。 パラメーターには定数または変数の式を使用できます。 オプションは、「opt1=val1 opt2=val2」の形式で指定します。

オプションの指定方法について詳しくは、626 ページの『%DATA (文書 {:オプション})』を参照してください。

- *path* オプションは、要求される情報の文書内の場所を指定します。
- *doc* オプションは、 %DATA 組み込み関数の最初のパラメーターが文書であるか、または文書が含まれるファイルの名前であるかを指定します。
- *ccsid* オプションは、 文書の構文解析に使用される CCSID を指定します。
- *case* オプションは、 RPG フィールド名および path オプション内の名前に一致する名前を 検索する際に DATA-INTO が DATA 文書内の要素名および属性名を 解釈する方法を指定します。
- *trim* オプションは、 RPG 変数へ代入する前に、データからブランク、タブ、および行終了文字をトリムするかどうかを指定します。
- *allowmissing* オプションは、 データ構造のすべての RPG サブフィールドのデータを提供するために必要な 数の情報が文書にない場合に、RPG ランタイムが 状態を処理する方法を指定します。
- *allowextra* オプションは、 RPG 変数の設定に必要ない追加の情報が文書にある場 合に、RPG ランタイムが状態を処理する方法を指定します。
- データ・サブフィールド ・オプションは、 RPG データ構造に一致する項目にテキスト・データがある状態を処理するために使用される、追加のサブフィールドの名前を指定します。
- *countprefix* オプションは、 DATA-INTO 命令で設定された RPG 配列要素の数を受け取る追加サブフィールドの名前の接頭部を指定します。

期待される DATA-INTO の形式

文書内のデータの構造は、RPG 変数の構造と一致することが必要です。

- RPG 変数に一致する名前付きデータのネスティング・レベルは、文書の任意のレベルに設定可能ですが、その名前付きデータが文書の想定されたネスティング・レベルにない場合は *path* オプションを指定する必要があります。 *path* オプションが指定されない場合は、以下の前提条件が適用されます。
  - パーサーが文書の最外部項目の名前を報告しない場合、ターゲット変数の名前が想定されます。 例えば、DATA-INTO 命令のターゲットが myDs.mySubfield である場合、 RPG により文書の最外部の項目の名前は「MYSUBFIELD」であると想定されます。
  - パーサーが文書の最外部項目の名前を報告する場合、その名前は、DATA-INTO 命令のターゲット変数の名前と一致する必要があります。
- RPG 変数がデータ構造の場合、文書はサブフィールドを含む構造を表すことも必要です。 パーサーは構造を検出したことをまず報告し、その後に構造のサブフィールドの名前と値を報告する必要があります。 文書内のサブフィールド情報の順序は、RPG データ構造内のサブフィールドの順序と一致する必要はありません。
- RPG 配列に一致する項目は、同じ親項目の子である必要があります。 それらの子項目が文書内で順番に報告される必要はなく、別の項目を挿入することも可能です。

このセクションの例では、架空の文書形式を使用しています。 大部分の行は名前で開始し、その後に「StartStruct」(構造の場合)、「StartArray」(配列の場合)、または引用符で囲んだ値 (スカラー値の場合) のいずれかが続きます。 配列の値には名前がありません。

以下にこの架空の形式の文書例を示します。 同等の XML 文書と JSON 文書も表示します。

架空の形式	同等の XML 文書	同等の JSON 文書
`info StartStruct` `  team "A"` `  leader "Jack"` `  members StartArray` `  "Mary"` `  "Adam"` `  EndArray` `EndStruct`	`<info>info` `  <team>A</team>` `  <leader>Jack</leader>` `  <members>Mary</members>` `  <members>Adam</members>` `</info>`	`"info": {` `  "team" : "A",` `  "leader" : "Jack",` `  "members" : [` `  "Mary",` `  "Adam" ] }`

スカラー変数

スカラー変数には、スタンドアロン・フィールド (1)、テーブル名 (2)、配列の要素 (3)、またはサブフィールド (4) を指定できます。 4).

```
DCL-S libname CHAR(10);
DCL-S tab01 CHAR(10) DIM(3);
DCL-S arr CHAR(10) DIM(3);
DCL-DS ds;
   subfield CHAR(10);
END-DS;
DCL-DS qualDs QUALIFIED;
   subfield CHAR(10);
END-DS;

DATA-INTO libname %DATA(document : options)  // 1
                  %PARSER(parser : parserOptions);

DATA-INTO tab01 %DATA(document : options)  // 2
                %PARSER(parser : parserOptions);

DATA-INTO arr(1) %DATA(document : options)  // 3
                 %PARSER(parser : parserOptions);

DATA-INTO subfield %DATA(document : options)  // 4
                   %PARSER(parser : parserOptions);

DATA-INTO qualDs.subfield %DATA(document : options)  // 4
                          %PARSER(parser : parserOptions);
```

架空言語のサンプル文書	%DATA の *path* オプション
`"MYLIB"`	ブランク
`library "MYLIB"`	'path=library'
`info StartStruct` `  library "MYLIB"` `EndStruct`	'path=info/library'

単純なデータ構造または複数回繰り返しデータ構造

```
DCL-DS pgm;
   name CHAR(10);
   lib CHAR(10);
END-DS;
```

```
OR

DCL-DS pgm OCCURS(5);
   name CHAR(10);
   lib CHAR(10);
END-DS;

DATA-INTO pgm %DATA(doc : option)
              %PARSER(parser : parserOption);
```

架空言語のサンプル文書	%DATA の *path* オプション
`StartStruct` `  name "data"` `  lib "data"` `EndStruct`	ブランク
`pgm StartStruct` `  lib "data"` `  name "data"` `EndStruct`	ブランク
`program StartStruct` `  name "data"` `  lib "data"` `EndStruct`	'path=program'
`api StartStruct` `  program StartStruct` `  name "data"` `  lib "data"` `  EndStruct` `EndStruct`	'path=api/program'

スカラー型の配列

```
DCL-S sites CHAR(25) DIM(3);

DATA-INTO sites %DATA(doc : option)
                %PARSER(parser : parserOption);
```

架空言語のサンプル文書	%DATA の *path* オプション
`StartArray` `  "data 1"` `  "data 2"` `  "data 3"` `EndArray`	ブランク
`info StartStruct` `  custsites "data 1"` `  custsites "data 2"` `  custsites "data 3"` `EndStruct`	'path=info/custsites'

データ構造の配列

```
DCL-DS pgm QUALIFIED DIM(3);
```

```
  name CHAR(10);
  lib CHAR(10);
END-DS;

DATA-INTO pgm %DATA(doc : option)
              %PARSER(parser : parserOption);
```

架空言語のサンプル文書	%DATA の *path* オプション
`StartArray` `  StartStruct` `  name "name1"` `  lib "lib1"` `  EndStruct` `  StartStruct` `  name "name2"` `  lib "lib2"` `  EndStruct` `  StartStruct` `  name "name3"` `  lib "lib3"` `  EndStruct` `EndArray`	ブランク
`programs StartStruct` `  pgm StartArray` `  StartStruct` `  name "name1"` `  lib "lib1"` `  EndStruct` `  StartStruct` `  name "name2"` `  lib "lib2"` `  EndStruct` `  StartStruct` `  name "name3"` `  lib "lib3"` `  EndStruct` `  EndArray` `EndStruct`	'path=programs/pgm'

複雑なデータ構造

```
DCL-DS dtaaraInfo QUALIFIED;
   DCL-DS dtaara;
     name CHAR(10);
     lib CHAR(10);
   END-DS;
   type INT(10);
   value CHAR(100);
END-DS;

DATA-INTO dtaaraInfo %DATA(doc : option)
                     %PARSER(parser : parserOption);
```

架空言語のサンプル文書	%DATA の *path* オプション
`dtaarainfo StartStruct` `  type "data"` `  value "data"` `  dtaara StartStruct` `  name "data"` `  lib "data"` `  EndStruct` `EndStruct`	ブランク

架空言語のサンプル文書	%DATA の *path* オプション
`` sys StartStruct sysName "data" obj StartStruct dta StartStruct dtaara StartStruct name "data" lib "data" EndStruct type "data" value "data" EndStruct EndStruct EndStruct ```	'path=sys/obj/dta'

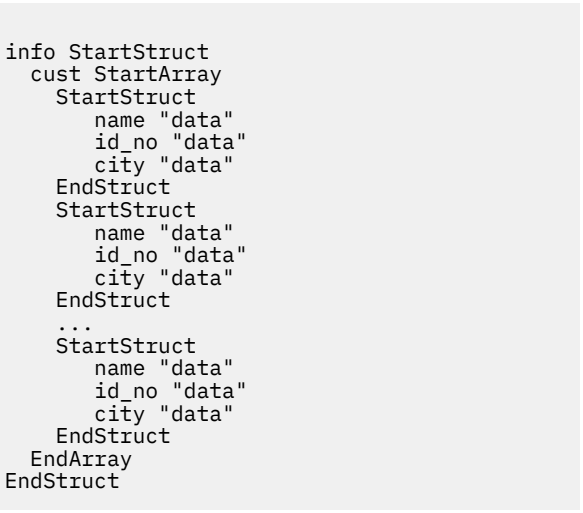
データ構造の配列がある %HANDLER プロシージャー

```
DCL-DS myCommArea;
   total UNS(20);
END-DS;
DCL-DS custType QUALIFIED;
   name VARCHAR(50);
   id_no INT(10);
   city CHAR(20);
END-DS;
DCL-PR custHdlr;
   commArea LIKEDS(myCommArea);
   custinfo LIKEDS(custType) DIM(5) CONST;
   numElems UNS(10) CONST;
END-PR;

DATA-INTO %HANDLER(custHdlr : myCommArea);
          %DATA(doc : option)
          %PARSER(parser : parserOption);
```

注 : %HANDLER が指定された場合には path オプションが必要です。

架空言語のサンプル文書	*path* オプション
`` cust StartArray StartStruct name "data" id_no "data" city "data" EndStruct StartStruct name "data" id_no "data" city "data" EndStruct ... StartStruct name "data" id_no "data" city "data" EndStruct EndArray ```	'path=cust'

架空言語のサンプル文書	*path* オプション
 `info StartStruct` `  cust StartArray` `  StartStruct` `  name "data"` `  id_no "data"` `  city "data"` `  EndStruct` `  StartStruct` `  name "data"` `  id_no "data"` `  city "data"` `  EndStruct` `  ...` `  StartStruct` `  name "data"` `  id_no "data"` `  city "data"` `  EndStruct` `  EndArray` `EndStruct`	'path=info/cust'

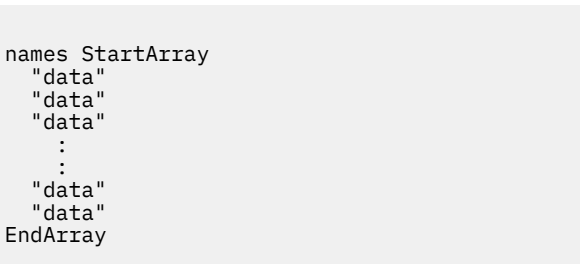
スカラー型の配列があるハンドラー・プロシージャー

```
DCL-S total UNS(20);

DCL-PR nameHdlr;
   commArea LIKEDS(myCommArea);
   names CHAR(10) DIM(5) CONST;
   numNames UNS(10) CONST;
END-PR;

DATA-INTO %HANDLER(nameHdlr : total);
          %DATA(doc : option)
          %PARSER(parser : parserOption);
```

注 : %HANDLER が指定された場合には path オプションが必要です。

架空言語のサンプル文書	*path* オプション
 `names StartArray` `  "data"` `  "data"` `  "data"` `  :` `  :` `  "data"` `  "data"` `EndArray`	'path=names'

DATA-INTO パーサーの例

DATA-INTO パーサーの詳細な例については、Rational Open Access: RPG 版 のトピックを参照してください。

一部のサンプル・パーサーは、ライブラリー QOAR の SAMPLE ファイルにも提供されています。 サンプル・パーサーをそのまま使用することも、コピーして必要に応じて変更することも可能です。

- PARSJSON は JSON のパーサーです。
- PARSPROP1 は、プロパティー・ファイルのパーサーです。

DEALLOC (記憶域の解放)

自由形式構文	DEALLOC {(EN) } ポインター名

コード	演算項目 1	演算項目 2	結果フィールド	標識		
DEALLOC (E/N)			ポインター名	_	ER	_

DEALLOC 命令は、動的記憶域の割り振りの直前の 1 つを解放します。ポインター名 は、動的記憶域割り振り命令 (RPG 内の ALLOC 命令、または他の何らかの動的記憶域割り振りメカニズム) によって直前に設定された値である必要があるポインターです。 このポインターは、動的記憶域を単純に指し示すだけのものでは ありません。割り振りの始めに設定されていることも必要です。

このポインターによって指し示されている記憶域は、このプログラムによって、 あるいは活動化グループ内の他のプログラムによって、これ以降の割り振りの ために解放されます。

命令コード拡張 N が指定されている場合、ポインターは再割り振りが正常に 行われた後、*NULL に設定されます。

DEALLOC 例外 (プログラム状況コード 426) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。 エラーが発生した場合、'N' が指定されていても、結果フィールドのポインターは変更されません。エラー処理について詳しくは、163 ページの『プログラム例外/エラー』 を参照してください。

ポインター名 は、基底ポインター・スカラー変数 (独立フィールド、データ構造サブフィールド、テーブル名、または配列要素) でなければなりません。

このポインターがすでに *NULL であれば、実行時にエラーには なりません。

モジュールの RPG メモリー管理命令が、制御仕様書の ALLOC キーワードにより、単一レベル・ヒープ記憶域を使用している場合、 DEALLOC 命令は、単一レベル・ヒープ記憶域へのポインターのみを処理できます。 モジュールの RPG メモリー管理命令が、テラスペース・ヒープ記憶域を使用している場合、DEALLOC 命令は、単一レベルとテラスペースの両方のヒープ記憶域へのポインターを処理できます。

詳しくは、581 ページの『メモリー管理命令』 を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
 *
D Ptr1            S               *
D Fld1            S              1A
D BasedFld        S              7A   BASED(Ptr1)

 /FREE
    // 7 bytes of storage are allocated from the heap and
    // Ptr1 is set to point to it
    Ptr1 = %alloc (7);

    // The DEALLOC frees the storage.  This storage is now available
    // for allocation by this program or any other program in the
    // activation group.  (Note that the next allocation may or
    // may not get the same storage back).
    dealloc Ptr1;

    // Ptr1 still points at the deallocated storage, but this pointer
    // should not be used with its current value.  Any attempt to
    // access BasedFld which is based on Ptr1 is invalid.
    Ptr1 = %addr (Fld1);

    // The DEALLOC is not valid because the pointer is set to the
    // address of program storage.  %ERROR is set to return '1',
    // the program status is set to 00426 (%STATUS returns 00426),
    // and the pointer is not changed.
    dealloc(e) Ptr1;

    // Allocate and deallocate storage again.  Since operational
    // extender N is specified, Ptr1 has the value *NULL after the
    // DEALLOC.
    Ptr1 = %alloc (7);
    dealloc(n) Ptr1;
 /END-FREE
```

図 *307. DEALLOC* 命令

DEFINE (フィールド定義)

自由形式構文	(許可されていない - 定義仕様書で LIKE キーワードまたは DTAARA キーワードを使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
DEFINE	*LIKE	被参照フィールド	定義されるフィールド			
DEFINE	*DTAARA	外部データ区域	内部フィールド			

演算項目 1 の指定によって、宣言の DEFINE 命令では次のいずれかを実行す ることができます。

- フィールドを、別のフィールドの属性 (長さおよび小数点以下の桁数) に基づいて定義する 。
- フィールドを、データ域として定義する 。

DEFINE 命令は演算内の任意の場所に指定できますが、サブプロシージャー内で *DTAARA DEFINE を指定することも、これを UCS-2 結果フィールドで使用することもできません。 制御レベル の指定 (7 から 8 桁目) はブランクにするか、 あるいは L1 から L9 標識、LR 標識、または L0 項目を入れてプログラムの該当する セクション内のステートメントをグループにまとめることができます。 制御レベルの指定は文書化のために使用されるだけです。 条件付け標識の指定 (9 から 11 桁目) は使用できません。

*LIKE DEFINE

演算項目 1 に *LIKE がある 754 ページの『DEFINE (フィールド定義)』 命令は、別のフィールドの属性 (長さおよび小数点以下の桁数) に基づいてフィールドを定義します。

演算項目 2 には参照するフィールドの名前を、結果フィールドには定義する フィールドの名前を入れなければなりません。 演算項目 2 に指定されるフィー ルドはプログラム内または外部で定義することができ、定義するフィールドの 属性を入れます。 演算項目 2 は、リテラル、名前付き定数、浮動数値フィールド、またはオブジェクトにすることはできません。 演算項目 2 が配列、配列要素、またはテーブル名の場合には、 配列またはテーブルの要素の属性がそのフィールドの定義に使用されます。 結果フィールドを配列、配列要素、データ構造、またはテーブル名にすること はできません。 ALTSEQ(*NO)、NOOPT、ASCEND、CONST またはヌル値可能などの属性 は、演算項目 2 から、結果フィールドによって引き継がれません。 データ・タイプ、 長さ、小数点以下の桁数だけが引き継がれます。

64 から 68 桁目 (フィールドの長さ) は、結果フィールドの指定を演算項目 2 の 指定より長くしたり短くするために使用することができます。 数字の前のプラス符号 (+) は増やす長さを示し、マイナス符号 (-) は減らす長さを示します。 65 から 68 桁目には、長さの増減 (右寄せ) を入れるか、あるいはブランクにする ことができます。 64 から 68 桁目がブランクの場合には、結果フィールドの指定は 演算項目 2 の指定と同じ長さで定義されます。 定義するフィールドの小数点以下の桁数を変更することはできません。 フィールド長の指定を使用できるのは、図形、UCS-2、数値、および文字フィールドの場合だけです。

図形フィールドまたは UCS-2 フィールドの場合には、フィールド長の 差は 2 バイト文字で計算されます。

演算項目 2 が図形フィールドまたは UCS-2 フィールドの 場合、結果フィールドは同じタイプ、すなわち、図形または UCS-2 として定義 されます。 新しいフィールドには、モジュールのデフォルトの図形 CCSID または UCS-2 CCSID が使用されます。 新しいフィールドに、演算項目 2 のフィールドと同じ CCSID を使用したい 場合には、定義仕様書で LIKE キーワードを使用します。 長さの調整は、2 バイト単位で表されます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
*
 *  FLDA is a 7-position character field.
 *  FLDB is a 5-digit field with 2 decimal positions.
 *
 *
 *  FLDP is a 7-position character field.
C     *LIKE         DEFINE    FLDA          FLDP
 *
 *  FLDQ is a 9-position character field.
C     *LIKE         DEFINE    FLDA          FLDQ             +2
 *
 *  FLDR is a 6-position character field.
C     *LIKE         DEFINE    FLDA          FLDR            - 1
 *
 *  FLDS is a 5-position numeric field with 2 decimal positions.
C     *LIKE         DEFINE    FLDB          FLDS
 *
 *  FLDT is a 6-position numeric field with 2 decimal positions.
C     *LIKE         DEFINE    FLDB          FLDT           +  1
 *
 *  FLDU is a 3-position numeric field with 2 decimal positions.
C     *LIKE         DEFINE    FLDB          FLDU            - 2
 *
 *  FLDX is a 3-position numeric field with 2 decimal positions.
C     *LIKE         DEFINE    FLDU          FLDX
```

図 *308. *LIKE* を指定した *DEFINE* 命令

数値フィールドの *LIKE DEFINE について、以下の点に注意してください。

- そのフィールドが定義仕様書で完全に定義されている場合、*LIKE DEFINE に よって形式は変更されません。
- そうでない場合、フィールドがデータ構造のサブフィールドであれば、 そのフィールドはゾーン形式で定義されます。
- 上記が該当しない場合、フィールドはパック形式で定義されます。

```
D           DS
D  Fld1
D  Fld2     S              7P 2
 *
 * Fld1 will be defined as zoned because it is a subfield of a
 * data structure and numeric subfields default to zoned format.
 *
C    *LIKE      DEFINE      Fld2       Fld1
 *
 * Fld3 will be defined as packed because it is a standalone field
 * and all numeric items except subfields default to packed format.
C    *LIKE      DEFINE      Fld1       Fld3
```

図 *309. *LIKE DEFINE* の使用

*DTAARA DEFINE

演算項目 1 に *DTAARA がある 754 ページの『DEFINE (フィールド定義)』 命令は、フィールド、データ構造、データ構造サブフィールド、またはデータ域データ構造 (ILE RPG プログラム内にある) を、システム上の *DTAARA オブジェクト (ILE RPG プログラムの外側にある) と関連付けます。

注 : サブプロシージャー内、または UCS-2 結果フィールド では *DTAARA DEFINE を使用できません。

演算項目 2 にはデータ域の外部名を指定します。 内部データ域の名前には *LDA を使用し、プログラム初期化パラメーター (PIP) データ域には *PDA を使用します。 演算項目 2 をブランクのままにした場合には、結果フィールドの指定は RPG IV 名とそのデータ域の外部名の両方になります。

結果フィールドには、プログラム内で定義したフィールド、データ構造、デ ータ構造サブフィールド、またはデータ域データ構造のいずれかの名前を指定 します。 演算項目 2 に指定されたデータ域からデータを検索したりこのデータ域にデータを書き出したりするには、この名前を IN 命令および OUT 命令と一緒に使用します。 結果フィールドにデータ域データ構造を指定すると、ILE RPG プログラムは、暗黙のうちにプログラムの開始時にデータ域からデータを検索して、プログラムの終了時にデータ域にデータを書き出します。

結果フィールドの指定を、プログラム状況データ構造、ファイル情報データ 構造 (INFDS)、複数オカレンス・データ構造、入力レコード・フィールド、配列、 配列要素、またはテーブルの名前にすることはできません。 また、複数オカレンス・データ構造のサブフィールド、データ域データ構造、プログラム状況データ 構造、ファイル情報データ構造 (INFDS)、またはすでに *DTAARA DEFINE ステ ートメントに現れているかあるいは定義仕様書で DTAARA キーワードを使用し てすでにデータ域として定義されているデータ構造の名前にすることもできま せん。

3 種類のデータ域を作成することができます。

- *CHAR 文字
- *DEC 数値
- *LGL 論理

さらに、リモート・システム上の上記の 3 つのタイプのいずれかのデータ域を 指し示す DDM データ域 (タイプ *DDM) も作成することができます。

データ域に関連付けることができるのは、文字と数値タイプ (浮動数値を除く) だけ です。 システム上の実際のデータ域は、プログラム内のフィールドと同じタイプ で、長さと小数点以下の桁数も同じでなければなりません。 標識フィールドは論理データ域または文字データ域と 関連付けることができます。

数値データ域の場合、最大長は、9 桁の小数部を含む 24 桁です。 小数部の左側には、小数部の桁数が 9 桁に満たない場合でも、最大 15 桁は あることに注意してください。

64 から 70 桁目には、結果フィールドの指定項目の長さおよび小数点以下の桁数 を定義することができます。 これらの指定は、演算項目 2 に指定されたデータ域の外部記述と一致している必要があります。 内部データ域は長さが 1024 桁の文字データですが、プログラムの中では長さが 1024 桁またはそれ以下のものとして内部データ域にアクセスすることができます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  The attributes (length and decimal positions) of
 *  the data area (TOTGRS) must be the same as those for the
 *  external data area.
C
C     *DTAARA       DEFINE                  TOTGRS           10 2
C
 *
 *  The result field entry (TOTNET) is the name of the data area to
 *  be used within the ILE RPG program.  The factor 2 entry (TOTAL)
 *  is the name of the data area as defined to the system.
C
C     *DTAARA       DEFINE    TOTAL         TOTNET
C
 *
 *  The result field entry (SAVTOT) is the name of the data area to
 *  be used within the ILE RPG program.  The factor 2 entry (*LDA)
 *  indicates the use of the local data area.
C
C     *DTAARA       DEFINE    *LDA          SAVTOT
```

図 *310. *DTAARA* を指定した *DEFINE* 命令

DELETE (レコードの削除)

自由形式構文	DELETE{(EHMR) }{検索引数 } 名前

コード	演算項目 1	演算項目 2	結果フィールド	標識		
DELETE (E)	検索引数	名前 (ファイルまたはレコード様式)		NR	ER	_

DELETE 命令は、データベース・ファイルからレコードを削除します。 ファイルは、削除可能ファイル (自由形式定義の USAGE キーワードに指定される *UPDATE または *DELETE で 示されるか、または、固定形式ファイル記述仕様書の 17 桁目の U で示されます) でなければなりません。 削除されたレコードは復元できません。

検索引数 (検索引数) が指定されていない場合、DELETE 命令は 現在のレコード (最後に検索されたレコード) を削除します。 レコードは前の入力命令 (例えば、CHAIN または READ) によってロックされ ていなければなりません。

検索引数 (検索引数) は、削除するレコードの検索に使用するキーまたは相対レコード番号でなければなりません。 アクセスがキー による場合には、検索引数 はフィールド名、名前の付いた定数、形象定数、またはリテラル形式の単一キーにすることができます。

ファイルが外部記述ファイルの場合、検索引数 は KLIST 名、値のリスト、または %KDS の形式の複合キーにすることもできます。 図形および UCS-2 のキー・フィールド には、そのファイル内のキーと同じ CCSID がなければなりません。%KDS の例については、652 ページの『%KDS (データ構造の検索引数)』 の終わりにある例を参照してください。 アクセスが相対レコード番号による場合には、検索引数 に整数のリテラルまたは小数点以下の桁数がゼロの数値フィールドを入れなければなりません。削除するレコードの検索に値のリストを使用する例については、730 ページの図 295 を参照してください。

制御キーワード EXPROPTS(*STRICTKEYS) が値のリストまたは %KDS でキーを指定する規則に与える影響について詳しくは、335 ページの『*STRICTKEYS』 を参照してください。

名前オペランドは、更新ファイル名またはレコードを削除するファイルのレコード様式名でなければなりません。 レコード様式名を使用で きるのは、外部記述ファイルの場合だけです。 検索引数が指定されていない場合には、レコード様式名はそのファイルから最後に読み取っ たレコードの名前でなければならず、そうでない場合にはエラーが起こります。

検索引数が指定される場合、71 桁目と 72 桁目には、ファイルに削除する レコードが見付からない場合にオンに設定される標識を入れなければなりません。 検索引数 が指定されない場合は、これらの桁はブランクのままにしておきます。 この情報は %FOUND 組み込み関数 からも入手することができます。この関数は、レコードが見付からない場合は '0' を 戻し、レコードが見付かった場合は '1' を戻します。

DELETE 例外 (ファイル状況コードが 1000 より大きい) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、146 ページの『ファイル例外/エラー』を参照してください。

IBM i オペレーティング・システムでは、ファイル名に指定されたファイルの DELETE 命令が正常に完了した後でそのファイルに対する読み取り操作が実行されると、削除されたレコードの次のレコードが取得されます。

ヌル値可能フィールドおよびキーを持つレコードの処理については、286 ページの『データベースのヌル値サポート』を参照してください。

詳しくは、576 ページの『ファイル操作』 を参照してください。

注 :

1. 命令コード拡張 H、M、および R は、検索引数がリストまたは %KDS() である場合にのみ使用できます。579 ページの『ファイル命令のキー』および 558 ページの『精度の確認』を参照してください。
2. 75 桁目と 76 桁目はブランクのままにしてください。

DIV (除算)

自由形式構文	(許可されていない - / または /= 演算子、 あるいは %DIV 組み込み関数を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
DIV (H)	被除数	除数	商	+	-	Z

演算項目 1 が指定されている場合には DIV 命令によって演算項目 1 が演算項目 2 で除算され、指定されていない場合には結果フィールドが演算項目 2 で除算されます。 商 (結果) は、結果フィールドに入れられます。 演算項目 1 が 0 の場合、除算命令の結果は 0 になります。 演算項目 2 を 0 にすることはできません。 0 の場合には、エラーが発生し、RPG IV 例外/エラー処理ルーチンが制御を受け取ります。 演算項目 1 が指定されていない場合には、結果フィールド (被除数) が演算項目 2 (除数) で除算されて、結果 (商) が結果フィールドに入れられます。 演算項目 1 と演算項目 2 は数値でなければならず、それぞれに配列、配列要し、 フィールド、形象定数、リテラル、名前のついた定数、サブフィールド、 またはテーブル名を入れることができます。

除算命令から生ずる剰余は、次の命令として剰余の転送 (MVR) 命令が指定 されていなければ失われます。 条件付け標識を使用する場合には、MVR 命令の 直前に DIV 命令が処理されることを確認してください。 MVR 命令が DIV 命令の前に処理されると、好ましくない結果になります。 移動命令が次の命令である場合には、除算命令の結果を四捨五入する (丸める) ことはできません。

DIV 命令に関する規則について詳しくは、557 ページの『算術演算』を参照してください。

560 ページの図 181 に、DIV 命令の例を示します。

注 : MVR 命令は、DIV 命令のいずれかのオペランドが浮動形式のものである 場合、その DIV 命令の後に続けることはできません。 ただし、浮動変数を命令コード MVR の結果として指定することはできます。

DO (命令グループの開始)

自由形式構文	(許可されていない - FOR 命令コードを使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
DO	開始値	限界値	指標値			

DO 命令は、命令のグループを開始してそのグループを処理する回数を指示 します。 命令グループを処理する回数を指示するためには、指標フィールド、開始値、 および限界値を指定します。 対応する ENDDO ステートメントは、このグループの終わりを示します。DO グループについて詳しくは、591 ページの『構造化プログラミング命令』を参照してください。

演算項目 1 には、数値リテラル、名前のついた定数、またはフィールド 名を使用して、小数点以下の桁数がゼロの開始値を指定します。 演算項目 1 を 指定しないと、開始値は 1 になります。

演算項目 2 には、数値フィールド名、リテラル、または名前のついた固定情 報を使用して、小数点以下の桁数がゼロの限界値を指定します。 演算項目 2 を指定しないと、限界値は 1 になります。

結果フィールドには、現在の指標値が入る数値フィールド名を指定します。 結果フィールドには、限界値に増分値を加えた値が入る十分な大きさが必要で す。 指標フィールドを指定しないと、内部使用のために指標フィールドが生成され ます。 指標フィールドの値は、DO 命令が開始されると演算項目 1 と置き換えられます。

対応する ENDDO 命令の演算項目 2 には、指標フィールドに加算する値を指 定します。 この値は、数値リテラルまたは小数点以下の桁数がゼロの数値フィールドとす ることができます。 これをブランクにすると、指標フィールドに加算される値は 1 になります。

DO グループは、DO 命令自体のほかに、DO および ENDDO ステートメントの 条件付け標識によっても制御されます。 DO ステートメントの条件付け標識は、 DO 命令を開始するかどうかを制御します。 これらの標識は DO ループの始めに 1 回だけ検査されます。 対応する ENDDO ステートメントの条件付け標識は、DO グループをもう一度繰 り返すかどうかを制御します。 これらの標識はそれぞれのループの終わりに検査されます。

DO 命令は次の 7 つのステップに従って実行されます。

1. DO 命令は、DO ステートメント行の条件付け標識が満たされた場合に処理 されます (ステップ 2)。 この標識が満たされない場合には、対応する ENDDO ステートメントの後で処理される次の命令に制御が渡されます (ステップ 7)。
2. DO 命令が開始されると、開始値 (演算項目 1) が指標フィールド (結果フ ィールド) に転送されます。
3. 指標値が限界値より大きい場合には、対応する ENDDO ステートメントの後 の演算命令に制御が渡されます (ステップ 7)。 そうでない場合には、DO ステ ートメントの後の最初の命令に制御が渡されます (ステップ 4)。
4. DO グループのそれぞれの命令が処理されます。
5. ENDDO ステートメントの条件付け標識が満たされない場合には、対応する ENDDO ステートメントの後の演算命令に制御が渡されます (ステップ 7)。 そうでない場合には ENDDO 命令が処理されます (ステップ 6)。
6. ENDDO 命令は指標フィールドに増分値を加算して処理されます。 制御はステップ 3 に渡されます。 (制御がステップ 3 に渡されたときに DO ステートメントの条件付け標識は再びテスト (ステップ 1) されないことに注意してください)。
7. DO ステートメントまたは ENDDO ステートメントの条件付け標識が満たさ れない時 (ステップ 1 または 5)、または指標値が限界値より大きい時 (ステ ップ 3) には、ENDDO ステートメントの後のステートメントが処理されます。

DO 命令を指定する場合には次のことに留意してください。

- 指標値、増分値、限界値、および標識は、DO グループの終わりに影響を与え るためにループ内で変更することができます。
- DO グループが明細演算と合計演算の両方にまたがることはできません。

これらの命令が DO 命令に与える影響については、796 ページの『LEAVE (Do/For グループからの抜け出し)』 および 792 ページの『ITER (繰り返し)』 を参照してください。

初期値、増分値、および限界値の**自由形式の式**での、反復ループの実行については、784 ページの『FOR (For)』 を参照してください。

詳しくは、「591 ページの『構造化プログラミング命令』」を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  The DO group is processed 10 times when indicator 17 is on;
 *  it stops running when the index value in field X, the result
 *  field, is greater than the limit value (10) in factor 2.  When
 *  the DO group stops running, control passes to the operation
 *  immediately following the ENDDO operation.  Because factor 1
 *  in the DO operation is not specified, the starting value is 1.
 *  Because factor 2 of the ENDDO operation is not specified, the
 *  incrementing value is 1.
C
C   17              DO        10            X                3 0
C                   :
C                   ENDDO
 *
 *  The DO group can be processed 10 times.  The DO group stops
 *  running when the index value in field X is greater than
 *  the limit value (20) in factor 2, or if indicator 50 is not on
 *  when the ENDDO operation is encountered.  When indicator 50
 *  is not on, the ENDDO operation is not processed; therefore,
 *  control passes to the operation following the ENDDO operation.
 *  The starting value of 2 is specified in factor 1 of the DO
 *  operation, and the incrementing value of 2 is specified in
 *  factor 2 of the ENDDO operation.
 *
C     2             DO        20            X                3 0
C                   :
C                   :
C                   :
C   50              ENDDO     2
```

図 *311. DO* 命令

DOU (条件が真になるまでの繰り返し)

自由形式構文	DOU {(MR) } 標識式

コード	演算項目 1	拡張演算項目 2
DOU (M/R)		標識式

DOU 命令コードは、最低 1 回 (おそらくはそれ以上) 実行したい命令のグループ の前に置かれます。 その機能は DOUxx 命令コードの機能と似ています。 対応する ENDDO ステートメントは、このグループの終わりを示します。 相違点は、標識の値を示す式 (標識式) によって論理条件が表されることです。 DOU 命令によって制御される操作は、標識式 が真になるまで、実行されます。命令拡張 M および R の使用法については、608 ページの『数値演算の精度の規則』を参照してください。

固定形式構文の場合、レベルおよび条件付け標識は有効です。 演算項目 1 はブランクでなければなりません。 拡張演算項目 2 には評価する式が入ります。

詳しくは、567 ページの『比較命令』または 591 ページの『構造化プログラミング命令』を参照してください。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/FREE
   // In this example, the do loop will be repeated until the F3
   // is pressed.
   dou *inkc;
      do_something();
   enddo;

   // The following do loop will be repeated until *In01 is on
   // or until FIELD2 is greater than FIELD3
   dou *in01 or (Field2 > Field3);
      do_something_else ();
   enddo;

   // The following loop will be repeated until X is greater than
   // the number of elements in Array
   dou X > %elem (Array);
      Total = Total + Array(x);
      X = X + 1;
   enddo;
/END-FREE
```

図 *312. DOU* 命令

DOUxx (条件までの繰り返し)

自由形式構文	(許可されていない - DOU 命令コードを使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
DOUxx	被比較値	被比較値				

DOUxx 命令コードは、最低 1 回 (おそらくはそれ以上) 実行したい命令のグルー プの前に置かれます。 対応する ENDDO ステートメントは、このグループの終わりを示します。DO グループの詳細と xx の意味については、591 ページの『構造化プログラミング命令』を参照してください。

演算項目 1 と演算項目 2 には、リテラル、名前のついた定数、フィールド 名、テーブル名、配列要素、形象定数、またはデータ構造名を入れなければな りません。 演算項目 1 と演算項目 2 は同じデータ・タイプでなければなりませ ん。

DOUxx ステートメントでは、関係 xx を指示します。 より複雑な条件を指定するため には、DOUxx ステートメントの直後に ANDxx または ORxx ステートメントを続けます。 DOUxx グループの命令は 1 回処理された後で、 次のいずれかに該当するまで繰り返されます。

- 演算項目 1 と演算項目 2 の間にこの関係が存在する
- DOUxx、ANDxx、または ORxx 命令を組み合わせて指定した条件が存在する

このグループは、グループの開始時にこの条件が 真でない場合であっても、常に最低 1 回は処理されます。

DOUxx 命令自体のほかに、 DOUxx グループは、DOUxx および ENDDO ステートメントの条件付け標識によっても制御されます。 DOUxx ステートメントの条件付け標識は、DOUxx 命令を開始するかどうかを 制御します。 対応する ENDDO ステートメントの条件付け標識では、DO ループを早期に終了 させることができます。

DOUxx 命令は次のステップに従って実行されます。

1. DOUxx 命令は、DOUxx ステートメント行の条件付け標識が満たされた場合 に処理されます (ステップ 2)。 標識が満たされない場合には、対応する ENDDO ステートメントの後で処理され る次の命令に制御が渡されます (ステップ 6)。
2. DOUxx 命令が処理されて、処理可能な次の命令に制御が渡されます (ステ ップ 3)。 DOUxx 命令では、この時点で演算項目 1 と演算項目 2 が比較されたり、指定さ れた条件がテストされることはありません。

3. DO グループのそれぞれの命令が処理されます。
4. ENDDO ステートメントの条件付け標識が満たされない場合には、対応する ENDDO ステートメントの後の次の演算命令に制御が渡されます (ステップ 6)。 そうでない場合には ENDDO 命令が処理されます (ステップ 5)。
5. ENDDO 命令が処理されて、DOUxx 命令の演算項目 1 と演算項目 2 が比較さ れるか、あるいは命令を組み合わせて指定した条件がテストされます。 演算項目 1 と演算項目 2 の間に関係 xx が存在するか、または指定された条件が存在 する場合には、DO グループが終了されて、ENDDO ステートメントの後の次の演 算命令に制御が渡されます (ステップ 6)。 演算項目 1 と演算項目 2 の間に関 係 xx が存在しないか、または指定された条件が存在しない場合には、DO グル ープの命令が繰り返されます (ステップ 3)。
6. DOUxx または ENDDO ステートメントの条件付け標識が満たされない場合 (ステップ 1 または 4)、あるいはステップ 5 で演算項目 1 と演算項目 2 の間 に関係 xx が存在するかまたは指定された条件が存在する場合には、ENDDO ス テートメントの後のステートメントが処理されます。

これらの命令が DOUxx 命令に与える影響については、796 ページの『LEAVE (Do/For グループからの抜け出し)』 および 792 ページの『ITER (繰り返し)』 を参照してください。

詳しくは、567 ページの『比較命令』 または 591 ページの『構造化プログラミング命令』 を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  The DOUEQ operation runs the operation within the DO group at
 *  least once.
C
C     FLDA          DOUEQ     FLDB
C
 *
 *  At the ENDDO operation, a test is processed to determine whether
 *  FLDA is equal to FLDB.  If FLDA does not equal FLDB, the
 *  preceding operations are processed again.  This loop continues
 *  processing until FLDA is equal to FLDB.  When FLDA is equal to
 *  FLDB, the program branches to the operation immediately
 *  following the ENDDO operation.
C
C                   SUB       1             FLDA
C                   ENDDO
C
 *
 *  The combined DOUEQ ANDEQ OREQ operation processes the operation
 *  within the DO group at least once.
C
C     FLDA          DOUEQ     FLDB
C     FLDC          ANDEQ     FLDD
C     FLDE          OREQ      100
C
 *
 *  At the ENDDO operation, a test is processed to determine whether
 *  the specified condition, FLDA equal to FLDB and FLDC equal to
 *  FLDD, exists.  If the condition exists, the program branches to
 *  the operation immediately following the ENDDO operation.  There
 *  is no need to test the OREQ condition, FLDE equal to 100, if the
 *  DOUEQ and ANDEQ conditions are met.  If the specified condition
 *  does not exist, the OREQ condition is tested.  If the OREQ
 *  condition is met, the program branches to the operation
 *  immediately following the ENDDO.  Otherwise, the operations
 *  following the OREQ operation are processed and then the program
 *  processes the conditional tests starting at the second DOUEQ
 *  operation.  If neither the DOUEQ and ANDEQ condition nor the
 *  OREQ condition is met, the operations following the OREQ
 *  operation are processed again.
C
C                   SUB       1             FLDA
C                   ADD       1             FLDC
C                   ADD       5             FLDE
C                   ENDDO
```

図 *313. DOUxx* 命令

DOW (条件が真の間繰り返し)

自由形式構文	DOW {(MR) } 標識式

コード	演算項目 1	拡張演算項目 2
DOW (M/R)		標識式

DOW 命令コードは、指定された条件が存在する時に処理したい命令のグループ の前に置かれます。 その機能は DOWxx 命令コードの機能と似ています。 対応する ENDDO ステートメントは、このグループの終わりを示します。 相違点は、標識の値を示す式 (標識式) によって論理条件が表されることです。 DOW 命令によって制御される操作は、標識式 が真である間、 実行されます。 式について詳しくは、596 ページの『式』を参照してください。命令拡張 M および R の使用法については、608 ページの『数値演算の精度の規則』を参照してください。

固定形式構文の場合、レベルおよび条件付け標識は有効です。 演算項目 1 はブランクでなければなりません。 演算項目 2 には評価する式が入ります。

詳しくは、567 ページの『比較命令』または 591 ページの『構造化プログラミング命令』を参照してください。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
* In this example, the do loop will be repeated until the condition
* is false. That is when A > 5 or B+C are not equal to zero.

/FREE
   dow (a <= 5) and (b + c = 0);
      do_something (a:b:c);
   enddo;
/END-FREE
```

図 *314. DOW* 命令

DOWxx (条件が真の間繰り返し)

自由形式構文	(許可されていない - DOW 命令コードを使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
DOWxx	被比較値	被比較値				

DOWxx 命令コードは、指定された条件が存在する時に処理したい命令のグルー プの前に置かれます。 より複雑な条件を指定するためには、DOWxx ステートメント の直後に ANDxx または ORxx ステートメントを続けます。 対応する ENDDO ステートメントは、このグループの終わりを示します。DO グループの詳細と xx の意味については、591 ページの『構造化プログラミング命令』を参照してください。

演算項目 1 と演算項目 2 には、リテラル、名前のついた定数、形象定数、 フィールド名、テーブル名、配列要素、またはデータ構造名を入れなければな りません。 演算項目 1 と演算項目 2 は同じデータ・タイプでなければなりません。 演算項目 1 と 2 の比較は、 比較命令の場合と同じ規則に従って行われます。567 ページの『比較命令』を参照してください。

DOWxx 命令自体のほかに、DO グループは、DOWxx ステートメントおよび ENDDO ステートメントの条件付け標識によっても制御されます。 DOWxx ステートメントの条件付け標識は、DOWxx 命令を開始するかどうかを 制御します。 対応する ENDDO ステートメントの条件付け標識は、DOW グループ をもう一度繰り返すかどうかを制御します。

DOWxx 命令は次のステップに従って実行されます。

1. DOWxx 命令は、DOWxx ステートメント行の条件付け標識が満たされた場合 に処理されます (ステップ 2)。 標識が満たされない場合には、対応する ENDDO ステートメントの後で処理される次の命令に制御が渡されます (ステッ プ 6)。
2. DOWxx 命令は、演算項目 1 と演算項目 2 を比較するか、または組み合わさ れた DOWxx、ANDxx、または ORxx 命令によって指定された条件をテストして 処理されます。 演算項目 1 と演算項目 2 の間に関係 xx が存在するか、組み合 わされた命令によって指定された条件が存在しない場合には、DO グループは終 了して ENDDO ステートメントの後の次の演算命令に制御が渡されます (ステッ プ 6)。 演算項目 1 と演算項目 2 の間に関係 xx が存在するかまたは組み合わ された命令によって指定された条件が存在する場合には、DO グループの命令が 繰り返されます (ステップ 3)。
3. DO グループのそれぞれの命令が処理されます。
4. ENDDO ステートメントの条件付け標識が満たされない場合には、対応する ENDDO ステートメントの後で実行される次の命令に制御が渡されます (ステッ プ 6)。 そうでない場合には ENDDO 命令が処理されます (ステップ 5)。
5. ENDDO 命令が処理されて、DOWxx 命令に制御が渡されます (ステップ 2)。 (ステップ 1 では DOWxx ステートメントの条件付け標識は再びテストされない ことに注意してください。)
6. DOWxx または ENDDO ステートメントの条件付け標識が満たされない時 (ス テップ 1 または 4)、あるいはステップ 2 で演算項目 1 と演算項目 2 の間に xx 関係または指定した条件が存在しない場合には、ENDDO ステートメントの後の ステートメントが処理されます。

これらの命令が DOWxx 命令に与える影響については、796 ページの『LEAVE (Do/For グループからの抜け出し)』 および 792 ページの『ITER (繰り返し)』 を参照してください。

詳しくは、567 ページの『比較命令』または 591 ページの『構造化プログラミング命令』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  The DOWLT operation allows the operation within the DO group
 *  to be processed only if FLDA is less than FLDB.  If FLDA is
 *  not less than FLDB, the program branches to the operation
 *  immediately following the ENDDO operation.  If FLDA is less
 *  than FLDB, the operation within the DO group is processed.
C
C     FLDA          DOWLT     FLDB
C
 *
 *  The ENDDO operation causes the program to branch to the first
 *  DOWLT operation where a test is made to determine whether FLDA
 *  is less than FLDB.  This loop continues processing until FLDA
 *  is equal to or greater than FLDB; then the program branches
 *  to the operation immediately following the ENDDO operation.
C
C                   MULT      2.08          FLDA
C                   ENDDO
C
 *  In this example, multiple conditions are tested.  The combined
 *  DOWLT ORLT operation allows the operation within the DO group
 *  to be processed only while FLDA is less than FLDB or FLDC. If
 *  neither specified condition exists, the program branches to
 *  the operation immediately following the ENDDO operation. If
 *  either of the specified conditions exists, the operation after
 *  the ORLT operation is processed.
C
C     FLDA          DOWLT     FLDB
C     FLDA          ORLT      FLDC
C
 *  The ENDDO operation causes the program to branch to the second
 *  DOWLT operation where a test determines whether specified
 *  conditions exist.  This loop continues until FLDA is equal to
 *  or greater than FLDB and FLDC; then the program branches to the
 *  operation immediately following the ENDDO operation.
C
C                   MULT      2.08          FLDA
C                   ENDDO
```

図 *315. DOWxx* 命令

DSPLY (メッセージ表示)

自由形式構文	DSPLY {(E) }{メッセージ{メッセージ待ち行列{応答 }}}

コード	演算項目 1	演算項目 2	結果フィールド	標識		
DSPLY (E)	メッセージ	メッセージ・キュー	応答	_	ER	_

DSPLY 命令では、プログラムがそのプログラムを要求したディスプレイ・ ワークステーションと通信することができます。 メッセージ、応答、または両方のオペランドが指定される必要があります。 この命令では、メッセージを表示し、応答を受け入れることができます。

表示されるメッセージの作成には、メッセージ・オペランドおよび場合により応答 オペランドの値が使用されます。メッセージ は、その値を使用して表示されるメッセージを作成するフィールド名、リテラル、名前付き定数、テーブル名、または配列要素にすることができます。 自由形式演算では、メッセージ・オペランドは式にすることができます (式が括弧で囲まれている場合)。 メッセージ・オ ペランドには、*M に続けて、メッセージ・ファイル QUSERMSG から検索されるメッセージを識別するメッセージ識別コードも入れることができます。 別のメッセージ・ファイルを使用するためには、OVRMSGF コマンドを使用し ます。QUSERMSG は、メッセージを受け取るジョブのライブラリー・リストに入ってい るライブラリー中になければなりません。

メッセージ識別コードは、3 桁の英字と 4 桁の数字で構成される 7 桁の長 さでなければなりません (例えば *MUSR0001、これはメッセージ USR0001 が 使用されることを意味しています)。

これを指定する場合、メッセージ待ち行列 オペランドは、文字フィールド、リテラル、名前のついた定数、テーブル名、または配列要素にすることができます。 この値は、メッセージを受け取るためのオブジェクト (オプションで応答の送信も可能) の記号名になります。 プログラム・メッセージ待ち行列以外の待ち行列名を、メッセージ待ち行列 オペランドに含まれる値にすることができます。 この待ち行列をプログラムの実行時に使用するには、その前にオペレーティング・システムに宣言する必要があります。 (待ち行列の作成方法については、「*CL* プログラミング」を参照してください。) 定義済みの待ち行列には次の 2 つがあります。

待ち行列
値

QSYSOPR
メッセージはシステム操作員に送られます。 DSPLY 命令で直ちにシステム 操作員にメッセージを表示できるようにするためには、QSYSOPR メッセージ待 ち行列の重大度レベルがゼロ (00) でなければならないことに注意してくださ い。

***EXT**
メッセージは外部メッセージ待ち行列に送られます。

注 : バッチ・ジョブでメッセージ待ち行列 の値が指定されていない場合、デフォルト値は QSYSOPR になります。 対話式ジョブの場合には、デフォルト値は *EXT です。

応答 オペランドは任意指定です。 指定した場合には、ここに応答が入ります。応答 には応答が入るフィールド名、テーブル名、または配列要素を入れることができます。 データが入力されなければ、応答 は変わりません。 自由形式仕様書において、応答を指定してメッセージ・キューは指定しないようにするには、メッセージ・キュー にブランクを指定してください。

```
DSPLY fld1 ' ' fld2;
```

自由形式演算仕様書でコーディングされる場合、完全修飾名は結果フィールド・オペランドとして指定すること ができ、式は演算項目 1 および演算項目 2 として使用することができます。 しかし、オペランドが完全修飾名よりも複雑である場合には、 式は括弧で囲む必要があります。

DSPLY 例外 (プログラム状況コード 333) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。 命令でエラーが発生した場合、例外は 指定された方法によって処理されます。エラー処理について詳しくは、163 ページの『プログラム例外/エラー』を参照してください。

DSPLY 命令をメッセージ・オペランドにメッセージ識別コードを入れないで 指定した場合には、この命令の機能は次のようになります。

- メッセージ・オペランドが指定され、応答 オペランドが指定されない場合、メッセージ・オペランドの内容が表示されます。 プログラムは、ワークステーションで 形式を表示するためにパラメーター RSTDSP (*NO) を指定した表示装置ファイ ルが使用されていない限り、応答を待機しません。 その後でプログラムは、ユーザーが実行キーを押すまで待機します。
- メッセージ・オペランドが指定されずに応答 オペランドが指定されている場合、応答 オペランドの内容が表示され、プログラムはユーザーが応答のためにデータを入力するのを待ちます。 この応答は応答 オペランドに入れられます。
- メッセージ・オペランドと応答 オペランドの両方が指定されている場合は、それらの内容が結合されて表示されます。 プログラ ムはユーザーが応答のためにデータを入力するまで待機します。 この応答は結果フィー ルドに入れられます。
- メッセージについてのヘルプを要求すると、必要なデータのタイプおよび 属性と 正常に実行されなかった試行回数を知ることができます。

  表示可能な情報の最大長は 52 バイトです。

メッセージ・オペランドでメッセージ識別コードが指定されていない DSPLY 命令によって書き出されるレコードの形式は次のとおりです。

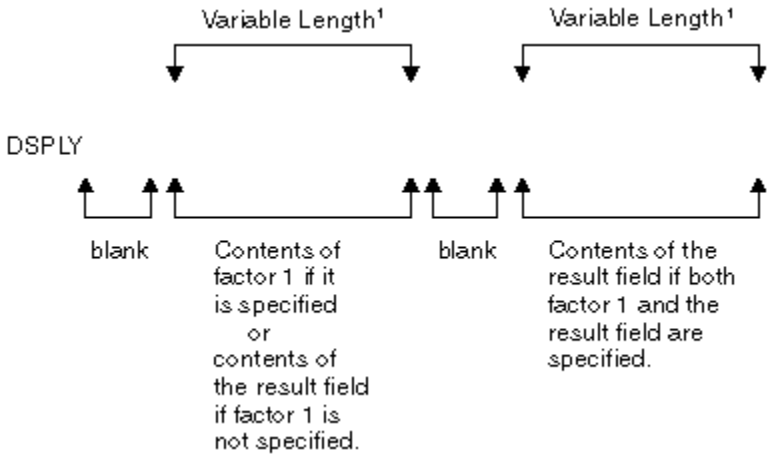


図 *316. DSPLY* 命令のレコード様式

メッセージ・オペランドにメッセージ識別コードを入れた DSPLY 命令を指定した場合には、この命令の機能は次のようになります。 すなわち、メッセージ・オペランドによって識別されるメッセージが QUSERMSG から検索され、そのメッセージが表示されて、プログラムは (応答 オペランドが指定されている場合に) ユーザーがデータを入力して応答するまで待機します。 この応答は結果フィー ルドに入れられます。

メッセージに応答する場合には、次のことに留意してください。

- 表示装置に送られた非浮動数値フィールドは右寄せされてゼロが消去 されます。
- 長さが結果フィールドの桁数より大きい非浮動数値 フィールドが入力されて右端の文字がマイナス符号 (-) でない場合には、エラーが検出されて 2 回目の待機状態になります。 ユーザーはこのフィールドに入力し直さなければなりません。
- 浮動値は、外部表示表現で入力されます。 この値は、4 バイト浮動フィールドの場合には 14 桁より長くなっては ならず、8 バイト浮動フィールドの場合には 23 桁より長くなっては なりません。
- 図形データ、UCS-2 データ、または文字データを入力する場合には、その長さは受け取りフィールドの長さと等しいかまたはそれ以下でなければなりません。
- 結果のフィールドが可変長の場合、その長さはユーザーが入力した長さに設定されます。
- 日付、時刻、またはタイム・スタンプ・フィールドが入力される場合には、そ の形式および区切り記号は結果フィールドの形式および区切り記号と一致しな ければなりません。 形式または区切り記号が一致しないか、または値が正しくない場合 (例えば、1999/99/99 の日付) には、エラーが検出されて 2 回目の待機状態になります。 ユーザーはこのフィールドに入力し直さなければなりません。
- DSPLY 命令によって、ワークステーション・ユーザーは最高 5 回 までメッセージへの応答を試みることができます。 5 回目の試みが失敗した 後、DSPLY 命令は失敗します。 DSPLY 命令にメッセージ・オペランドで指定されたメッセージ識別コードがない場合、ユーザーはメッセージに関するヘルプを要求して、期待されている応答のタイプと属性を知ることができます。
- システム操作員待ち行列 (QSYSOPR) にヌルの応答を入力するためには、文字 *N を入力して実行キーを押さなければなりません。
- 図形フィールド、UCS-2 フィールド、または文字フィールドには、すべての文字が入力された後で右側にブランクが埋められます。
- UCS-2 フィールドは、1 バイト文字として表示、入力されます。
- 数値フィールドは、すべての文字が入力された後で右寄せされて左側にゼ ロが埋められます。
- 小文字は大文字に変換されません。
- 演算項目 1 または結果フィールドが図形データ・タイプの場合には、表示 される時に SO/SI で囲まれます。 SO/SI は、入力では図形の結果フィールドに 割り当てられる値から除かれます。
- 浮動フィールドは、外部表示表現で表示されます。 浮動値は、数値リテラルまたは浮動リテラルとして入力することができます。 応答を入力するときは、浮動値を正規化する必要はありません。

詳しくは、「583 ページの『メッセージ命令』」を参照してください。

```
/free
    // Display prompt and wait for response:
    dsply prompt '' result;
    // Display string constructed in an expression:
    dsply ('Length of name is ' + %char(%len(str)) + ' bytes.');
/end-free
```

図 *317. DSPLY* 命令コードの例

DUMP (プログラム・ダンプ)

自由形式構文	DUMP {(A) } {識別コード }

コード	演算項目 1	演算項目 2	結果フィールド	標識		
DUMP (A)	ID					

DUMP 命令によって、モジュールのダンプ (定義されたすべてのフィールド、すべてのファイル、標識、データ構造、配列、およびテーブル) が実行されます。 この命令は単独に使用するか、または IBM i のテストおよびデバッグ機能と組み合わせて使用できます。 CRTBNDRPG または CRTRPGMOD コマンド上で、あるいは制御仕様書の キーワードとして OPTIMIZE(*FULL) コンパイラー・オプションを選択すると、 最適化の影響で、ダンプに示されるフィールド値に実際に内容が反映されない場合があります。

DBGVIEW(*NONE) コンパイラー・オプションが指定されている場合、ダンプは、プログラム状況データ構造、ファイル情報データ構造、および *IN 標識だけを表示します。 その他の変数は、オブジェクトが必要なプログラム識別情報を含まないために、表示される内容はないことになります。

DEBUG(*NO) 制御仕様書キーワードが指定されている場合、ダンプは実行されません。 このキーワードは、命令拡張 A を指定することによって指定変更することができます。 この命令拡張は、DEBUG キーワードの値にかかわらず必ずダンプを実行することを意味します。

任意指定の識別コード オペランドの内容は、DUMP 命令を識別します。 指定すると、ダンプ・リストのデフォルトの見出しが置き換えられます。 これには文字または図形の指定を入れなければならず、その内容がダンプを識 別するフィールド名、リテラル、名前のついた定数、テーブル名、または 配列要素のいずれかとすることができます。 識別コード・オペランドにグラフィック項目が入っている場合には、64 桁の 2 バイト文字に制限されます。識別コード は形象定数であってはなりません。

プログラムは、DUMP 命令の後の次の演算ステートメントから処理を続行します。

DUMP 命令が実行されるのは、制御仕様書に DEBUG キーワードが指定されているか、または DUMP 命令で A 命令拡張がコーディングされている場合です。 それ以外の場合には、DUMP 命令のエラーが検査されてリストにステートメントが印刷されますが、DUMP 命令は処理されません。

ファイルのダンプ時には、DUMP は INFDS のファイル・フィードバック情報部 分をダンプしますが、INFDS のオープン・フィードバック情報または入出力フ ィードバック情報部分はダンプしません。 その代わりに、DUMP はファイルの実際のオープン・フィードバックおよび装置 フィードバック情報をダンプします。

INFDS のファイル・フィードバック情報は常に更新されるので、宣言した INFDS に十分な容量がなくてオープン・フィードバックまたは入出力フィード バック情報を入れられない場合でも、DUMP の前に POST を実行することを心配 する必要はありません。

サブプロシージャーが活動状態でない場合には、サブプロシージャーの変数の 値が正しくない場合があります。 サブプロシージャーが繰り返し呼び出される場合には、一番新しい呼び出しからの 値が表示されます。

Java オブジェクト変数は期待された値を表示しない場合があります。 RPG モジュールは、あるオブジェクトがすでに存在しなくなった後にもそのオブジェクトを参照し続ける場合があります。 これはオブジェクト参照が再使用されて、ダンプされようとしている RPG モジュールとは関係のない異なるオブジェクトを参照する可能性があるということです。 この異なるオブジェクトが、定様式ダンプに表示されるオブジェクトになります。

サンプル・ダンプ・リストについては、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」のダンプの入手に 関する章を参照してください。

詳細については、580 ページの『情報命令』 を参照してください。

ELSE (他の場合)

自由形式構文	ELSE

コード	演算項目 1	演算項目 2	結果フィールド	標識		
ELSE						

ELSE 命令は、IFxx および IF 命令の任意指定部分です。 IFxx の比較が満たされた場合には ELSE の前の演算が処理され、そうでない 場合には、ELSE の後の演算が処理されます。

合計演算の中では、制御レベルの指定 (7 から 8 桁目) は ブランクにするか、あるいはプログラムの該当するセクション内のステートメントを グループにまとめる L1-L9 標識、LR 標識、または L0 の指定を入れることができます。 制御レベルの指定は文書化のためだけのものです。 条件付け標識の指定 (9 から 11 桁目) は使用できません。 IFxx/ELSE グループをクローズするためには ENDIF 命令を使用します。

791 ページの図 332 に、IFxx 命令を伴う ELSE 命令の例を示します。

詳しくは、「591 ページの『構造化プログラミング命令』」を参照してください。

ELSEIF (ELSE IF)

自由形式構文	ELSEIF {(MR) } 標識式

コード	演算項目 1	拡張演算項目 2
ELSEIF (M/R)	ブランク	標識式

ELSEIF 命令は、ELSE 命令と IF 命令を組み合わせたものです。 これにより、ネスティングのレベルがさらに深くなるのを避けることができます。

IF では、条件が満たされた場合に一連の命令コードを処理することができます。 その機能は IFxx 命令コードの機能と似ています。 相違点は、標識の値を示す式 (標識式) によって論理条件が表されることです。 ELSEIF 命令でコントロールされる操作は、標識式 オペランドが真 (かつ直前の IF または ELSEIF ステートメントの式が偽) の時に実行されます。

命令拡張 M および R の使用法については、608 ページの『数値演算の精度の規則』 を参照してください。

詳しくは、「591 ページの『構造化プログラミング命令』」を参照してください。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/free

   IF keyPressed = HELPKEY;
      displayHelp();
   ELSEIF keyPressed = EXITKEY;
      return;
   ELSEIF keyPressed = ROLLUP OR keyPressed = ROLLDOWN;
      scroll (keyPressed);
   ELSE;
      signalError ('Key not defined');
   ENDIF;

/end-free
```

図 *318. ELSEIF* 命令

ENDyy (構造化グループの終わり)

自由形式構文	ENDDO ENDFOR ENDIF ENDMON ENDSL (END および ENDCS は許可されていない)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
END		増分値				
ENDCS						
ENDDO		増分値				
ENDFOR						
ENDIF						
ENDMON						
ENDSL						

ENDyy 命令は、命令の CASxx、DO、DOU、DOW、DOUxx、DOWxx、FOR、 IF、IFxx、MONITOR、または SELECT グループを終了します。

ENDyy 命令を下にリストします。

END
CASxx、DO、DOU、DOUxx、DOW、DOWxx、FOR、IF、IFxx、または SELECT グループを終了します。

ENDCS
CASxx グループを終了します。

ENDDO
DO、DOU、DOUxx、DOW、または DOWxx グループを終了します。

ENDFOR
FOR または FOR-EACH グループを終了します。

ENDIF
IF または IFxx グループを終了します。

ENDMON
MONITOR グループを終了します。

ENDSL
SELECT グループを終了します。

増分値 オペランドは、DO グループの範囲を区切る ENDyy 命令においてのみ、使用できます。 演算項目 2 には、DO グループの増分値が入ります。 演算項目 2 は正または負とすることができ、小数点以下の桁数はゼロで、配列 要素、テーブル名、データ構造、フィールド、名前のついた定数、または 数値リテラルのいずれかとすることができます。 増分値 が ENDDO で指定されていない場合、デフォルト値 1 が使用されます。 増分値 が負の場合、DO グループは終了しません。

条件付け標識は ENDDO または ENDFOR では任意指定ですが、ENDCS、 ENDIF、ENDMON、および ENDSL には指定することができません。

結果の標識は使用できません。 オペランドは、ENDCS、ENDIF、ENDMON、および ENDSL では指定できません。

ENDyy 形式を別の命令グループで使用すると (例えば、構造化グループで ENDIF)、コンパイル時にエラーが起こります。

ENDyy 命令の使用例については、CASxx、DO、DOUxx、DOWxx、FOR、FOR-EACH、IFxx、および DOU、DOW、IF、MONITOR、および SELECT を参照してください。

詳しくは、575 ページの『エラー処理命令』または 591 ページの『構造化プログラミング命令』を参照してください。

ENDSR (サブルーチンの終了)

自由形式構文	ENDSR {戻り点 }

コード	演算項目 1	演算項目 2	結果フィールド	標識		
ENDSR	ラベル	戻り点				

ENDSR 命令は、RPG IV サブルーチンの終わりおよびサイクル・メイン・プログラムへの戻り点 (戻り点) を定義します。 ENDSR は、サブルーチンの最後のステートメントでなければなりません。 従来型の構文では、サブルーチン内の GOTO 命令の分岐先として使用できるラベル・オペランドが指定できます。 (自由形式構文では、ラベル は指定できません。) 制御レベル項目 (7 から 8 桁目) は SR またはブランクにすることができます。 条件付け標識は指定することができません。

ENDSR はサブルーチンを終了して、EXSR または CASxx 命令の直後のステー トメントに分岐して戻ります。ただし、そのサブルーチンがプログラム例外/エラー処理サブルーチン (*PSSR) またはファイル例外/エラー処理サブルーチン (INFSR) でない場合に限られます。 これらのサブルーチンの場合には、ENDSR 命令の戻り点 オペランドに、サブルーチンの処理後に制御を戻す場所を指定する項目を入れることができます。この項目 は、予約キーワードが入っているフィールド名、または予約キーワードである リテラルまたは名前のついた定数を入れるフィールド名とすることができ ます。 正しくない戻り点が指定された場合には、RPG IV エラー処理プログラムに制御が渡されます。

注 : サブプロシージャー (リニア・メイン・プロシージャーを含む) 内に現れる ENDSR 命令の場合には、戻り点 オペランドは指定できません。

戻り点について詳しくは、160 ページの『ファイル例外/エラー処理サブルーチン (INFSR)』 を参照してください。

RPG IV サブルーチンのコーディングの例については、595 ページの図 192 を参照してください。

詳しくは、「594 ページの『サブルーチン命令』」を参照してください。

EVAL (式の評価)

自由形式構文	{EVAL {(HMR) } } 結果 = 式
	{EVAL{(HMR) } } 結果 += 式
	{EVAL{(HMR) } } 結果 -= 式
	{EVAL{(HMR) } } 結果 *= 式
	{EVAL{(HMR) } } 結果 /= 式
	{EVAL{(HMR) } } 結果 **= 式

コード	演算項目 1	拡張演算項目 2
EVAL (H M/R)		割り当てステートメント

EVAL 命令コードは、"結果 = 式" または "結果命令 = 式" の形式の割り当てステートメントを評価します。 式が評価されると結果が**結果**に入れられます。 したがって、**結果**をリテラルまたは定数とすることはできず、 フィールド名、配列名、配列要素、データ構造、データ構造サブフィールド、 または %SUBST 組み込み関数を使用したストリングとしなければなりません。

式では任意の RPG データ・タイプを生成することができます。 式のタイプは結果のタイプと同じでなければなりません。 文字、図形、または UCS-2 の結果は左寄せされ、必要に応じて右側にブランクが埋め込まれるか切り捨てられます。 **結果**が可変長フィールドの場 合、その長さは式の結果の長さに設定されます。

結果が指標のない配列や配列 (*) として指定された配列を表す場合には、241 ページの『演算での配列の指定』に説明されている規則に従って、結果のそれぞれの要素に式の値が割り当てられます。 そうでない場合には、式が一度評価されて、その値が配列またはサブ配列のそ れぞれの要素に入れられます。 数値式の場合には、四捨五入の命令コード拡張を使用することができます。 四捨五入の規則は、算術演算の場合の規則と同じです。

自由形式演算仕様においては、拡張が不要である場合、および変数に命令コードと同じ名前がない場合は、命令コード名を省略できます。

割り当て演算子 +=、-=、*=、/=、および **= の場合、該当する演算は結果および式に適用され、 結果は結果に割り当てられます。 例えば、ステートメント X+=Y は X=X+Y とほぼ等しくなります。 この 2 つのステートメントの違いは、これらの割り当て演算子の場合、結果オペランドが評価されるのは 1 回のみであるということです。 これらの違いは、 結果命令を評価するときにサブプロシージャーを呼び出す場合に重要になります (例えば、次のような副産物があります)。

```
     warnings(getNextCustId(OVERDRAWN)) += 1;
```

式の概要については、596 ページの『式』を参照してください。 数値式の精度に関する規則については、608 ページの『数値演算の精度の規則』を参照してください。 これは、式に除算命令が入っている場合、または EVAL が いずれかの命令拡張を使用する場合には、特に重要です。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
*                   Assume FIELD1 = 10
*                          FIELD2 =  9
*                          FIELD3 =  8
*                          FIELD4 =  7
*                          ARR is defined with DIM(10)
*                          *IN01 = *ON
*                          A = 'abcdefghijklmno' (define as 15 long)
*                          CHARFIELD1 = 'There'  (define as  5 long)

/FREE
  // The content of RESULT after the operation is 20
  eval RESULT=FIELD1 + FIELD2+(FIELD3-FIELD4);
  // The indicator *IN03  will be set to *ON
  *IN03 = *IN01 OR (FIELD2 > FIELD3);
  //  Each element of array ARR will be assigned the value 72
  ARR(*) = FIELD2 * FIELD3;
  //  After the operation, the content of A = 'Hello There   '
  A = 'Hello ' + CHARFIELD1;
  //  After the operation the content of A = 'HelloThere    '
  A = %TRIMR('Hello ') + %TRIML(CHARFIELD1);
  //  Date in assignment
  ISODATE = DMYDATE;
  //  Relational expression
  //  After the operation the value of *IN03 = *ON
  *IN03 = FIELD3 < FIELD2;
  //  Date in Relational expression
  // After the operation, *IN05 will be set to *ON if Date1 represents
  // a date that is later that the date in Date2
  *IN05 = Date1 > Date2;
  // After the EVAL the original value of A contains 'ab****ghijklmno'
  %SUBST(A(3:4))= '****';
  // After the EVAL PTR has the address of variable CHARFIELD1
  PTR = %ADDR(CHARFIELD1);
  //  An example to show that the result of a logical expression is
  //  compatible with the character data type.
  //  The following EVAL statement consisting of 3 logical expressions
  //  whose results are concatenated using the '+' operator
  //  The resulting value of the character field RES is '010'
  RES = (FIELD1<10) + *in01 + (field2 >= 17);
  // An example of calling a user-defined function using EVAL.
  // The procedure FormatDate converts a date field into a character
  // string, and returns that string.  In this EVAL statement, the
  // field DateStrng1 is assigned the output of formatdate.
  DateStrng1 = FormatDate(Date1);
  // Subtract value in complex data structure.
  cust(custno).account(accnum).balance -= purchase_amount;
  // Append characters to varying length character variable
  line += '<br />';
/END-FREE
```

図 *319. EVAL* 命令

EVALR (式の評価、右寄せ)

自由形式構文	EVALR {(MR) } 結果 = 式

コード	演算項目 1	拡張演算項目 2
EVALR (M/R)		割り当てステートメント

EVALR 命令コードは、結果=式の形式の割り当てステートメントを評価 します。 式が評価されると、結果が結果の中で右寄せされます。 したがって、結果をリテラルまたは定数とすることはできず、固定長 の文字、図形、または UCS-2 のフィールド名、配列名、配列要素、データ構造、 データ構造サブフィールド、または %SUBST 組み込み関数を使用したストリングと しなければなりません。 式のタイプは結果のタイプと同じでなければなりません。 結果は右寄せ され、必要に応じて、左側がブランクで埋め込まれるか、左側で切り捨てられます。

注 : EVAL 命令と異なり、EVALR の結果のタイプは文字、図形、または UCS-2 に しかできません。 また、固定長の結果フィールドしか使用できませんが、%SUBST に は、この組み込み関数が式の左の部分を成すのであれば、可変長フィールドを 含むことができます。

結果が指標のない配列や配列 (*) として指定された配列を表す場合には、241 ページの『演算での配列の指定』に説明されている規則に従って、結果のそれぞれの要素に式の値が割り当てられます。 そうでない場合には、式が一度評価されて、その値が配列またはサブ配列のそ れぞれの要素に入れられます。

式の概要については、596 ページの『式』を参照してください。 数値式の精度に関する規則については、608 ページの『数値演算の精度の規則』を参照してください。 これは、式に 除算命令が入っている場合、または EVALR が いずれかの命令拡張を使用する場合には、特に重要です。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
D*Name++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
D Name            S             20A

 /FREE
    eval  Name = 'Kurt Weill';
    // Name is now 'Kurt Weill          '
    evalr Name = 'Johann Strauss';
    // Name is now '      Johann Strauss'
    evalr %SUBST(Name:1:12) = 'Richard';
    // Name is now '     Richard Strauss'
    eval  Name = 'Wolfgang Amadeus Mozart';
    // Name is now 'Wolfgang Amadeus Moz'
    evalr Name = 'Wolfgang Amadeus Mozart';
    // Name is now 'fgang Amadeus Mozart'
 /END-FREE
```

図 *320. EVALR* 命令

EVAL-CORR (対応するサブフィールドの代入)

自由形式構文	EVAL-CORR{(HMR) } ターゲット = ソース;

コード	演算項目 1	拡張演算項目 2
EVAL-CORR		ターゲット = ソース

EVAL-CORR 命令は、ソースのデータ構造の対応するサブフィールドからターゲットのデータ構造のサブフィールドにデータおよびヌル標識を代入します。 代入されるサブフィールドは、両方のデータ構造に同じ名前および互換データ・タイプがあるサブフィールドです。 例えば、データ構造 DS1 に文字サブフィールド A、B、および C があり、データ構造 DS2 に文字サブフィールド B、C、および D がある場合、ステートメント

```
EVAL-CORR DS1 = DS2
```

は、サブフィールド DS2.B および DS2.C から DS1.B および DS1.C にデータを代入します。 EVAL-CORR 命令が反映されたターゲットのデータ構造内のヌル対応サブフィールドでは、ソースのデータ構造のサブフィールドのヌル標識からヌル標識が設定されるか、またはソースのサブフィールドがヌルに非対応である場合は *OFF に設定されます。

命令コード拡張 H が指定された場合、すべての数値代入に対して四捨五入 機能が適用されます。 EVAL-CORR の拡張は、自由形式演算にのみ指定できます。

命令コード拡張 M または R が指定された場合、ソースまたはターゲットの式の一部として指定された任意のプロシージャー呼び出しの引数に適用されます。 EVAL-CORR の拡張は、自由形式演算にのみ指定できます。

コンパイラー・リストの *EVAL-CORR* 要約セクション が以下の判別に使用できます。

- どのサブフィールドが選択され、EVAL-CORR 命令が反映されるか
- 選択されないサブフィールドの場合は、そのサブフィールドが選択されなかった理由

- 選択されたサブフィールドの場合は、ディメンションの違いやサブフィールドのヌル対応など、そのサブフィールドについての追加情報。

EVAL-CORR 要約セクションについて詳しくは、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」を参照してください。

EVAL-CORR 命令を使用する場合は、次のことに留意してください。

- 命令コード EVAL-CORR は、自由形式演算または固定形式演算のいずれかでコーディングされます。 固定形式演算でコーディングする場合、代入式は拡張演算項目 2 記入項目でコ ーディングされ、演算項目 1 記入項目はブランクのままになります。
- ソースおよびターゲットのオペランドは、LIKEDS または LIKEREC で定義されたデータ構造サブフィールドも含め、両方ともデータ構造変数にする必要があります。
- オペランドには、修飾または非修飾のデータ構造を使用できます。 ただし、命令を正常に終了するには、少なくとも 1 つのオペランドは修飾データ構造にする必要があります。 そうしないと、2 つのデータ構造が同じ名前のサブフィールドを持つことができません。
- 代入に関連するサブフィールドは、両方のデータ構造に同じ名前があり、EVAL を使用した代入と互換性があるデータ・タイプのサブフィールドです。
- サブフィールド名を比較して対応するサブフィールドを検出する場合、使用される名前は内部プログラム名です。 その内部プログラム名は、外部記述ファイルまたはデータ構造からのフィールドの場合、外部名と異なることがあります。 外部で定義されたフィールドで名前変更または接頭部が付けられた場合、使用される名前は、名前変更または接頭部が付けられた後の名前です。
- 名前で照応するソースおよびターゲットのサブフィールドで、両方のデータ構造が LIKEDS または LIKEREC で定義されている場合、代入されるサブフィールドは、サブフィールド・データ構造の対応するサブフィールドです。 ソースおよびターゲットの 2 つのサブフィールドに同じ名前があっても、一方は LIKEDS または LIKEREC で定義されたデータ構造であるが、もう一方はデータ構造ではない場合、そのサブフィールドは EVAL-CORR 命令では代入されません。
- ソース・サブフィールドからターゲット・サブフィールドへのデータの代入は、命令コード EVAL と同じ規則に従います。 例えば、文字値の長さが等しくない場合は、切り捨てされるか、またはブランクが埋め込まれて、左寄せで代入されます。
- データは、サブフィールドごとに、ソース・データ構造内のサブフィールドの順番で代入されます。 開始/終了位置の重複、または OVERLAY キーワードのいずれかの理由により、ターゲット・データ構造でサブフィールドが重複している場合は、先に移動したデータが後の代入により上書きされます。
- ソースおよびターゲットのデータ構造、または対応するソースおよび ターゲットのサブフィールドの両方のデータ構造が LIKEDS または LIKEREC と同じ方法で定義されている場合、つまり両方のデータ構造が同一のデータ構造であるかのように定義されている場合、コンパイラーは代入を最適化し、個々の一連のサブフィールドの代入ではなく、データ構造全体を代入します。
- ソースまたはターゲットのオペランドのいずれかが複数回繰り返しデータ構造である場合、現在のオカレンスが使用されます。
- 配列を使用する場合は、以下が適用されます。
  - ソース・オペランドが索引なし配列データ構造である場合、ターゲット・データ構造も配列データ構造である必要があります。
  - ターゲット・オペランドが索引なし配列データ構造である場合、配列の結果がある EVAL と同じ規則に従い、配列データ構造の各要素で命令が処理されます。%SUBARR を使用して、ソースまたはターゲットのデータ構造配列のいずれかで使用される要素の数を制限できます。
  - 一方のサブフィールドが配列である場合、もう一方のサブフィールドも配列である必要があります。 一方の配列サブフィールドのディメンションがもう一方のディメンションより小さい場合は、小さい方の数の配列の要素のみが代入されます。 ターゲット・サブフィールドの要素が多い場合、EVAL-CORR 命令では余計な要素が変更されません。
- ヌルに対応したサブフィールドの場合は、以下が適用されます。
  - EVAL-CORR により、データ構造サブフィールドではないヌルに対応したサブフィールドのヌル標識の代入が自動的に処理されます。

- ソースおよびターゲットのサブフィールドの両方がヌル対応である場合、ソース・サブフィールドのヌル標識がターゲット・サブフィールドのヌル標識にコピーされます。
- ターゲット・サブフィールドがヌル対応であり、ソース・サブフィールドがヌルに非対応である場合、ターゲット・サブフィールドのヌル標識が *OFF に設定されます。
- ソース・サブフィールドがヌル対応であり、ターゲット・サブフィールドがヌルに非対応である場合、ソース・サブフィールドのヌル標識は無視されます。
- EVAL-CORR 命令は、スカラーおよび配列のサブフィールドに対してのみヌル標識を設定します。 ヌル対応サブフィールドがデータ構造である場合、EVAL-CORR 命令ではそのヌル標識を設定しません。同様に、ターゲット・データ構造そのものがヌル対応である場合、EVAL-CORR 命令ではそのヌル標識を設定しません。

– サブフィールドがデータ構造であり、そのデータ構造そのものにヌル標識が代入された場合、EVAL-CORR 命令はそのヌル標識に影響を及ぼしません。

EVAL-CORR 命令の例

```
 * Physical file EVALCORRPF
A          R PFREC
A            NAME          25A
A            IDNO          10P 0
A            CITY          20A
 * Display file EVALCORRDF
A          R DSPFREC
A                                  3  2'Name'
A            NAME          25A  O  3 15
A                                  4  2'City'
A            CITY          20A  B  4 15CHECK(LC)
 * RPG program
Fevalcorrpfuf   e             disk
Fevalcorrdfcf   e             workstn

D pf_ds         e ds                    extname(evalcorrpf : *input)
D                                       qualified
D pf_save_ds      ds                    likeds(pf_ds)
D dspf_ds       e ds                    extname(evalcorrdf : *all)
D                                       qualified
 /free
       read pfrec pf_ds;
       dow not %eof;
          // Assign all subfields with the same name and type
          // to the data structure for the EXFMT operation
          // to the display file (NAME and CITY)
          eval-corr dspf_ds = pf_ds;

          // Show the screen to the user
          exfmt dspfrec dspf_ds;
          // Save the original physical file record
          // and assign the display file subfields to the
          // physical file data structure.  Then compare
          // the physical file data structure to the saved
          // version to see if any fields have changed.
          eval pf_save_ds = pf_ds;
          eval-corr pf_ds = dspf_ds;
          if pf_ds <> pf_save_ds;
             // Some of the fields have changed
             update pfrec pf_ds;
          endif;
          read pfrec pf_ds;
       enddo;
       *inlr = '1';
```

図 *321.* 外部記述データ構造 *I/O* がある *EVAL-CORR*

```
 * The two data structures ds1 and ds2 have several
 * subfields, some having the same names and
 * compatible types:
 *    num     - appears in both, has compatible type
 *    extra   - appears only in ds1
 *    char    - appears in both, has identical type
 *    other   - appears only in ds1
 *    diff    - appears in both, types are not compatible
 *    another - appears only in ds2
D ds1             ds                  qualified
D   num                            10i 0
D   extra                            d
D   char                           20a
D   otherfld                        1a
D   diff                            5p 0
D ds2             ds                  qualified
D   char                           20a
D   diff                            5a
D   another                         5a
D   num                            15p 5

 /free
       // assign corresponding fields from DS1 to DS2
       EVAL-CORR ds2 = ds1;
       // this EVAL-CORR is equivalent to these EVAL operations
       // between all the subfields which have the same name
       // in both data structures and which have a compatible
       // data type
       //   EVAL ds2.num = ds1.num;
       //   EVAL ds2.char = ds1.char;
       // - Subfields "extra" and "another" are not affected
       //   because there is no subfield of the same name in
       //   the other data structure.
       // - Subfield "diff" is not selected because the
       //   subfields do not a compatible type
```

図 *322. プログラム記述データ構造の間の EVAL-CORR*

```
 * DDS for file EVALCORRN1
A          R REC1
A            FLD1          10A         ALWNULL
A            FLD2          10A         ALWNULL
A            FLD3          10A
A            FLD4          10A
A            FLD5           5P 0       ALWNULL
 * DDS for file EVALCORRN2
A          R REC2
A            FLD1          10A         ALWNULL
A            FLD2          10A
A            FLD3          10A         ALWNULL
A            FLD4          10A
A            FLD5           5A         ALWNULL
 * In the following example, data structure "ds1"
 * is defined from REC1 in file EVALCORRN1 and
 * data structure "ds2" is defined from REC2 in
 * file EVALCORRN2 above.  The EVAL-CORR operation
 * does the following:
 * 1. DS2.FLD1 is assigned the value of DS1.FLD1
 *    and %NULLIND(DS2.FLD1) is assigned the value of
 *    %NULLIND(DS1.FLD1)
 * 2. DS2.FLD2 is assigned the value of DS1.FLD2
 * 3. DS2.FLD3 is assigned the value of DS1.FLD3
 *    and %NULLIND(DS2.FLD3) is assigned *OFF
 * 4. DS2.FLD4 is assigned the value of DS1.FLD4
 * The null-indicator for DS1.FLD2 is ignored because
 * the target subfield DS2.FLD2 is not null-capable.
 * DS2.FLD5 is ignored because DS1.FLD5 has a different
 * data type, so the subfields do not correspond.
H ALWNULL(*USRCTL)
FEVALCORRN1IF   E             DISK
FEVALCORRN2O    E             DISK
D ds1             DS                  LIKEREC(REC1:*INPUT)
D ds2             DS                  LIKEREC(REC2:*OUTPUT)
C                   READ      REC1          ds1
C                   EVAL-CORR ds2 = ds1
C                   WRITE     REC2          ds2
```

図 *323.* ヌル対応サブフィールドがある *EVAL-CORR*

```
D ds0             ds                  qualified
D   num                          10i 0
D   char                         20a   varying
 * A data structure with a nested subfield data structure
D ds1             ds                  qualified
D   a                                 likeds(ds0)
D   b                                 likeds(ds0)
D   char                         20a   varying
D   otherfld                      1a
 * A data structure with a nested subfield data structure
D ds2             ds                  qualified
D   char                         20a
D   another                       5a
D   b                                 likeds(ds0)

 /free
       // assign corresponding fields from DS1 to DS2
       EVAL-CORR ds2 = ds1;
       // this EVAL-CORR is equivalent to these EVAL operations
       //   EVAL ds2.b.num = ds1.b.num;
       //   EVAL ds2.b.char = ds1.b.char;
       //   EVAL ds2.char = ds1.char;
       // assign corresponding fields from DS1.A to DS0
       EVAL-CORR(H) ds0 = ds1.a;
       // this EVAL-CORR is equivalent to these EVAL operations
       //   EVAL(H) ds0.num = ds1.a.num;
       //   EVAL ds0.char = ds1.a.char;
       // assign corresponding fields from DS1.A to DS2.B
       EVAL-CORR ds2.b = ds1.a;
       // this EVAL-CORR is equivalent to these EVAL operations
       //   EVAL ds2.b.num = ds1.a.num;
       //   EVAL ds2.b.char = ds1.a.char;
```

図 *324.* ネストされたサブフィールド・データ構造がある *EVAL-CORR*

EXCEPT (演算時出力)

自由形式構文	EXCEPT {例外名 }

コード	演算項目 1	演算項目 2	結果フィールド	標識		
EXCEPT		例外名				

EXCEPT 命令では、明細演算時または合計演算時に 1 つまたは複数のレコードを書き出すことができます。EXCEPT 命令の例については、780 ページの図 325 を参照してください。

EXCEPT 命令を指定する場合には、次のことに留意してください。

- 演算時に書き出される例外レコードは、出力仕様 の 17 桁目の E によって示されます。 EXCEPT 命令の例外名 オペランドに指定されたものと同じ名前の EXCEPT 名は、例外レコードの出力仕様の 30 から 39 桁目に指定することができます。
- EXCEPT 名を入れられるのは例外レコードだけで、見出し、明細、または 合計レコードには入れることができません。
- 例外名 オペランドに指定した名前の EXCEPT 命令が処理されると、条件付け標識が満たされた場合に、同じ EXCEPT 名の例外レコードだけが検査されて書き出されます。
- 例外名 が指定されていない場合には、条件付け標識が満たされた場合に、出力仕様の 30 から 39 桁目に名前が指定されていない例外レコードだけが検査されて書き出されます。
- 例外レコードが出力仕様のオーバーフロー標識で条件付けられている場合にレコードが書き出されるのは、RPG IV サイクルのオーバーフロー部分中またはフェッチ・オーバーフロー中のみです。 EXCEPT 命令の処理時にはレコードは書き出されません。
- フィールドが入っていない形式に例外出力が指定された場合には、次のようになります。
  - 出力ファイルが指定されている場合には、レコードはデフォルトの値で書き出 されます。

– レコードがロックされている場合には、システムはこの命令をレコードの アンロック要求として処理します。 これはアンロック要求の代替形式です。 UNLOCK 命令による方法が好ましい方式です。

詳しくは、576 ページの『ファイル操作』 を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  When the EXCEPT operation with HDG specified in factor 2 is
 *  processed, all exception records with the EXCEPT name HDG are
 *  written.  In this example, UDATE and PAGE would be printed
 *  and then the printer would space 2 lines.
 *  The second HDG record would print a line of dots and then the
 *  printer would space 3 lines.
 *
C                   EXCEPT    HDG
 *
 *  When the EXCEPT operation with no entry in factor 2 is
 *  processed, all exception records that do not have an EXCEPT
 *  name specified in positions 30 through 39 are written if the
 *  conditioning indicators are satisfied.  Any exception records
 *  without conditioning indicators and without an EXCEPT name
 *  are always written by an EXCEPT operation with no entry in
 *  factor 2.  In this example,  if indicator 10 is on, TITLE and
 *  AUTH would be printed and then the printer would space 1 line.
 *
C                   EXCEPT
O*
OFilename++DF..N01N02N03Excnam++++B++A++Sb+Sa+.............................
O..............N01N02N03Field+++++++++YB.End++PConstant/editword/DTformat++
O
O          E    10                     1
O                         TITLE
O                         AUTH
O          E              HDG            2
O                         UDATE
O                         PAGE
O          E              HDG            3
O                                                '.............'
O                                                '.............'
O          E              DETAIL         1
O                         AUTH
O                         VERSNO
```

図 *325.* 演算項目 *2* に指定がある場合/指定がない場合の *EXCEPT* 命令

EXFMT (形式の書き出し、その後読み取り)

自由形式構文	EXFMT{(E)} 様式名 *{データ構造}*

コード	演算項目 1	演算項目 2	結果フィールド	標識		
EXFMT (E)		様式名	データ構造	_	ER	_

EXFMT 命令は、WRITE とそれに続く同じレコード様式の READ の組み合わせです。 EXFMT が有効なのは、外部記述の全手順入出力共用ファイル として定義された WORKSTN ファイルの場合のみです。

形式名 オペランドは、書き出されてから読み取られるレコード様式の名前である必要があります。

データ構造オペランドが指定されている場合、レコードの書き込みおよび読み込みの対象はデータ構造になります。 このデータ構造は、EXTNAME(...:*ALL) または LIKEREC(...:*ALL) で定義されるデータ構造にする必要があります。 データ構造の定義方法、およびファイルとデータ構造の間のデータ転送方法については、576 ページの『ファイル操作』 を参照してください。

EXFMT 例外 (ファイル状況コードが 1000 より大きい) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。 エラーが発生した場合には、この命令

の読み取り 部分は処理されません (レコード識別標識およびフィールドは変更されません)。エラー処理について詳しくは、146 ページの『ファイル例外/エラー』を参照してください。

71 桁目、72 桁目、75 桁目、および 76 桁目はブランクでなければなりません。

複数装置ファイルでの EXFMT の使用法について は、READ (形式名による) および WRITE 命令の説明を参照してください。

詳しくは、「576 ページの『ファイル操作』」を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
F*Flename++IPEASFRLen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++
 *
 * PROMTD is a WORKSTN file which prompts the user for an option.
 * Based on what user enters, this program executes different
 * subroutines to add, delete, or change a record.
 *
FPROMTD    CF   E             WORKSTN

 /free
    // If user enters F3 function key, indicator *IN03 is set
    // on and the do while loop is exited.
    dow not *in03;

       // EXFMT writes out the prompt to the screen and expects user to
       // enter an option. SCR1 is a record format name defined in the
       // WORKSTN file and OPT is a field defined in the record.
       exfmt SCR1;
       select;
       when opt = 'A';
          exsr AddRec;
       when opt = 'D';
          exsr DelRec;
       when opt = 'C';
          exsr ChgRec;
       endsl;
    enddo;
    do_something ();
    do_more_stuff ();
 /end-free
```

図 *326. EXFMT* 命令

```
  * DDS for display file MYDSPF
A          R REC
A            QUESTION      40A  O  5  2
A            NAME          20A  I  7  5
A            CITY          20A  B  8  5
 * RPG program using MYDSPF
Fmydspf    cf   e             workstn
 * Define a data structure for use with EXFMT REC
D recDs           ds                  likerec(rec : *all)
 /free
        // Set the output-capable fields
        recDs.question = 'What is your name?';
        recDs.city = 'Toronto';        // Show the screen to the user
        exfmt rec recDs;
        // Use the input-capable fields
        // Since the "city" field is both input and output
        // capable, its value may have changed during EXFMT
        dsply ('Hello ' + recDs.name + ' in ' + recDs.city);
```

図 *327. EXFMT* が指定されているデータ構造の使用

EXSR (サブルーチンの呼び出し)

自由形式構文	EXSR サブルーチン名

コード	演算項目 1	演算項目 2	結果フィールド	標識		
EXSR		サブルーチン名				

EXSR 命令では、サブルーチン名 オペランドに指定された RPG IV サブルーチンが処理されます。 サブルーチン名は固有の記号名でなければならず、BEGSR 命令のサブルーチン名 オペランドとして現れている必要があります。 EXSR 命令は、演算仕様書の任意の場所に入れることができます。 この命令が現れると、指定されたサブルーチンが処理されます。 サブルーチン内の命令が処理されると、EXSR 命令の後のステートメントが処理されます。 ただし、サブルーチン内の GOTO にこのサブルーチンの外のラベルが渡された場合、またはサブルーチンが ENDSR 命令の戻り点 オペランドに指定された例外/エラー処理サブルーチンである場合を除きます。

サブルーチン名 オペランドに使用される *PSSR は、プログラム例外/エラー処理サブルーチンを処理することを指定します。サブルーチン名 オペランドに使用される *INZSR は、プログラム初期化サブルーチンを処理することを指定します。

詳しくは、594 ページの『サブルーチンのコーディング』、594 ページの『サブルーチン命令』、または 567 ページの『比較命令』を参照してください。

EXTRCT (日付/時刻/タイム・スタンプの抽出)

自由形式構文	(許可されていない - %SUBDT 組み込み関数を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
EXTRCT (E)		日付/時刻: 期間コード	ターゲット	_	ER	_

EXTRCT 命令は次の 1 つを結果フィールドに指定されたフィールドに戻します。

- 日付またはタイム・スタンプ・フィールドの年、月、または日の部分
- 時刻またはタイム・スタンプ・フィールドの時、分、または秒の部分
- タイム・スタンプ・フィールドのマイクロ秒部分

EXTRCT 命令は、タイム・スタンプ・フィールド からマイクロ秒 (秒の小数部 6 桁) を抽出する操作のみをサポートします。 異なる桁数の秒の小数部を抽出するには、%SUBDT 組み込み関数を使用します。

to the field specified in the result field.

情報を要求する日付、時刻、またはタイム・スタンプは、演算項目 2 に指定して、 その後に期間コードを続けます。 演算項目 2 に指定する記入項目 は、フィールド、サブフィールド、テーブル要素、または配列要素と することができます。 期間コードは、演算項目 2 のデータ・タイプと一致している必要があります。 有効な期間コードについては、572 ページの『日付命令』を参照してください。

演算項目 1 はブランクでなければなりません。

結果フィールドは、数値または文字フィールド、サブフィールド、配列/テー ブル要素とすることができます。 結果フィールドには、その内容が消去されてから、取り出されたデータが割り 当てられます。 文字の結果フィールドの場合には、データは左寄せされて結果フィールドに入 れられます。

注 : 年間通算日 (形式 *JUL) を指定した EXTRCT 命令を使用している場合には、 期間コード *D を指定するとその月の日が戻されて、*M を指定するとその年の 月が戻されます。 日と月を 3 桁の形式にしたい場合には、基底ポインターを使用して取り出すこ とができます。 年間通算日形式で表示する場合の例については、282 ページの図 105 を参照してください。

EXTRCT 例外 (プログラム状況コード 112) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、163 ページの『プログラム例外/エラー』を参照してください。

詳しくは、「572 ページの『日付命令』」を参照してください。

```
     D LOGONDATE       S               D
     D DATE_STR        S              15
     D MONTHS          S               8    DIM(12) CTDATA
     C*0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
      * Move the job date to LOGONDATE.  By default, LOGONDATE has an *ISO
      * date format, which contains a 4-digit year.  *DATE also contains a
      * 4-digit year, but in a different format, *USA.
     C     *USA          MOVE      *DATE         LOGONDATE
      *
      * Extract the month from a date field to a 2-digit field
      * that is used as an index into a character array containing
      * the names of the months.  Then extract the day from the
      * timestamp to a 2-byte character field which can be used in
      * an EVAL concatenation expression to form a string.
      * For example, if LOGONDATE is March 17, 1996, LOGMONTH will
      * contain 03, LOGDAY will contain 17, and DATE_STR will contain
      * 'March 17'.
     C                   EXTRCT    LOGONDATE:*M  LOGMONTH          2 0
     C                   EXTRCT    LOGONDATE:*D  LOGDAY            2
     C                   EVAL      DATE_STR = %TRIMR(MONTHS(LOGMONTH))
     C                                        + ' ' + LOGDAY
     C                   SETON                                       LR
** CTDATA MONTHS
January
February
March
April
May
June
July
August
September
October
November
December
```

図 *328. EXTRCT* 命令

FEOD (データの強制終了)

自由形式構文	FEOD{(EN)} ファイル名

コード	演算項目 1	演算項目 2	結果フィールド	標識		
FEOD (EN)		ファイル名		_	ER	_

FEOD 命令は、1 次ファイル、2 次ファイル、または全手順ファイルの論理 的なデータの終わりを知らせます。 FEOD 機能は、ファイル・タイプおよび装置によって 異なります (ファイル・タイプと装置による FEOD の違いについては、IBM i Information Center の「データベースおよびファイル・システム」のカテゴリーを参照してください。)

FEOD は次の点で CLOSE 命令と異なります。すなわち、プログラムが装置ま たはファイルから切り離されないので、ファイルに対する明示の OPEN 命令なし で、以後のファイル操作に再びそのファイルを使用することができます。

条件付け標識を指定することができます。 ファイル名 オペランドには FEOD を指定するファイルの名前を入れます。

ブロック化を使用する出力可能な DISK ファイルまたは SEQ ファイルに対し、FEOD に命令拡張 N を指定できます (157 ページの『ブロック化の考慮事項』を参照)。 命令拡張 N が指定された場合、ブロック内の書き出されていないレコードはデータベースに書き出されますが、 不揮発性の記憶域には必ずしも書き出されません。 N 拡張を使用することでパフォーマンスが改善されます。

FEOD 例外 (ファイル状況コードが 1000 より大きい) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、146 ページの『ファイル例外/エラー』を参照してください。

FEOD 命令の後でファイルに対する以後の順次命令 (例えば、READ または READP) を処理するためには、再度ファイルを位置決めしなければなりません。

詳しくは、「576 ページの『ファイル操作』」を参照してください。

FOR (For)

自由形式構文	FOR{(MR) } 指標名 {= 開始値 } {BY 増分 } {TO|DOWNTO 限界 }

コード	演算項目 1	拡張演算項目 2
FOR		指標名 = 開始値 BY 増分 TO | DOWNTO 限界

FOR 命令は、命令のグループを開始してそのグループを処理する回数を制御 します。 命令グループを処理する回数を指示するためには、指標名、開始値、 増分値、および限界値を指定します。 オプションの開始値、増分値、 および限界値は、自由形式の式にすることができます。 対応する END または ENDFOR ステートメント が、このグループの終わりを指示します。FOR グループについて詳しくは、591 ページの『構造化プログラミング命令』を参照してください。

FOR 命令の構文は次のとおりです。

```
      FOR         index-name { = starting-value }
                  { BY increment-value }
                  { TO | DOWNTO limit-value }
         { loop body }
      ENDFOR | END
```

開始値、増分値、および限界値 は、小数点以下の桁数がゼロの数値または式にすることができます。 増分値が指定されている場合、この値はゼロにはできません。

BY および TO (または DOWNTO) 文節は、どの順序でも指定できます。 "BY 2 TO 10" と "TO 10 BY 2" は両方とも可能です。

FOR 命令自体のほかに、FOR グループは、FOR および ENDFOR (また は END) ステートメントおよび ENDFOR ステートメントの条件付け標識によっても制御 されます。 FOR ステートメントの条件付け標識は、FOR 命令を開始するかどうかを 制御します。 これらの標識は for ループの始めに 1 回だけ 検査されます。 対応する END または ENDFOR ステートメントの条件付け標識 は、FOR グループをもう一度繰り返すかどうかを制御します。 これらの標識はそれぞれのループの終わりに検査されます。

FOR 命令は、次のように実行されます。

1. FOR 命令は、FOR ステートメント行の条件付け標識が満たされた場合 に処理されます (ステップ 2)。 この標識が満たされない場合には、対応する END、または ENDFOR ステートメントの後で処理される次の命令に 制御が渡されます (ステップ 8)。
2. 初期値が指定されている場合、指標名に割り当てられます。 指定されていなければ、指標名は、ループの開始前と同じ値を保存します。
3. 限界値が指定されている場合、指標名に対して評価、比較されます。 限界値が指定されていない場合、ループは、ループを終了する ステートメント (LEAVE や GOTO など) またはプログラムかプロシージャー を終了するステートメント (RETURN など) を検出するまで、無限に繰り返します。

   TO 文節が指定され、指標名の値が限界値より大きい場合、制御 は、ENDFOR ステートメント以降の最初のステートメントに渡されます。 DOWNTO が指定され、指標名が限界値より小さい場合、制御は、ENDFOR の 後の最初のステートメントに渡されます。
4. FOR グループ内の命令が処理されます。
5. END または ENDFOR ステートメントの条件付き標識が 満たされない場合、制御は、対応する END または ENDFOR の後のステートメントに 渡され、ループは終了します。
6. 増分値が指定されている場合、その値が評価されます。 指定されていなければ、デフォルト値として 1 を使用します。

7. 増分値は、指標名に加算 (TO の場合) されるか、指標名から減算 (DOWNTO の 場合) されます。 制御はステップ 3 に渡されます。(制御がステップ 3 に渡された時に FOR ステートメントの条件付け標識は再びテスト (ステップ 1) されないことに注意してください)。
8. FOR、END、または ENDFOR ステートメントの条件付け標識が 満たされない (ステップ 1 または 5) とき、指標値が限界値より 大きい (TO の場合) か小さい (DOWNTO の場合) とき (ステップ 3)、また は指標値がオーバーフローしたときには、END または ENDFOR ステートメント の後のステートメントが処理されます。

注 : FOR ループが *n* 回実行された 場合、限界値は *n+1* 回評価され、増分値 は *n* 回評価されます。 このことは、限界値 または増分値が複雑で、評価に時間がかかる場合や、限界値または増分値 に副次作用のあるサブプロシージャーへの呼び出しが含まれている場合 には、重要になる可能性があります。 限界または増分の多重評価が不要な場合は、FOR ループの前の一時の中の値を 計算し、FOR ループ内の一時を使用します。

FOR 命令を指定する場合には次のことに留意してください。

- FOR 命令では指標名を宣言できません。 変数は、定義仕様書内で宣言する必要があります。
- 指標名 は、指標付き配列要素を含む完全修飾名にすることができます。

これらの命令が FOR 命令に与える影響については、796 ページの『LEAVE (Do/For グループからの抜け出し)』 および 792 ページの『ITER (繰り返し)』 を参照してください。

詳しくは、「591 ページの『構造化プログラミング命令』」を参照してください。

```
*..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
/free
   // Example 1
   // Compute n!

   factorial = 1;
   for i = 1 to n;
      factorial = factorial * i;
   endfor;

   // Example 2
   // Search for the last nonblank character in a field.
   // If the field is all blanks, "i" will be zero.
   // Otherwise, "i" will be the position of nonblank.

   for i = %len (field) downto 1;
      if %subst(field: i: 1) <> ' ';
         leave;
      endif;
   endfor;

   // Example 3
   // Extract all blank-delimited words from a sentence.

   WordCnt = 0;
   for i = 1 by WordIncr to %len (Sentence);
      // Is there a blank?
      if %subst(Sentence: i: 1) = ' ';
         WordIncr = 1;
         iter;
      endif;

      // We've found a word - determine its length:
      for j = i+1 to %len(Sentence);
         if %subst (Sentence: j: 1) = ' ';
            leave;
         endif;
      endfor;

      // Store the word:
      WordIncr = j - i;
      WordCnt = WordCnt + 1;
      Word (WordCnt) = %subst (Sentence: i: WordIncr);
   endfor;

/end-free
```

図 *329. FOR* 命令の例

FOR-EACH (それぞれの場合)

自由形式構文	FOR-EACH{(H)} 項目 IN 配列 | %LIST | %SPLIT | %SUBARR

コード	演算項目 1	拡張演算項目 2
FOR-EACH		項目 IN 配列 | %LIST | %SPLIT | %SUBARR

FOR-EACH 命令は、配列、サブ配列、または %LIST 内の項目を一度に 1 つずつ処理するための命令のグループを開始します。

FOR-EACH の第 1 オペランドは変数です。 配列にはできません。

FOR-EACH の第 2 オペランドには、配列、%LIST、%SPLIT、または %SUBARR を指定できます。 これは、第 1 オペランドと互換性のあるデータ・タイプである必要があります。 FOR-EACH の第 1 オペランドがデータ構造である場合、第 2 オペランドは、LIKEDS キーワードによって第 1 オペランドに関連付けられている必要があります。

第 2 オペランドの各要素は、第 1 オペランドに反復的に割り当てられます。 FOR-EACH 命令と ENDFOR 命令の間の命令のグループ内では、第 1 オペランドは処理対象要素の値を保持します。

数値の場合は、四捨五入命令コード拡張「H」が許可されます。 四捨五入の規則は、算術演算の規則に相当します。558 ページの『精度の確認』

例

以下の例では、*order_states* 配列の各要素が *state* 変数に割り当てられています。

```
DCL-S order_states CHAR(10) DIM(3);
DCL-S state CHAR(20);

order_states(1) = 'Open';
order_states(2) = 'Active';
order_states(3) = 'Closed';
FOR-EACH state in order_states;
   DSPLY state;
ENDFOR;
```

プログラムは以下の出力を表示します。

```
DSPLY  Open
DSPLY  Active
DSPLY  Closed
```

以下の例の 2 つの FOR-EACH ステートメントは同じものです。ただし、2 番目の FOR-EACH 命令には命令拡張「H」があり、四捨五入が実行されることを示しています。

値 5.279 の *price(1)* を *price* に代入すると、小数点以下が 2 桁のみの *price* の値は、最初の FOR-EACH ループでは 5.27、四捨五入した 2 番目の FOR-EACH ループでは 5.28 となります。

```
DCL-S prices PACKED(15:5) DIM(2);
DCL-S price PACKED(15:2);

prices(1) = 5.279;
prices(2) = 5.271;

FOR-EACH price in prices;
   DSPLY (%char(price));
ENDFOR;

FOR-EACH(H) price in prices;
   DSPLY (%char(price));
ENDFOR;
```

プログラムは以下の出力を表示します。

```
DSPLY  5.27
DSPLY  5.27
DSPLY  5.28
DSPLY  5.27
```

組み込み関数 %LIST で指定されたリスト内の各項目の FOR-EACH 処理の例については、657 ページの『%LIST の例』 を参照してください。

組込み関数 %SPLIT が戻す一時配列の各項目を FOR-EACH で処理する例については、689 ページの『%SPLIT の例』を参照してください。

FORCE (次のサイクルでのファイルの強制読み取り)

自由形式構文	FORCE ファイル名

コード	演算項目 1	演算項目 2	結果フィールド	標識		
FORCE		ファイル名				

FORCE 命令では、次のレコードを読み取るファイルを選択することができます。 この命令は 1 次ファイルまたは 2 次ファイルにしか使用できません。

ファイル名 オペランドは、次のレコードを選択するファイルの名前でなければなりません。

FORCE 命令が処理されると、次のプログラム・サイクルの始めにそのレコードが読み取られます。 同じプログラム・サイクルの中で複数の FORCE 命令が処理される場合には、最後の命令以外はすべて無視されます。 FORCE は、合計時でなく、明細 時に出されなければなりません。

FORCE 命令は、プログラムが通常レコードを選択する複数のファイルの処理 方式を一時変更します。 しかし、最初に処理されるレコードは常に通常の方式で選択されます。 残りのレコードは FORCE 命令で選択することができます。FORCE 命令が突き合わせフィールドの処理に与える影響については、104 ページの図 11 を参照してください。

ファイルの終わりになっているファイルに FORCE を指定しても、そのファイ ルからはレコードは検索されません。 プログラム・サイクルによって次に読み 取るレコードが決定されます。

詳しくは、「576 ページの『ファイル操作』」 を参照してください。

GOTO (演算命令のスキップ)

自由形式構文	(許可されていない - LEAVE、LEAVESR、ITER、 および RETURN などの他の命令コードを使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
GOTO		ラベル				

GOTO 命令では、プログラムがそのプログラム内の別の演算命令に進む (ま たは分岐する) ように指示して、演算命令をスキップすることができます。 892 ページの『TAG (タグ)』 命令は、GOTO 命令の宛先を指定します。 TAG は、GOTO の前または後に指定することができます。 GOTO 命令は次の分岐を指定するために使用します。

- 明細演算行から別の明細演算行への分岐
- 合計演算行から別の合計演算行への分岐
- 明細演算行から合計演算行への分岐
- サブルーチンから同じサブルーチン内の TAG または ENDSR への分岐
- サブルーチンから明細演算行または合計演算行への分岐

サイクル・メイン・プロシージャーのサブルーチン内の GOTO は、同じサブルーチン、明細演算、または合計演算の中の TAG に出すことができます。 サブプロシージャーのサブ ルーチン内の GOTO は、同じサブルーチン内またはサブプロシージャーの本体 内の TAG に出すことができます。

RPG IV 論理サイクルの一部から別の部分に分岐すると、無限ループに入ることがあります。 ユーザーの責任で、ユーザー・プログラムの論理から好ま しくない結果が生じることのないようにしてください。

演算項目 2 には、プログラムの分岐先のラベルを入れなければなりません。 このラベルは、TAG または ENDSR 命令の演算項目 1 に入れます。 ラベルは固有 の記号名でなければなりません。

詳しくは、「562 ページの『分岐命令』」 を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  If indicator 10, 15, or 20 is on, the program branches to
 *  the TAG label specified in the GOTO operations.
 *  A branch within detail calculations.
C   10              GOTO      RTN1
 *
 *  A branch from detail to total calculations.
C   15              GOTO      RTN2
 *
C     RTN1          TAG
 *
C                      :
C                      :
C:
C   20              GOTO      END
 *
C                      :
C                      :
C                      :
C     END           TAG
 *  A branch within total calculations.
CL1                 GOTO      RTN2
CL1                 :
CL1   RTN2          TAG
```

図 *330. GOTO* および *TAG* 命令

IF (If)

自由形式構文	IF{(MR)} 標識式

コード	演算項目 1	拡張演算項目 2
IF (M/R)	ブランク	標識式

IF では、条件が満たされた場合に一連の命令コードを処理することができます。 その機能は IFxx 命令コードの機能と似ています。 相違点は、標識の値を示す式 (標識式) によって論理条件が表されることです。 IF 命令で制御される操作は、標識式 オペランドの式が真になった時に実行されます。命令拡張 M および R の使用法については、608 ページの『数値演算の精度の規則』を参照してください。

詳しくは、「591 ページの『構造化プログラミング命令』」を参照してください。

```
CL0N01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++..
C                           Extended-factor2-continuation+++++++++++++++
 * The operations controlled by the IF operation are performed
 * when the expression is true.  That is A is greater than 10 and
 * indicator 20 is on.
C
C                   IF        A>10 AND *IN(20)
C                   :
C                   ENDIF
 *
 * The operations controlled by the IF operation are performed
 * when Date1 represents a later date then Date2
C
C                   IF        Date1 > Date2
C                   :
C                   ENDIF
 *
```

図 *331. IF* 命令

IFxx (満たされた条件の処理)

自由形式構文	(許可されていない - IF 命令コードを使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
IFxx	被比較値	被比較値				

IFxx 命令では、演算項目 1 と演算項目 2 の間に xx で指定された特定の関係が存在する場合に演算のグループを処理することができます。 713 ページの『ANDxx (かつ)』 命令および 843 ページの『ORxx (または)』 命令を IFxx と一緒に使用するときに、命令を組み合わせて指定した条件が存在すると、演算のグループが実行されます (xx の意味については、591 ページの『構造化プログラミング命令』を参照してください)。

条件付け標識を使用することができます。 演算項目 1 と演算項目 2 には、リテラル、名前のついた定数、形象定数、 テーブル名、配列要素、データ構造名、またはフィールド名を入れなければな りません。 演算項目 1 と演算項目 2 は、同じデータ・タイプでなければなりません。

IFxx および対応する ANDxx または ORxx 命令によって指定された関係が存在しない 場合には、対応する ENDIF 命令の直後の演算命令に制御 が渡されます。 769 ページの『ELSE (他の場合)』 命令も指定されている場合、ELSE 命令の後で処理可能な最初の演算命令に制御が渡されます。

IFxx に対応する ENDIF 命令の条件付け標識の指定はブランクでなければな りません。

IFxx グループをクローズするためには、ENDIF ステートメントを使用しな ければなりません。 IFxx ステートメントの後に ELSE ステートメントが続く場 合には、IFxx ステートメントの後でなく、ELSE ステートメントの後に ENDIF ステートメントが必要です。

読みやすくするために、コンパイル・リストの DO ステートメント、IF-ELSE 文節、および SELECT-WHENxx-OTHER 文節は字下げすることができます。 ソース・リストのステートメントの字下げの方法については、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」のコンパイル・リストのセクションを参照してください。

詳しくは、567 ページの『比較命令』または 591 ページの『構造化プログラミング命令』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  If FLDA equals FLDB, the calculation after the IFEQ operation
 *  is processed.  If FLDA does not equal FLDB, the program
 *  branches to the operation immediately following the ENDIF.
C
C     FLDA          IFEQ      FLDB
C                       :
C                       :
C                       :
C                   ENDIF
C
 *  If FLDA equals FLDB, the calculation after the IFEQ operation
 *  is processed and control passes to the operation immediately
 *  following the ENDIF statement.  If FLDA does not equal FLDB,
 *  control passes to the ELSE statement and the calculation
 *  immediately following is processed.
C
C     FLDA          IFEQ      FLDB
C                       :
C                       :
C                       :
C                   ELSE
C                       :
C                       :
C                       :
C                   ENDIF
```

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  If FLDA is equal to FLDB and greater than FLDC, or if FLDD
 *  is equal to FLDE and greater than FLDF, the calculation
 *  after the ANDGT operation is processed.  If neither of the
 *  specified conditions exists, the program branches to the
 *  operation immediately following the ENDIF statement.
C
C     FLDA          IFEQ      FLDB
C     FLDA          ANDGT     FLDC
C     FLDD          OREQ      FLDE
C     FLDD          ANDGT     FLDF
C                       :
C                       :
C                       :
C                   ENDIF
```

図 *332. IFxx/ENDIF* および *IFxx/ELSE/ENDIF* 命令

IN (データ域の検索)

自由形式構文	IN{(E)}{*LOCK} データ域名

コード	演算項目 1	演算項目 2	結果フィールド	標識		
IN (E)	*LOCK	データ域名		_	ER	_

IN 命令では、データ域を検索して、別のプログラムによって更新されない ようにデータ域をロックするかどうかをオプションで指定することができます。 IN 命令 でデータ域を検索する場合には、*DTAARA DEFINE ステートメント の結果フィールドに指定するか、あるいは定義仕様書の DTAARA キーワードを 使用して指定しなければなりません。 (*DTAARA DEFINE 命令については 754 ページの『DEFINE (フィールド定義)』 を、DTAARA キーワードについては『定義仕様書』を参照してください。)

DTAARA(*VAR) がフィールドの定義に指定されたために、データ域の名前が実行時に判別される場合、そのデータ域名を含む変数は IN 命令の実行 前に設定する必要がありま す。 しかし、データ域が以前に実行した *LOCK IN 命令が原因でロックされている場合は、以前にロックさ れたデータ域が使用され、データ域の名前を含む変数は参照されません。

予約語 *LOCK を演算項目 1 で指定することにより、(1) UNLOCK 命令が処理されるか、(2) *data-area-name* オペランド が指定されていない OUT 命令が処理されるか、 あるいは (3) プログラムの終了時に RPG IV プログラムが暗黙にデータ域をアンロックしない限り、 データ域を別のプログラムが更新またはロックできないことを指示することができます。

データ域名 オペランドが内部データ域またはプログラム初期化パラメーター (PIP) データ域の名前である場合には、*LOCK は指定できません。

プログラムがすでにロックしているデータ域に *LOCK IN ステートメントを 指定することができます。 データ域名 オペランドが指定されていない場合には、ロック状況は、データ域が検索される前の状況と同じになります。 すなわち、ロックされていた場合にはロックされたままで、アンロックされていた場合にはアンロックされたままとなります。

データ域名は、DTAARA キーワード、*DTAARA DEFINE 命令の結果フィールド、または予約語 *DTAARA を使用して定義された定義名でなければなりません。 *DTAARA が指定された場合には、プログラム内で定義されているすべてのデー タ域が検索されます。 データ域の検索でエラーが発生した (例えば、データ域を検索できるが、ロックできない) 場合には、IN 命令でエラーが発生して RPG IV 例外/エラー処理ルーチンに制御が渡されます。 要求元にメッセージが出された場合には、そのメッセージによってエラーのあ るデータ域が識別されます。

IN 例外 (プログラム状況コード 401 から 421、431、 または 432) を処理するために、命令コード拡張 'E' またはエラー標識 ER を 指定できますが、両方を指定することはできません。エラー処理について詳しくは、163 ページの『プログラム例外/エラー』を参照してください。

固定形式演算では、71 から 72 桁目と 75 から 76 桁目はブランクでなければなりません。

IN 命令に関する規則について詳しくは、571 ページの『データ域命令』を参照してください。

```
 *..1....+....2....+....3....+....4....+....5....+....6....+....7...+....
 * Define Data areas
D TotAmt          s              8p 2 dtaara
D TotGrs          s             10p 2 dtaara
D TotNet          s             10p 2 dtaara

 *  TOTAMT, TOTGRS, and TOTNET are defined as data areas.  The IN
 *  operation retrieves all the data areas defined in the program
 *  and locks them.  The program processes calculations, and at
 *  LR time it writes and unlocks all the data areas.
 *  The data areas can then be used by other programs.

 /free

      in *lock *dtaara;
      TotAmt = TotAmt + Amount;
      TotGrs = TotGrs + Gross;
      TotNet = TotNet + Net;

 /end-free
 * To start total calcs, code a fixed format calc statement with a
 * level entry specified.
CL0   total_calcs   tag
 /free

      if *inlr
         out *dtaara
      endif
 /end-free
```

図 *333. IN および OUT 命令*

ITER (繰り返し)

自由形式構文	ITER

コード	演算項目 1	演算項目 2	結果フィールド	標識		
ITER						

ITER 命令は、DO または FOR グループの中からそのグループの ENDDO または ENDFOR ステートメントに制御を渡します。 この命令は、DO、DOU、DOUxx、 DOW、DOWxx、および FOR ループ内で、ループの ENDDO または ENDFOR ステートメントにただちに制御を渡すために使用できます。 ITER 命令によって、ループの次の繰り返しが直ちに実行されます。 ITER は一番内側のループに影響を与えます。

制御が渡される ENDDO または ENDFOR ステートメントに条件付け標識があってその条件が満たされない場合には、その ENDDO または ENDFOR 命令の後のステートメントから処理が続行されます。

796 ページの『LEAVE (Do/For グループからの抜け出し)』 命令は ITER 命令に似ていますが、LEAVE は、ENDDO 命令または ENDFOR 命令の**次の** ステートメントに制御を渡します。

詳しくは、562 ページの『分岐命令』または 591 ページの『構造化プログラミング命令』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 * The following example uses a DOU loop containing a DOW loop.
 * The IF statement checks indicator 01.  If indicator 01 is ON,
 * the LEAVE operation is executed, transferring control out of
 * the innermost DOW loop to the Z-ADD instruction.  If indicator
 * 01 is not ON, subroutine PROC1 is processed.  Then indicator
 * 12 is checked.  If it is OFF, ITER transfers control to the
 * innermost ENDDO and the condition on the DOW is evaluated
 * again.  If indicator 12 is ON, subroutine PROC2 is processed.
C
C                   DOU       FLDA = FLDB
C                   :
C     NUM           DOWLT     10
C                   IF        *IN01
C                   LEAVE
C                   ENDIF
C                   EXSR      PROC1
C     *IN12         IFEQ      *OFF
C                   ITER
C                   ENDIF
C                   EXSR      PROC2
C                   ENDDO
C                   Z-ADD     20            RSLT             2 0
C                   :
C                   ENDDO
C                   :
```

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 * The following example uses a DOU loop containing a DOW loop.
 * The IF statement checks indicator 1. If indicator 1 is ON, the
 * MOVE operation is executed, followed by the LEAVE operation,
 * transferring control from the innermost DOW loop to the Z-ADD
 * instruction.  If indicator 1 is not ON, ITER transfers control
 * to the innermost ENDDO and the condition on the DOW is
 * evaluated again.
C                   :
C     FLDA          DOUEQ     FLDB
C                   :
C     NUM           DOWLT     10
C     *IN01         IFEQ      *ON
C                   MOVE      'UPDATE'      FIELD            20
C                   LEAVE
C                   ELSE
C                   ITER
C                   ENDIF
C                   ENDDO
C                   Z-ADD     20            RSLT             2 0
C                   :
C                   ENDDO
C                   :
```

図 *334. ITER* 命令

KFLD (キーの構成部分定義)

自由形式構文	(許可されていない - %KDS を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
KFLD		標識	キー・フィールド			

KFLD 命令は宣言命令で、フィールドが KLIST 名で識別される検索引数の一部 であることを示します。

KFLD 命令は、合計演算を含む演算内の任意の場所に指定することができます。 制御レベル項目 (7 から 8 桁目) はブランクにするか、 あるいは L1 から L9 標識、LR 標識、または L0 項目を入れてプログラムの該当する セクション内のステートメントをグループにまとめることができます。 条件付け標識の指定 (9 から 11 桁目) は使用できません。

KFLD は大域または内部のいずれにすることもできます。 サイクル・メイン・プロシージャーの KLIST は、対応する大域の KFLD だけを持つことができます。 サブプロシージャーの KLIST は内部および大域の KFLD を持つことが できます。 詳しくは、95 ページの『定義の有効範囲』 を参照してください。

制御仕様書のキーワードとして、またはコマンド・パラメーターとし て、ALWNULL(*USRCTL) が指定されている場合、 演算項目 2 にはヌル値可能キー・フィールドの標識を入れることができます。

この標識がオンの場合、ヌル値のあるキー・フィールドが選択されます。 この標識がオフになっている場合、あるいは標識が指定されていない場合、 ヌル値のあるキー・フィールドは選択されません。 ヌル値可能キーへのアクセス方法については、289 ページの『キー順命令』 を参照してください。

結果フィールドには、検索引数の一部となるフィールドの名前を入れなければ なりません。 結果フィールドに配列名を入れることはできません。 それぞれの KFLD フィールドは、レコードまたはファイルの複合キーの対応す るフィールドと、長さ、データ・タイプ、および小数点以下の桁数が一致して いなければなりません。 ただし、レコードに可変長 KFLD フィールド がある場合、複合キー内の対応するフィールドは可変長でなければなりませんが、 同じ長さである必要はありません。 それぞれの KFLD は、複合キーの対応するフィールドと同じ名前であ る必要はありません。 KLIST に指定された KFLD フィールドの順序で、複合キーの特定のフィールド に対応する KFLD が決定されます。 例えば、KLIST 命令の後の最初の KFLD フィールドは複合キーの左端 (高位) のフィールドに対応します。

図形および UCS-2 のキー・フィールド には、そのファイル内のキーと同じ CCSID がなければなりません。

796 ページの図 335 に、KFLD 命令を伴う KLIST 命令の例を示します。

291 ページの図 111 は、ヌル・キーによってレコードを位置付け、検索するためのキー付き命令の使用法を示しています。

詳しくは、575 ページの『宣言命令』 を参照してください。

KLIST (複合キーの定義)

自由形式構文	(許可されていない - %KDS を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
KLIST	KLIST 名					

KLIST 命令は宣言命令で、KFLD のリストに名前を付けます。 このリストを 検索引数として使用すれば、複合キーを持つファイルからレコードを検索する ことができます。

KLIST は演算の中の任意の場所に指定することができます。 制御レベル項目 (7 から 8 桁目) はブランクにするか、 あるいは L1 から L9 標識、LR 標識、または L0 項目を入れてプログラムの該当する セクション内のステートメントをグループにまとめることができます。 条件付け標識の指定 (9 から 11 桁目) は使用できません。 演算項目 1 には固有の名前を入れなければなりません。

KLIST 命令を指定する場合には、次のことに留意してください。

- 検索引数が複数のフィールド (複合キー) から構成される場合には、複数 の KFLD を持つ KLIST を指定しなければなりません。
- KLIST 名を検索引数として指定できるのは、外部記述ファイルの場合だけ です。
- KLIST とそれに対応する KFLD フィールドは演算の中の任意の場所に入れ ることができます。
- KLIST の直後には少なくとも 1 つの KFLD がなければなりません。
- KLIST は、KFLD 以外の命令が見付かると終了します。
- KLIST 名は、CHAIN、DELETE、READE、READPE、SETGT、または SETLL 命令 の演算項目 1 に入れることができます。

- 同じ KLIST 名を複数のファイルに検索引数として使用するか、あるいは同 じファイルに検索引数として複数回使用することができます。
- サイクル・メイン・プロシージャーの KLIST は、対応する大域の KFLD だけを持つことができます。 サブプロシージャーの KLIST は内部および大域の KFLD を持つことが できます。 詳しくは、「95 ページの『定義の有効範囲』」を参照してください。

詳しくは、「575 ページの『宣言命令』」を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
A*  DDS source
A          R RECORD
A            FLDA           4
A            SHIFT          1  0
A            FLDB          10
A            CLOCK#         5  0
A            FLDC          10
A            DEPT           4
A            FLDD           8
A          K DEPT
A          K SHIFT
A          K CLOCK#
A*
A*  End of DDS source
A*
A******************************************************************************
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  The KLIST operation indicates the name, FILEKY, by which the
 *  search argument can be specified.
 *
C     FILEKY        KLIST
C                   KFLD                    DEPT
C                   KFLD                    SHIFT
C                   KFLD                    CLOCK#
```

次の図に、検索引数がどのようなものであるかを示します。 フィールド DEPT、SHIFT、および CLOCK# はこのレコードのキー・フィールド です。

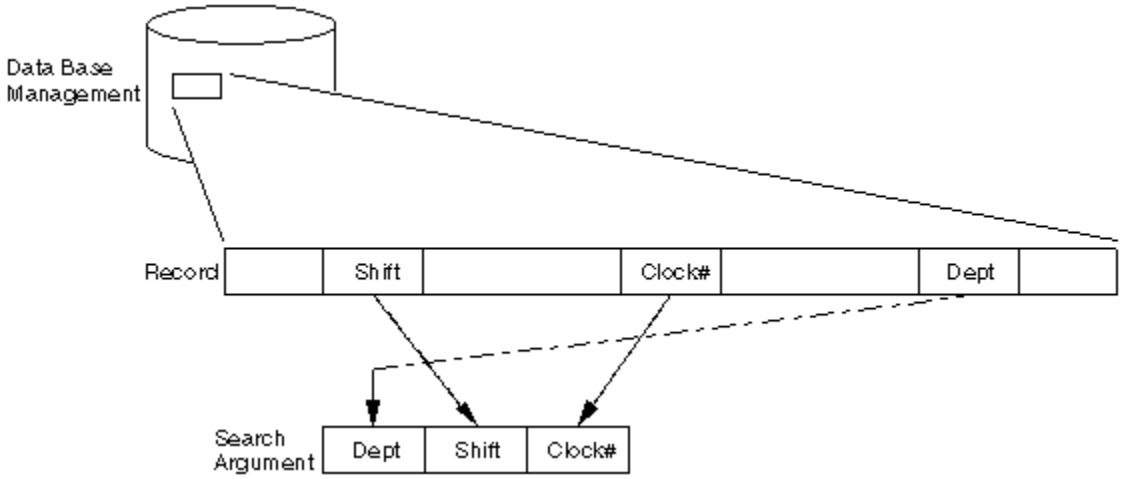


図 *335. KLIST* および *KFLD* 命令

LEAVE (Do/For グループからの抜け出し)

自由形式構文	LEAVE

コード	演算項目 1	演算項目 2	結果フィールド	標識		
LEAVE						

LEAVE 命令は、DO または FOR グループの中から ENDDO または ENDFOR 命令の後のステートメントに制御を渡します。

LEAVE は、DO、DOU、DOUxx、DOW、DOWxx、または FOR ループの中で、一番内側のループから、一番内側のループの ENDDO または ENDFOR 命令の後のステートメントにただちに制御を渡すために使用できます。 DO または FOR グループから抜けるために LEAVE を使用しても指標の増分は行われません。

ネストされたループでは、LEAVE によって 1 レベルのみ「外側」へ制御が渡されます。 DO または FOR グループの外側では LEAVE は使用できません。

792 ページの『ITER (繰り返し)』 命令は LEAVE 命令に似ていますが、ITER は ENDDO または ENDFOR ステートメントに制御を渡します。

詳しくは、562 ページの『分岐命令』または 591 ページの『構造化プログラミング命令』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 * The following example uses an infinite loop.  When the user
 * types 'q', control transfers to the LEAVE operation, which in
 * turn transfers control out of the loop to the Z-ADD operation.
 *
C     2             DOWNE     1
C                   :
C                   IF        ANSWER = 'q'
C                   LEAVE
C                   ENDIF
C                   :
C                   ENDDO
C                   Z-ADD     A             B
 *
 * The following example uses a DOUxx loop containing a DOWxx.
 * The IF statement checks indicator 1. If it is ON, indicator
 * 99 is turned ON, control passes to the LEAVE operation and
 * out of the inner DOWxx loop.
 *
 * A second LEAVE instruction is then executed because indicator 99
 * is ON, which in turn transfers control out of the DOUxx loop.
 *
C                   :
C     FLDA          DOUEQ     FLDB
C     NUM           DOWLT     10
C     *IN01         IFEQ      *ON
C                   SETON                                        99
C                   LEAVE
C                   :
C                   ENDIF
C                   ENDDO
C   99              LEAVE
C                   :
C                   ENDDO
C                   :
```

図 *336. LEAVE* 命令

LEAVESR (サブルーチンから抜け出す)

自由形式構文	LEAVESR

コード	演算項目 1	演算項目 2	結果フィールド	標識		
LEAVESR						

LEAVESR 命令は、サブルーチン内の任意のポイントからそのサブルーチンを 終了します。 制御は、そのサブルーチンの ENDSR 命令に渡されます。 LEAVESR はサブルーチンの中からしか使用できません。

制御レベル項目 (7 から 8 桁目) は SR または ブランクにすることができます。 条件付け標識項目 (9 から 11 桁目) は指定できます。

詳しくは、「594 ページの『サブルーチン命令』」を参照してください。

```
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq...
 *
C     CheckCustName BEGSR
C     Name          CHAIN     CustFile
 *
 * Check if the name identifies a valid customer
 *
C                   IF        not %found(CustFile)
C                   EVAL      Result = CustNotFound
C                   LEAVESR
C                   ENDIF
 *
 * Check if the customer qualifies for discount program
C                   IF        Qualified = *OFF
C                   EVAL      Result = CustNotQualified
C                   LEAVESR
C                   ENDIF
 *
 * If we get here, customer can use the discount program
C                   EVAL      Result = CustOK
C                   ENDSR
```

図 *337. LEAVESR* 命令

LOOKUP (テーブルまたは配列要素の検索)

自由形式構文	(許可されていない - %LOOKUP 組み込み関数または %TLOOKUP 組み込み関数を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
LOOKUP						
(配列)	検索引数	配列名		HI	LO	EQ
(テーブル)	検索引数	テーブル名	テーブル名	HI	LO	EQ

LOOKUP 命令では、配列またはテーブルの特定の要素の検索を行うことが できます。 演算項目 1 は検索引数 (指定された配列またはテーブル内で一致す るものを見付けたいデータ) です。 演算項目 1 は、リテラル、フィールド名、配列要素、テーブル名、名前のつい た定数、または形象定数とすることができます。 比較の性質はデータ・タイプによって異なります。

文字データ

制御仕様書に ALTSEQ(*EXT) が指定されている場合には、 文字の LOOKUP に代替照合順序が使用されます。 ただし、演算項目 1 または演算項目 2 のいずれかが、定義仕様書で ALTSEQ(*NONE) を指定 して定義されている場合は除きます。 ALTSEQ(*SRC) または代替順序が指定されてい ない場合には、文字の LOOKUP に代替順序は使用されません。

図形および UCS-2 データ

比較は 16 進数です。どのような場合にも代替照合順序は使用されません。

数値データ
演算項目 2 の配列またはテーブルのタイプが浮動である場合を除き、 数値データ内の小数点は無視されます。

その他のデータ・タイプ
567 ページの『比較命令』で説明されている比較に関する考慮事項は、その他のタイプにも適用されます。

演算項目 1 にテーブルが指定されている場合には、使用される検索引数は LOOKUP 命令で最後に選択されたテーブルの要素か、または前の LOOKUP が処理 されていなければテーブルの最初の要素になります。 検索する配列またはテーブルは演算項目 2 に指定します。

テーブルの LOOKUP の場合には、結果フィールドに 2 番目のテーブルの名前 を入れて、そこから要素 (最初のテーブルの要素と対応する位置にある) を検 索することができます。 検索したい要素を参照するために 2 番目のテーブルの 名前を使用することができます。 演算項目 2 に配列名が入っている場合には、結果フィールドはブランクでなければなりません。

結果の標識は LOOKUP の検索条件を指定します。 この標識は、最初に、実行される検索 を判別し、次にその検索の結果を反映するために、71 から 76 桁目に指定しなければなりません。 指定された標識がオンに設定されるのは、検索が正常に実行された場合だけです。 使用できる標識の数は 2 つ以下です。 結果の標識は、等 (EQ) と高 (HI) または等 (EQ) と低 (LO) に割り当てることができます。 プログラムは、指定された優先順位が等しいどちらかの 条件を満たす項目を検索します (すなわち、等しい項目が見付からない場合には、 低い方または高い方で一番近い項目が選択されます)。

75 から 76 桁目に標識を指定した場合、%EQUAL 組み込み関数は、検索引数に正確に一致 する要素が見付かった場合に '1' を戻します。 %FOUND 組み込み関数は、指定された検索が成功した場合に '1' を 戻します。

結果の標識は、等と低または等と高に割り当てることができます。 同じ LOOKUP 命令で高と低を指定することはできません。 コンパイラーは、LOOKUP 命令に高または低の標識が指定されている場合には、 配列またはテーブルが分類され、順序付けられていると見なします。 LOOKUP 命令は、指定された優先順位が等しいか低い/等しいか大きい/等 条件を満たす項目を検索します。

- 高 *(71-72):* プログラムに、検索引数に一番近くて、しかも順序が高い項目を見付けるように指示します。そのような高位の項目が見付かった場合には、高の標識がオンに設定されま す。 例えば、昇順の配列に値 A B C C C D E が入っていて、検索引数が B の場 合には、最初の C がこの検索の条件を満たすことになります。 降順の配列に E D C C C B A が入っていて、検索引数が B の場合には、最後 の C がこの検索の条件を満たすことになります。 配列またはテーブルに検索引数より高位の項目が見付からない場合には、検索 は失敗となります。
- 低 *(73-74):* プログラムに、検索引数に一番近くて、しかも順序が低い項目を見付けるように指示します。そのような低位の項目が見付かった場合には、低の標識がオンに設定されま す。 例えば、昇順の配列に値 A B C C C D E が入っていて、検索引数が D の場 合には、最後の C がこの検索の条件を満たすことになります。 降順の配列に E D C C C B A が入っていて、検索引数が D の場合には、最初 の C がこの検索の条件を満たすことになります。 配列またはテーブルに検索引数より低位の項目が見付からない場合には、検索 は失敗となります。
- 等 *(75-76):* プログラムに、検索引数に等しい項目を 見付けるように指示します。 等しい最初の項目が見付かると、 等の標識がオンに設定されます。 検索引数に等しい項目が見付からない場合には、検索は失敗となります。

LOOKUP 命令を使用する場合には、次のことに留意してください。

- 検索引数と配列またはテーブルは、同じタイプで同じ長さでなければなり ません (長さが異なる場合がある時刻および日付フィールドを除く)。 配列またはテーブルが固定長文字、図形、または UCS-2 である場合、検索引数も固定長でなければなりません。 可変長の 場合、検索引数の長さは、配列またはテーブルと異なる長さでもかまいません。
- 配列で LOOKUP が処理されて指標が使用されている場合には、LOOKUP はそ の指標によって指定された要素から開始されます。 指標値は見付かった要素の 位置番号に設定されます。 検索の開始時に、指標がゼロに等しいかまたは配 列内の要素の数より大きい場合にはエラーになります。 検索が失敗となった場合には、指標は 1 に設定されます。 指標が名前のついた定数の場合には、指標値は変わりません。
- 高、低、高と等、または低と等について検索できるのは、定義仕様書で配列 またはテーブルに ASCEND または DESCEND キーワードで順序が指定されている 場合だけです。

- 検索が失敗となった場合には、結果の標識はオンに設定されません。
- 等の標識 (75 から 76 桁目) だけが使用されている場合には、LOOKUP 命令で 配列またはテーブル全体が検索されます。 配列またはテーブルが昇順になっていて、等しいかどうかだけを比較したい場 合には、高の標識を指定することによって配列またはテーブル全体の検索を避 けることができます。
- 配列が昇順または降順になっていない場合には、LOOKUP 命令で予期しない 結果になることがあります。
- すべての定義済み要素が割り振り済みでない動的割り振り配列に 対する LOOKUP 命令によって、エラーが発生する可能性があります。

詳しくは、「561 ページの『配列命令』」を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  In this example, the programmer wants to know which element in
 *  ARY the LOOKUP operation locates.  The Z-ADD operation sets the
 *  field X to 1.  The LOOKUP starts at the element ARY that is
 *  indicated by field X and continues running until it finds the
 *  first element equal to SRCHWD.  The index value, X, is set to
 *  the position number of the element located.
C
C                   Z-ADD     1             X               3 0
C     SRCHWD        LOOKUP    ARY(X)                                26
C
 *  In this example, the programmer wants to know if an element
 *  is found that is equal to SRCHWD.  LOOKUP searches ARY until it
 *  finds the first element equal to SRCHWD.  When this occurs,
 *  indicator 26 is set on and %EQUAL is set to return '1'.
C
C     SRCHWD        LOOKUP    ARY                                   26
C
 *  The LOOKUP starts at a variable index number specified by
 *  field X.  Field X does not have to be set to 1 before the
 *  LOOKUP operation.  When LOOKUP locates the first element in
 *  ARY equal to SRCHWD, indicator 26 is set on and %EQUAL is set
 *  to return '1'.  The index value, X, is set to the position
 *  number of the element located.
 *
C
C     SRCHWD        LOOKUP    ARY(X)                                26
```

図 *338.* 配列での *LOOKUP* 命令

```
  * In this example, an array of customer information actually consists
  * of several subarrays.  You can search either the main array or the
  * subarrays overlaying the main array.
 D custInfo        DS
 D  cust                              DIM(100)
 D   name                       30A   OVERLAY(cust : *NEXT)
 D   id_number                  10I 0 OVERLAY(cust : *NEXT)
 D   amount                     15P 3 OVERLAY(cust : *NEXT)

  * You can search for a particular set of customer information
  * by doing a search on the "cust" array
 C     custData      LOOKUP    cust(i)                               10

  * You can search on a particular field of the customer information
  * by doing a search on one of the overlay arrays
 C     custName      LOOKUP    name(i)                               11
  * After the search, the array index can be used with any of the
  * overlaying arrays.  If the search on name(i) is successful,
  * the id_number and amount for that customer are available
  * in id_number(i) and amount(i).
```

図 *339.* 副配列を使用した *LOOKUP* 命令

MHHZO (上位桁から上位桁へのゾーンの転送)

自由形式構文	(許可されていない - %BITAND および %BITOR 組み込み関数を使用。620 ページの図 206 を参照。)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
MHHZO		ソース・フィールド	ターゲット・フィールド			

MHHZO 命令では、演算項目 2 の左端のゾーンから結果フィールドの左端のゾーンに文字のゾーン部分が転送されます。 演算項目 2 と結果フィールドは両方とも文字フィールドとして定義されていなければなりません。MHHZO 命令の機能について詳しくは、588 ページの『ゾーン移動命令』 を参照してください。

MHHZO 命令の機能は、589 ページの図 189 に示しています。

MHLZO (上位桁から下位桁へのゾーンの転送)

自由形式構文	(許可されていない - %BITAND および %BITOR 組み込み関数を使用。620 ページの図 206 を参照。)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
MHLZO		ソース・フィールド	ターゲット・フィールド			

MHLZO 命令では、演算項目 2 の左端のゾーンから結果フィールドの右端のゾーンに文字のゾーン部分が転送されます。 演算項目 2 は文字フィールドとして定義されていなければなりません。 結果フィールドは文字データでも数値データでもかまいません。MHLZO 命令について詳しくは、588 ページの『ゾーン移動命令』を参照してください。

MHLZO 命令の機能は、589 ページの図 189 に示しています。

MLHZO (下位桁から上位桁へのゾーンの転送)

自由形式構文	(許可されていない - %BITAND および %BITOR 組み込み関数を使用。620 ページの図 206 を参照。)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
MLHZO		ソース・フィールド	ターゲット・フィールド			

MLHZO 命令では、演算項目 2 の右端のゾーンから結果フィールドの左端のゾーンに文字のゾーン部分が転送されます。 演算項目 2 は数値フィールドまたは文字フィールドとして定義できますが、結果フィールドは文字フィールドでなければなりません。MLHZO 命令について詳しくは、588 ページの『ゾーン移動命令』 を参照してください。

MLHZO 命令の機能は、589 ページの図 189 に示しています。

MLLZO (下位桁から下位桁へのゾーンの転送)

自由形式構文	(許可されていない - %BITAND および %BITOR 組み込み関数を使用。620 ページの図 206 を参照。)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
MLLZO		ソース・フィールド	ターゲット・フィールド			

MLLZO 命令では、演算項目 2 の右端のゾーンから結果フィールドの右端のゾ ーンに文字のゾーン部分が転送されます。 演算項目 2 と結果フィールドは文字データまたは数値データのいずれであって もかまいません。MLLZO について詳しくは、588 ページの『ゾーン移動命令』を参照してください。

MLLZO 命令の機能は、589 ページの図 189 に示してあります。

MONITOR (監視グループの始め)

自由形式構文	MONITOR

コード	演算項目 1	演算項目 2	結果フィールド	標識		
MONITOR						

MONITOR グループは、状況コードに基づいて、条件付きエラー処理を実行します。 グループは次のものから構成されます。

- MONITOR ステートメント
- 1 つまたは複数の ON-ERROR グループ
- 1 つの ENDMON ステートメント

MONITOR ステートメントの後は、制御は次のステートメントに渡されます。 MONITOR ブロックは、MONITOR ステートメントから最初の ON-ERROR ステートメントまでのすべてのステートメントで構成されます。 MONITOR ブロックの処理の際にエラーが発生すると、適切な ON-ERROR グループに制御が渡ります。

MONITOR ブロックのすべてのステートメントがエラーなしに処理された場合は、 ENDMON ステートメントの次のステートメントへ制御が渡されます。

MONITOR グループは演算の中の任意の場所に指定することができます。 IF、DO、SELECT、または他の MONITOR グループの中でネストすることもできます。 IF、DO、および SELECT グループは、MONITOR グループの中でネストすることができます。

MONITOR グループが別の MONITOR グループの中でネストされている場合、エラーが起こると、最も深いグループが最初に考慮されます。 その MONITOR グループがそのエラー条件を処理しない場合は、次のグループが考慮されます。

MONITOR 命令ではレベル標識が使用され、MONITOR グループが合計演算の一部であることを示します。文書化を目的としてレベル標識を ON-ERROR または ENDMON 命令に指定することもできますが、このレベル標識は無視されることになります。

MONITOR ステートメントでは、条件付け標識を使用することができます。 条件付け標識が満たされない場合には、MONITOR グループの ENDMON 命令の後のステートメントに即時に制御が渡されます。 ON-ERROR 命令に単独で条件付け標識を使用することはできません。

MONITOR ブロックがサブプロシージャーの呼び出しを含んでおり、かつそのサブプロシージャーにエラーがある場合は、そのサブプロシージャーのエラー処理が優先順位を有します。 例えば、サブプロシージャーに *PSSR サブルーチンがあれば呼び出されます。 呼び出しを含む MONITOR グループは、サブプロシージャーがエラー処理に失敗し、00202 の「呼び出し中のエラー」 状況で呼び出しが失敗した場合にのみ、考慮されることになります。

MONITOR グループはサブルーチンで発生したエラーを処理します。 サブルーチンが自分自身の MONITOR グループを持っている場合は、それらが最初に考慮されます。

MONITOR ブロックの中での分岐命令は許されませんが、ON-ERROR ブロックの中であれば許されます。

MONITOR ブロックの中での LEAVE 命令または ITER 命令は、その MONITOR ブロックを含んでいるすべての活動状態の DO グループに対して適用されます。 MONITOR ブロックの中での LEAVESR 命令または RETURN 命令は、その MONITOR ブロックを含んでいるすべてのサブルーチン、サブプロシージャー、またはプロシージャーに対して適用されます。

詳しくは、「575 ページの『エラー処理命令』」を参照してください。

```
 * The MONITOR block consists of the READ statement and the IF
 * group.
 * - The first ON-ERROR block handles status 1211 which
 *   is issued for the READ operation if the file is not open.
 * - The second ON-ERROR block handles all other file errors.
 * - The third ON-ERROR block handles the string-operation status
 *   code 00100 and array index status code 00121.
 * - The fourth ON-ERROR block (which could have had a factor 2
 *   of *ALL) handles errors not handled by the specific ON-ERROR
 *   operations.
 *
 * If no error occurs in the MONITOR block, control passes from the
 * ENDIF to the ENDMON.
C                   MONITOR
C                   READ       FILE1
C                   IF         NOT %EOF
C                   EVAL       Line = %SUBST(Line(i) :
C                                            %SCAN('***': Line(i)) + 1)
C                   ENDIF
C                   ON-ERROR   1211
C                    ... handle file-not-open
C                   ON-ERROR   *FILE
C                    ... handle other file errors
C                   ON-ERROR   00100 : 00121
C                    ... handle string error and array-index error
C                   ON-ERROR
C                    ... handle all other errors
C                   ENDMON
```

図 *340. MONITOR* 命令

MOVE (転送)

自由形式構文	(許可されていない - EVAL または EVALR 命令、 または %CHAR、%DATE、 %DEC、%DECH、%GRAPH、%INT、%INTH、 %TIME、%TIMESTAMP、%UCS2 、%UNS、あるいは %UNSH などの組み込み関数を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
MOVE(P)	データ属性	ソース・フィールド	ターゲット・フィールド	+	-	ZB

MOVE 命令では、演算項目 2 から結果フィールドに文字が転送されます。 転送は演算項目 2 の右端の文字から開始されます。

日付、時刻またはタイム・スタンプ・データを転送する 場合には、転送元または転送先が文字または数値フィールドのいずれかでない 限り、演算項目 1 はブランクでなければなりません。

そうでない場合には、演算項目 1 にこの命令の転送元または転送先の文字また は数値フィールドに対応する日付または時刻の形式が入ります。 使用可能な形式については、273 ページの『日付データ・タイプ』、275 ページの『時刻データ・タイプ』、および 277 ページの『タイム・スタンプ・データ・タイプ』を参照してください。

転送元または転送先が文字フィールドである場合、オプションで、 演算項目 1 内の形式の後に区切り記号を示すことができます。 ただし、その形式で有効な区切り記号だけを使用する ことができます。

演算項目 2 が *DATE または UDATE で、結果が日付フィールドである 場合、演算項目 1 は必要ありません。演算項目 1 に日付形式が含まれている場合、演算項目 1 は、制御仕様書 の DATEDIT キーワードによって指定されている *DATE または UDATE の 形式と互換性を持っている必要があります。

文字データ、図形データ、UCS-2 データ、または 数値データを転送する場合に、演算項目 2 が結果フィールドより長い場合 には、演算項目 2 の左端の余分な文字または数字は 転送されません。 結果フィールドが演算項目 2 より長い場合には、埋め込みが指定されていない限 り、結果フィールドの左端の余分な文字または数字は変更されません。

結果フィールドが配列の場合には、結果の標識を指定することはできません。 結果の標識を指定できるのは、結果フィールドが配列要素または配列以外の フィールドの場合です。

演算項目 2 が結果フィールドの長さより短い場合には、命令拡張桁に P を指定 することによって、転送後に結果フィールドの左側に埋め込みが行うことがで きます。

浮動数値フィールドおよびリテラルを、演算項目 2 または結果フィールドの 指定として使用することはできません。

モジュールについて CCSID(*GRAPH : IGNORE) が指定または想定されている 場合、UCS-2 データと図形データの間で MOVE 命令は使用できません。

可変長文字、図形、または UCS-2 データを 転送する場合は、可変長フィールドは、現在の長さが同じ固定長フィールドと まったく同様に機能します。 MOVE 命令では、可変長結果フィールドの長さは変わりません。 例については、808 ページの図 345 から 810 ページの図 350 を参照してください。 この例の GRAPHIC リテラルは、無効な GRAPHIC リテラルです。 詳しくは、252 ページの『グラフィック形式』を参照してください。

例の後の表に、演算項目 2 から結果フィールドにどのようにデータが転送され るかを示します。 MOVE 命令について詳しくは、583 ページの『移動命令』または 569 ページの『変換命令』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
 *  Control specification date format
H DATFMT(*ISO)
 *
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++++++
D DATE_ISO       S               D
D DATE_YMD       S               D   DATFMT(*YMD)
D                                    INZ(D'1992-03-24')
D DATE_EUR       S               D   DATFMT(*EUR)
D                                    INZ(D'2197-08-26')
D DATE_JIS       S               D   DATFMT(*JIS)
D NUM_DATE1      S              6P 0 INZ(210991)
D NUM_DATE2      S              7P 0
D CHAR_DATE      S              8    INZ('02/01/53')
D CHAR_LONGJUL   S              8A   INZ('2039/166')
D DATE_USA       S               D   DATFMT(*USA)
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+H1LoEq..
 * Move between Date fields. DATE_EUR will contain 24.03.1992
 *
C                   MOVE      DATE_YMD      DATE_EUR
 *
 * Convert numeric value in ddmmyy format into a *ISO Date.
 * DATE_ISO will contain 1991-09-21 after each of the 2 moves.
C     *DMY          MOVE      210991        DATE_ISO
C     *DMY          MOVE      NUM_DATE1     DATE_ISO
 *
 * Move a character value representing a *MDY date to a *JIS Date.
 * DATE_JIS will contain 1953-02-01 after each of the 2 moves.
C     *MDY/         MOVE      '02/01/53'    DATE_JIS
C     *MDY/         MOVE      CHAR_DATE     DATE_JIS
 *
 *  Move a date field to a character field, using the
 *  date format and separators based on the job attributes
C     *JOBRUN       MOVE (P)  DATE_JIS      CHAR_DATE
 *
 *  Move a date field to a numeric field, using the
 *  date format based on the job attributes
 *
 *  Note:  If the job format happens to be *JUL, the date will
 *         be placed in the rightmost 5 digits of NUM_DATE1.
 *         The MOVEL operation might be a better choice.
 *
C     *JOBRUN       MOVE (P)  DATE_JIS      NUM_DATE1
 *
 *  DATE_USA will contain 12-31-9999
C                   MOVE      *HIVAL        DATE_USA
 *
 *  Execution error, resulting in error code 114.  Year is not in
 *  1940-2039 date range.  DATE_YMD will be unchanged.
C                   MOVE      DATE_USA      DATE_YMD
 *
 *  Move a *EUR date field to a numeric field that will
 *  represent a *CMDY date.  NUM_DATE2 will contain 2082697
 *  after the move.
C     *CMDY         MOVE      DATE_EUR      NUM_DATE2
 *
 *  Move a character value representing a *LONGJUL date to
 *  a *YMD date.  DATE_YMD will be 39/06/15 after the move.
C     *LONGJUL      MOVE      CHAR_LONGJUL  DATE_YMD
```

図 *341.* 日付の *MOVE* 命令

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
 *  Specify default format for date fields
H DATFMT(*ISO)
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++++++
D date_USA       S               D   DATFMT(*USA)
D datefld        S               D
D timefld        S               T   INZ(T'14.23.10')
D chr_dateA      S              6    INZ('041596')
D chr_dateB      S              7    INZ('0610807')
D chr_time       S              6
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+H1LoEq..
 * Move a character value representing a *MDY date to a D(Date) value.
 * *MDY0 indicates that the character date in Factor 2 does not
 * contain separators.
 * datefld will contain 1996-04-15 after the move.
C     *MDY0         MOVE      chr_dateA     datefld
 * Move a field containing a T(Time) value to a character value in the
 * *EUR format.  *EUR0 indicates that the result field should not
 * contain separators.
 * chr_time will contain '142310' after the move.
C     *EUR0         MOVE      timefld       chr_time
 * Move a character value representing a *CYMD date to a *USA
 * Date. Date_USA will contain 08/07/1961 after the move.
 * 0 in *CYMD indicates that the character value does not
 * contain separators.
C     *CYMD0        MOVE      chr_dateB     date_USA
```

図 *342.* 区切り記号を使用しない 日付と時刻の *MOVE* 命令

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
 *  Control specification DATEDIT format
 *
H DATEDIT(*MDY)
 *
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++
D Jobstart        S               Z
D Datestart       S               D
D Timestart       S               T
D Timebegin       S               T   inz(T'05.02.23')
D Datebegin       S               D   inz(D'1991-09-24')
D TmStamp         S               Z   inz
 *
 * Set the timestamp Jobstart with the job start Date and Time
 *
 * Factor 1 of the MOVE *DATE (*USA = MMDDYYYY) is consistent
 * with the value specified for the DATEDIT keyword on the
 * control specification, since DATEDIT(*MDY) indicates that
 * *DATE is formatted as MMDDYYYY.
 *
 * Note:   It is not necessary to specify factor 1 with *DATE or
 *         UDATE.
 *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
C     *USA          MOVE      *DATE         Datestart
C                   TIME                    StrTime           6 0
C     *HMS          MOVE      StrTime       Timestart
C                   MOVE      Datestart     Jobstart
C                   MOVE      Timestart     Jobstart
 *
 * After the following C specifications are performed, the field
 * stampchar will contain '1991-10-24-05.17.23.000000'.
 *
 * First assign a timestamp the value of a given time+15 minutes and
 * given date + 30 days. Move tmstamp to a character field.
 * stampchar will contain '1991-10-24-05.17.23.000000'.
 *
C                   ADDDUR    15:*minutes   Timebegin
C                   ADDDUR    30:*days      Datebegin
C                   MOVE      Timebegin     TmStamp
C                   MOVE      Datebegin     TmStamp
C                   MOVE      TmStamp       stampchar        26
 * Move the timestamp to a character field without separators. After
 * the move, STAMPCHAR will contain '      19911024051723000000'.
C     *ISO0         MOVE(P)   TMSTAMP       STAMPCHAR0
```

図 *343.* タイム・スタンプを使用した *MOVE* 命令

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
 *
 * Example of MOVE between graphic and character fields
 *
D char_fld1      S             10A   inz('oK1K2K3  i')
D dbcs_fld1      S              4G
D char_fld2      S             10A   inz(*ALL'Z')
D dbcs_fld2      S              3G   inz(G'oK1K2K3i')
 *
 *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
 *
 * Value of dbcs_fld1 after MOVE operation is 'K1K2K3  '
 * Value of char_fld2 after MOVE oepration is 'ZZoK1K2K3i'
 *
C                   MOVE      char_fld1     dbcs_fld1
C                   MOVE      dbcs_fld2     char_fld2
```

図 *344.* 文字と図形フィールドの間の *MOVE*

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName+++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
 *
 * Example of MOVE from variable to variable length
 * for character fields
 *
D var5a          S              5A   INZ('ABCDE') VARYING
D var5b          S              5A   INZ('ABCDE') VARYING
D var5c          S              5A   INZ('ABCDE') VARYING
D var10a         S             10A   INZ('0123456789') VARYING
D var10b         S             10A   INZ('ZXCVBNM') VARYING
D var15a         S             15A   INZ('FGH') VARYING
D var15b         S             15A   INZ('FGH') VARYING
D var15c         S             15A   INZ('QWERTYUIOPAS') VARYING
 *
 *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
 *
C                   MOVE      var15a        var5a
 * var5a = 'ABFGH' (length=5)
C                   MOVE      var10a        var5b
 * var5b = '56789' (length=5)
C                   MOVE      var5c         var15a
 * var15a = 'CDE' (length=3)
C                   MOVE      var10b        var15b
 * var15b = 'BNM' (length=3)
C                   MOVE      var15c        var10b
 * var10b = 'YUIOPAS' (length=7)
```

図 *345.* 可変長フィールドから 可変長フィールドへの *MOVE*

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName+++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
 *
 * Example of MOVE from variable to fixed length
 * for character fields
 *
D var5           S              5A   INZ('ABCDE') VARYING
D var10          S             10A   INZ('0123456789') VARYING
D var15          S             15A   INZ('FGH') VARYING
D fix5a          S              5A   INZ('MNOPQ')
D fix5b          S              5A   INZ('MNOPQ')
D fix5c          S              5A   INZ('MNOPQ')
 *
 *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
 *
C                   MOVE      var5          fix5a
 * fix5a = 'ABCDE'
C                   MOVE      var10         fix5b
 * fix5b = '56789'
C                   MOVE      var15         fix5c
 * fix5c = 'MNFGH'
```

図 *346.* 可変長フィールドから 固定長フィールドへの *MOVE*

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName+++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
 *
 * Example of MOVE from fixed to variable length
 * for character fields
 *
D var5           S              5A   INZ('ABCDE') VARYING
D var10          S             10A   INZ('0123456789') VARYING
D var15          S             15A   INZ('FGHIJKL') VARYING
D fix5           S              5A   INZ('.....')
D fix10          S             10A   INZ('PQRSTUVWXY')
 *
 *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
 *
C                   MOVE      fix10         var5
 * var5 = 'UVWXY' (length=5)
C                   MOVE      fix5          var10
 * var10 = '01234.....' (length=10)
C                   MOVE      fix10         var15
 * var15 = 'STUVWXY' (length=7)
```

図 *347.* 固定長フィールドから 可変長フィールドへの *MOVE*

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName+++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
 *
 * Example of MOVE(P) from variable to variable length
 * for character fields
 *
D var5a          S              5A   INZ('ABCDE') VARYING
D var5b          S              5A   INZ('ABCDE') VARYING
D var5c          S              5A   INZ('ABCDE') VARYING
D var10          S             10A   INZ('0123456789') VARYING
D var15a         S             15A   INZ('FGH') VARYING
D var15b         S             15A   INZ('FGH') VARYING
D var15c         S             15A   INZ('FGH') VARYING
 *
 *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
 *
C                   MOVE(P)   var15a        var5a
 * var5a = '  FGH' (length=5)
C                   MOVE(P)   var10         var5b
 * var5b = '56789' (length=5)
C                   MOVE(P)   var5c         var15b
 * var15b = 'CDE' (length=3)
C                   MOVE(P)   var10         var15c
 * var15c = '789' (length=3)
```

図 *348.* 可変長フィールドから 可変長フィールドへの *MOVE(P)*

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
 *
 * Example of MOVE(P) from variable to fixed length
 * for character fields
 *
D var5           S              5A   INZ('ABCDE') VARYING
D var10          S             10A   INZ('0123456789') VARYING
D var15          S             15A   INZ('FGH') VARYING
D fix5a          S              5A   INZ('MNOPQ')
D fix5b          S              5A   INZ('MNOPQ')
D fix5c          S              5A   INZ('MNOPQ')
 *
 *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
 *
C                   MOVE(P)   var5          fix5a
 * fix5a = 'ABCDE'
C                   MOVE(P)   var10         fix5b
 * fix5b = '56789'
C                   MOVE(P)   var15         fix5c
 * fix5c = '  FGH'
```

図 *349.* 可変長フィールドから 固定長フィールドへの *MOVE(P)*

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
 *
 * Example of MOVE(P) from fixed to variable length
 * for character fields
 *
D var5           S              5A   INZ('ABCDE') VARYING
D var10          S             10A   INZ('0123456789') VARYING
D var15a         S             15A   INZ('FGHIJKLMNOPQR') VARYING
D var15b         S             15A   INZ('FGHIJ') VARYING
D fix5           S              5A   INZ('')
D fix10          S             10A   INZ('PQRSTUVWXY')
 *
 *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
 *
C                   MOVE(P)   fix10         var5
 * var5 = 'UVWXY' (length=5 before and after)
C                   MOVE(P)   fix10         var10
 * var10 = 'PQRSTUVWXY' (length=10 before and after)
C                   MOVE(P)   fix10         var15a
 * var15a = '   PQRSTUVWXY' (length=13 before and after)
C                   MOVE(P)   fix10         var15b
 * var15b = 'UVWXY' (length=5 before and after)
```

図 *350.* 固定長フィールドから 可変長フィールドへの *MOVE(P)*

表 *125.* 日付時刻フィールドへの文字フィールドの転送. 演算項目 1 は演算項目 2 の指定の形式を示します。

演算項目 1 **記入**	**演算項目 2** **(文字)**	**結果フィールド**	
		値	**DTZ タイプ**
*MDY-	11-19-75	75/323	D(*JUL)
*JUL	92/114	23/04/92	D(*DMY)
*YMD	14/01/28	01/28/2014	D(*USA)
*YMD0	140128	01/28/2014	D(*USA)
*USA	12/31/9999	31.12.9999	D(*EUR)
*ISO	2036-05-21	21/05/36	D(*DMY)

表 *125.* 日付時刻フィールドへの文字フィールドの転送. 演算項目 1 は演算項目 2 の指定の形式を示します。 *(続き)*

演算項目 1 **記入**	**演算項目 2** **(文字)**	**結果フィールド**	
		値	**DTZ タイプ**
*JUL	45/333	11/29/1945	D(*USA)
*MDY/	03/05/33	03.05.33	D(*MDY.)
*CYMD&	121 07 08	08.07.2021	D(*EUR)
*CYMD0	1210708	07,08,21	D(*MDY,)
*CMDY.	107.08.21	21-07-08	D(*YMD-)
*CDMY0	1080721	07/08/2021	D(*USA)
*LONGJUL-	2021-189	08/07/2021	D(*EUR)
*HMS&	23 12 56	23.12.56	T(*ISO)
*USA	1:00 PM	13.00.00	T(*EUR)
*EUR	11.10.07	11:10:07	T(*JIS)
*JIS	14:16:18	14.16.18	T(*HMS.)
*ISO	24.00.00	12:00 AM	T(*USA)
ブランク	1991-09-14-13.12.56.123456	1991-09-14-13.12.56.123456	Z(*ISO)
*ISO	1991-09-14-13.12.56.123456	1991-09-14-13.12.56.123456	Z(*ISO)

表 *126.* 日付時刻フィールドへの数値フィールドの転送. 演算項目 1 は演算項目 2 の指定の形式を示します。

演算項目 1 **Entry**[1]	**演算項目 2** **(数値)**	**結果フィールド**	
		値	**DTZ タイプ**
*MDY	111975	75/323	D(*JUL)
*JUL	92114	23/04/92	D(*DMY)
*YMD	140128	01/28/2014	D(*USA)
*USA[2]	12319999	31.12.9999	D(*EUR)
*ISO	20360521	21/05/36	D(*DMY)
*JUL	45333	11/29/1945	D(*USA)
*MDY	030533	03.05.33	D(*MDY.)
*CYMD	1210708	08.07.2021	D(*EUR)
*CMDY	1070821	21-07-08	D(*YMD-)
*CDMY	1080721	07/08/2021	D(*USA)
*LONGJUL	2021189	08/07/2021	D(*EUR)
*USA	*DATE (092195)[3]	1995-09-21	D(*JIS)
ブランク	*DATE (092195)[3]	1995-09-21	D(*JIS)
*MDY	UDATE (092195)[3]	21.09.1995	D(*EUR)
*HMS	231256	23.12.56	T(*ISO)

表 *126.* 日付時刻フィールドへの数値フィールドの転送. 演算項目 1 は演算項目 2 の指定の形式を示します。 *(続き)*

演算項目 1 Entry[1]	演算項目 2 (数値)	結果フィールド	
		値	DTZ タイプ
*EUR	111007	11:10:07	T(*JIS)
*JIS	141618	14.16.18	T(*HMS.)
*ISO	240000	12:00 AM	T(*USA)
Blank[4]	19910914131256123456	1991-09-14-13.12.56.123456	Z(*ISO)

注 :

1. 区切り記号のゼロ (0) は、演算項目 1 で、日付、時刻、または タイム・スタンプ・フィールドと数値クラスの間の転送には使用することはできません。
2. 時刻の形式 *USA は、時刻と数値クラスの間の転送には使用することがで きません。
3. *DATE および UDATE の場合には、ジョブ記述のジョブ日付は *MDY 形式で 092195 が入っていると見なされます。 演算項目 1 は、オプションであり、正しい形式にデフォルト指定されます。 演算項目 2 が *DATE で、演算項目 1 がコーディングされている場合、演算項目 1 は 4 桁の年の日付形式でなければなりません。 演算項目 2 が UDATE で、演算項目 1 がコーディングされている場合、 演算項目 1 は 2 桁の年の日付形式でなければなりません。
4. タイム・スタンプ・フィールドの転送の場合、演算項目 1 はオプションです。 演算項目 1 は、コーディングされる場合、*ISO または *ISO0 でなければなり ません。

表 *127.* 文字フィールドへの日付/時刻フィールドの転送

演算項目 1 記入	演算項目 2		結果フィールド (文字)
	値	DTZ タイプ	
*JUL	11-19-75	D(*MDY-)	75/323
*DMY-	92/114	D(*JUL)	23-04-92
*USA	14/01/28	D(*YMD)	01/28/2014
*EUR	12/31/9999	D(*USA)	31.12.9999
*DMY,	2036-05-21	D(*ISO)	21,05,36
*USA	45/333	D(*JUL)	11/29/1945
*USA0	45/333	D(*JUL)	11291945
*MDY&	03/05/33	D(*MDY)	03 05 33
*CYMD,	03 07 08	D(*MDY&);	108,03,07
*CYMD0	21/07/08	D(*DMY)	1080721
*CMDY	21-07-08	D(*YMD-)	107/08/21
*CDMY-	07/08/2021	D(*USA)	108-07-21
*LONGJUL&	08/07/2021	D(*EUR)	2021 189
*ISO	23 12 56	T(*HMS&);	23.12.56
*EUR	11:00 AM	T(*USA)	11.00.00
*JIS	11.10.07	T(*EUR)	11:10:07

表 *127.* 文字フィールドへの日付/時刻フィールドの転送 (続き)

演算項目 1 記入	演算項目 2		結果フィールド (文字)
	値	DTZ タイプ	
*HMS,	14:16:18	T(*JIS)	14,16,18
*USA	24.00.00	T(*ISO)	12:00 AM
ブランク	2045-10-27-23.34.59.123456	Z(*ISO)	2045-10-27-23.34.59.123456

表 *128.* 数値フィールドへの日付時刻フィールドの転送

演算項目 1 記入	演算項目 2		結果フィールド (数値)
	値	DTZ タイプ	
*JUL	11-19-75	D(*MDY-)	75323
*DMY-	92/114	D(*JUL)	230492
*USA	14/01/28	D(*YMD)	01282014
*EUR	12/31/9999	D(*USA)	31129999
*DMY,	2036-05-21	D(*ISO)	210536
*USA	45/333	D(*JUL)	11291945
*MDY&	03/05/33	D(*MDY)	030533
*CYMD,	03 07 08	D(*MDY&);	1080307
*CMDY	21-07-08	D(*YMD-)	1070821
*CDMY-	07/08/2021	D(*USA)	1080721
*LONGJUL&	08/07/2021	D(*EUR)	2021189
*ISO	23 12 56	T(*HMS&);	231256
*EUR	11:00 AM	T(*USA)	110000
*JIS	11.10.07	T(*EUR)	111007
*HMS,	14:16:18	T(*JIS)	141618
*ISO	2045-10-27-23.34.59.123456	Z(*ISO)	20451027233459123456

表 *129.* 日付時刻フィールドへの日付時刻フィールドの転送. タイム・スタンプの初期値が `1985-12-03-14.23.34.123456` で あるとします。

演算項目 1	演算項目 2		結果フィールド	
	値	DTZ タイプ	値	DTZ タイプ
N/A	1986-06-24	D(*ISO)	86/06/24	D(*YMD)
N/A	23 07 12	D(*DMY&);	23.07.2012	D(*EUR)
N/A	11:53 PM	T(USA)	23.53.00	T(*EUR)
N/A	19.59.59	T(*HMS.)	19:59:59	T(*JIS)
N/A	1985-12-03-14.23.34.123456	Z(*ISO.)	1985-12-03-14.23.34.123456	Z(*ISO)

表 *129.* 日付時刻フィールドへの日付時刻フィールドの転送. タイム・スタンプの初期値が 1985-12-03-14.23.34.123456 で あるとします。 *(続き)*

演算項目 1	演算項目 2		結果フィールド	
	値	DTZ タイプ	値	DTZ タイプ
N/A	75.06.30	D(*YMD.)	1975-06-30-14.23.34.123456	Z(*ISO)
N/A	09/23/2234	D(*USA)	2234-09-23-14.23.34.123456	Z(*ISO)
N/A	18,45,59	T(*HMS,)	1985-12-03-18.45.59.000000	Z(*ISO)
N/A	2:00 PM	T(*USA)	1985-12-03-14.00.00.000000	Z(*ISO)
N/A	1985-12-03-14.23.34.123456	Z(*ISO.)	12/03/85	D(*MDY)
N/A	1985-12-03-14.23.34.123456	Z(*ISO.)	12/03/1985	D(*USA)
N/A	1985-12-03-14.23.34.123456	Z(*ISO.)	14:23:34	T(*HMS)
N/A	1985-12-03-14.23.34.123456	Z(*ISO.)	02:23 PM	T(*USA)

表 *130.* 文字フィールドへの日付フィールドの転送. 結果のフィールドが演算項目 2 より大きい。 演算項目 1 に *ISO が入っていて、結果のフィールドが次のように定義されているとします。

```
D     Result_Fld        20A   INZ('ABCDEFGHIJabcdefghij')
```

命令コード	演算項目 2		結果フィールドの値 (移動命令の後の)
	値	DTZ タイプ	
MOVE	11 19 75	D(*MDY&);	'ABCDEFGHIJ1975-11-19'
MOVE(P)	11 19 75	D(*MDY&);	'1975-11-19 '
MOVEL	11 19 75	D(*MDY&);	'1975-11-19abcdefghij'
MOVEL(P)	11 19 75	D(MDY&);	'1975-11-19 '

表 *131.* 数値フィールドへの時刻フィールドの転送. 結果のフィールドが演算項目 2 より大きい。 演算項目 1 に *ISO が入っていて、結果のフィールドが次のように定義されているとします。

```
D     Result_Fld        20S   INZ(11111111111111111111)
```

命令コード	演算項目 2		結果フィールドの値 (移動命令の後の)
	値	DTZ タイプ	
MOVE	9:42 PM	T(*USA)	11111111111111214200
MOVE(P)	9:42 PM	T(*USA)	00000000000000214200
MOVEL	9:42 PM	T(*USA)	21420011111111111111
MOVEL(P)	9:42 PM	T(*USA)	21420000000000000000

表 *132.* 時刻フィールドへの数値フィールドの転送. 演算項目 2 が結果のフィールドより大きい。 強調表示されている部分は、演算項目 2 フィールドの転送される部分を示しま す。

命令コード	演算項目 2	結果フィールド	
		DTZ タイプ	値
MOVE	11:**12:13:14**	T(*EUR)	12.13.14

表 *132.* 時刻フィールドへの数値フィールドの転送. 演算項目 2 が結果のフィールドより大きい。 強調表示されている部分は、演算項目 2 フィールドの転送される部分を示しま す。 *(続き)*

命令コード	演算項目 2	結果フィールド	
		DTZ タイプ	値
MOVEL	**11:12:13**:14	T(*EUR)	11.12.13

表 *133.* タイム・スタンプ・フィールドへの数値フィールドの転送. 演算項目 2 が結果のフィールドより大きい。 強調表示されている部分は、演算項目 2 フィールドの転送される部分を示しま す。

命令コード	演算項目 2	結果フィールド	
		DTZ タイプ	値
MOVE	123406**18230323123420123456**	Z(*ISO)	1823-03-23-12.34.20.123456
MOVEL	**12340618230323123420**123456	Z(*ISO)	1234-06-18-23.03.23.123420

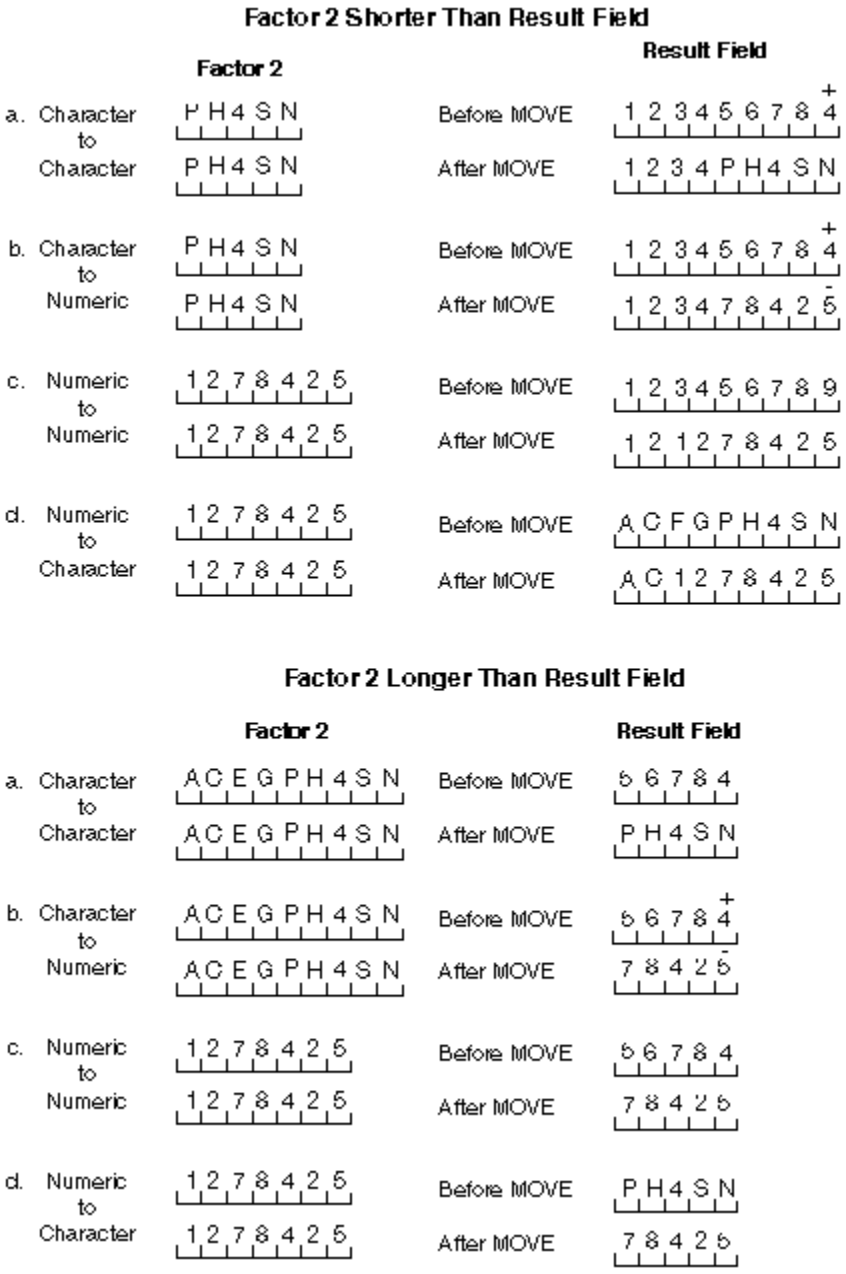
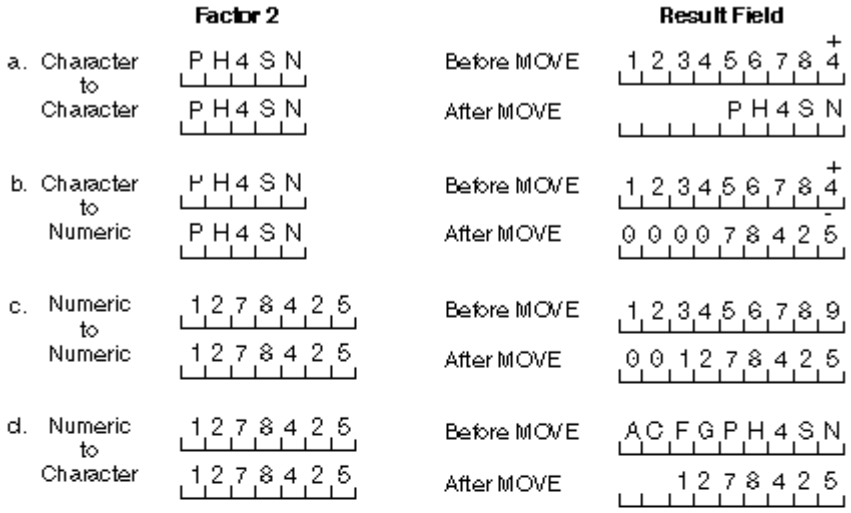
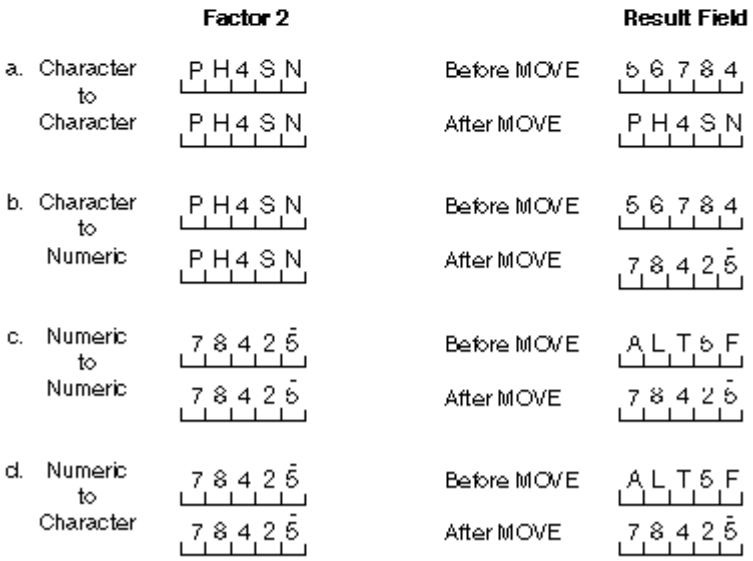

Factor 2 Shorter Than Result Field
Factor 2
Result Field
a. Character to Character
P H 4 S N
Before MOVE
1 2 3 4 5 6 7 8 4 +
P H 4 S N
After MOVE
1 2 3 4 P H 4 S N
b. Character to Numeric
P H 4 S N
Before MOVE
1 2 3 4 5 6 7 8 4 +
P H 4 S N
After MOVE
1 2 3 4 7 8 4 2 5 -
c. Numeric to Numeric
1 2 7 8 4 2 5
Before MOVE
1 2 3 4 5 6 7 8 9
1 2 7 8 4 2 5
After MOVE
1 2 1 2 7 8 4 2 5
d. Numeric to Character
1 2 7 8 4 2 5
Before MOVE
A C F G P H 4 S N
1 2 7 8 4 2 5
After MOVE
A C 1 2 7 8 4 2 5
Factor 2 Longer Than Result Field
Factor 2
Result Field
a. Character to Character
A C E G P H 4 S N
Before MOVE
5 6 7 8 4
A C E G P H 4 S N
After MOVE
P H 4 S N
b. Character to Numeric
A C E G P H 4 S N
Before MOVE
5 6 7 8 4 +
A C E G P H 4 S N
After MOVE
7 8 4 2 5 -
c. Numeric to Numeric
1 2 7 8 4 2 5
Before MOVE
5 6 7 8 4
1 2 7 8 4 2 5
After MOVE
7 8 4 2 5
d. Numeric to Character
1 2 7 8 4 2 5
Before MOVE
P H 4 S N
1 2 7 8 4 2 5
After MOVE
7 8 4 2 5

図 *351. MOVE* 命令

Factor 2 Shorter Than Result Field With P in Operation Extender Field



Factor 2 and Result Field Same Length



Note: 4+ = letter D, and 5- = letter N.

MOVEA (配列の転送)

自由形式構文	(許可されていない - %SUBARR または 1 つ以上のストリング命令を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
MOVEA (P)		ソース	ターゲット	+	-	ZB

MOVEA 命令では、演算項目 2 から結果のフィールドに文字、図形、UCS-2、または数値が転送されます。(数値を転送する場合には、いくつかの制約事項が適用されま す。) 演算項目 2 または結果のフィールドには配列が入っていなければなりません。 配列が指標付きの場合であっても、演算項目 2 と結果のフィールドに同じ配列 を指定することはできません。 次のことができます。

- 連続するいくつかの配列要素を 1 つのフィールドへ転送する。

- 1 つのフィールドを連続するいくつかの配列要素へ転送する。
- 連続する配列要素を別の配列の連続する要素へ転送する。

データの転送は、配列が指標付きでない場合には配列の最初の要素から、ま た配列が指標付きの場合には指定された要素から開始されます。 データの転送は、配列の最後の要素が転送されるか、または埋められた時に 終了します。 結果のフィールドに標識配列が入る場合には、MOVEA 命令によって影響を受け るすべての標識が相互参照表に示されます。

MOVEA 命令のコーディングとその結果を 819 ページの図 352 に示します。

詳しくは、561 ページの『配列命令』、583 ページの『移動命令』、または 572 ページの『日付命令』 を参照してください。

文字、図形、および UCS-2 の MOVEA 命令

演算項目 2 と結果のフィールドは両方とも同じタイプ (すなわち、 文字、図形、または UCS-2 のいずれか) でなければなりません。 図形または UCS-2 の CCSID は、それらの CCSID の 1 つが 65535 で ない限り、または、図形フィールドの場合には、CCSID(*GRAPH: *IGNORE) が 制御仕様書で指定されていない限り、同じでなければなりません。

文字、図形、または UCS-2 の MOVEA 命令では、データの転送は、転送された文字数が演算項目 2 と結果のフィールドで指定されたフィールドの短い方の長さに等しくなったときに終了します。 したがって、MOVEA 命令は配列要素の中間で終了することがあります。 可変長配列は使用できません。

数字の MOVEA 命令

転送が有効となるのは、同じ数字の長さが定義されているフィールドと配列 要素の間だけです。 演算項目 2 と結果のフィールドには、数値フィールド、数値配列要素、または 数値配列を指定することができます。少なくとも 1 つは配列か配列要素でなけ ればなりません。 数値タイプは、2 進数、パック 10 進数、またはゾーン 10 進数とすることが できますが、演算項目 2 と結果のフィールドの間で同じにする必要はありません。

結果のフィールドに数値配列または数値配列要素を指定した場合には、演算項目 2 に数値リテラルを入れることができます。

- この数値リテラルに小数点を入れることはできません。
- この数値リテラルの長さを結果のフィールドに指定された配列または配列 要素の要素の長さより大きくすることはできません。

小数点以下の桁数は転送中に無視されるので、対応する必要はありません。 数値は、定義された小数点以下の桁数の差が明らかになるようには変換されま せん。

数値配列で MOVEA 命令の演算項目 2 に形象定数の *BLANK、*ALL、*ON、お よび *OFF を使用することはできません。

一般的な MOVEA 命令

アプリケーション・プログラムに MOVEA 命令を使用する必要があって、数値の MOVEA 命令に対する制約事項のために使用できない場合には、 文字の MOVEA 命令を使用できる場合があります。 数値配列がゾーン 10 進数形式の場合には次のようにしてください。

- 数値配列をデータ構造のサブフィールドとして定義します。
- データ構造の中の数値配列を文字配列として再定義します。

MOVEA で形象定数を指定した場合には、生成される定数の長さは、指定 した配列の部分に等しくなります。 数値配列の形象定数の場合には、それぞれの配列要素に入れられる符号を除い て、要素の境界は無視されます。 例は次のようになります。

- MOVEA *BLANK ARR(X)

  要素 X から始まり、ARR の残りの部分にはブランクが入ります。
- MOVEA *ALL‘XYZ’ ARR(X)

ARR には 4 バイトの文字要素があります。 文字の MOVEA の場合には常に要素の境界は無視されます。 要素 X から始まり、配列の残りには「XYZXYZXYZXYZ. . .」が入ります。

文字、図形、UCS-2、および数値の MOVEA 命令の場合には、命令拡張 P を指定して、結果に右から埋め込みを行うことができます。

MOVEA 命令について詳しくは、583 ページの『移動命令』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA     ARRX          ARRY
 *  Array-to-array move.  No indexing; different length array,
 *  same element length.
```

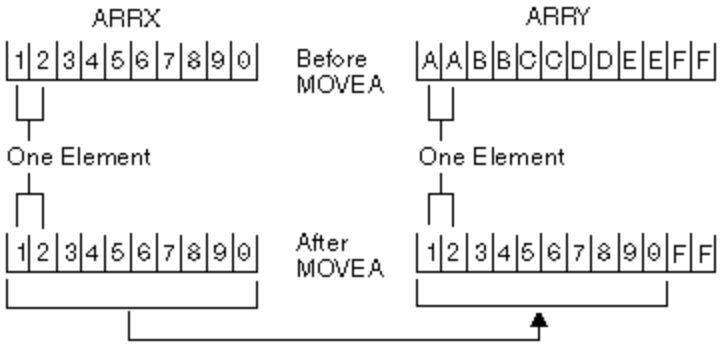
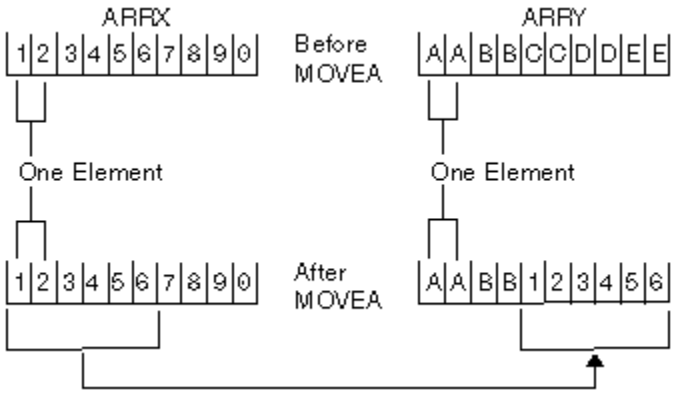
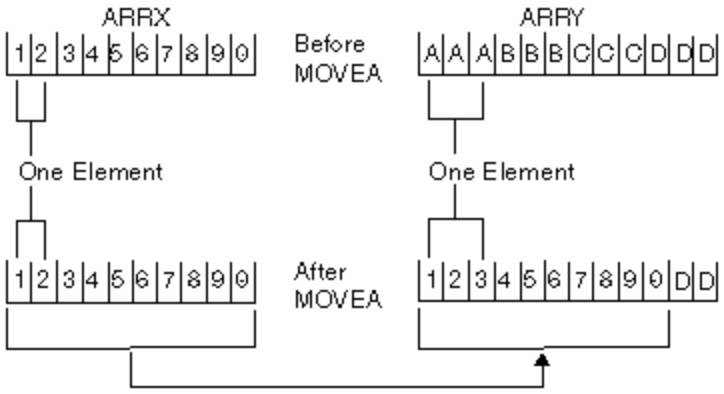


図 *352. MOVEA* 命令

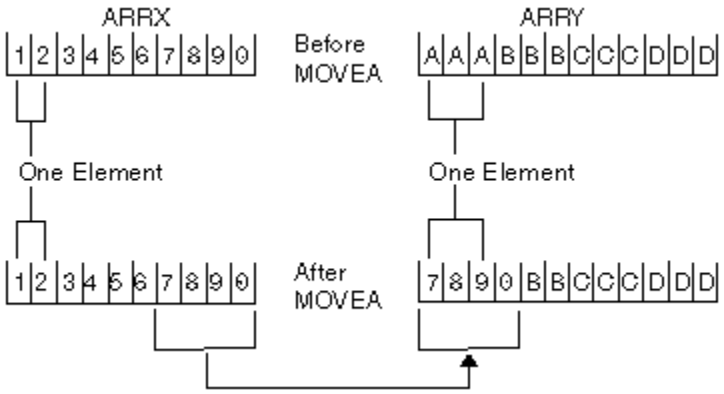
```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA     ARRX          ARRY(3)
 *  Array-to-array move with index result field.
```



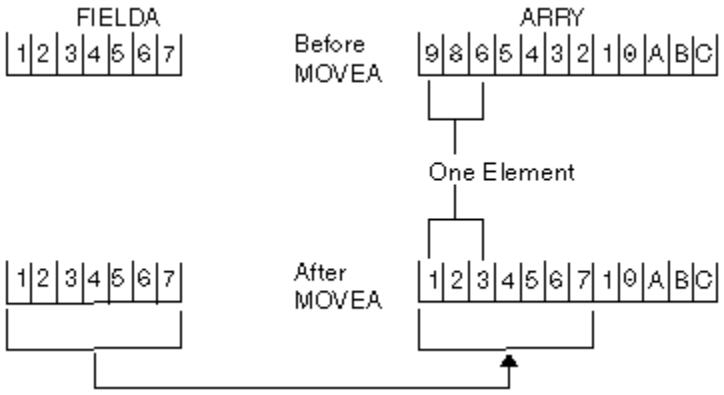
```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA     ARRX          ARRY
 *  Array-to-array move, no indexing and different length array
 *  elements.
```



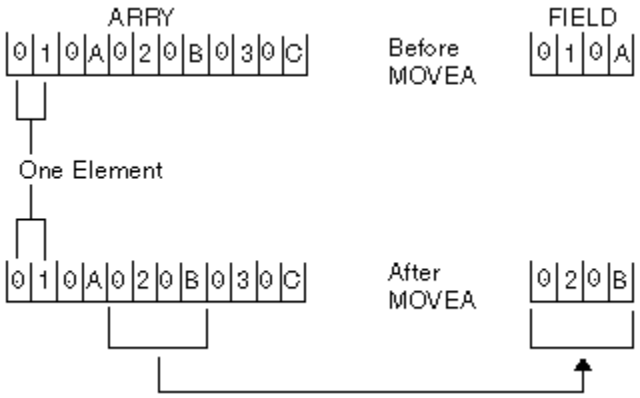

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA     ARRX(4)       ARRY
 *  Array-to-array move, index factor 2 with different length array
 *  elements.
```



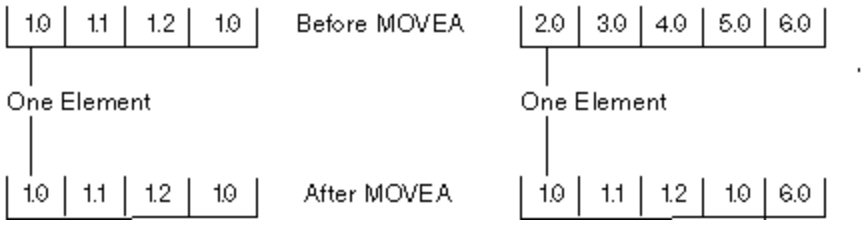
```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA     FIELDA        ARRY
 *  Field-to-array move, no indexing on array.
```



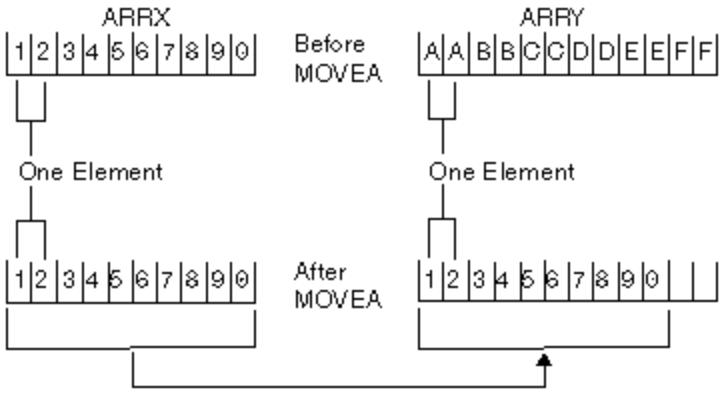

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 * In the following example, N=3.  Array-to-field move with variable
 * indexing.
C                   MOVEA     ARRX(N)       FIELD
 *
```



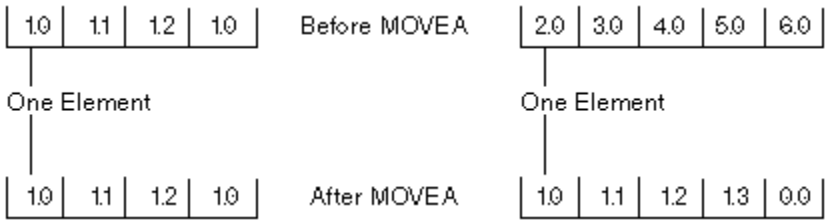
```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA     ARRB          ARRZ
 *
 * An array-to-array move showing numeric elements.
```



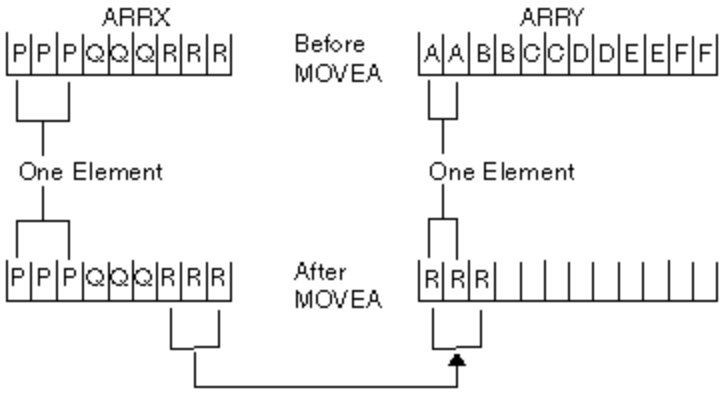
```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA(P)  ARRX          ARRY
 *  Array-to-array move with padding.  No indexing; different length
 *  array with same element length.
```




```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA(P)  ARRB          ARRZ
 *
 * An array-to-array move showing numeric elements with padding.
```



```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   MOVEA(P)  ARRX(3)       ARRY
 *  Array-to-array move with padding.  No indexing; different length
 *  array with different element length.
```



MOVEL (左につめて転送)

自由形式構文	許可されていない - EVAL、または%CHAR、%DATE、%DEC、%DECH、%GRAPH、%INT、%INTH、%TIME、%TIMESTAMP、%UCS2、%UNS、または %UNSH などの組み込み関数を使用

コード	演算項目 1	演算項目 2	結果フィールド	標識		
MOVEL(P)	データ属性	ソース・フィールド	ターゲット・フィールド	+	-	ZB

MOVEL 命令では、演算項目 2 から結果フィールドに文字が転送されます。 転送は演算項目 2 の左端の文字から開始されます。 結果フィールドが配列の場合には、結果の標識を指定することはできません。 この標識は、結果フィールドが配列要素または配列以外のフィールドの場合に は指定することができます。

データが数値フィールドに転送される場合には、結果フィールドの符号 (+ または -) が保持されます。ただし、演算項目 2 が結果フィールドと同じ長 さか、またはそれ以上の場合を除きます。 この場合には、演算項目 2 の符号は 結果フィールドの符号として使用されます。

演算項目 1 には、この命令の転送元または転送先の文字または数値フィールド の形式を指定するために、日付または時刻の形式を入れることができます。使用可能な形式に ついては、273 ページの『日付データ・タイプ』、275 ページの『時刻データ・タイプ』、および 277 ページの『タイム・スタンプ・データ・タイプ』 を参照してください。

転送元または転送先が文字フィールドである場合、オプションで、 演算項目 1 内の形式の後に区切り記号を示すことができます。 ただし、その形式で有効な区切り記号だけを使用する ことができます。

演算項目 2 が *DATE または UDATE で、結果が日付フィールドである 場合、演算項目 1 は必要ありません。演算項目 1 に日付形式が含まれている場合、演算項目 1 は、制御仕様書 の DATEDIT キーワードによって指定されている *DATE または UDATE の 形式と互換性を持っている必要があります。

演算項目 2 が結果フィールドより長い場合には、演算項目 2 の余分な右端の文 字は転送されません。 結果フィールドが演算項目 2 より長い場合には、埋め込み が指定されていない限り、結果フィールドの右端の余分な文字は変更されませ ん。

浮動数値フィールドおよびリテラルを、演算項目 2 または結果フィールドの 指定として使用することはできません。

演算項目 2 が UCS-2 で、結果フィールドが文字の場合、または 演算項目 2 が文字で結果フィールドが UCS-2 の場合、転送される 文字数は、文字にシフト文字と図形文字が含まれる場合と含まれない 場合があるために変化します。 例えば、5 つの UCS-2 文字は、次の文字に変換できます。

- 5 個の 1 バイト文字
- 5 個の 2 バイト文字
- モードを分離するシフト文字を含む、1 バイト文字と 2 バイト文字の 組み合わせ

結果のデータが長過ぎて、結果フィールドに入りきらない場合、そのデータは 切り捨てられます。 結果が 1 バイト文字である場合、その結果に完全な文字が含まれている こと、および突き合わせられた SO/SI のペアが含まれていることは、ユーザーが確認 してください。

MOVEL 命令は、827 ページの図 353 でまとめられています。

MOVEL 命令の、フィールド長に基づいた 4 つの条件での規則の要約を下に 示します。

1. 演算項目 2 が結果フィールドと同じ長さの場合。
   a. 演算項目 2 と結果フィールドが数値の場合には、符号は右端の桁に転送さ れます。
   b. 演算項目 2 が数値で結果フィールドが文字の場合には、符号は右端の桁に 転送されます。
   c. 演算項目 2 が文字で結果フィールドが数値の場合には、演算項目 2 の右端 の桁のゾーンが 16 進数の D (マイナス・ゾーン) であれば、マイナス・ゾー ンが結果フィールドの右端の桁に転送されます。 しかし、演算項目 2 の右端の桁のゾーンが 16 進数の D でない場合には、正の ゾーンが結果フィールドの右端の桁に転送されます。 数字部分は対応する数字 に変換されます。 数字部分が有効な数字でない場合には、データ例外エラーが起こります。
   d. 演算項目 2 と結果フィールドが文字の場合には、すべての文字が転送され ます。
   e. 演算項目 2 と結果フィールドが、両方とも図形また は UCS-2 の場合には、すべての図形文字または UCS-2 文字が転送されます。
   f. 演算項目 2 が図形で結果フィールドが文字の場合には、文字の結果フィー ルドの 2 桁 (バイト) がコンパイラーによって挿入される SO/SI を入れるた めに使用されるので、1 つの図形文字が失われます。
   g. 演算項目 2 が文字で結果フィールドが図形の場合には、演算項目 2 の文字 データは一対の SO/SI で完全に囲まれていなければなりません。 この SO/SI は、図形の結果フィールドにデータを転送する前にコンパイラーによって除去 されます。
2. 演算項目 2 が結果フィールドより長い場合。
   a. 演算項目 2 と結果フィールドが数値の場合には、演算項目 2 の右端の桁の 符号が結果フィールドの右端の桁に転送されます。
   b. 演算項目 2 が数値で結果フィールドが文字の場合には、結果フィールドに は数字だけが入ります。
   c. 演算項目 2 が文字で結果フィールドが数値の場合には、演算項目 2 の右端 の桁のゾーンが 16 進数の D (マイナス・ゾーン) であれば、マイナス・ゾー ンが結果フィールドの右端の桁に転送されます。 しかし、演算項目 2 の右端の桁のゾーンが 16 進数の D でない場合には、正の ゾーンが結果フィールドの右端の桁に転送されます。 結果フィールドのその他 の桁には数字だけが入ります。
   d. 演算項目 2 と結果フィールドが文字の場合には、結果フィールドを埋める ために必要な数の文字だけが転送されます。
   e. 演算項目 2 と結果フィールドが図形または UCS-2 の 場合、結果フィールドを埋めるために必要な数の図形文字 または UCS-2 文字だけが転送されます。
   f. 演算項目 2 が図形で結果フィールドが文字の場合には、図形データが切り 捨てられて、SO/SI がコンパイラーによって挿入されます。
   g. 演算項目 2 が文字で結果が図形の場合には、文字データは切り捨てられま す。 文字データは一対の SO/SI で完全に囲まれていなければなりません。
3. 演算項目 2 が結果フィールドより短い場合。
   a. 演算項目 2 が数値または文字のいずれかで結果フィールドが数値の場合に は、演算項目 2 の数字部分が結果フィールドの左端の桁の内容と置き換えられ ます。 結果フィールドの右端の桁の符号は変更されません。

b. 演算項目 2 が数値または文字のいずれかで結果フィールドが文字データの 場合には、演算項目 2 の文字が結果フィールドの左端の同じ桁数と置き換えら れます。 結果フィールドの右端の桁のゾーンは変更されません。

c. 演算項目 2 が図形で結果フィールドが文字の場合には、図形データの直前 直後に SO/SI が追加されます。 これによって、このフィールドの残りのデータ のために、文字フィールドの SO/SI の数が合わなくなる場合がありますが、こ れはユーザーの責任になります。

d. 文字フィールドから図形フィールドに転送する場合には、文字フィールド 全体を SO/SI で囲まなければならないことに注意してください。 例えば、文字フィールドの長さが 8 桁の場合には、フィールドの文字データ は "oAABBbbi" になります。"oAABBibb" にはなりません。

4. 演算項目 2 が結果フィールドより短く、命令拡張フィールドに P が指定さ れている場合。

a. 転送は上記のように実行されます。

b. 結果フィールドには右から埋め込みが行われます。 埋め込み規則について詳しくは、583 ページの『移動命令』を参照してください。

可変長文字、図形、または UCS-2 データを転送する場合、可変長フィールドは、現在の長さが同じ固定長フィールドとまったく同様に機能します。 MOVEL 命令では、可変長結果フィールドの長さは変わりません。 例については、831 ページの図 356 から 833 ページの図 361 を参照してください。

MOVEL 命令について詳しくは、583 ページの『移動命令』、572 ページの『日付命令』、または 569 ページの『変換命令』を参照してください。

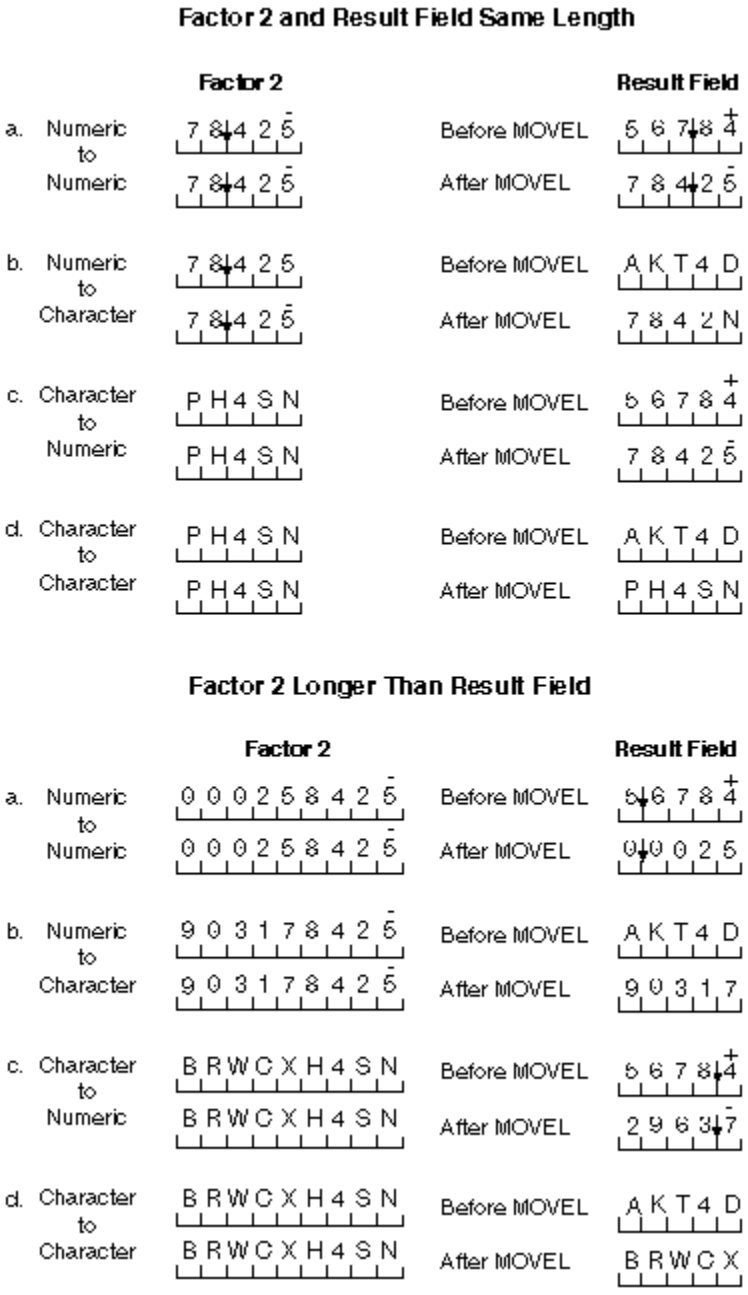
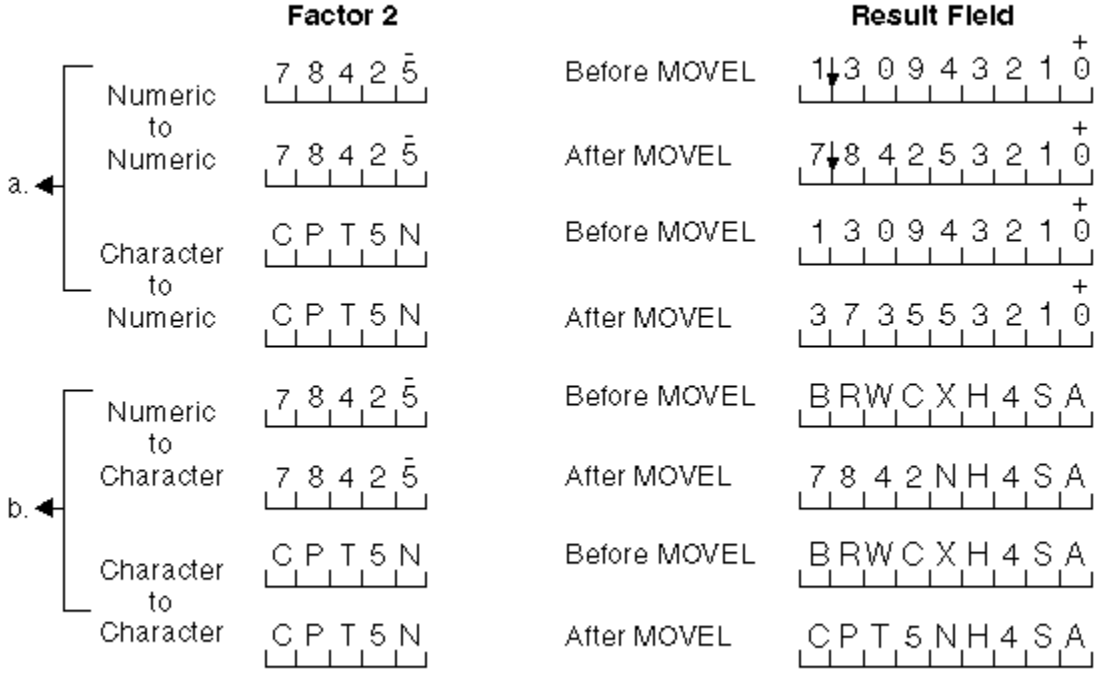


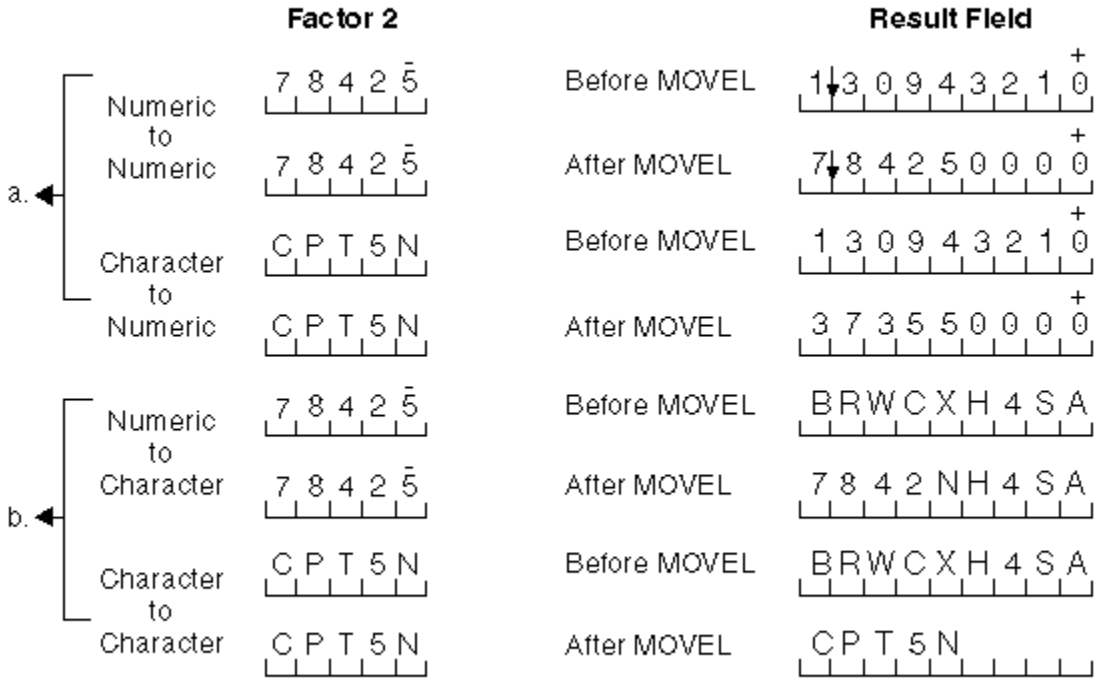
図 *353. MOVEL* 命令

Factor 2 Shorter Than Result Field



Note: + 4 = letter D, and − 5 = letter N; arrow ↓ is decimal point.

Factor 2 Shorter Than Result Field With P in Operation Extender Field



Note: + 4 = letter D, and − 5 = letter N; arrow ↓ is decimal point.

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++++++
D
 *
 * Example of MOVEL between graphic and character fields
 *
D char_fld1      S              8A   inz(' ')
D dbcs_fld1      S              4G   inz('oAABBCCDDi')
D char_fld2      S              4A   inz(' ')
D dbcs_fld2      S              3G   inz(G'oAABBCCi')
D char_fld3      S             10A   inz(*ALL'X')
D dbcs_fld3      S              3G   inz(G'oAABBCCi')
D char_fld4      S             10A   inz('oAABBCC  i')
D dbcs_fld4      S              2G
 *
 *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
 *
 *  The result field length is equal to the factor 2 length in bytes.
 *  One DBCS character is lost due to insertion of SO/SI.
 *  Value of char_fld1 after MOVEL operation is 'oAABBCCi'
 *
C                   MOVEL     dbcs_fld1     char_fld1
 *
 *  Result field length shorter than factor 2 length. Truncation occurs.
 *  Value of char_fld2 after MOVEL operation is 'oAAi'
 *
C                   MOVEL     dbcs_fld2     char_fld2
 *
 *  Result field length longer than factor 2 length. Example shows
 *  SO/SI are added immediately before and after graphic data.
 *  Before the MOVEL, Result Field contains 'XXXXXXXXXX'
 *  Value of char_fld3 after MOVEL operation is 'oAABBCCiXX'
 *
C                   MOVEL     dbcs_fld3     char_fld3
 *
 *  Character to Graphic MOVEL
 *  Result Field shorter than Factor 2. Truncation occurs.
 *  Value of dbcs_fld4 after MOVEL operation is 'AABB'
 *
C                   MOVEL     char_fld4     dbcs_fld4
```

図 *354.* 文字フィールドと図形フィールドの間の *MOVEL*

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
HKeywords++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
 *
 * Example of MOVEL between character and date fields
 *
 *  Control specification date format
H DATFMT(*MDY)
 *
DName+++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++++++
D datefld        S               D   INZ(D'04/15/96')
D char_fld1      S              8A
D char_fld2      S             10A   INZ('XXXXXXXXXX')
D char_fld3      S             10A   INZ('04/15/96XX')
D date_fld3      S               D
D char_fld4      S             10A   INZ('XXXXXXXXXX')
D char_fld5      S              9A   INZ('015/04/50')
D date_fld2      S               D   INZ(D'11/16/10')
 *
 *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+H1LoEq..
 * Date to Character MOVEL
 * The result field length is equal to the factor 2 length.  Value of
 * char_fld1 after the MOVEL operation is '04/15/96'.
C     *MDY          MOVEL     datefld       char_fld1
 * Date to Character MOVEL
 * The result field length is longer than the factor 2 length.
 * Before MOVEL, result field contains 'XXXXXXXXXX'
 * Value of char_fld2 after the MOVEL operation is '04/15/96XX'.
C     *MDY          MOVEL     datefld       char_fld2
 * Character to Date MOVEL
 * The result field length is shorter than the factor 2 length.
 * Value of date_fld3 after the MOVEL operation is '04/15/96'.
C     *MDY          MOVEL     char_fld3     date_fld3
 * Date to Character MOVEL (no separators)
 * The result field length is longer than the factor 2 length.
 * Before MOVEL, result field contains 'XXXXXXXXXX'
 * Value of char_fld4 after the MOVEL operation is '041596XXXX'.
C     *MDY0         MOVEL     datefld       char_fld4
 *  Character to date MOVEL
 *  The result field length is equal to the factor 2 length.
 *  The value of date_fld3 after the move is 04/15/50.
C     *CDMY         MOVEL     char_fld5     date_fld3
 *  Date to character MOVEL (no separators)
 *  The result field length is longer than the factor 2 length.
 *  The value of char_fld4 after the move is '2010320XXX'.
C     *LONGJUL0     MOVEL     date_fld2     char_fld4
```

図 *355.* 文字フィールドと日付フィールドの間の *MOVEL*

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName+++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
 *
 * Example of MOVEL from variable to variable length
 * for character fields
 *
D var5a          S              5A   INZ('ABCDE') VARYING
D var5b          S              5A   INZ('ABCDE') VARYING
D var5c          S              5A   INZ('ABCDE') VARYING
D var10          S             10A   INZ('0123456789') VARYING
D var15a         S             15A   INZ('FGH') VARYING
D var15b         S             15A   INZ('FGH') VARYING
 *
 *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
 *
C                   MOVEL     var15a        var5a
 * var5a = 'FGHDE' (length=5)
C                   MOVEL     var10         var5b
 * var5b = '01234' (length=5)
C                   MOVEL     var5c         var15a
 * var15a = 'ABC' (length=3)
C                   MOVEL     var10         var15b
 * var15b = '012' (length=3)
```

図 *356.* 可変長フィールドから 可変長フィールドへの *MOVEL*

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName+++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
 *
 * Example of MOVEL from variable to fixed length
 * for character fields
 *
D var5           S              5A   INZ('ABCDE') VARYING
D var10          S             10A   INZ('0123456789') VARYING
D var15          S             15A   INZ('FGH') VARYING
D fix5a          S              5A   INZ('MNOPQ')
D fix5b          S              5A   INZ('MNOPQ')
D fix5c          S              5A   INZ('MNOPQ')
D fix10          S             10A   INZ('')
 *
 *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
 *
C                   MOVEL     var5          fix5a
 * fix5a = 'ABCDE'
C                   MOVEL     var10         fix5b
 * fix5b = '01234'
C                   MOVEL     var15         fix5c
 * fix5c = 'FGHPQ'
```

図 *357.* 可変長フィールドから 固定長フィールドへの *MOVEL*

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
 *
 * Example of MOVEL from fixed to variable length
 * for character fields
 *
D var5           S              5A   INZ('ABCDE') VARYING
D var10          S             10A   INZ('0123456789') VARYING
D var15a         S             15A   INZ('FGHIJKLMNOPQR') VARYING
D var15b         S             15A   INZ('WXYZ') VARYING
D fix10          S             10A   INZ('PQRSTUVWXY')
 *
 *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
 *
C                   MOVEL     fix10         var5
 * var5 = 'PQRST' (length=5)
C                   MOVEL     fix10         var10
 * var10 = 'PQRSTUVWXY' (length=10)
C                   MOVEL     fix10         var15a
 * var15a = 'PQRSTUVWXYPQR' (length=13)
C                   MOVEL     fix10         var15b
 * var15b = 'PQRS' (length=4)
```

図 *358.* 固定長フィールドから 可変長フィールドへの *MOVEL*

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
 *
 * Example of MOVEL(P) from variable to variable length
 * for character fields
 *
D var5a          S              5A   INZ('ABCDE') VARYING
D var5b          S              5A   INZ('ABCDE') VARYING
D var5c          S              5A   INZ('ABCDE') VARYING
D var10          S             10A   INZ('0123456789') VARYING
D var15a         S             15A   INZ('FGH') VARYING
D var15b         S             15A   INZ('FGH') VARYING
D var15c         S             15A   INZ('FGHIJKLMN') VARYING
 *
 *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
 *
C                   MOVEL(P)  var15a        var5a
 * var5a = 'FGH  ' (length=5)
C                   MOVEL(P)  var10         var5b
 * var5b = '01234' (length=5)
C                   MOVEL(P)  var5c         var15b
 * var15b = 'ABC' (length=3)
C                   MOVEL(P)  var15a        var15c
 * var15c = 'FGH      ' (length=9)
```

図 *359.* 可変長フィールドから 可変長フィールドへの *MOVEL(P)*

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
 *
 * Example of MOVEL(P) from variable to fixed length
 * for character fields
 *
D var5           S              5A   INZ('ABCDE') VARYING
D var10          S             10A   INZ('0123456789') VARYING
D var15          S             15A   INZ('FGH') VARYING
D fix5a          S              5A   INZ('MNOPQ')
D fix5b          S              5A   INZ('MNOPQ')
D fix5c          S              5A   INZ('MNOPQ')
 *
 *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
 *
C                   MOVEL(P)  var5          fix5a
 * fix5a = 'ABCDE'
C                   MOVEL(P)  var10         fix5b
 * fix5b = '01234'
C                   MOVEL(P)  var15         fix5c
 * fix5c = 'FGH  '
```

図 *360.* 可変長フィールドから 固定長フィールドへの *MOVEL(P)*

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++
 *
 * Example of MOVEL(P) from fixed to variable length
 * for character fields
 *
D var5           S              5A   INZ('ABCDE') VARYING
D var10          S             10A   INZ('0123456789') VARYING
D var15a         S             15A   INZ('FGHIJKLMNOPQR') VARYING
D var15b         S             15A   INZ('FGH') VARYING
D fix5           S             10A   INZ('.....')
D fix10          S             10A   INZ('PQRSTUVWXY')
 *
 *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiL
 *
C                   MOVEL(P)  fix10         var5
 * var5 = 'PQRST' (length=5)
C                   MOVEL(P)  fix5          var10
 * var10 = '.....     ' (length=10)
C                   MOVEL(P)  fix10         var15a
 * var15a = 'PQRSTUVWXY   ' (length=13)
C                   MOVEL(P)  fix10         var15b
 * var15b = 'PQR' (length=3)
```

図 *361.* 固定長フィールドから 可変長フィールドへの *MOVEL(P)*

MULT (乗算)

自由形式構文	(許可されていない - * または *= 演算子を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
MULT (H)	被乗数	乗数	積	+	-	Z

演算項目 1 が指定されている場合には、演算項目 1 に演算項目 2 が乗算され て、その積が結果フィールドに入れられます。 結果フィールドに、積を十分入れられるだけの大きさがあることを確認してく ださい。結果フィールドの最大長を調べるためには次の規則を使用します。すなわち、 結果フィールドの長さは演算項目 1 の長さに演算項目 2 の長さを加えた長さに なります。 演算項目 1 が指定されていない場合には、演算項目 2 に結果フィー ルドが乗算されて、その積が結果フィールドに入れられます。 演算項目 1 および

演算項目 2 は数値でなければならず、それぞれ配列、配列要 素、フィールド、形象定数、リテラル、名前付き定数、サブフィール ド、またはテーブル名のいずれかを入れることができます。 結果フィールドは 数値でなければならず、名前のついた定数またはリテラルにすることはで きません。 結果を丸めるために四捨五入を指定することができます。

MULT 命令について詳しくは、557 ページの『算術演算』を参照してください。

MULT 命令の例については、560 ページの図 181 を参照してください。

MVR (剰余の転送)

自由形式構文	(許可されていない - %REM 組み込み関数を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
MVR			剰余	+	-	Z

MVR 命令は、前の DIV 命令で生じた剰余を結果フィールドに指定された別 のフィールドに転送します。 演算項目 1 と演算項目 2 はブランクでなければなりません。 MVR 命令は DIV 命令の直後になければなりません。 条件付け標識を使用する場 合には、MVR 命令が DIV 命令の直後に処理されることを確認してください。 MVR 命令が DIV 命令の前に処理されると、好ましくない結果になります。 結果フィールドは数値でなければならず、配列、配列要素、サブフィールド、 またはテーブル名のいずれかを入れることができます。

DIV 命令で小数点以下の桁数を持つ演算項目が使用される場合には、 結果フィールドに十分な余地を残しておいてください。 小数点以下の有効桁数は次のうちの大きい方になります。

- 前の除算命令の演算項目 1 の小数点以下の桁数
- 前の除算命令の演算項目 2 と結果フィールドの小数点以下の桁数の和

剰余の符号 (+ または -) は被除数 (演算項目 1) と同じです。

直後に MVR 命令が続く DIV 命令に四捨五入を指定することはできません。

剰余の整数部分の最大桁数は、前の除算命令の演算項目 2 の整数部分の桁数 と同じです。

前の除算命令で結果フィールドに配列を指定している場合には、MVR 命令を 使用することはできません。また、直前の DIV 命令に少なくとも 1 つの浮動オペランドがある場合に は、MVR 命令を使用することはできません。

MVR 命令について詳しくは、557 ページの『算術演算』を参照してください。

MVR 命令の例については、560 ページの図 181 を参照してください。

NEXT (次の入力の取り出し)

自由形式構文	NEXT{(E)} プログラム装置 ファイル名

コード	演算項目 1	演算項目 2	結果フィールド	標識		
NEXT (E)		ファイル名		_	ER	_

NEXT 命令コードは、複数装置ファイルの次の入力をプログラム装置 オペランドに 指定されたプログラム装置から強制的に取り出します。 ただし、その入力命令がサイクル読み取りまたは「ファイル名による READ」の場合に限られます。 読み取り操作 (CHAIN、EXFMT、READ、および READC を含む) によって、前の NEXT 命令の影響は終了します。 入力操作間に NEXT が複数回指定されている場合には、最後の命令だけが処理 されます。 NEXT 命令コードを使用できるのは、複数装置ファイルの場合だけで す。

プログラム装置 オペランドには、プログラム装置名を含む 10 桁のフィールドの名前、またはプログラム装置名である文字リテラルあるいは名前のついた定数を入力 します。 ファイル名 オペランドには、命令を要求する複数装置 ワークステーション・ファイルの名前を入力します。

NEXT 例外 (ファイル状況コードが 1000 より大きい) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、146 ページの『ファイル例外/エラー』を参照してください。

詳しくは、「576 ページの『ファイル操作』」を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C
 * Assume devices Dev1 and Dev2  are connected to the WORKSTN file
 * DEVICEFILE. The first READ reads data from DEV1, the second READ
 * reads data from DEV2. The NEXT operation will direct the program
 * to wait for data from the device specified in factor 1 (i.e. DEV1)
 * for the third READ.
C
C                   READ (E)  Devicefile
C                   :
C                   READ (E)  Devicefile
C                   :
C     'DEV1'        NEXT
C                   :
C                   READ (E)  Devicefile
```

図 *362. NEXT* 命令

OCCUR (データ構造のオカレンスの設定/取り出し)

自由形式構文	(許可されていない - %OCCUR 組み込み関数を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
OCCUR (E)	オカレンス値	データ構造	オカレンス値	_	ER	_

OCCUR 命令コードは、RPG IV プログラムの中で次に使用されるデータ構造のオカレンスを指定します。

OCCUR 命令は、プログラムの中で次に使用される複数オカレンス・データ構造の オカレンスを設定します。 一度に使用できるオカレンスは 1 つだけです。 操作に複数オカレンス・データ構造またはそのデータ構造のサブフィールドが指定 されている場合には、OCCUR 命令が指定されるまで、そのデータ構造の最初の オカレンスが使用されます。 OCCUR 命令が指定されると、その OCCUR 命令によって設定されたデータ構造の オカレンスが使用されます。

演算項目 1 は任意指定です。指定する場合には、数値、小数点以下の桁数が ゼロのリテラル、フィールド名、名前のついた定数、またはデータ構造名 を入れることができます。 演算項目 1 は、OCCUR 命令の実行中に演算項目 2 に指定されたデータ構造のオ カレンスを設定するために使用されます。 演算項目 1 がブランクの場合には、 OCCUR 命令の実行中に演算項目 2 の現在のデータ構造のオカレンスの値が結果 フィールドに入れられます。

演算項目 1 がデータ構造名の場合には、複数オカレンス・データ構造でなければ なりません。 演算項目 1 の現在のデータ構造のオカレンスが、演算項目 2 のデ ータ構造のオカレンスを設定するために使用されます。

演算項目 2 は必須で、複数オカレンス・データ構造の名前でなければなりません。

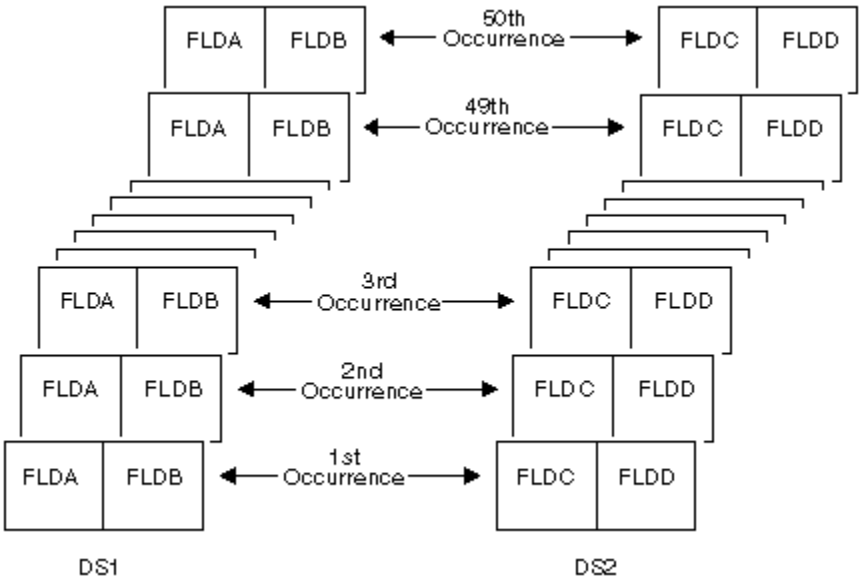
結果フィールドは任意指定です。指定する場合には、小数点以下の桁数がゼ ロの数値フィールド名でなければなりません。 OCCUR 命令の実行中に、演算項目 2 に指定された現在のデータ構造のオカレン スの値が (演算項目 1 に任意に指定された値またはデータ構造によって設定 された後で) 結果フィールドに入れられます。

少なくとも演算項目 1 または結果フィールドのいずれかを指定しなければな りません。

オカレンスがデータ構造について設定された有効な範囲外の場合には、エラーが 発生し、演算項目 2 のデータ構造のオカレンス は、OCCUR 命令が処理される前と同じままで変更されません。

OCCUR 例外 (プログラム状況コード 122) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、163 ページの『プログラム例外/エラー』を参照してください。

複数オカレンス・データ構造がインポートまたはエクスポートされる場合には、 現在のオカレンスについての情報はインポートまたはエクスポートされません。 詳しくは、428 ページの『EXPORT{(外部名)}』 キーワードおよび 441 ページの『IMPORT{(外部名)}』 キーワードを参照してください。




```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
 *
 *  DS1 and DS2 are multiple occurrence data structures.
 *  Each data structure has 50 occurrences.
D DS1              DS                  OCCURS(50)
D  FLDA                       1      5
D  FLDB                       6     80
 *
D DS2              DS                  OCCURS(50)
D  FLDC                       1      6
D  FLDD                       7     11
```

図 *363. OCCUR* 命令の使用法

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *  DS1 is set to the third occurrence.  The subfields FLDA
 *  and FLDB of the third occurrence can now be used.  The MOVE
 *  and Z-ADD operations change the contents of FLDA and FLDB,
 *  respectively, in the third occurrence of DS1.
C
C     3             OCCUR     DS1
C                   MOVE      'ABCDE'       FLDA
C                   Z-ADD     22            FLDB
 *
 *  DS1 is set to the fourth occurrence.  Using the values in
 *  FLDA and FLDB of the fourth occurrence of DS1, the MOVE
 *  operation places the contents of FLDA in the result field,
 *  FLDX, and the Z-ADD operation places the contents of FLDB
 *  in the result field, FLDY.
C
C     4             OCCUR     DS1
C                   MOVE      FLDA          FLDX
C                   Z-ADD     FLDB          FLDY
 *
 *  DS1 is set to the occurrence specified in field X.
 *  For example, if X = 10, DS1 is set to the tenth occurrence.
C     X             OCCUR     DS1
 *
 *  DS1 is set to the current occurrence of DS2.  For example, if
 *  the current occurrence of DS2 is the twelfth occurrence, DSI
 *  is set to the twelfth occurrence.
C     DS2           OCCUR     DS1
 *
 *  The value of the current occurrence of DS1 is placed in the
 *  result field, Z.  Field Z must be numeric with zero decimal
 *  positions.  For example, if the current occurrence of DS1
 *  is 15, field Z contains the value 15.
C                   OCCUR     DS1           Z
C
 *  DS1 is set to the current occurrence of DS2.  The value of the
 *  current occurrence of DS1 is then moved to the result field,
 *  Z.  For example, if the current occurrence of DS2 is the fifth
 *  occurrence, DS1 is set to the fifth occurrence.  The result
 *  field, Z, contains the value 5.
C
C     DS2           OCCUR     DS1           Z
 *
 *  DS1 is set to the current occurrence of X.  For example, if
 *  X = 15, DS1 is set to the fifteenth occurrence.
 *  If X is less than 1 or is greater than 50,
 *  an error occurs and %ERROR is set to return '1'.
 *  If %ERROR returns '1', the LR indicator is set on.
C
C     X             OCCUR (E) DS1
C                   IF        %ERROR
C                   SETON                                       LR
C                   ENDIF
```

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
 *
 * Procedure P1 exports a multiple occurrence data structure.
 * Since the information about the current occurrence is
 * not exported, P1 can communicate this information to
 * other procedures using parameters, but in this case it
 * communicates this information by exporting the current
 * occurrence.
 *
D EXP_DS          DS                  OCCURS(50) EXPORT
D  FLDA                      1      5
D NUM_OCCUR       C                   %ELEM(EXP_DS)
D EXP_DS_CUR      S               5P 0 EXPORT
 *
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq.
 *
 * Loop through the occurrences.  For each occurrence, call
 * procedure P2 to process the occurrence. Since the occurrence
 * number EXP_DS_CUR is exported, P2 will know which occurrence
 * to process.
 *
C                   DO        NUM_OCCUR     EXP_DS_CUR
C     EXP_DS_CUR    OCCUR     EXP_DS
C                     :
C                   CALLB     'P2'
C                   ENDDO
C                     :
```

図 *364.* 複数オカレンス *DS* のエクスポート

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++
 *
 * Procedure P2 imports the multiple occurrence data structure.
 * The current occurrence is also imported.
 *
D EXP_DS          DS                  OCCURS(50) IMPORT
D  FLDA                      1      5
D EXP_DS_CUR      S               5P 0 IMPORT
 *
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq.
 *
 * Set the imported multiple-occurrence data structure using
 * the imported current occurrence.
 *
C     EXP_DS_CUR    OCCUR     EXP_DS
 *
 * Process the current occurrence.
C                     :
```

ON-ERROR (エラーの時)

自由形式構文	ON-ERROR {例外 *ID1* {:例外 *ID2* ...}}

コード	演算項目 1	拡張演算項目 2
ON-ERROR		例外 ID のリスト

例外 ID リスト (例外 *ID1*:例外 *ID2*...) の中のどのエラー条件を ON-ERROR ブロックで処理するかを指定します。. 次のいずれの組み合わせでも、コロンで区切って指定することができます。

nnnnn
状況コード

***PROGRAM**
00100 から 00999 のすべてのプログラム・エラー状況コードを処理します。

***FILE**
01000 から 09999 のすべてのファイル・エラー状況コードを処理します。

***ALL**
00100 から 09999 のプログラム・エラー・コードとファイル・エラー・コードの両方を処理します。これはデフォルト値。

00100 から 09999 の範囲外の状況コード (例えば 0 から 99 のコード) は、モニターの対象にはなりません。 これらの値を ON-ERROR グループに対して指定することはできません。 また、使用されているコンパイラーの特定バージョンに対して無効ないかなる状況コードも、指定することはできません。

複数の ON-ERROR グループが同一の状況コードをカバーしている場合は、最初の 1 つだけが使用されます。 この理由により、特定の状況コードの後に *ALL などの特殊値を指定する必要があります。

ON-ERROR グループの中で発生するいかなるエラーも、MONITOR グループによって処理されることはありません。 エラーを処理するには、ON-ERROR グループの中で MONITOR グループを指定することができます。

ON-ERROR ブロックのすべてのステートメントが処理された場合、ENDMON ステートメントの次のステートメントへ制御が渡されます。

ON-ERROR ステートメントの例については、802 ページの『MONITOR (監視グループの始め)』 を参照してください。

詳しくは、「575 ページの『エラー処理命令』」を参照してください。

ON-EXIT (終了時)

自由形式構文	ON-EXIT {*status*}

コード	演算項目 1	拡張演算項目 2
ON-EXIT		状況標識

ON-EXIT 命令コードが ON-EXIT セクションを開始します。 ON-EXIT セクションには、プロシージャーが終了 (正常終了でも異常終了でも) するたびに実行されるコードが含まれています。 ON-EXIT セクションは、以下の条件下で実行されます。

- プロシージャーが、プロシージャーのメイン部分の終わりに達したとき。
- プロシージャーが RETURN 命令に達したとき。
- プロシージャーが未処理の例外で終了したとき。
- プロシージャーが、ジョブやサブシステムが終了したために、あるいは呼び出しスタック内でより高い優先順位でプロシージャーに送信される例外メッセージのためにキャンセルされたとき。

ON-EXIT セクションにクリーンアップ・コード (一時ファイルの削除やヒープ記憶域の割り振り解除など) を入れることで、プロシージャーが未処理の例外で終了したり、キャンセルされた場合でも、常にこのセクションが実行されるように設定できます。

拡張演算項目 2 には、プロシージャーの終了が正常だったか異常だったかを示すオプションの標識変数が含まれています。 プロシージャーが正常に返された場合、標識は *OFF に設定されます。 プロシージャーが未処理の例外で終了したり、その他の理由でキャンセルされたりした場合、標識は *ON に設定されます。

ON-EXIT セクションは、サブルーチンの後、サブプロシージャーの終わりにコーディングされます。

ON-EXIT セクションでは、プロシージャーで定義されるすべての変数とファイルを利用できます。

プロシージャーの ON-EXIT 部分を実行しても、プロシージャーのメインの部分の状態は変更されません。プロシージャーが未処理の例外のためにキャンセルされた場合、その例外はプロシージャーの ON-EXIT 部分が実行された後も活動状態のままです。

値を返すプロシージャーの場合、そのプロシージャーが正常に終了すると、RETURN ステートメントによって設定された値は ON-EXIT プロシージャーの RETURN ステートメントにオーバーライドされる可能性があります。 ON-EXIT に RETURN ステートメントが含まれていない場合は、プロシージャーはメインの本体の RETURN によって設定された値を返します。841 ページの『ON-EXIT セクションが新規値を返す場合の例』を参照してください。

ON-EXIT の処理中に例外が発生した場合、その例外の影響はプロシージャーが終了した状況により異なります。

- プロシージャーが正常に終了した場合、そのプロシージャーは新規例外で終了します。
- プロシージャーが処理できない例外によって異常終了していた場合、ON-EXIT セクションは即座に終了し、元の例外の例外処理が続行されます。 新規例外は、元のプロシージャー呼び出しの例外処理に影響しません。 このプロシージャーはそのまま元の例外のために終了したと見なされます。
- プロシージャーがその他の理由 (ジョブの終了など) のためにキャンセルされていた場合、ON-EXIT セクションは即座に終了します。 ON-EXIT セクションでの例外は、キャンセル処理全体に影響しません。

ON-EXIT 処理中の例外は、次のような場合に発生します。

- 状況標識の設定中に発生する例外。 これは、状況標識が配列要素であり指標が境界外にある場合、または設定されていないポインターに状況標識が基づいている場合に発生する可能性があります。
- ON-EXIT ステートメントに続くコード内で発生する例外。

ON-EXIT セクションは別個のサブプロシージャーで実行されます。 ON-EXIT サブプロシージャーの名前は「_QRNI_ON_EXIT_」で開始し、 ON-EXIT 命令を含むプロシージャーの名前で終了します。 例えば、外部名が「myProc」のプロシージャーの ON-EXIT サブプロシージャーの外部名は「_QRNI_ON_EXIT_myProc」となります。

ON-EXIT セクションは、プロシージャーへの呼び出しごとに 1 回ずつ起動されます。 正常な戻りの後に ON-EXIT セクションで例外が発生した場合、その ON-EXIT セクションは再実行されません。

ON-EXIT セクションの制約事項

- ON-EXIT セクションを含むプロシージャーで許可されていないものを以下に示します。
  - ローカルの SPECIAL ファイル
  - ローカルのオープン・アクセス・ファイル
  - *PSSR サブルーチン。 代わりに MONITOR グループを使用してください。
  - 複数オカレンス・データ構造。 代わりにデータ構造配列を使用してください。
  - テーブル。 代わりに配列を使用してください。
- Java ネイティブ・メソッドでは ON-EXIT は許可されていません。 ネイティブ・メソッドの論理を、そのネイティブ・メソッドによって呼び出される別のプロシージャーに移動してください。
- サイクル・メイン・プロシージャーでは ON-EXIT は許可されていません。 サイクル・メイン・プロシージャーの論理を、そのサイクル・メイン・プロシージャーによって呼び出される別のプロシージャーに移動するか、リニア・メイン・プロシージャーを使用するようにモジュールを変更してください。
- ON-EXIT セクションで許可されていないものを以下に示します。
  - API の CEEDOD、CEEGSI、および CEETSTA への呼び出し。 これらのプロシージャーは、プロシージャーのメインの本体内で呼び出し、結果をローカル変数に保存してください。
  - 命令コードの BEGSR、DUMP、EXSR、GOTO、RESET、TAG。

デバッグの注意点

- ON-EXIT 命令の停止点に達したときには、すでに状況標識が設定されています。
- ON-EXIT セクションが終了すると、プロシージャーを終了するステートメントの停止点に達します。 プロシージャーのメインの本体の終了時にはこの停止点に達しません。
- ON-EXIT セクションは別個のプロシージャーとして実装されているため、step-over デバッガー・コマンドを使用してプロシージャーのメイン部分から ON-EXIT セクションに簡単に移動することはできませ

ん。 ON-EXIT セクションをデバッグするには、プロシージャーメインの本体の最後のステートメントで step-into debugger コマンドを使用するか、ON-EXIT 命令に停止点を設定するかのいずれかを実行します。

ON-EXIT セクションを使用したプロシージャーの例

以下の例では、*num_orders* で 1 とマークされた行の値がゼロである場合、ゼロ除算例外となり、制御は、 3 とマークされた行で開始する ON-EXIT セクションにすぐに渡されます。標識 *isAbnormalReturn* の値は「1」です。

num_orders がゼロではない場合、プロシージャーは 2 とマークされている行の RETURN 命令まで続行し、制御は 3 の ON-EXIT セクションに渡されます。標識 *isAbnormalReturn* の値は「0」です。

```
dcl-proc myproc;
   dcl-s isAbnormalReturn ind;
   ...
   p = %alloc(100);
   price = total_cost / num_orders; 1
   filename = crtTempFile();
   return; 2
on-exit isAbnormalReturn; 3
   dealloc(n) p;
   if filename <> blanks;
      dltTempFile (filename);
   endif;
   if isAbnormalReturn;
      reportProblem ();
   endif;
end-proc;
```

ON-EXIT セクションが新規値を返す場合の例

以下の例では、プロシージャーで 3 とマークされた行のプロシージャーのメイン部分と、 4 とマークされた ON-EXIT セクションに RETURN 命令が含まれています。ON-EXIT セクションの RETURN は、*returnFromOnExit* パラメーターの値が *ON である場合のみ実行されます。

最初にこのプロシージャーが 1 とマークされた行で呼び出され、パラメーターの値が *OFF である場合、ON-EXIT セクションの RETURN 命令は実行されません。 このプロシージャーは、プロシージャーのメイン部分の RETURN 命令により設定された値を返します。

2 回目にこのプロシージャーが 2 とマークされた行で呼び出され、パラメーターの値が *ON である場合、ON-EXIT セクションの RETURN 命令は実行されます。 このプロシージャーは、ON-EXIT セクションの RETURN 命令により設定された値を返します。

```
ctl-opt dftactgrp(*no);
dcl-s rc packed(5);

rc = myProc2 (*off);  // 1
dsply ('value returned is ' + %char(rc));
rc = myProc2 (*on);   // 2
dsply ('value returned is ' + %char(rc));
return;

dcl-proc myproc2;
   dcl-pi *n packed(5);
      returnFromOnExit ind const;
   end-pi;
   dsply 'myproc2';
   dsply 'myproc2 returning 5';
   return 5; // 3
on-exit;
   dsply 'myproc2 on-exit';
   if returnFromOnExit;
      dsply 'myproc2 ON-EXIT returning 6';
      return 6; // 4
   endif;
end-proc;
```

プロシージャーからの出力値は次のようになります

```
DSPLY  myproc2
DSPLY  myproc2 returning 5
DSPLY  myproc2 on-exit
DSPLY  value returned is 5
DSPLY  myproc2
DSPLY  myproc2 returning 5
DSPLY  myproc2 on-exit
DSPLY  myproc2 ON-EXIT returning 6
DSPLY  value returned is 6
```

OPEN (処理のためのファイルのオープン)

自由形式構文	OPEN{(E)} ファイル名

コード	演算項目 1	演算項目 2	結果フィールド	標識		
OPEN (E)		ファイル名		_	ER	_

明示的な OPEN 命令では、ファイル名 オペランドに指定されたファイルがオープンされます。 指定されたファイルを 1 次ファイル、2 次ファイル、またはテーブル・ファイルにすることはできません。

OPEN 例外 (ファイル状況コードが 1000 より大きい) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、146 ページの『ファイル例外/エラー』を参照してください。

OPEN 命令を明示的に使用して、ファイル名 オペランドで指定されたファイルを、モジュールまたはサブプロシージャー内で初めてオープンするには、ファイル仕様書で USROPN キーワードを指定してください。(USROPN キーワードを使用するときの制約事項については、349 ページの『ファイル仕様書』を参照してください。)

モジュールまたはサブプロシージャー中でファイルがオープンされて、後で CLOSE 命令によってクローズされた場合には、プログラマーは OPEN 命令でこのファイルを再度オープンすることができるため、ファイル仕様書に USROPN キーワードを指定する必要はありません。 ファイル仕様書に USROPN キーワードが指定されていない時は、モジュールの初期化時か (グローバル・ファイルの場合)、サブプロシージャー

の初期化時に (ローカル・ファイルの場合)、ファイルがオープンされます。 すでにオープンされているファ イルに OPEN 命令を指定するとエラーが起こります。

プログラム中での同じファイルに対する複数の OPEN 命令は、OPEN 命令が 出された時にそのファイルがクローズされていれば有効です。

DEVID キーワードを (ファイル仕様書に) 指定してファイルをオープンすると、DEVID キーワードにパラメーターとして指定されたフィールド名はブランクに 設定されます。349 ページの『ファイル仕様書』 の DEVID キーワードの説明を参照してください。

詳しくは、「576 ページの『ファイル操作』」を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords++++++++++++++++++++++++++++++
F
FEXCEPTN   O    E             DISK    USROPN
FFILEX     F    E             DISK
F
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
CL0N01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++..
 *
 *  The explicit OPEN operation opens the EXCEPTN file for
 *  processing if indicator 97 is on and indicator 98 is off.
 *  Note that the EXCEPTN file on the file description
 *  specifications has the USROPN keyword specified.
 *  %ERROR is set to return '1' if the OPEN operation fails.
 *
C                   IF        *in97 and not *in98
C                   OPEN(E)   EXCEPTN
C                   IF        not %ERROR
C                   WRITE     ERREC
C                   ENDIF
C                   ENDIF
 *
 *  FILEX is opened at program initialization.  The explicit
 *  CLOSE operation closes FILEX before control is passed to RTNX.
 *  RTNX or another program can open and use FILEX.  Upon return,
 *  the OPEN operation reopens the file.  Because the USROPN
 *  keyword is not specified for FILEX, the file is opened at
 *  program initialization
 *
C                   CLOSE     FILEX
C                   CALL      'RTNX'
C                   OPEN      FILEX
```

図 *365. CLOSE* 命令を伴う *OPEN* 命令

ORxx (または)

自由形式構文	(許可されていない - OR 演算子を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
ORxx	被比較値	被比較値				

ORxx 命令は、DOUxx、DOWxx、 IFxx、WHENxx、 および ANDxx 命令と一緒に任意に指定します。 ORxx は DOUxx、DOWxx、IFxx、WHENxx、ANDxx、または ORxx ステートメントの 直後に指定します。 ORxx は、DOUxx、DOWxx、IFxx、および WHENxx 命令と複合条件を指定す るために使用します。

制御レベル項目 (7 から 8 桁目) はブランクにするか、 あるいは L1 から L9 標識、LR 標識、または L0 項目を入れてプログラムの該当する セクション内のステートメントをグループにまとめることができます。 制御レベルの指定は、対応する DOUxx、DOWxx、IFxx、または WHENxx 命令の 指定と同じでなければなりません。 条件付け標識の 指定 (9 から 11 桁目) は使用できません。

演算項目 1 と演算項目 2 には、リテラル、名前のついた定数、形象定数、 テーブル名、配列要素、データ構造名、またはフィールド名を入れなければな りません。 演算項目 1 と 2 は同じタイプでなければなりません。 演算項目 1 と 2 の比較は、 比較命令の場合と同じ規則に従って行われます。567 ページの『比較命令』を参照してください。

762 ページの図 313 に、ORxx 命令と ANDxx 命令、および DOUxx 命令の例を示します。

詳しくは、「591 ページの『構造化プログラミング命令』」を参照してください。

OTHER (その他の場合の選択)

自由形式構文	OTHER

コード	演算項目 1	演算項目 2	結果フィールド	標識		
OTHER						

OTHER 命令は、SELECT グループの WHENxx または 902 ページの『WHEN (真の場合に選択)』 (真の場合に選択) 条件が満たされなかった場合に、一連の命令の処理を開始します。 この一連の命令 は ENDSL または END 命令で終了します。

OTHER 命令を使用する場合は、以下の規則に留意してください。

- SELECT グループの中では、OTHER 命令の指定は任意です。
- SELECT グループの中では、OTHER 命令は 1 つだけ指定できます。
- 同じ SELECT グループの OTHER 命令の後に、WHENxx または WHEN 命令を 指定することはできません。
- OTHER グループ内の一連の演算命令を空にすることができます。 効果は OTHER ステートメントを指定しないのと同じです。
- 合計演算の中では、制御レベルの指定 (7 から 8 桁目) は ブランクにするか、あるいはプログラムの該当するセクション内のステートメントを グループにまとめる L1-L9 標識、LR 標識、または L0 の指定を入れることができます。 制御レベルの指定は文書化のためだけのものです。 条件付け標識の指定 (9 から 11 桁目) は使用できません。

詳しくは、「591 ページの『構造化プログラミング命令』」を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 * Example of a SELECT group with WHENxx and OTHER.  If X equals 1,
 * do the operations in sequence 1; if X does not equal 1 and Y
 * equals 2, do the operations in sequence 2.  If neither
 * condition is true, do the operations in sequence 3.
 *
C                   SELECT
C     X             WHENEQ    1
 *
 * Sequence 1
 *
C                     :
C                     :
C     Y             WHENEQ    2
 *
 * Sequence 2
 *
C                     :
C                     :
C                   OTHER
 *
 * Sequence 3
 *
C                     :
C                     :
C                   ENDSL
```

図 *366. OTHER* 命令

詳細および例については、SELECT 命令および WHENxx 命令を参照してくだ さい。

OUT (データ域の書き出し)

自由形式構文	OUT{(E)}{*LOCK} データ域名

コード	演算項目 1	演算項目 2	結果フィールド	標識		
OUT (E)	*LOCK	データ域名		_	ER	_

OUT 命令は、データ域名 オペランドに指定されたデータ域を更新します。 OUT 命令のデータ域名 オペランドにデータ域を指定するためには、次の 2 つの事項について確認する必要があります。

- データ域が、*DTAARA DEFINE ステートメントの 結果フィールドにも指定されているか、あるいは定義仕様書で DTAARA キーワードを 使用して定義されていなければなりません。
- データ域は、前に *LOCK IN ステートメントによって ロックされているか、あるいは定義仕様書の 23 桁目の U に よってデータ域データ構造として指定されていなければなりません。 (RPG IV 言語では、プログラムの初期化時にデータ域データ構造が暗黙に検索されてロックされます。)

任意指定で予約語 *LOCK を指定することができます。 *LOCK を指定した場合には、データ域は更新された後でロックされたままになります。 *LOCK を指定しない場合には、データ域は更新された後でアンロックされます。

データ域名 オペランドが内部データ域またはプログラム初期化パラメーター (PIP) データ域の名前である場合には、*LOCK は指定できません。

データ域名 オペランドは、データ域の名前であるかまたは予約語 *DTAARA かいずれかである必要があります。 *DTAARA が指定された場合には、プログラム内で定義されているすべてのデー タ域が更新されます。 1 つ以上のデータ域の更新時にエラーが発生した場合 (例えば、プログラムによってロックされていないデータ域に OUT 命令を指定した場合) には、OUT 命令でエラーが発生し、RPG IV 例外/エラー処理ルーチンに制御が渡されます。 要求元にメッセージが出された場合には、そのメッセージによってエラーのあ るデータ域が識別されます。

OUT 例外 (プログラム状況コード 401 から 421、431、または 432) を処理するために、命令コード拡張 'E' またはエラー標識 ER を 指定できますが、両方を指定することはできません。エラー処理について詳しくは、163 ページの『プログラム例外/エラー』を参照してください。

71 から 72 桁目と 75 から 76 桁目はブランクでなければなりません。

OUT 命令に関する規則について詳しくは、571 ページの『データ域命令』を参照してください。

OUT 命令の例については、792 ページの図 333 を参照してください。

PARM (パラメーターの識別)

自由形式構文	(許可されていない - 224 ページの『プロトタイプおよびパラメーター』 および CALLP を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
PARM	ターゲット・フィールド	ソース・フィールド	パラメーター			

宣言命令の PARM は、パラメーター・リスト (PLIST) を構成するパラメー ターを定義します。 PARM 命令は、この命令が参照している PLIST、CALL、また は CALLB 命令の直後であれば、演算の中の任意の場所に入れることができます。 PARM ステートメントは、呼び出されるプログラムまたはプロシージャーが必要とする順序でなければなりません。 PARM ステートメントは 1 つ使用するか、 あるいは CALL の場合には最大 255、 CALLB または PLIST の場合には最大 399 まで使用することができます。

PARM 命令は、演算 (合計演算を含む) の中の任意の場所に指定することが できます。 制御レベルの指定 (7 から 8 桁目) は ブランクにするか、あるいはプログラムの適切なセクション内のステートメントを グループにまとめるために L1 から 19 標識、LR 標識、または L0 項目を指定することができます。 条件付け標識の指定 (9 から 11 桁目) は使用できません。

演算項目 1 と演算項目 2 は任意指定です。 指定する場合には、これらの指定は結果フィールドに指定されたものと同じタ イプでなければなりません。 ターゲット・フィールドが可変長の場合、その長さはソース・フィールドの長さの値に設定されます。 リテラルまたは名前付き定数は、演算項目 1 には指定できません。 結果フィールドに複数オカレンス・データ構造の名前または *OMIT が入っている場合には、演算項目 1 と演算項目 2 はブランクである必要があります。

ヒント:

アプリケーションにとってパラメーター・タイプの検査が重要である場合には、PLIST および PARM 命令を使用しないで、呼び出しインターフェースのプロトタイプお よびプロシージャー・インターフェースの定義を定義してください。

結果フィールドには次の名前を指定しなければなりません。

- すべての PARM ステートメントに対して
  - フィールド
  - データ構造
  - 配列
- *ENTRY PLIST PARM 以外のステートメントの場合には次も入れることが できます。
  - 配列要素
  - *OMIT (CALLB のみ)

PARM 命令の結果フィールドの指定に次を入れることはできません。

- *IN, *INxx, *IN(xx)
- リテラル
- 名前のついた定数

- テーブル名

また、*ENTRY PLIST の PARM 命令の結果フィールドの指定には次も使用できま せん。

- *OMIT
- 大域的に初期化されたデータ構造
- 初期化されたサブフィールドを持つデータ構造
- サブフィールドとしてコンパイル時配列を持つデータ構造
- キーワード BASED、IMPORT、または EXPORT によって定義されたフィール ドまたはデータ構造
- 配列要素
- データ域名
- データ域データ構造名
- データ構造サブフィールド
- コンパイル時配列
- プログラム状況データ構造 (PSDS) またはファイル情報 データ構造 (INFDS)

*ENTRY PLIST には、フィールド名を 1 回だけ指定することができます。

結果フィールドに配列が指定されて場合には、呼び出されたプログラムまた はプロシージャーにその配列用に定義された区域が渡されます。 呼び出されたプログラムまたはプロシージャーに複数オカレンス・データ構造が渡 される場合には、そのデータ構造のすべてのオカレンスが単一のフィールドと して渡されます。 しかし、結果フィールドに複数オカレンス・データ構造のサブフィールドが指定さ れている場合には、呼び出されたプログラムまたはプロシージャーにそのサブ フィールドの現在のオカレンスだけが渡されます。

それぞれのパラメーター・フィールドには 1 つの記憶位置があるだけです。 それは呼び出し元のプログラムまたはプロシージャー内にあります。 結果フィールドの記憶位置のアドレスは、呼び出されたプログラムまたはプロ シージャーに PARM 命令で渡されます。 呼び出されたプログラムまたはプロシージャーによってパラメーターの値が変 更される場合には、その記憶位置のデータが変更されます。 呼び出し側プログラムまたはプロシージャーに制御が戻された場合には、呼び出し側プログラムまたはプロシージャーのパラメーター (すなわち、結果フィールド) が変更されています。 呼び出されたプログラムまたはプロシージャーがパラメ ーターの値を変更した後でエラーで終了した場合にも、変更された値は呼び出し側プログラムまたはプロシージャー内に存在します。 呼び出されたプログラムまたはプロシージャーに渡された情報を保存して後で 使用するためには、演算項目 2 に呼び出されたプログラムまたはプロシージャ ーに渡したい情報が入っているフィールドの名前を指定してください。 演算項目 2 が結果フィールドにコピーされて、結果フィールドの記憶域アドレ スが、呼び出されたプログラムまたはプロシージャーに渡されます。

パラメーター・フィールドはフィールド名でなくアドレスでアクセスされる ので、呼び出し元と呼び出し先のパラメーターが受け渡しするフィールドに同じフィ ールド名を使用する必要はありません。 呼び出し元と呼び出し先のプログラムまた はプロシージャーの対応するパラメーター・フィールドの属性は同じでなけれ ばなりません。 そうでない場合には、好ましくない結果が生ずる場合がありま す。

CALL または CALLB 命令が実行されると、次のことが行われます。

1. 呼び出し元のプロシージャーでは、PARM 命令の演算項目 2 のフィールドの 内容が同じ PARM 命令の結果フィールド (レシーバー・フィールド) にコピー されます。
2. CALLB で結果フィールドが *OMIT の場合には、呼び出されたプロシージャ ーにヌルのアドレスが渡されます。
3. 呼び出されたプロシージャーでは、制御が渡されて通常のプログラムの 初期化が行われた後で、PARM 命令の結果フィールドの内容が同じ PARM 命令の演算項目 1 のフィールド (レシーバー・フィールド) にコピーされます。
4. 呼び出されたプロシージャーでは、呼び出し元のプロシージャーに制御が戻 される時に、PARM 命令の演算項目 2 のフィールドの内容が同じ PARM 命令の 結果フィールドにコピーされます。 呼び出されたプロシージャーが異常終了した場合には、この転送は行われま せん。 この転送でエラーが起こると、転送の結果は予測できないものになります。
5. 呼び出し元のプロシージャーに戻ると、呼び出し元のプロシージャーの PARM 命令の結果フィールドの内容が同じ PARM 命令の演算項目 1 フィールド (レシ ーバー・フィールド) にコピーされます。 呼び出

されたプロシージャーが異常終了した場合、または呼び出し命令でエラー が起こった場合には、この転送は行われません。

注 : データは、EVAL 命令コードを使用してデータを転送するのと同じ方法で 転送されます。 タイプの互換性が厳密に適用されます。 CL を介してプログラムを呼び出し、パラメーターを渡す方法に ついては、「*CL プログラミング*」マニュアルを参照してください。

詳しくは、563 ページの『呼び出し命令』または 575 ページの『宣言命令』を参照してください。

849 ページの図 367 は、PARM 命令を示します。

PLIST (パラメーター・リストの識別)

自由形式構文	(許可されていない - 224 ページの『プロトタイプおよびパラメーター』 および CALLP を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
PLIST	PLIST 名					

PLIST 宣言命令は、CALL または CALLB 命令で指定されるパラメーター・ リストに固有の記号名を定義します。

PLIST 命令は、演算内 (合計演算内およびサブルーチン間を含む) の任意の 場所に指定することができます。制御レベルの指定 (7 から 8 桁目) は ブランクにするか、あるいはプログラムの適切なセクション内のステートメントを グループにまとめるために L1 から 19 標識、LR 標識、または L0 項目を指定することができます。 PLIST 命令の直後には、少なくとも 1 つの PARM 命令が続いていなければなり ません。 条件付け標識の指定 (9 から 11 桁目) は使用できません。

演算項目 1 には、パラメーター・リストの名前を入れなければなりません。 パラメーター・リストが入力パラメーター・リストの場合には、 演算項目 1 に *ENTRY を入れなければなりません。 1 つのプログラムまたはプロシージャーに指定できる *ENTRY パラメーター・リストは 1 つだけです。 パラメーター・リストは、PARM 命令以外の命令が見付かった時に終了します。

ヒント :

アプリケーションにとってパラメーター・タイプの検査が重要である場合には、 PLIST および PARM 命令を使用しないで、呼び出しインターフェースのプロトタイプお よびプロシージャー・インターフェースの定義を定義してください。

詳しくは、563 ページの『呼び出し命令』または 575 ページの『宣言命令』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  In the calling program, the CALL operation calls PROG1 and
 *  allows PROG1 to access the data in the parameter list fields.
C                   CALL     'PROG1'        PLIST1
 *
 *  In the second PARM statement, when CALL is processed, the
 *  contents of factor 2, *IN27, are placed in the result field,
 *  BYTE.  When PROG1 returns control, the contents of the result
 *  field, BYTE, are placed in the factor 1 field, *IN30.  Note
 *  that factor 1 and factor 2 entries on a PARM are optional.
 *
C     PLIST1        PLIST
C                   PARM                    Amount            5 2
C     *IN30         PARM      *IN27         Byte              1
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C                   CALLB     'PROG2'
 *  In this example, the PARM operations immediately follow a
 *  CALLB operation instead of a PLIST operation.
C                   PARM                    Amount            5 2
C     *IN30         PARM      *IN27         Byte              1
*...1....+....2....+....3....+....4....+....5....+....6....+....7....+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *  In the called procedure, PROG2,  *ENTRY in factor 1 of the
 *  PLIST statement identifies it as the entry parameter list.
 *  When control transfers to PROG2, the contents of the result
 *  fields (FieldC and FieldG) of the parameter list are placed in
 *  the factor 1 fields (FieldA and FieldD).  When the called procedure
 *  returns, the contents of the factor 2 fields of the parameter
 *  list (FieldB and FieldE) are placed in the result fields (FieldC
 *  and FieldG).  All of the fields are defined elsewhere in the
 *  procedure.
C     *ENTRY        PLIST
C     FieldA        PARM      FieldB        FieldC
C     FieldD        PARM      FieldE        FieldG
```

図 *367. PLIST/PARM* 命令

POST (転記)

自由形式構文	POST{(E)}{プログラム装置} ファイル名

コード	演算項目 1	演算項目 2	結果フィールド	標識		
POST (E)	プログラム装置	ファイル名	INFDS 名	_	ER	_

POST 命令は、情報を INFDS (ファイル情報データ構造) に入れます。 この情報には次のものが含まれます。

- ファイルに関する RPG I/O に固有のファイル・フィードバック情報
- ファイルのオープン・フィードバック情報
- ファイルの入出力フィードバック情報および装置従属フィードバック情報 または属性入手情報

プログラム装置 オペランドにはプログラム装置名を指定して、その特定のプログラム装置についての情報を入手します。 プログラム装置を指定する場合には、ファイルはワークステーション・ファイルとして定義しなければなりません。 プログラム装置 が指定されている場合は、INFDS にオープン・フィードバック情報に続いて属性入手情報が入ります。 長さが 10 桁以下の文字フィールド、文字リテラル、または文字の名前のつい た定数のいずれかを使用してください。 プログラム装置 が指定されていない場合は、INFDS ではオープン・フィードバック情報に続いて入出力フィードバック情報と装置依存情報が入ります。

ファイル名 オペランドにはファイルの名前を指定します。 このファイルの情報が、この ファイルに対応する INFDS に転記されます。

自由形式構文では必ずファイル名 を指定する必要があり、INFDS 名は指定できません。 従来型の構文ではファイル名 も INFDS 名もどちらでも指定できます。

- INFDS 名を指定しない場合、ファイル仕様書の中で INFDS キーワードを使用している、このファイルに関連した INFDS が使用されます。
- 従来型の構文において INFDS 名を指定しなかった場合には、ファイル仕様書の INFDS キーワードに使用されたデータ構造名を結果フィールドに指定しなければなりません。 ファイル仕様書の対応するファイルから情報が転記されます。

POST 例外 (ファイル状況コードが 1000 より大きい) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、146 ページの『ファイル例外/エラー』を参照してください。

POST 命令コードが処理されない場合でも、プログラム内にこの命令コードが存在すると、RPG IV 言語の動作に影響を与える可能性があります。 プログラム装置が指定されていない場合に POST 命令が存在していると、1 つまたは複数のファイルへのフィードバックの転送に影響することがあります。

- グローバル・ファイル仕様書で定義されているファイルにプログラム装置が指定されていない場合に、POST 命令が存在していると、モジュール内のすべてのグローバル・ファイルの INFDS に対する、暗黙的なフィードバック転送に影響が生じます。
- グローバル・ファイルにプログラム装置が指定されていない場合に POST 命令が存在していても、サブプロシージャーで定義されているファイルの INFDS に対する、暗黙的なフィードバック転送には何ら影響はありません。
- ローカルで定義されているファイルにプログラム装置が指定されていない場合に POST 命令が存在していると、そのファイルの INFDS に対する暗黙的なフィードバック転送にのみ影響が生じます。 グローバル・ファイルや、そのサブプロシージャーで定義されている他のファイルについては、何ら影響はありません。
- パラメーターとして渡されるファイルの INFDS への暗黙的なフィードバック転送が実行されるかどうかは、ファイルが定義されているモジュールによって決まります。 フィードバック情報が常にそのファイルの INFDS に転送されている場合、プログラム装置が指定されていない POST 命令が、ファイル・パラメーターに対して冗長に実行されてしまうことがあります。
- 同じモジュール内の他のプロシージャーに、グローバル・ファイルがパラメーターとして渡され、そのプロシージャーによってパラメーターに対する POST 命令が実行された場合、その POST 命令はグローバル・ファイルに対する POST 命令とはみなされません。

通常 INFDS は、入出力命令または命令ブロックごとに更新されます。 ただし、POST 命令の存在によって、ファイルの INFDS へのフィードバック転送に影響が生じる場合には、そのファイルに対する POST 命令が処理されたときにのみ、ファイルの INFDS 内の入出力フィードバック情報域および装置従属フィードバック情報域が、RPG IV によって更新されます。 INFDS のファイル従属情報は、すべての入出力操作で 更新されます。 複数メンバーの処理用にファイルをオープンしている場合には、INFDS のオー プン・フィードバック情報は、入力操作 (READ、READP、READE READPE) によっ て新しいメンバーがオープンされる時に更新されます。

DUMP はその情報を、INFDS からでなく、オープン・データ・パスから直接 検索するので、DUMP のファイル情報セクションは POST に依存しないことに注 意してください。

プログラムに POST 命令コードがないか、またはプログラム装置 を指定した POST 命令コードだけがある場合には、入出力フィードバックおよび装置従属フィードバック・セクションは、それぞれの入出力命令または命令ブロックごとに更新されます。 RPG でレコードがブロック化されている場合には、INFDS の情報のほとんどは 最後に処理された完全なレコード・ブロックに有効となるだけです。 ブロック化された入力をデータベース・ファイルから実行する場合には、RPG は、最後に処理されたレコード・ブロックだけでなく、それぞれの読み取りで INFDS の相対レコード番号およびキー情報を更新します。 より正確な情報が必要な場合には、レコードのブロック化を使用しないでくだ さい。 レコード・ブロック化について詳しくは、146 ページの『ファイル情報データ構造』参照してください。 入出力操作の後で毎回 フィードバック情報を必要としない場合には、フィードバック情報が必要な時に だけ POST 命令を使用してパフォーマンスを改善することができます。

POST 命令が処理される時には、対応するファイルがオープンされていなけ ればなりません。 POST 命令でプログラム装置を指定する場合には、そのファイ ルによって装置が入手されている必要はありません。

詳しくは、「576 ページの『ファイル操作』」を参照してください。

READ (レコードの読み取り)

自由形式構文	READ{(EN) } 名前 {データ構造 }

コード	演算項目 1	演算項目 2	結果フィールド	標識		
READ (E N)		名前 (ファイルまたはレコード様式)	データ構造	_	ER	EOF

READ 命令は、全手順ファイルから、現在ポイントされているレコードを読み取ります。

名前 オペランドは必須で、ファイルまたはレコード様式の名前でなければなりません。 レコード様式名を使用できるのは、外部記述ファイルの場合のみです。 「形式名による READ」命令では、名前 オペランドに指定した形式とは別の形式を受け取る場合があります。 その場合には、READ 命令はエラーで終了します。

データ構造 オペランドが指定されている場合、レコードはデータ構造に直接読み込まれます。 名前 がプログラム記述ファイルを参照する場合、データ構造は、宣言されたファイルのレコード長と同じ長さの任意のデータ構造にできます。 名前 が外部記述ファイルまたは外部記述ファイルのレコード様式を参照する場合、データ構造は EXTNAME(...:*INPUT or *ALL) または LIKEREC(...:*INPUT or *ALL) で定義されているデータ構造にする必要があります。 データ構造の定義方法、およびファイルとデータ 構造の間でどのようにデータが転送されるかについては、576 ページの『ファイル操作』 を参照してください。

READ 命令が正常に実行された場合には、ファイルはその読み取りを満たす次のレ コードに位置付けられます。 レコード・ロック・エラー (状況コード 1218) が発生した場合、ロックされたレコードにファイルが置かれたままになり、そのレコードの読み取りが次の読み取り操作において再試行されます。 その他の何らかのエラーが発生するか、ファイルの終わり条件が存在する場合には、(CHAIN 命令、SETLL 命令、または SETGT 命令を使用して) ファイルの位置を変更する必要があります。

読み取るファイルが更新ディスク・ファイルの場合には、命令拡張 N を指 定して、読み取り時にレコードをロックする必要がないことを指示することがで きます。 詳細については、*Rational Development Studio for i: ILE RPG* プログラマーの手引き を参照してください。

READ 例外 (ファイル状況コードが 1000 より大きい) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。 エラー処理の詳細については、146 ページの『ファイル例外/エラー』 を 参照してください。

75-76 桁目に、READ 命令でファイルの終わりになったかどうかを知らせる標識を指定 することができます。 この標識は READ 命令が実行されるたびに オン (EOF 条件) またはオフに設定されます。 この情報は %EOF 組み込み関数からも入手することができます。この関数 は、EOF 条件が発生した場合に '1' を戻します。それ以外の場合には '0' を戻します。 EOF 条件の後でさらにファイルに対する正常な 順次操作 (例えば、READ または READP) を処理 するためには、ファイルの位置を決め直さなければなりません。

852 ページの図 368 に、READ 命令を示します。

名前 が複数装置ファイルを指定している場合には、READ 命令によって次のいずれかが実行されます。

- 最後の NEXT 命令で指定された装置からデータが読み取られます (そのよ うな NEXT 命令が処理された場合)。
- ファイルに対して入手されていて、DDS の INVITE キーワードによって “送信勧誘状態” に指定されている装置から最初の応答を受け入れます。送信勧誘された装置がない場合には、この命令はファイルの終わりを受け取ります。 入力は対応する形式に従って処理されます。 装置がワークステーションの場合 には、最後に書き込まれた形式が使用されます。 装置が通信装置の場合には、ユーザーが形式を選択できます。

  ICF ファイルの形式選択処理について詳しくは、「*ICF Programming*」*(SC41-5442)* を参照してください。

  READ 命令は、一定の時間何も入力されない場合、または制御された方式による オプションを指定して次の CL コマンドの 1 つが入力された場合には、待機を 停止します。

  – ENDJOB (ジョブ終了)
  – ENDSBS (サブシステム終了)

- PWRDWNSYS (システム電源遮断)
- ENDSYS (システム終了)

この結果はファイル例外/エラーとなり、ユーザーのプログラムに指定された方法 によって処理されます (146 ページの『ファイル例外/エラー』を参照してください)。ファイルを作成または変更するコマンドでの WAITRCD パラメーターについては、「*ICF Programming*」*(SC41-5442)* を参照してください。 このパラメーターは、READ 命令が入力を待機する時間の長さを制御します。

名前 に形式名を指定し、その形式名が複数装置ファイルに対応する場合には、データはファイル仕様書の DEVID キーワードに指定されたフィールドによって識別される装置から読み取られます。 そのような指定がない場合には、 データは最後の正常な入力操作で使用された装置から読み取られます。

ヌル値可能フィールドを持つレコードの読み取りについて は、286 ページの『データベースのヌル値サポート』を参照してください。

詳細については、576 ページの『ファイル操作』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  READ retrieves the next record from the file FILEA, which must
 *  be a full procedural file.
 *  %EOF is set to return '1' if an end of file occurs on READ,
 *  or if an end of file has occurred previously and the file
 *  has not been repositioned.  When %EOF returns '1',
 *  the program will leave the loop.
 *
C                   DOW       '1'
C                   READ      FILEA
C                   IF        %EOF
C                   LEAVE
C                   ENDIF
 *
 *  READ retrieves the next record of the type REC1 (factor 2)
 *  from an externally described file.  (REC1 is a record format
 *  name.)  Indicator 64 is set on if an end of file occurs on READ,
 *  or if it has occurred previously and the file has not been
 *  repositioned.  When indicator 64 is set on, the program
 *  will leave the loop.  The N operation code extender
 *  indicates that the record is not locked.
 *
C                   READ(N)   REC1                                  64
C   64              LEAVE
C                   ENDDO
```

図 *368. READ* 命令

READC (次の変更レコードの読み取り)

自由形式構文	READC{(E) } レコード名 *{データ構造 }*

コード	演算項目 1	演算項目 2	結果フィールド	標識		
READC (E)		レコード名	データ構造	_	ER	EOF

READC 命令は、サブファイルの次の変更済みレコードを取り出すために、外部記述ワークステーション・ファイルのみで使用することができます。 レコード名 オペランドは必須で、ファイル仕様書の SFILE キーワードによってサブファイルとして定義されたレコード様式の名前でなければなりません (SFILE キーワードについては、387 ページの『SFILE(レコード様式:RRN フィールド)』 を参照してください)。

複数装置ファイルの場合には、データはプログラム装置に対応するサブファ イル・レコードから読み取られます。このプログラム装置は、ファイル仕様書の DEVID キーワードに指定されたフィールドによって識

別されます。 そのような 指定がない場合には、データは最後の正常な入力操作で使用されたプログラム 装置から読み取られます。

READC 例外 (ファイル状況コードが 1000 より大きい) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、146 ページの『ファイル例外/エラー』を参照してください。

サブファイルにそれ以上変更済みレコードがない時にオンに 設定される標識を 75 から 76 桁目に指定することができます。 この情報 は %EOF 組み込み関数からも入手することができます。この関数は、サブファイルに それ以上変更済みレコードがない場合に '1' を戻します。それ以外の場合には '0' を戻します。

データ構造 オペランドが指定されている場合、レコードはデータ構造に直接読み込まれます。 データ構造は、EXTNAME(...:*INPUT or *ALL) または LIKEREC(...:*INPUT or *ALL) で定義されているデータ構造にする必要があります。 データ構造の定義方法、およびファイルとデータ構造の間のデータ転送方法については、576 ページの『ファイル操作』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++
 * CUSSCR is a WORKSTN file which displays a list of records from
 * the CUSINFO file. SFCUSR is the subfile name.
 *
FCUSINFO   UF   E             DISK
FCUSSCR    CF   E             WORKSTN SFILE(SFCUSR:RRN)
F
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 * After the subfile has been loaded with the records from the
 * CUSINFO file. It is written out to the screen using EXFMT with
 * the subfile control record, CTLCUS. If there are any changes in
 * any one of the records listed on the screen, the READC operation
 * will read the changed records one by one in the do while loop.
 * The corresponding record in the CUSINFO file will be located
 * with the CHAIN operation and will be updated with the changed
 * field.
C                   :
C                   EXFMT     CTLCUS
C                   :
 * SCUSNO, SCUSNAM, SCUSADR, and SCUSTEL are fields defined in the
 * subfile. CUSNAM, CUSADR, and CUSTEL are fields defined in a
 * record, CUSREC which is defined in the file CUSINFO.
 *
C                   READC     SFCUSR
C                   DOW       %EOF = *OFF
C     SCUSNO        CHAIN (E) CUSINFO
 * Update the record only if the record is found in the file.
C                   :
C                   IF        NOT %ERROR
C                   EVAL      CUSNAM = SCUSNAM
C                   EVAL      CUSADR = SCUSADR
C                   EVAL      CUSTEL = SCUSTEL
C                   UPDATE    CUSREC
C                   ENDIF
C                   READC (E) SFCUSR
C                   ENDDO
```

図 *369. READC* の例

READE (等しいキーのレコードの読み取り)

自由形式構文	READE{(ENHMR) } 検索引数|*KEY 名前 {データ構造 }

コード	演算項目 1	演算項目 2	結果フィールド	標識		
READE (E N)	検索引数	名前 (ファイルまたはレコード様式)	データ構造	_	ER	EOF

READE 命令は、レコードのキーが検索引数と一致する場合に、全手順ファイルから次の順次レコードを 取り出します。 レコードのキーが検索引数と一致しない場合 には、EOF 条件が発生し、レコードはプログラム に戻されません。 EOF 条件は、ファイルの終わりになった場合に も適用されます。

検索引数 (検索引数) は、検索するレコードを識別します。 検索引数 オペランドは、従来型の構文においては任意指定ですが、自由形式構文では必須です。検索引数 は次のいずれかが可能です。

- フィールド名、リテラル、名前のついた定数、または形象定数
- 外部記述ファイルの場合は KLIST 名
- 括弧で囲まれたキー値のリスト。 キー値のリストを使用した検索の例については、730 ページの図 295 を参照してください。
- 検索引数はデータ構造のサブフィールドであることを指示する %KDS データ構造内の検索引数の図については、652 ページの『%KDS (データ構造の検索引数)』 の終わりにある例を参照してください。
- *KEY または (従来型の構文の場合のみ) 値なし。 次のレコードの全キーが現行レコードの全キーと等しい場合には、ファイルの次のレコードが検索されます。 全キーは、名前 で指定されているレコード様式またはファイルによって定義されます。

制御キーワード EXPROPTS(*STRICTKEYS) が値のリストまたは %KDS でキーを指定する規則に与える影響について詳しくは、335 ページの『*STRICTKEYS』を参照してください。

注 : ファイルが更新として定義され、N 命令拡張が指定されていない場合、キー値が検索引数と一致しないレコードに対する一時的なレコード・ロックにより、READE 命令が待たされることがあります。 一時的ロックが取得されると、キー値が検索引数に一致しない場合に一時的ロックがリリースされます。

ほとんどの場合、RPG は、一致するレコードがあるかどうかの判別に一時的レ コード・ロックを取得する必要がないシステム・サポートを使用して READE を実行できます。 ただし、RPG がこのサポートを使用できない場合があり、その場合は次のレコードを要求してから、READE 要求でレコードの一致を判別する必要があります。

以下の理由により、RPG は、READE 命令に対し次のレコードの一時的ロックを取得することを必要とします。

- 現行レコードのキーが検索引数と一致しない
- 現行レコードが要求されたレコードと一致しない
- ファイルにヌル可能フィールドがある
- モジュールに ALWNULL(*USRCTL) が指定されている
- ファイルにファイル終了の遅延がある

注 :

図形キーと UCS-2 キーは CCSID が同じでなければなりません。

名前 オペランドは、検索するファイルまたはレコード様式の名前でなければなりません。 レコード様式名を使用できるのは、外部記述ファイルの場合だけです。

データ構造 オペランドが指定されている場合、レコードはデータ構造に直接読み込まれます。 名前 が プログラム記述ファイルを参照する場合、データ構造は、 宣言されたファイルのレコード長と同じ長さの任意のデータ構造にできます。 名前 が外部記述ファイルまたは外部記述ファイルのレコード様式を参照する場合、データ構造は EXTNAME(...:*INPUT or *ALL) または LIKEREC(...:*INPUT or *ALL) で定義されているデータ構造にする必要があります。 データ構造の定義方法、およびファイルとデータ構造の間のデータ転送方法については、576 ページの『ファイル操作』を参照してください。

読み取るファイルが更新ディスク・ファイルの場合には、命令拡張 N を指 定して、読み取り時にレコードをロックする必要がないことを指示することが できます。 詳しくは、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」を参照してください。

READE 例外 (ファイル状況コードが 1000 より大きい) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、146 ページの『ファイル例外/エラー』を参照してください。

EOF 条件が発生した場合、すなわち、検索引数と等しいキーを持つレコードが見付から ないか、またはファイルの終わりになった場合にオンに設定される標識 を 75 から 76 桁目に指定することができます。 こ

の情報は %EOF 組み込み関数からも入手することができます。この関数 は、EOF 条件が発生した場合に '1' を戻します。それ以外の場合には '0' を戻します。

READE 命令が正常に実行された場合には、その命令を満たす次のレコードに、ファイルが置かれます。 レコード・ロック・エラー (状況コード 1218) が発生した場合、ロックされたレコードにファイルが置かれたままになり、そのレコードの読み取りが次の読み取り操作において再試行されます。 その他の何らかのエラーが発生するか、ファイルの終わり条件が存在する場合には、(CHAIN 命令、SETLL 命令、または SETGT 命令を使用して) ファイルの位置を変更する必要があります。 728 ページの『CHAIN (ファイルからのランダム検索)』、873 ページの『SETGT (より大きい設定)』、または 876 ページの『SETLL (下限の設定)』 を参照してください。

通常、指定されたキーとファイル内の実際のキーの比較は、データ管理機能により実行されます。 データ管理機能でこれを実行できない場合は、16 進数照合シーケンスを使用して比較が実行されます。 この場合、予想通りの結果が得られないことがあります。 詳しくは、「*Rational Development Studio for i: ILE RPG プログラマーの手引き*」のセクション『キー付きファイルの使用による予期しない結果』を参照してください。

OPEN 命令または EOF 条件の直後に指定された、検索引数 オペランドを持つ READE では、レコードのキーが検索引数と一致した場合に、ファイルの最初のレコードが検索されます。 検索引数 が指定されて**いない**、OPEN 命令または EOF 条件の直後の READE は、エラー条件を招きます。 73 桁目 と 74 桁目のエラー標識が指定されている場合、オンに設定されるか、 または %ERROR によって検査される 'E' 拡張が指定されている場合、オンに 設定されます。 ファイルが正常にクローズされて再びオープンされるまでは、それ以上入出力 命令を出すことはできません。

ヌル値可能フィールドおよびキーを持つレコードの処理について詳しくは、286 ページの『データベースのヌル値サポート』を参照してください。

詳しくは、「576 ページの『ファイル操作』」を参照してください。

注 : 命令コード拡張 H、M、および R は、検索引数がリストまたは %KDS() である場合にのみ使用できます。579 ページの『ファイル命令のキー』 および 558 ページの『精度の確認』 を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  With Factor 1 Specified...
 *
 *  The READE operation retrieves the next record from the file
 *  FILEA and compares its key to the search argument, KEYFLD.
 *
 *  The %EOF built-in function is set to return '1' if KEYFLD is
 *  not equal to the key of the record read or if end of file
 *  is encountered.
 *
C     KEYFLD        READE     FILEA
 *
 *  The READE operation retrieves the next record of the type REC1
 *  from an externally described file and compares the key of the
 *  record read to the search argument, KEYFLD.  (REC1 is a record
 *  format name.)  Indicator 56 is set on if KEYFLD is not equal to
 *  the key of the record read or if end of file is encountered.
C     KEYFLD        READE     REC1                                 56
 *
 *  With No Factor 1 Specified...
 *
 *  The READE operation retrieves the next record in the access
 *  path from the file FILEA if the key value is equal to
 *  the key value of the record at the current cursor position.
 *
 *  If the key values are not equal, %EOF is set to return '1'.
C                   READE     FILEA
 *
 *  The READE operation retrieves the next record in the access
 *  path from the file FILEA if the key value equals the key value
 *  of the record at the current position.  REC1 is a record format
 *  name.  Indicator 56 is set on if the key values are unequal.
 *  N indicates that the record is not locked.
C                   READE(N)  REC1                                 56
```

図 *370. READE* 命令

READP (前のレコードの読み取り)

自由形式構文	READP {(EN) } 名前 {データ構造 }

コード	演算項目 1	演算項目 2	結果フィールド	標識		
READP (E N)		名前 (ファイルまたはレコード様式)	データ構造	_	ER	BOF

READP 命令は、全手順ファイルから、前のレコードを読み取ります。

名前 オペランドは、読み取るファイルまたはレコード様式の名前でなければなりません。 レコード様式名を使用できるのは、外部記述ファイルの場合だけです。 名前 にレコード様式名が指定されている場合には、検索されるレコードは、指定したタイプで前にある最初のレコードです。 間にあるレコードは回避されます。

データ構造 オペランドが指定されている場合、レコードはデータ構造に直接読み込まれます。 名前 がプログラム記述ファイルを参照する場合、データ構造には、宣言されたファイルのレコード長と同じ長さの任意のデータ構造を設定できます。 名前 が外部記述ファイルまたは外部記述ファイルのレコード様式を参照する場合、データ構造は EXTNAME(...:*INPUT or *ALL) または LIKEREC(...:*INPUT or *ALL) で定義されているデータ構造にする必要があります。 データ構造の定義方法、およびファイルとデータ構造の間のデータ転送方法については、576 ページの『ファイル操作』 を参照してください。

READP 命令が正常に実行された場合には、ファイルは、その読み取りを満たす 前のレコードに位置付けられます。

読み取るファイルが更新ディスク・ファイルの場合には、命令拡張 N を指 定して、読み取り時にレコードをロックする必要がないことを指示することがで きます。 詳しくは、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」を参照してください。

READP 例外 (ファイル状況コードが 1000 より大きい) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、146 ページの『ファイル例外/エラー』を参照してください。

ファイルに前のレコードが存在しない (ファイルの始め状態) 場合にオンに設定 される標識を 75 から 76 桁目に指定することができます。 この情報は %EOF 組み込み関数からも入手することができます。この関数 は、BOF 条件が発生した場合に '1' を戻します。それ以外の場合には '0' を戻します。

レコード・ロック・エラー (状況コード 1218) が発生した場合、ロックされたレコードにファイルが置かれたままになり、そのレコードの読み取りが次の読み取り操作において再試行されます。 その他の何らかのエラーが発生するか、ファイルの先頭条件が存在する場合には、(CHAIN 命令、SETLL 命令、または SETGT 命令を使用して) ファイルの位置を変更する必要があります。

ヌル値可能フィールドを持つレコードの読み取りについては、286 ページの『データベースのヌル値サポート』を参照してください。

詳しくは、576 ページの『ファイル操作』 を参照してください。

857 ページの図 371 に、演算項目 2 にファイル名およびレコード様式名を指定した READP 命令を示します。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  The READP operation reads the prior record from FILEA.
 *
 *  The %EOF built-in function is set to return '1' if beginning
 *  of file is encountered.  When %EOF returns '1', the program
 *  branches to the label BOF specified in the GOTO operation.
C                   READP     FILEA
C                   IF        %EOF
C                   GOTO      BOF
C                   ENDIF
 *
 *  The READP operation reads the next prior record of the type
 *  REC1 from an externally described file.  (REC1 is a record
 *  format name.)  Indicator 72 is set on if beginning of file is
 *  encountered during processing of the READP operation.  When
 *  indicator 72 is set on, the program branches to the label BOF
 *  specified in the GOTO operation.
C                   READP     PREC1                                  72
C   72              GOTO      BOF
 *
C     BOF           TAG
```

図 *371. READP* 命令

READPE (等しいキーの前のレコードの読み取り)

自由形式構文	READPE{(ENHMR) } 検索引数|*KEY 名前 {データ構造 }

コード	演算項目 1	演算項目 2	結果フィールド	標識		
READPE (E N)	検索引数	名前 (ファイルまたはレコード様式)	データ構造	_	ER	BOF

READPE 命令は、レコードのキーが検索引数と一致した場合に、全手順ファ イルから 1 つ前の順次レコードを検索します。 レコードのキーが検索引数と一致しない場合には、BOF 条件が発生し、 レコードはプログラムに戻されません。 BOF 条件 は、ファイルの先頭になった場合にも適用されます。

検索引数 (検索引数) は、検索するレコードを識別します。 検索引数 オペランドは、従来型の構文においては任意指定ですが、自由形式構文では必須です。検索引数 は次のいずれかが可能です。

- フィールド名、リテラル、名前のついた定数、または形象定数
- 外部記述ファイルの場合は KLIST 名
- 括弧で囲まれたキー値のリスト。 キー値のリストを使用した検索の例については、730 ページの図 295 を参照してください。
- 検索引数はデータ構造のサブフィールドであることを指示する %KDS データ構造内の検索引数の図については、652 ページの『%KDS (データ構造の検索引数)』 の終わりにある例を参照してください。
- *KEY または (従来型の構文の場合のみ) 値なし。 1 つ前のレコードの全キーが現行レコードの 全キーと等しい場合には、ファイルの 1 つ前のレコードが検索されます。 全キーは、演算項目 2 に使用されたレコード様式またはファイルによって定義されます。

KLIST を 使用して指定されたキーの場合、キー・フィールドはファイル内のキーと同じ CCSID である必要があります。

制御キーワード EXPROPTS(*STRICTKEYS) が値のリストまたは %KDS でキーを指定する規則に与える影響について詳しくは、335 ページの『*STRICTKEYS』 を参照してください。

名前 オペランドは、検索するファイルまたはレコード様式の名前でなければなりません。 レコード様式名を使用できるのは、外部記述ファイルの場合だけです。

データ構造 オペランドが指定されている場合、レコードはデータ構造に直接読み込まれます。 名前 がプログラム記述ファイルを参照する場合、データ構造には、宣言されたファイルのレコード長と同じ長さの任意のデータ構造を設定できます。 名前 が外部記述ファイルまたは外部記述ファイルのレコード様式を参照する場合、データ構造は EXTNAME(...:*INPUT or *ALL) または LIKEREC(...:*INPUT or *ALL) で定義されているデータ構造にする必要があります。 データ構造の定義方法、およびファイルとデータ構造の間のデータ転送方法については、576 ページの『ファイル操作』 を参照してください。

読み取るファイルが更新ディスク・ファイルの場合には、命令拡張 N を指 定して、読み取り時にレコードをロックする必要がないことを指示することがで きます。 詳しくは、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」を参照してください。

READPE 例外 (ファイル状況コードが 1000 より大きい) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、146 ページの『ファイル例外/エラー』 を参照してください。

BOF 条件が発生した場合、すなわち、検索引数と等しいキーを持つレコードが見付から ないか、またはファイルの先頭になった場合にオンに設定される標識 を 75 から 76 桁目に指定することができます。 この情報は %EOF 組み込み関数からも入手することができます。この関数 は、BOF 条件が発生した場合に '1' を戻します。それ以外の場合には '0' を戻します。

レコード・ロック・エラー (状況コード 1218) が発生した場合、ロックされたレコードにファイルが置かれたままになり、そのレコードの読み取りが次の読み取り操作において再試行されます。 その他の何らかのエラーが発生するか、ファイルの先頭条件が存在する場合には、(CHAIN 命令、SETLL 命令、または SETGT 命令を使用して) ファイルの位置を変更する必要があります。 728 ページの『CHAIN (ファイルからのランダム検索)』、873 ページの『SETGT (より大きい設定)』、または 876 ページの『SETLL (下限の設定)』 を参照してください。

注 : ファイルが更新として定義され、N 命令拡張が指定されていない場合、キー値が検索引数と一致しないレコードに対する一時的なレコード・ロックにより、READPE 命令が待たされることがあります。 一時的ロックが取得されると、キー値が検索引数に一致しない場合に一時的ロックがリリースされます。

ほとんどの場合、 RPG は、一致するレコードがあるかどうかの判別に一時的レ コード・ロックを取得する必要がないシステム・サポートを使用して READPE を実行できます。 ただし、RPG がこのサポートを使用できない場合があり、その場合は次のレコードを要求してから、READPE 要求でレコードの一致を判別する必要があります。

以下の理由により、RPG は、READPE 命令に対し次のレコードの一時的ロックを取得することを必要とします。

- 現行レコードのキーが検索引数と一致しない

- 現行レコードが要求されたレコードと一致しない
- ファイルにヌル可能フィールドがある
- モジュールに ALWNULL(*USRCTL) が指定されている
- ファイルにファイル終了の遅延がある

通常、指定されたキーとファイル内の実際のキーの比較は、データ管理機能により実行されます。 データ管理機能でこれを実行できない場合は、16 進数照合シーケンスを使用して比較が実行されます。 この場合、予想通りの結果が得られないことがあります。 詳しくは、「*Rational Development Studio for i: ILE RPG プログラマーの手引き*」のセクション『キー付きファイルの使用による予期しない結果』を参照してください。

OPEN 命令または BOF 条件の直後の READPE に検索引数 オペランドが指定された場合は BOF を戻します。 検索引数 が指定されて**いない**、OPEN 命令または BOF 条件の直後の READPE は、エラー条件を招きます。 73 桁目 と 74 桁目のエラー標識が指定されている場合、オンに設定されるか、 または %ERROR によって検査される 'E' 拡張が指定されている場合、オンに 設定されます。 ファイルは、演算項目 1 がブランクの READPE 命令を出す前に、検索引数 が指定された CHAIN、SETLL、READ、READE、または READP を使用して再度位置決めされなければなりません。 READPE (検索引数 を指定しないで) を出す前に SETGT 命令コードを使用してファイルの位置決めを行うと該当レコードなしの状況になります。 そのような使い方はしないでください (SETGT が出された後では、現行レコードの前のレコードが現行レコードと同じキーを持つことはないからです)。 両方の命令コードに同じキーを使用して検索引数 を指定した場合は、このエラー状況は生じません。

ヌル値可能フィールドおよびキーを持つレコードの処理について詳しくは、286 ページの『データベースのヌル値サポート』を参照してください。

詳しくは、「576 ページの『ファイル操作』」を参照してください。

注 : 命令コード拡張 H、M、および R は、検索引数がリストまたは %KDS() である場合にのみ使用できます。579 ページの『ファイル命令のキー』 および 558 ページの『精度の確認』 を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 * With Factor 1 Specified...
 *
 * The previous record is read and the key compared to FieldA.
 * Indicator 99 is set on if the record's key does not match
 * FieldA.
C     FieldA        READPE    FileA                                  99
 *
 * The previous record is read from FileB and the key compared
 * to FieldB.  The record is placed in data structure Ds1.  If
 * the record key does not match FieldB, indicator 99 is set on.
C     FieldB        READPE    FileB         Ds1                      99
 *
 * The previous record from record format RecA is read, and
 * the key compared to FieldC.  Indicator 88 is set on if the
 * operation is not completed successfully, and 99 is set on if
 * the record key does not match FieldC.
C     FieldC        READPE    RecA                                   8899
 *
 * With No Factor 1 Specified...
 *
 * The previous record in the access path is retrieved if its
 * key value equals the key value of the current record.
 * Indicator 99 is set on if the key values are not equal.
C                   READPE    FileA                                  99
 *
 * The previous record is retrieved from FileB if its key value
 * matches the key value of the record at the current position
 * in the file.  The record is placed in data structure Ds1.
 * Indicator 99 is set on if the key values are not equal.
C                   READPE    FileB         Ds1                      99
 *
 * The previous record from record format RecA is retrieved if
 * its key value matches the key value of the current record in
 * the access path.  Indicator 88 is set on if the operation is
 * not successful; 99 is set on if the key values are unequal.
C                   READPE    RecA                                   8899
```

図 *372. READPE* 命令

REALLOC (新しい長さでの記憶域の再割り振り)

自由形式構文	(許可されていない - %REALLOC 組み込み関数を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
REALLOC (E)		長さ	ポインター	_	ER	_

REALLOC 命令は、結果フィールド・ポインターによって指し示されている動的記憶域の長さを、演算項目 2 に指定された長さに変更します。 REALLOC の結果フィールドには基底ポインター変数が入っています。 結果フィールド・ポインター には、動的記憶域割り振り命令 (RPG 内の ALLOC または REALLOC 命令、あるいは CEEGTST などの他の何らかの動的記憶域機能) によって 直前に設定された値が入っていなければなりません。 このポインターは、動的記憶域を単純に指し示すだけのものでは ありません。割り振りの始めに設定されていることも必要です。

指定されたサイズの新しい記憶域が割り振られ、古い記憶域の 値は、新しい記憶域にコピーされます。 古い記憶域は割り振り解除されます。 新しい長さが短い場合、値の右辺が切り捨てられます。 新しい長さが長い場合には、新しい記憶域のうち、コピーされたデータから 右側は初期化されません。

結果フィールド・ポインターは、新しい記憶域を指し示すように 設定されます。

この命令が正常に行われないと、エラー条件が発生しますが、 結果フィールド・ポインターは変更されません。 元のポインターが有効であり、使用可能な新しい記憶域が不十分であるために (状況 425) 命令が失

敗した場合は、 元の記憶域は割り振り解除されず、そのため、 結果フィールド・ポインターはその元の値のまま、引き続き有効になります。

ポインターが有効でも、割り振り解除できる記憶域を指し示していない 場合、426 (記憶域管理命令内のエラー) が設定されます。

プログラム状況コードが 425 または 426 である 例外を処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、163 ページの『プログラム例外/エラー』を参照してください。

演算項目 2 には、割り振られる新しいサイズの記憶域 (バイト単位) を 示す数値変数または定数が入ります。 演算項目 2 は小数点以下の桁がない数値でなければなりません。 値は、1 から最大許容サイズまでの範囲でなければなりません。

最大許容サイズは、制御仕様書の ALLOC キーワードによる、メモリー管理命令に使用されるヒープ記憶域のタイプによって異なります。 モジュールがメモリー管理命令にテラスペース記憶域モデルを使用することがコンパイル時に分かっている場合、 許容最大サイズは 4294967295 バイトです。 それ以外の場合、最大許容サイズは 16776704 バイトです。

実行時に使用可能な最大サイズは、RPG の許容最大サイズより小さい場合があります。

モジュールの RPG メモリー管理命令が、制御仕様書の ALLOC キーワードにより、単一レベル・ヒープ記憶域を使用している場合、 REALLOC 命令は、単一レベル・ヒープ記憶域へのポインターのみを処理できます。 モジュールの RPG メモリー管理命令が、テラスペース・ヒープ記憶域を使用している場合、REALLOC 命令は、単一レベルとテラスペースの両方のヒープ記憶域へのポインターを処理できます。

詳しくは、「581 ページの『メモリー管理命令』」を参照してください。

```
D Ptr1            S               *
D Fld             S          32767A     BASED(Ptr1)
 * The ALLOC operation allocates 7 bytes to the pointer Ptr1.
 * After the ALLOC operation, only the first 7 bytes of variable
 * Fld can be used.
C                   ALLOC     7                 Ptr1
C                   EVAL      %SUBST(Fld : 1 : 7) = '1234567'
C                   REALLOC   10                Ptr1
 * Now 10 bytes of Fld can be used.
C                   EVAL      %SUBST(Fld : 1 : 10) = '123456789A'
```

図 *373. REALLOC* 命令

REL (解放)

自由形式構文	REL {(E) } プログラム装置 ファイル名

コード	演算項目 1	演算項目 2	結果フィールド	標識		
REL (E)		ファイル名		_	ER	_

REL 命令では、プログラム装置 に指定されたプログラム装置が、ファイル名 に指定されたワークステーション・ファイルから解放されます。

プログラム装置名はプログラム装置 オペランドに指定します。 長さが 10 桁以下の文字 フィールド、文字リテラル、または名前のついた定数のいずれかを使用 してください。 ファイル名はファイル名 オペランドに指定します。

REL 例外 (ファイル状況コードが 1000 より大きい) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、146 ページの『ファイル例外/エラー』を参照してください。

ワークステーション・ファイルに入手されたプログラム装置がない場合には、次のファイ ル名別 READ またはサイクル読み取りでファイルの終わり条件になります。 プログラムが次に何を実行するかはユーザー

が決定しなければなりません。 REL 命令は複数装置ファイルで使用するか、または (エラー回復のために) 単 一装置ファイルで使用することができます。

注 : レコード・ロックを解除するためには、UNLOCK 命令を使用してください。 更新ディスク・ファイルのレコード・ロックの解除の詳細については、UNLOCK 命令を参照してください。

詳しくは、「576 ページの『ファイル操作』」を参照してください。

RESET (リセット)

自由形式構文	RESET{(E)}{*NOKEY}{*ALL} 名前

コード	演算項目 1	演算項目 2	結果フィールド	標識		
RESET (E)	*NOKEY	*ALL	名前 (変数またはレコード様式)	–	ER	–

RESET 命令は、変数を *INIT 段階の終わりに持っていた値に復元するために使用 します。 この値を**リセット値**と呼びます。 *INZSR サブルーチンがない場合には、リセット値は初期値 (442 ページの『INZ{(初期値)}』 で指定された値またはデフォルトの値のいずれか) と同じです。 *INZSR サブルーチンがある場合には、リセット値は *INZSR サブルーチンが完 了した時にその変数が持っていた値です。

RESET 命令は、レコード様式中のすべてのフィールドをそれらのリセット値に 復元するためにも使用することができます。

*INIT 段階について詳しくは、104 ページの図 11 を参照してください。

注 : サブプロシージャーの内部変数の場合には、 リセット値は、演算が開始される前でなく、そのサブプロシージャーが最初に 呼び出された時の変数の値です。

RESET 例外 (プログラム状況コード 123) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、163 ページの『プログラム例外/エラー』 を参照してください。

詳しくは、「580 ページの『初期化命令』」を参照してください。

変数のリセット

*ALL は任意指定です。 *ALL が指定されて、名前 オペランドに複数オカレンス・データ構造またはテーブル名が入っている場合には、すべてのオカレンスまたはテーブル要素がリセットされて、オカレンス・レベルまたはテーブル指標は 1 に設定されます。

名前 オペランドには、リセットされる変数を指定します。 このオペランドに対する特定な値によって、リセットの処置は次のように決定されます。

単一オカレンス・データ構造
すべてのフィールドが構造内で宣言されている順序にリセットされます。

複数回繰り返しデータ構造
*ALL が指定されていない場合は、現在の オカレンスのすべてのフィールドだけがリセットされます。
*ALL が指定されている場合は、すべての オカレンスのすべてのフィールドがリセットされます。

テーブル名
*ALL が指定されていない場合は、現在の テーブル要素がリセットされます。 *ALL が指定されている場合は、すべてのテーブル要素がリセットされます。

配列名
配列全体がリセットされます。

配列要素 (標識を含む)
指定された要素だけがリセットされます。

レコード様式のリセット

*NOKEY は任意指定です。 *NOKEY が指定されている場合には、キー・フィールドはそれらのリセット値に リセットされません。

*ALL は任意指定です。 *ALL が指定され、*NOKEY が指定されない場合は、レコード様式内のすべてのフィールドがリセットされます。 *ALL が指定されない場合は、そのレコード様式で出力されるフィールドだけが影響を受けます。 *NOKEY が指定されている場合には、*ALL が指定されていても、キー・フィールドがリセットされることはありません。

結果フィールドには、リセットされるレコード様式が入ります。 ワークステーション・ファイルのレコード様式 (ファイル仕様書の 36 から 42 桁目) で、*ALL が指定されていない場合には、用途が出力または入出力共用のフィールドだけが影響を受けます。 この命令によってすべてのフィールドの条件づけ標識が影響を受けます。 RESET 命令がレコード様式名に適用されて、DDS に INDARA が指定されている 場合には、そのレコード様式の標識はリセットされません。

DISK、SEQ、または PRINTER ファイル・レコード様式内のフィールドが影響を受けるのは、そのレコード様式がプログラム内で出力される場合か、プログラム内にサブプロシージャーが定義されている場合だけです。 入力専用フィールドは RESET 命令による影響を受けません。 ただし、*ALL が指定されている場合を除きます。

*ALL が指定されているレコード様式の RESET 命令は、次の場合には有効ではありません。

- フィールドが入力専用として外部で定義されて、レコードが入力用に使用 されていない場合。
- フィールドが出力専用として外部で定義されて、レコードが出力用に使用 されていない場合。
- フィールドが入出力共用として外部で定義されて、レコードが入力または 出力用に使用されていない場合。

注 : 論理ファイルの入力専用フィールドは、実際にそのファイルに書き出されない 場合でも、出力仕様に現れます。 これらのフィールドが入っているレコードに、*ALL が指定されていない CLEAR または RESET が実行されると、 これらのフィールドは、出力仕様に出ているために消去またはリセットされます。

追加の考慮事項

RESET 命令をコーディングする場合には、次のことに留意してください。

- RESET は、基礎となる変数およびインポートされた変数、 またはサブプロシージャーのパラメーターには使用できません。
- RESET 命令では、プログラムが必要とする記憶域の容量が増えます。 変数がリセットされる場合には、必要な記憶域は 2 倍になります。 複数オカレンス・データ構造、テーブル、および配列の場合には、 すべての繰り返しまたは要素のリセット値が保管されることに注意してください。
- プログラムの初期化ルーチン中で RESET が行われると、実行時にエラー・ メッセージが出されます。 *INZSR の処理中にサブルーチンの演算を終了するために GOTO または CABxx が使用されたか、またはエラー処理の結果としてサイクルの別の部分に制御が渡 された場合には、初期化ステップの保管域を初期化する部分には達しませ ん。 この場合には、実行時にプログラムのすべての RESET 命令にエラー・メッセー ジが出されます。
- サブプロシージャー内の大域変数または構造に対する RESET 命令は、次の 場合に有効となります。
  - *INZSR がない場合には常に有効です。
  - *INZSR がある場合には、*INZSR が最低 1 回完了するまで有効にはなりま せん。 それ以降であれば、サイクル・メイン・プロシージャーが活動状態でない場合でも、常に有効になります。
- プログラムの呼び出し時に渡されない *ENTRY PLIST のパラメーターに対して RESET 命令を実行すると、予期せぬ結果が生じることがあります。 そのパラメーターが渡されるものであることが、%PARMS() が戻す値によって示されている場合には、代わりの方法として、パラメーターのように定義された変数に、パラメーター値を保存することができます。

注意!

RESET 値が保存されたときに、サイクル・モジュールで以下のすべてが 真である場合、ポインター非設定エラーが発生します。

- *INZSR がない。
- モジュール内のいずれかにおいて、サイクル・メイン・プロシージャーに対する入力パラメーターが RESET されている。
- サイクル・メイン・プロシージャーが呼び出される前に、サブプロシージャーが呼び出された。

詳しくは、「734 ページの『CLEAR (消去)』」を参照してください。

RESET の例

フィールドに実際に命令が実行される場合を除き、次の例に示す考慮事項は CLEAR 命令にも適用されます。864 ページの図 374 に、「*NOKEY を伴う RESET 命令」の例を示します。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++
EXTFILE   O    E             DISK
DName++++++++++ETDsFrom+++To/L+++IDc.Functions+++++++++++++++++++++++++++++
 * The file EXTFILE contains one record format RECFMT containing
 * the character fields CHAR1 and CHAR2 and the numeric fields
 * NUM1 and NUM2.  It has keyfields CHAR2 and NUM1.
D
D DS1             DS
D  DAY1                   1      8    INZ('MONDAY')
D  DAY2                   9     16    INZ('THURSDAY')
D  JDATE                 17     22
D
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq...
 *
 * The following operation sets DAY1, DAY2, and JDATE to blanks.
C
C                   CLEAR                   DS1
C
 * The following operation will set DAY1, DAY2, and JDATE to their
 * reset values of 'MONDAY', 'THURSDAY', and UDATE respectively.
 * The reset value of UDATE for JDATE is set in the *INZSR.
C
C                   RESET                   DS1
C
 * The following operation will set CHAR1 and CHAR2 to blanks and
 * NUM1 and NUM2 to zero.
C                   CLEAR                   RECFMT
 * The following operation will set CHAR1, CHAR2, NUM1, and
 * NUM2 to their reset values of 'NAME', 'ADDRESS', 1, and 2
 * respectively. These reset values are set in the *INZSR.
 *
C                   RESET                   RECFMT
 * The following operation sets all fields in the record format
 * to blanks, except the key fields CHAR2 and NUM1.
 *
C     *NOKEY        RESET     *ALL          RECFMT
C                   RETURN
C
C     *INZSR        BEGSR
C                   MOVEL     UDATE         JDATE
C                   MOVEL     'NAME    '    CHAR1
C                   MOVEL     'ADDRESS '    CHAR2
C                   Z-ADD     1             NUM1
C                   Z-ADD     2             NUM2
C                   ENDSR
ORCDNAME+++D...N01N02N03EXCNAM++++.....................................
O.............N01N02N03FIELD+++++++++.B.................................
ORECFMT    T
O                       CHAR1
O                       CHAR2
O                       NUM1
O                       NUM2
```

図 *374. *NOKEY を伴う RESET 命令*

```
     A          R RECFMT
     A            CHAR1         10A
     A            CHAR2         10A
     A            NUM1           5P 0
     A            NUM2           7S 2
```

図 *375. EXTFILE* ファイルの *DDS*

866 ページの図 376 に、2 つの外部記述ファイル RESETIB と RESETON を使用するプログラムのソース・リストの抜粋を示します。 それぞれには 2 つのレコード様式があって、それぞれのレコード様式には入力フィールド FLDIN、出力フィールド FLDOUT、および入出力可能なフィールド FLDBOTH が入っています。この DDS を 867 ページの図 377 および 867 ページの図 378 に示します。

RESETIB は入出力共用ファイルとして定義されているので、入出力可能として 定義されている RECBOTH のフィールドは、入力と出力の両方の指定に使用する ことができます。 逆に、RECIN のフィールドは入力の指定にしか使用することができません。

```
     1  * The file RESETIB contains 2 record formats RECIN and RECBOTH.
     2 FRESETIB   CF   E             WORKSTN
     3  * The file RESETON contains 2 record formats RECOUT and RECNONE.
     4 FRESETON   O    E             WORKSTN
     5
     6=IRECIN
     7=I                             A    1    1  *IN02
     8=I                             A    2   11  FLDIN
     9=I                             A   12   21  FLDBOTH
    10=IRECBOTH
    11=I                             A    1    1  *IN04
    12=I                             A    2   11  FLDIN
    13=I                             A   12   21  FLDBOTH
    14 C                   WRITE     RECOUT
    15 C                   WRITE     RECBOTH
    16 C                   READ      RECIN                              ----99
    17 C                   READ      RECBOTH                            ----99
    18
    19  * RESET without factor 2 means to reset only those fields which
    20  * appear on the output specifications for the record format.
    21  * Since only RECOUT and RECBOTH have write operations, the
    22  * RESET operations for RECNONE and RECIN will have no effect.
    23  * The RESET operations for RECOUT and RECBOTH will reset fields
    24  * FLDOUT and FLDBOTH.  FLDIN will not be affected.
    25 C                   RESET                RECNONE
    26 C                   RESET                RECIN
    27 C                   RESET                RECOUT
    28 C                   RESET                RECBOTH
    29
    30  * RESET with *ALL in factor 2 means to reset all fields.  Note
    31  * that this can only be done when all fields are used in at least
    32  * one of the ways they are defined (for example, an output-capable
    33  * field must be used for output by the record format)
    34  * Since RECNONE does not have either input or output operations,
    35  * the RESET *ALL for RECNONE will fail at compile time.
    36  * Since RECIN does not have any output operations, RESET *ALL RECIN
    37  * will fail because FLDOUT is not output.
    38  * Since RECOUT does not have any input operations, and is not defined
    39  * as input capable on the file specification, RESET *ALL RECOUT
    40  * will fail because FLDIN is not input.
    41  * The RESET *ALL for RECBOTH will reset all fields: FLDIN, FLDOUT
    42  * and FLDBOTH.
    43 C                   RESET     *ALL       RECNONE
    44 C                   RESET     *ALL       RECIN
    45 C                   RESET     *ALL       RECOUT
    46 C                   RESET     *ALL       RECBOTH
    47
    48 C                   SETON                                    LR----
    49=ORECBOTH
    50=O                       *IN14           1A CHAR        1
    51=O                       FLDOUT         11A CHAR       10
    52=O                       FLDBOTH        21A CHAR       10
    53=ORECOUT
    54=O                       *IN13           1A CHAR        1
    55=O                       FLDOUT         11A CHAR       10
    56=O                       FLDBOTH        21A CHAR       10
```

図 *376. *ALL による RESET* ― ソース・リストの抜粋

ソースのコンパイル時にはいくつかのエラーが示されます。 RECNONE と RECIN の両方には出力フィールドなしと示されます。 RESET *ALL は、RECBOTH レコードを除くすべてのレコードに使用できません。これは、すべてのフィールドが入力または出力のいずれかの指定に現れる 唯一のレコード様式であるためです。

```
     A          R RECIN                       CF02(02)
     A            FLDIN         10A  I  2  2
     A            FLDOUT        10A  O  3  2
     A  12        FLDBOTH       10A  B  4  2
     A          R RECBOTH                     CF04(04)
     A            FLDIN         10A  I  2  2
     A            FLDOUT        10A  O  3  2
     A  14        FLDBOTH       10A  B  4  2
```

図 *377. RESETIB の DDS*

```
     A          R RECNONE                     CF01(01)
     A            FLDIN         10A  I  2  2
     A            FLDOUT        10A  O  3  2
     A  11        FLDBOTH       10A  B  4  2
     A          R RECOUT                      CF03(03)
     A            FLDIN         10A  I  2  2
     A            FLDOUT        10A  O  3  2
     A  13        FLDBOTH       10A  B  4  2
```

図 *378. RESETON の DDS*

RETURN (呼び出し元への戻し)

自由形式構文	RETURN{(HMR)} 式

コード	演算項目 1	拡張演算項目 2
RETURN (H M/R)		式

RETURN 命令は呼び出し元へ戻します。 呼び出し元に値が戻されると、式 オペランドに戻り値が示されます。

RETURN 命令の結果として行われる処置は、その命令がサイクル・メイン・プロシージャーとサブプロシージャーのどちらで行われたかによって異なります。 サイクル・メイン・プロシージャーが戻した場合、以下のことが行われます。

1. 停止標識が検査されます。 停止標識がオンになっている場合には、プロシージャーは異常終了します (開かれているファイルはすべて閉じられ、呼び出し元ルーチンにプロシージャーが異常終了したことを示すエラー戻りコードが設定され、呼び出し元ルーチンに制御が戻されます)。
2. 停止標識がオンになっていない場合には、LR 標識が検査されます。 LR がオンになっている場合、プログラムは正常に終了します (ロックされたデータ域構造、配列、およびテーブルが書き出されて、外部標識がリセットされます)。
3. 停止標識がオンでなく LR がオンもなっていない場合には、プロシージャ ーは呼び出し元ルーチンに戻します。 データは次回のプロシージャーの実行に備えて保存されます。 ファイルおよびデータ域は書き出されません。 *NEW 活動化グループでの実行が RETURN の操作に与える影響について詳しくは、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」の呼び出し元のプログラムおよびプロシージャーに関する章を参照してください。

サブプロシージャーが戻すと、戻り値 (呼び出されたプログラムまたはプロシ ージャーのプロトタイプで指定された場合) が呼び出し元に渡されます。 自動ファイルがクローズされます。 自動的には何も行われません。 静的ファイルまたはグローバル・ファイルおよびデータ域はすべて手操作でクローズしなければなりません。 LR などの標識を設定できますが、 これでプログラムの終了を行うことはできません。

命令拡張 H、M、および R の使用法については、608 ページの『数値演算の精度の規則』を参照してください。

値を戻すサブプロシージャーでは、RETURN 命令はそのサブプロシージャー内で コーディングしなければなりません。 実際に戻された値は EVAL 式の左側と同じ働きをし、RETURN 命令の拡張演算項目 2 は右側と同じ働きをします。 配列を戻すことができるのは、プロトタイプが戻り値を配列として定義してい る場合だけです。

注意!

サブプロシージャーが値を戻した場合には、プロシージャーの終わりに達する 前に RETURN 命令が実行されていることを確認してください。 サブプロシージャーが RETURN 命令を 見付けないで終了している場合には、呼び出し元に例外が通知されます。

パフォーマンスのヒント

プロトタイプに RTNPARM キーワードを指定すると、大きな値を戻す場合に、パフォーマンスが著しく向上する可能性があります。 詳しくは、477 ページの『RTNPARM』を参照してください。

詳しくは、「563 ページの『呼び出し命令』」を参照してください。

```
     * This is the prototype for subprocedure RETNONE. Since the
     * prototype specification does not have a data type, this
     * subprocedure does not return a value.
    D RetNone         PR
     * This is the prototype for subprocedure RETFLD. Since the
     * prototype specification has the type 5P 2, this subprocedure
     * returns a packed value with 5 digits and 2 decimals.
     * The subprocedure has a 5-digit integer parameter, PARM,
     * passed by reference.
    D RetFld          PR             5P 2
    D   Parm                         5I 0
     * This is the prototype for subprocedure RETARR. The data
     * type entries for the prototype specification show that
     * this subprocedure returns a date array with 3 elements.
     * The dates are in *YMD/ format.
    D RetArr          PR              D   DIM(3) DATFMT(*YMD/)
     * This procedure (P) specification indicates the beginning of
     * subprocedure RETNONE. The data specification (D) specification
     * immediately following is the procedure-interface
     * specification for this subprocedure. Note that the
     * procedure interface is the same as the prototype except for
     * the definition type (PI vs PR).
    P RetNone         B
    D RetNone         PI
     * RetNone does not return a value, so the RETURN
     * operation does not have factor 2 specified.
    C                   RETURN
    P RetNone         E
     * The following 3 specifications contain the beginning of
     * the subprocedure RETFLD as well as its procedure interface.
    P RetFld          B
    D RetFld          PI             5P 2
    D   Parm                         5I 0
    D Fld             S             12S 1 INZ(13.8)
     * RetFld returns a numeric value.  The following RETURN
     * operations show returning a literal, an expression and a
     * variable. Note that the variable is not exactly the same
     * format or length as the actual return value.
    C                   RETURN       7
    C                   RETURN       Parm * 15
    C                   RETURN       Fld
    P RetFld          E
```

図 *379. RETURN* 命令の例

```
     * The following 3 specifications contain the beginning of the
     * subprocedure RETARR as well as its procedure interface.
    P RetArr          B
    D RetArr          PI              D   DIM(3)
    D SmallArr        S               D   DIM(2) DATFMT(*ISO)
    D BigArr          S               D   DIM(4) DATFMT(*USA)
     * RetArr returns a date array.  Note that the date
     * format of the value specified on the RETURN operation
     * does not have to be the same as the defined return
     * value.
     * The following RETURN operation specifies a literal.
     * The caller receives an array with the value of the
     * literal in every element of the array.
    C                   RETURN        D'1995-06-27'
     * The following return operation returns an array
     * with a smaller dimension than the actual return value.
     * In this case, the third element would be set to the
     * default value for the array.
    C                   RETURN        SmallArr
     * The following return operation returns an array
     * with a larger dimension than the actual return
     * value.  In this case, the fourth element of BigArr
     * would be ignored.
    C                   RETURN        BigArr
    P RetArr          E
```

ROLBK (ロールバック)

自由形式構文	ROLBK {(E) }

コード	演算項目 1	演算項目 2	結果フィールド	標識		
ROLBK (E)				_	ER	_

ROLBK 命令では次のことが行われます。

- 前の COMMIT または ROLBK 命令以後 (前に COMMIT または ROLBK 命令が ない場合にはコミットメント制御のもとの操作の始め以後) の、出力操作で 指定されているユーザー・ファイルの変更がすべて除去されます。
- コミットメント制御のもとに置かれているファイルのレコード・ロックが すべて解除されます。
- ファイルが前回の COMMIT 命令時 (または前に COMMIT 命令がない場合に はファイルの OPEN 時) の位置に再度位置決めされます。

コミットメント制御は、CL コマンド STRCMTCTL が実行された時点で開始され ます。 詳しくは、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」の『コミットメント制御』の章を参照してください。

ファイルの変更およびレコード・ロックの解除は、変更が ROLBK 命令を出 しているプログラムによって要求されたか、同じ活動化グループまたはジョブ の別のプログラムによって要求されたかに関係なく、その活動化グループまた はジョブのコミットメント制御のもとにあるすべてのファイルに適用されます。
ROLBK 命令を出しているプログラムは、ファイルをコミットメント制御のもと に置く必要はありません。例えば、プログラム A がプログラム B とプログラム C を 呼び出していると仮定します。 プログラム B にはコミットメント制御の下にあるファイルがあり、プログラム C にはありません。 プログラム C の ROLBK 命令は、この場合でもプログラム B によって変更され たファイルに影響を与えます。

ROLBK 例外 (プログラム状況コード 802 から 805) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、163 ページの『プログラム例外/エラー』を参照してください。

システムでのロールバック機能の実行方法について詳しくは、「バックアップおよび回復」*(SD88-5008)*を参照してください。

詳しくは、「576 ページの『ファイル操作』」を参照してください。

SCAN (ストリングの走査)

自由形式構文	(許可されていない - %SCAN 組み込み関数を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
SCAN (E)	比較ストリング:長さ	基本ストリング: 開始	左端の位置	_	ER	FD

SCAN 命令では、演算項目 2 に入っているストリング (基本ストリング) を走査して、演算項目 1 に入っているサブストリング (比較ストリング) を見つけます。 走査は、演算項目 2 に入っている指定位置から開始されて、演算項目 1 に指定されている比較ストリングの長さだけ続けられます。 比較ストリングと基本ストリングは両方とも同じタイプ (つまり、両方とも文字、両方ともグラフィック、または両方とも UCS-2) である必要があります。

演算項目 1 には、比較ストリングか、比較ストリングにコロンを付け長さを 指定したものを入れなければなりません。 演算項目 1 の比較ストリング部分には、フィールド名、配列要素、名前のつい た定数、データ構造名、リテラル、またはテーブル名のいずれかを入れる ことができます。 長さの部分は小数点以下の桁数がゼロの数値でなければならず、名前のついた 定数、配列要素、フィールド名、リテラル、またはテーブル名のいずれかを 入れることができます。 長さが指定されていない場合には、比較ストリング の長さになります。

演算項目 2 には、基本ストリングか、 または基本ストリングにコロンを付け SCAN の開始位置を指定したものを入れなければなりません。 演算項目 2 の基本ストリング部分には、フィールド名、配列要素、名前のつい た定数、データ構造名、リテラル、またはテーブル名のいずれかを入れる ことができます。 演算項目 2 の開始位置部分は、小数点以下の桁数がゼロの数値でなければなら ず、名前のついた定数、配列要素、フィールド名、リテラル、またはテー ブル名とすることができます。 図形または UCS-2 ストリングが使用された場合、開始位置と長さは 2 バイト単位で測定されます。 開始位置が指定されていない場合には、値 1 が使用されます。

結果フィールドには、基本ストリング中の比較ストリングの左端の位置の数 値 (見付かった場合) が入ります。 これは小数点以下の桁数がゼロの数値でな ければならず、フィールド名、配列要素、配列名、またはテーブル名のいずれ かを入れることができます。 ストリングが見付からない場合には、結果フィールドは 0 に設定されます。 結果フィールドに配列が入っている場合には、比較ストリングのそれぞれのオカレンス が配列に入れられて、左端のオカレンスが要素 1 に入れられます。 右端のオカレンスが入っている要素の後の配列要素はすべてゼロになります。 結果の配列は、演算項目 2 に指定された基本ストリングのフィールド長と同じ 長さでなければなりません。

注 :

1. ストリングには 1 桁目から指標が付きます。
2. 開始位置が 1 より大きい場合には、結果フィールドに、開始位置からでな く、基本ストリングの先頭からの比較ストリングの相対位置が入ります。
3. 演算項目 1、演算項目 2、または結果フィールドに形象定数を使用することは できません。
4. データ構造内に演算項目 1 と結果フィールドまたは演算項目 2 と結果フィ ールドのオーバーラップがあってはなりません。

SCAN 例外 (プログラム状況コード 100) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。 エラーは、開始位置が演算項目 2 の長さより大きい場合、または演算項目 1 の 値が大き過ぎる場合に起こります。エラー処理について詳しくは、163 ページの『プログラム例外/エラー』を参照してください。

走査中のストリングが見付かった場合にオンに設定される標識を 75 から 76 桁目に 指定することができます。 この情報は %FOUND 組み込み関数からも入手することが できます。この関数は、等しい項目が見付かった場合に '1' を戻します。

SCAN は、演算項目 2 の左端文字 (開始位置によって指定) から開始されて、 演算項目 2 の文字を演算項目 1 の文字と比較しながら、左から右へと 1 文字 ずつ続けられます。 結果フィールドが配列以外の場合には、SCAN 命令で比較ス トリングの最初のオカレンスだけが見付けられます。 最初のオカレンスを超えて走査

を続行するためには、前の SCAN 命令の結果フ ィールドを使用して、次の SCAN の開始位置を計算してください。 結果フィールドが数値配列の場合には、配列内の要素と同数のオカレンスが記 録されます。 オカレンスが見付からない場合には、結果フィールドはゼロに設定されます。結果フィールドが配列の場合には、そのすべての要素がゼロに 設定されます。

比較ストリングに指定された先行ブランク、後書きブランク、または組み込み ブランクは SCAN 命令に含まれます。

SCAN 命令では大文字と小文字が区別されます。 大文字で指定された基本ス トリングの中から小文字で指定された比較ストリングを見付けることはできま せん。

詳しくは、589 ページの『ストリング命令』 を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 * The SCAN operation finds the substring 'ABC' starting in
 * position 3 in factor 2;  3 is placed in the result field.
 * Indicator 90 is set on because the string is found.  Because
 * no starting position is specified, the default of 1 is used.
C     'ABC'         SCAN      'XCABCD'      RESULT                  90
 *
 * This SCAN operation scans the string in factor 2 for an
 * occurrence of the string in factor 1 starting at position 3.
 * The 'Y' in position 1 of the base string is ignored because
 * the scan operation starts from position 3.
 * The operation places the values 5 and 6 in the first and
 * second elements of the array.  Indicator 90 is set on.
C
C                   MOVE      'YARRYY'      FIELD1            6
C                   MOVE      'Y'           FIELD2            1
C     FIELD2        SCAN      FIELD1:3      ARRAY                   90
 *
 * This SCAN operation scans the string in factor 2, starting
 * at position 2, for an occurrence of the string in factor 1
 * for a length of 4.  Because 'TOOL' is not found in FIELD1,
 * INT is set to zero and indicator 90 is set off.
C
C                   MOVE      'TESTING'     FIELD1            7
C                   Z-ADD     2             X                 1 0
C                   MOVEL     'TOOL'        FIELD2            5
C     FIELD2:4      SCAN      FIELD1:X      INT90            20
C
 *
 * The SCAN operation is searching for a name.  When the name
 * is found, %FOUND returns '1' so HandleLine is called.
C     SrchName      SCAN      Line
C                   IF        %FOUND
C                   EXSR      HandleLine
C                   ENDIF
```

図 *380. SCAN* 命令

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName++++++++++ETDsFrom+++To/L+++IDc.Functions++++++++++++++++++++++++++
 *
 *       A Graphic SCAN example
 *
 *       Value of Graffld is graphic 'AACCBBGG'.
 *       Value of Number after the scan is 3 as the 3rd graphic
 *       character matches the value in factor 1
D Graffld       S                4G   inz(G'oAACCBBGGi')
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
 * The SCAN operation scans the graphic string in factor 2 for
 * an occurrence of the graphic literal in factor 1. As this is a
 * graphic operation, the SCAN will operate on 2 bytes at a time
C
C     G'oBBi'       SCAN      Graffld:2     Number            5 0    90
C
```

図 *381.* 図形を使用する *SCAN* 命令

SELECT (選択グループの始め)

自由形式構文	SELECT

コード	演算項目 1	演算項目 2	結果フィールド	標識		
SELECT						

選択グループは、一連の複数の代替命令の中から 1 つを条件付きで処理しま す。 グループは次のものから構成されます。

- SELECT ステートメント
- ゼロまたはそれ以上の WHENxx または WHEN グループ
- 任意指定の OTHER グループ
- ENDSL または END ステートメント

SELECT 命令の後で、最初に満たされた WHENxx 条件の後のステートメント に制御が渡されます。 その後で、次の WHENxx 命令までのすべてのステートメントが実行されます。 ENDSL ステートメントに (WHENxx が 1 つしか実行されない場合) 制御が渡され ます。 WHENxx 条件が満たされないで、OTHER 命令が指定されている場合には、OTHER 命令の後のステートメントに制御が渡されます。 WHENxx 条件が満たされないで、OTHER 命令が指定されていない場合には、選択 グループの ENDSL 命令の後のステートメントに制御が渡されます。

SELECT 命令では、条件付け標識を使用することができます。 条件付け標識が満たされない場合には、選択グループの ENDSL 命令の後のステ ートメントに即時に制御が渡されます。 WHENxx、WHEN、OTHER、および ENDSL 命令個々では、条件付け標識を使用する ことはできません。

選択グループは演算の中の任意の場所に指定することができます。 IF、DO、またはその他の選択グループの中でネストすることもできます。 IF および DO グループは選択グループの中でネストすることができます。

SELECT 命令が選択グループの中に指定されている場合には、 WHENxx および OTHER 命令は、ENDSL が指定されるまでその新しい選択グループに適用されます。

詳しくは、「591 ページの『構造化プログラミング命令』」を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 * In the following example, if X equals 1, do the operations in
 * sequence 1 (note that no END operation is needed before the
 * next WHENxx); if X does NOT equal 1, and if Y=2 and X<10, do the
 * operations in sequence 2.  If neither condition is true, do
 * the operations in sequence 3.
 *
C                   SELECT
C                   WHEN      X = 1
C                   Z-ADD     A             B
C                   MOVE      C             D
 *  Sequence 1
C                   :
C                   WHEN      ((Y = 2) AND (X < 10))
 *  Sequence 2
C                   :
C                   OTHER
 *  Sequence 3
C                   :
C                   ENDSL
 *
 * The following example shows a select group with conditioning
 * indicators.  After the CHAIN operation, if indicator 10 is on,
 * then control passes to the ADD operation.  If indicator 10 is
 * off, then the select group is processed.
 *
C     KEY           CHAIN     FILE                            10
C  N10              SELECT
C                   WHEN      X = 1
 *  Sequence 1
C                   :
C                   WHEN      Y = 2
 *  Sequence 2
C                   :
C                   ENDSL
C                   ADD       1             N
```

図 *382. SELECT* 命令

SETGT (より大きい設定)

自由形式構文	SETGT{(EHMR) } 検索引数 名前

コード	演算項目 1	演算項目 2	結果フィールド	標識		
SETGT (E)	検索引数	名前 (ファイルまたはレコード様式)		NR	ER	_

SETGT 命令は、演算項目 1 で指定されたキーまたは相対レコード番号より大きいキーまたは相対レコード番号を持つ次のレコードにファイルを位置付けます。 ファイルは全手順ファイルでなければなりません。

検索引数 (検索引数) は、レコードの検索に使用するキーまたは相対レコード番号でなければなりません。 アクセスがキーによる場合には、検索引数 はフィールド名、名前の付いた固定情報、 形象定数、またはリテラルの形式の単一キーにすることができます。 キー・フィールドの検索の例については、730 ページの図 295 を参照してください。

ファイルが外部記述ファイルの場合、検索引数 は KLIST 名、値のリスト、または %KDS の形式の複合キーにすることもできます。 KLIST を 使用して指定されたキーの場合、キー・フィールドはファイル内のキーと同じ CCSID である必要があります。 データ構造内の検索引数の図については、652 ページの『%KDS (データ構造の検索引数)』 の終わりにある例を参照してください。 アクセスが相対レコード番号による場合には、検索引数 に整数のリテラルまたは小数点以下の桁数がゼロの数値フィールドを入れなければなりません。

制御キーワード EXPROPTS(*STRICTKEYS) が値のリストまたは %KDS でキーを指定する規則に与える影響について詳しくは、335 ページの『*STRICTKEYS』を参照してください。

名前 は必須で、ファイル名またはレコード様式名のいずれかでなければなりません。 レコード様式名を使用できるのは、外部記述ファイルの場合だけです。

検索引数 (検索引数) に指定された検索引数より大きいキーまたは相対レコード番号のレコードが見付からない場合にオンに設定される標識を 71 から 72 桁目に指定することができます。 この情報は %FOUND 組み込み関数からも入手することができます。 この関数は、レコードが見付からない場合は '0' を戻し、レコードが見付かった場合は '1' を戻します。

SETGT 例外 (ファイル状況コードが 1000 より大きい) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、146 ページの『ファイル例外/エラー』を参照してください。

SETGT 命令が正常に実行されなかった場合 (レコード不在条件) には、ファ イルは、ファイルの終わりに位置付けられます。

ファイル中の最後のレコードに位置付けるには、SETLL 命令で *END を使用します。

ファイルの位置付けに形象定数を使用することもできます。

注 : 以下の形象定数の説明および使用例では、*LOVAL および *HIVAL はファイ ルの実際のキーとしては使用されないものとしています。

複合キーを持つファイルで使用する場合には、形象定数は、キーのそれぞれ のフィールドに形象定数の値が入っているものとして処理されます。 ほとんどの場合に、*LOVAL は、最初の読み取りでキーの値が一番小さいレコード が検索されるようにファイルを位置付けます。 ほとんどの場合に *HIVAL は、 READ でファイルの終わりの指示が受け取られるようにファイルを位置付けます。 以後の READP では、ファイルの最後のレコードが検索されます。 しかし、*LOVAL および *HIVAL を使用 する場合には次の事例に注意してください。

- 降順のキーを持つ外部記述ファイルでは、*HIVAL は最初の読み取り操作でフ ァイルの最初のレコード (一番大きいキーを持つレコード) が検索され、*LOVAL は READP 命令でファイルの最後のレコード (一番小さいキーを持つレコード) が検索されるようにファイルを位置付けます。
- *LOVAL または *HIVAL による SETGT 命令の後でレコードの追加または キー・フィールドの変更が行われている場合には、以後そのファイルを、最低または最高のキーを持つレコードに位置付けることはできません。数値キーの *LOVAL はキーの値 '99...9D' を表し、*HIVAL はキーの値 '99...9F' を表します。 キーが浮動数値の場合、*LOVAL と *HIVAL の定義は異なります。207 ページの『表意定数』を参照してください。 ファイル仕様書でプログラム記述ファイルにパック 10 進数の指定があって、実 際のファイルのキー・フィールドに文字データが入っている場合には、レコー ドが *LOVAL より小さいかまたは *HIVAL より大きいキーを持つことがありま す。 キー・フィールドに符号のない 2 進数データが入っている場合には、*LOVAL は 最低のキーでないことがあります。

*LOVAL または *HIVAL が日付または時刻データ・タイプのキー・フィール ドで使用される場合には、その値は使用される日付時刻の形式によって異なり ます。 これらの値について詳しくは、246 ページの『データ・タイプおよびデータ形式』を参照してください。

SETGT 命令の後では、ファイルは、そのキーまたは相対レコード番号が検索引数 に指定された検索引数より大きい最初のレコードの直前になるように位置付けられます。 ユーザーはこのレコードを、ファイルを読み取って検索しま す。 しかし、ファイルを読み取る前に、 レコードが別のジョブまたはユーザー・ジョブの別のファイルによってファ イルから削除されている場合があります。 このような場合には、必要なレコー ドを取り出すことができません。 ファイルの予期しない変更を防止する方法については、IBM i Information Center (URL http://www.ibm.com/systems/i/infocenter/)「プログラミング」のトピックを参照してください。

ヌル値可能フィールドおよびキーを持つレコードの処理について詳しくは、286 ページの『データベースのヌル値サポート』を参照してください。

詳しくは、「576 ページの『ファイル操作』」を参照してください。

注 : 命令コード拡張 H、M、および R は、検索引数がリストまたは %KDS() である場合にのみ使用できます。579 ページの『ファイル命令のキー』 および 558 ページの『精度の確認』 を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *  This example shows how to position the file so READ will read
 *  the next record. The search argument, KEY, specified for the
 *  SETGT operation has a value of 98; therefore, SETGT positions
 *  the file before the first record of file format FILEA that
 *  has a key field value greater than 98.  The file is positioned
 *  before the first record with a key value of 100.  The READ
 *  operation reads the record that has a value of 100 in its key
 *  field.
C
C     KEY           SETGT     FILEA
C                   READ      FILEA                                  64
 *
 *  This example shows how to read the last record of a group of
 *  records with the same key value and format from a program
 *  described file.  The search argument, KEY, specified for the
 *  SETGT operation positions the file before the first record of
 *  file FILEB that has a key field value greater than 70.
 *  The file is positioned before the first record with a key
 *  value  of 80.  The READP operation reads the last record that
 *  has a value of 70 in its key field.
C
C     KEY           SETGT     FILEB
C                   READP     FILEB                                  64
```

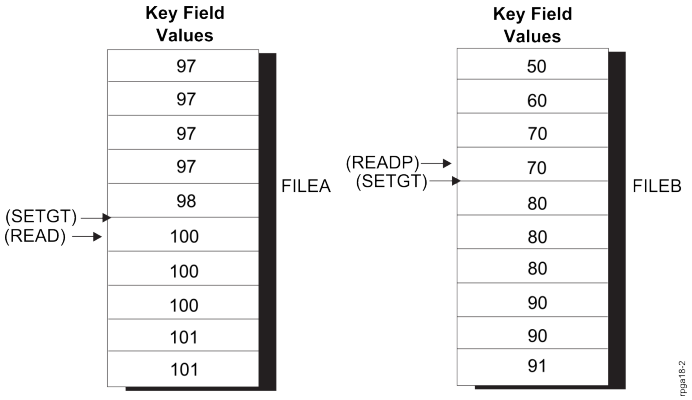
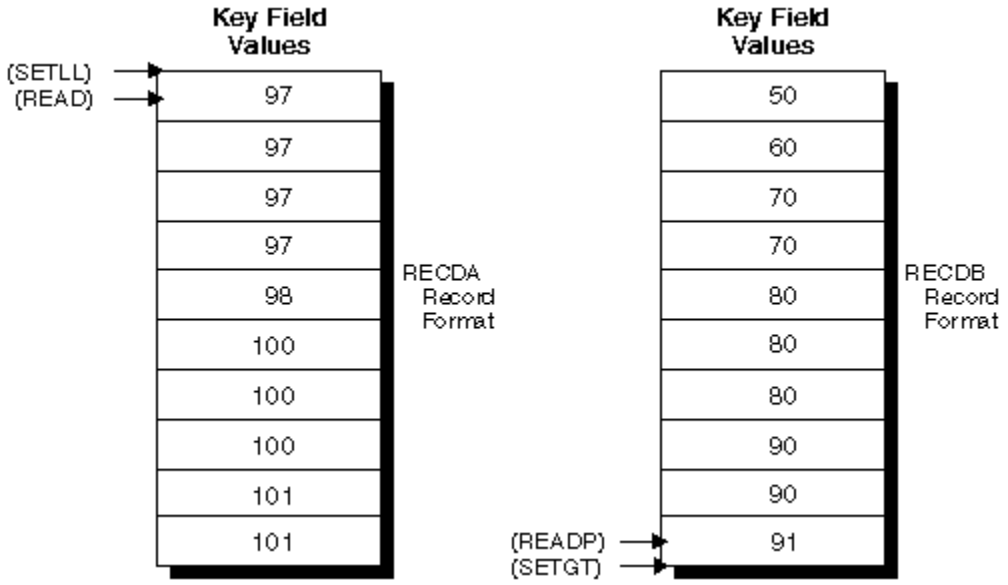


図 *383. SETGT* 命令

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  This example shows the use of *LOVAL.  The SETLL operation
 *  positions the file before the first record of a file in
 *  ascending order.  The READ operation reads the first record
 *  (key value 97).
C
C     *LOVAL        SETLL     RECDA
C                   READ      RECDA                                  64
C
 *  This example shows the use of *HIVAL.  The SETGT operation
 *  positions the file after the last record of a file in ascending
 *  order. The READP operation reads the last record (key value 91).
C
C     *HIVAL        SETGT     RECDB
C                   READP     RECDB                                  64
```



SETLL (下限の設定)

自由形式構文	SETLL{(EHMR) } 検索引数 名前

コード	演算項目 1	演算項目 2	結果フィールド	標識		
SETLL (E)	検索引数	名前 (ファイルまたはレコード様式)		NR	ER	EQ

SETLL 命令は、ファイルをキーまたは相対レコード番号が検索引数 に指定された検索引数 (キーまたは相対レコード番号) オペランドに等しいかそれより大きい次のレコードに位置付けます。 ファイルは全手順ファイルでなければなりません。

検索引数 (検索引数) は、レコードの検索に使用するキーまたは相対レコード番号でなければなりません。 アクセスがキーによる場合には、検索引数 はフィールド名、名前の付いた固定情報、 形象定数、またはリテラルの形式の単一キーにすることができます。 キー・フィールドの検索の例については、730 ページの図 295 を参照してください。

ファイルが外部記述ファイルの場合、検索引数 は KLIST 名、値のリスト、または %KDS の形式の複合キーにすることもできます。 KLIST を 使用して指定されたキーの場合、キー・フィールドはファイル内のキーと同じ CCSID である必要があります。 データ構造内の検索引数の図については、652 ページの『%KDS (データ構造の検索引数)』 の終わりにある例を参照してください。

制御キーワード EXPROPTS(*STRICTKEYS) が値のリストまたは %KDS でキーを指定する規則に与える影響について詳しくは、335 ページの『*STRICTKEYS』を参照してください。

アクセスが相対レコード番号による場合には、検索引数 に整数のリテラルまたは小数点以下の桁数がゼロの数値フィールドを入れなければなりません。

名前 オペランドは必須で、ファイル名またはレコード 様式名のいずれかが可能です。 レコード様式名を使用できるのは、外部記述ファイルの場合だけです。

結果の標識はこの命令の状況を反映します。 検索引数がファイルの一番大きいキーまたは相対レコード番号より大きい場合にオン に設定される標識を 71 から 72 桁目に指定することができます。 この情報は %FOUND 組み込み関数 からも入手することができます。この関数は、レコードが見付からない場合は '0' を 戻し、レコードが見付かった場合は '1' を戻します。

SETLL 例外 (ファイル状況コードが 1000 より大きい) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、146 ページの『ファイル例外/エラー』を参照してください。

そのキーまたは相対レコード番号が検索引数に等しいレコードが存在する場合にオン に設定される標識を 75-76 桁目に指定することができます。 この情報は %EQUAL 組み込み関数からも入手することが できます。この関数は、等しい項目が見付かった場合に '1' を戻します。

75 から 76 桁目の標識または %EQUAL を指定して SETLL を使用する場合、指定されたキーとファイル内の実際のキーの比較は通常、データ管理機能により実行されます。 データ管理機能でこれを実行できない場合は、16 進数照合シーケンスを使用して比較が実行されます。 この場合、予想通りの結果が得られないことがあります。 詳しくは、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」のセクション『キー付きファイルの使用による予期しない結果』を参照してください。

名前 が下限に設定されているファイル名である場合、ファイルは、指定された検索引数 (検索引数) に等しいかそれより大きいキーまたは相対レコード番号の最初のレコードに位置付けられます。

名前 に下限に設定されているレコード様式名が入っている場合、ファイルは、指定された検索引数 (検索引数) に等しいかそれより大きいキーまたは相対レコード番号の指定されたタイプの最初のレコードに位置付けられます。

特殊値 *START および *END を検索引数 として指定できます。*START はファイルの始めに位置指定し、*END はファイルの終わりに位置指定します。 どちらの位置付けも、キー付きファイルに使用される照合順序とは関係なく行なわれ、 ヌル値キー・フィールドにも関係なく行われます。 検索引数 に *START または *END のいずれかを指定する場合には、次のことに注意してください。

- ファイルの名前は、名前 オペランドとして指定しなければなりません。
- エラー標識 (73 から 74 桁目) または 'E' 拡張を 指定することができます。

ファイルの位置付けに形象定数を使用することもできます。 ただし、*LOVAL または *HIVAL を使用してもファイルの先頭または最終レコードに 正確にファイルが位置付けされないシチュエーションがいくつかあります。 ファイル中の先頭または最終レコードに位置付けしたい場合は、*START または *END を 使用するほうが確実です。

注 : 以下の形象定数の説明および使用例では、*LOVAL および *HIVAL はファイ ルの実際のキーとしては使用されないものとしています。

複合キーを持つファイルで使用する場合には、形象定数は、キーのそれぞれ のフィールドに形象定数の値が入っているものとして処理されます。 SETLL で *LOVAL を使用すると、 ヌル可能キー・フィールドのあるレコードがファイルに含まれていない場合は、最初の読み取り操作で 取り出されるのが一番小さいキーを持つレコードになるようにファイルが位置付けられます。 ほとんどの場合 (重複キーを使用できない場合) に、*HIVAL では、ファイルは、READP でファ イルの最後のレコードが検索されるか、または READ でファイルの終わりの指示が 受け取られるように位置付けられます。 しかし、*LOVAL および *HIVAL を使用する場合には次の事例に注意してください。

- 降順のキーを持つ外部記述ファイルでは、*HIVAL は最初の読み取り操作で取り出されるのが最も大きいキーを持つレコードになるようにファイルを位置付け、 *LOVAL は READP 命令で取り出されるのが最も小さいキーを持つレコードになるようにファイルを位置付けます。
- *LOVAL または *HIVAL のどちらかを用いた SETLL 命令の後でレコードを 追加したりキー・フィールドを変更した場合には、ファイルはもはや最低また は最高のキーに位置付けることができません。

- 数値キーの *LOVAL はキーの値 '99...9D' を表し、*HIVAL は キーの値 '99...9F' を表します。 キーが浮動数値の場合、*HIVAL と *LOVAL の定義は異なります。207 ページの『表意定数』を参照してください。ファイル仕様書でプログラム記述ファイルにパック 10 進数の指定があって、実 際のファイルのキー・フィールドに文字データが入っている場合には、レコー ドが *LOVAL より小さいかまたは *HIVAL より大きいキーを持つことがありま す。 キー・フィールドに符号のない 2 進数データが入っている場合には、*LOVAL は 最低のキーでないことがあります。
- *LOVAL または *HIVAL が日付または時刻データ・タイプのキー・フィール ドで使用される場合には、その値は使用される日付時刻の形式によって異なり ます。 これらの値について詳しくは、246 ページの『データ・タイプおよびデータ形式』を参照してください。
- ヌル可能キー・フィールドがある場合に *LOVAL や *HIVAL などの形象定数が使用され ると、ヌル値キーを持つレコードは検索で検出されません。

875 ページの図 383 (パート 3 / 4)に、SETGT 命令での形象定数の使用法を示します。 形象定数は、SETLL 命令で も同様に使用されます。

SETLL 命令を使用する場合には、次のことに留意してください。

- SETLL 命令が正常に実行されなかった場合 (レコード不在条件) には、 ファイルは、ファイルの終わりに位置付けられます。
- SETLL で処理されるファイルでファイルの終わりに達した場合には、別の SETLL を出して再度ファイルの位置決めを行うことができます。
- SETLL 命令でファイルが正常にレコードに位置付けられた後では、このレ コードはファイルを読み取って検索します。 しかし、ファイルを読み取る前に、 レコードが別のジョブまたはユーザー・ジョブの別のファイルによってファ イルから削除されている場合があります。 このような場合には、必要なレコードを取り出すことができません。 %EQUAL 組み込み関数もオンに設定されているか、または 75 桁目と 76 桁目の 結果の標識がオンに設定されて、一致するレコードが見付かったことが示されている場合でも、 そのレコードを取り出せないことがあります。 ファイルの予期しない変更を防止する方法については、IBM i Information Center (URL http://www.ibm.com/systems/i/infocenter/)「プログラミング」のトピックを参照してください。
- SETLL でシステムがデータ・レコードにアクセスすることはありません。 キーが実際に存在するかどうかを確認したいだけの場合には、多くの場合に、CHAIN 命令 を使用するよりも等しい標識 (75 から 76 桁目) または %EQUAL 組み込み関数を 指定した SETLL の方がパフォーマンスが向上します。管理レベルでキーの比較を 実行することはできません。 分散したキーを持つ複数形式論理ファイルの ような特殊な場合には、CHAIN の方が SETLL より速い場合もあります。

ヌル値可能フィールドおよびキーを持つレコードの処理について詳しくは、286 ページの『データベースのヌル値サポート』を参照してください。

詳しくは、「576 ページの『ファイル操作』」を参照してください。

注 : 命令コード拡張 H、M、および R は、検索引数がリストまたは %KDS() である場合にのみ使用できます。579 ページの『ファイル命令のキー』 および 558 ページの『精度の確認』 を参照してください。

次の例では、ファイル ORDFIL に注文レコードが入っています。 キー・フィールドは注文番号 (ORDER) フィールドです。 それぞれの注文には複数のレコードがあります。 演算仕様書では、ORDFIL は次の ようになっています。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  All the 101 records in ORDFIL are to be printed.  The value 101
 *  has previously been placed in ORDER.  The SETLL operation
 *  positions the file at the first record with the key value 101
 *  and %EQUAL will return '1'.
C
C     ORDER         SETLL     ORDFIL
C
 *  The following DO loop processes all the records that have the
 *  same key value.
C
C                   IF        %EQUAL
C                   DOU       %EOF
C     ORDER         READE     ORDFIL
C                   IF        NOT %EOF
C                   EXCEPT    DETAIL
C                   ENDIF
C                   ENDDO
C                   ENDIF
C
 *  The READE operation reads the second, third, and fourth 101
 *  records in the same manner as the first 101 record was read.
 *  After the fourth 101 record is read, the READE operation is
 *  attempted.  Because the 102 record is not of the same group,
 *  %EOF will return '1', the EXCEPT operation is bypassed, and
 *  the DOU loop ends.
```

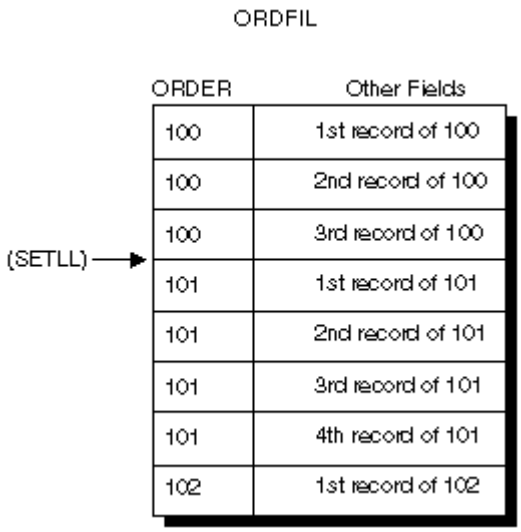


図 *384. SETLL* 命令

SETOFF (標識をオフに設定)

自由形式構文	(許可されていない - EVAL *INxx = *OFF を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
SETOFF				OF	OF	OF

SETOFF 命令は、71 から 76 桁目に指定された標識をオフに設定します。 71 から 76 桁目には少なくとも 1 つの結果の標識を指定しなければなりません。 1P および MR の指定は有効ではありません。 L1 から L9 の標識がオフに設定されても、 下位の制御レベルの標識が自動的にオフに設定されることはありません。

880 ページの図 385 は、SETOFF 命令を示しています。

詳しくは、580 ページの『標識設定命令』 を参照してください。

SETON (標識をオンに設定)

自由形式構文	(許可されていない - EVAL *INxx = *ON を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
SETON				ON	ON	ON

SETON 命令は 71 から 76 桁目に指定された標識をオンに設定します。 71 から 76 桁目には少なくとも 1 つの結果の標識を指定しなければなりません。 1P、MR、 KA-KN、および KP から KY の指定は有効ではありません。 L1 から L9 の標識がオンに設定されても、 低位の制御レベル標識が自動的にオンに設定されることはありません。

詳しくは、「580 ページの『標識設定命令』」を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  The SETON and SETOFF operations set from one to three indicators
 *  specified in positions 71 through 76 on and off.
 *  The SETON operation sets indicator 17 on.
C
C                   SETON                                      17
C
 *  The SETON operation sets indicators 17 and 18 on.
C
C                   SETON                                      1718
C
 *  The SETOFF operation sets indicator 21 off.
C
C                   SETOFF                                     21
```

図 *385. SETON および SETOFF 命令*

SHTDN (シャットダウン)

自由形式構文	(許可されていない - %SHUT 組み込み関数を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
SHTDN				ON	_	_

SHTDN 命令によって、プログラマーはシステム操作員がシャットダウンを要求している かどうかを調べることができます。 システム操作員がシャットダウンを要求している場合 には、71 桁目と 72 桁目に指定された結果の標識がオンに設定されます。 71 桁目と 72 桁目には、01 から 99、L1 から 9、U1 から U8、H1 から H9、LR、または RT の 標識の 1 つを入れなければなりません。

システム・オペレーターは、CL コマンド ENDJOB (ジョブ終了)、PWRDWNSYS (シス テム電源遮断)、ENDSYS (システム終了)、および ENDSBS (サブシステム終了) に *CNTRLD オプションを指定してシャットダウンを要求することができます。 これらのコマンドの詳細については、IBM i Information Center の「プログラミング」のカテゴリーを参照してください。

73 から 76 桁目はブランクでなければなりません。

詳細については、580 ページの『情報命令』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  When the SHTDN operation is run, a test is made to determine
 *  whether the system operator has requested shutdown.  If so,
 *  indicator 27 is set on.
C
C                   SHTDN                                       27
C   27              EXSR      Term_appl
C                   :
C                   :
C     Term_appl     BEGSR
C                   CLOSE     *ALL
C                   :
C                   ENDSR
```

図 *386. SHTDN* 命令

SORTA (配列の分類)

自由形式構文	SORTA{(A/D) } 配列名 | キー付きデータ構造配列
	SORTA{(A/D) } %SUBARR(配列名 | キー付きデータ構造配列: 開始要素 { : 要素の数 })
	SORTA{(A/D)} データ構造配列 %FIELDS(サブフィールド *1* { : サブフィールド *2* { ... } })
	SORTA{(A/D)} %SUBARR(データ構造配列 : 開始要素 {: 要素の数}) %FIELDS(サブフィールド *1*: { : サブフィールド *2* { ... } })

コード	演算項目 1	拡張演算項目 2
SORTA(A/D)		配列またはキー付きデータ構造配列
		%SUBARR(配列またはキー付きデータ構造配列 : 開始要素 {: 要素の数})
		SORTA{(A/D)} データ構造配列 %FIELDS(サブフィールド *1* { : サブフィールド *2* { ... } })
		SORTA{(A/D)} %SUBARR(データ構造配列 : 開始要素 {: 要素の数}) %FIELDS(サブフィールド *1*: { : サブフィールド *2* { ... } })

スカラー配列の場合、配列名 オペランドは、ソートされる配列の名前です。 配列 *IN を指定することはできません。 配列が交互形式のデータを持つコンパイル時配列または実行時前配列の場合に は、交互配列は分類されません。 配列名 として指定された配列だけが分類されます。

配列データ構造では、キー付きデータ構造配列構文を使用して 1 つのサブフィールドでソートすることも、%FIELDS を使用して必要なサブフィールドを指定し、複数のサブフィールドでソートすることもできます。

- キー付きデータ構造配列 オペランドは、ソート対象配列で構成される修飾名の後に、ソートのキーとして使用するサブフィールドを指定したものです。 ソート対象の配列データ構造は、配列の指標として * を指定して示します。 例えば、配列データ構造 INFO にサブフィールド NAME と SALARY がある場合、サブフィールド NAME をキーとして使用して配列 INFO をソートするには、 SORTA のオペランドとして INFO(*).NAME を指定します。 配列 INFO を SALARY でソートするには、SORTA のオペランドとして INFO(*).SALARY を指定します。
- 配列データ構造を複数のサブフィールドによってソートするには、%FIELDS を使用して、必要なサブフィールドをリストします。 例えば、配列データ構造 INFO にサブフィールド NAME および SALARY がある場合、配列 INFO を SALARY と NAME によってソートするには、SORTA の第 1 オペランドとして INFO を指定し、第 2 オペランドとして %FIELDS(SALARY:NAME) を指定します。 2 つの配列要素が比較され

るときに、SALARY サブフィールドが最初に比較されます。 SALARY サブフィールドが等しい場合、NAME サブフィールドが次に比較されます。 NAME サブフィールドが等しい場合、配列要素は等しいと見なされます。

```
SORTA INFO %FIELDS(SALARY : NAME):
```

詳しくは、「642 ページの『%FIELDS (ソートするためのサブフィールド)』」を参照してください。

配列の定義仕様書の ASCEND キーワードまたは DESCEND キーワードで配列の順序が定義されている場合、 配列は常にその順でソートされます。 配列の順序を指定しないと、順序はデフォルトで昇順になります。 命令拡張の 'A' が指定されている場合、配列は昇順でソートされます。 命令拡張の 'D' が指定されている場合、配列は降順でソートされます。

注 : ASCEND キーワードと DESCEND キーワードは、配列データ構造に指定できません。

配列が OVERLAY キーワードを指定して定義されていて、命令拡張の 'A' または 'D' が指定されていない場合には、基本配列はオーバーレイ配列によって定義された順序でソートされます。

図形配列および UCS-2 配列は、代替照合順序に関係なく、定義仕様書に指定された順序で、配列要素の 16 進数値によってソートされます。

配列の部分をソートするには、 %SUBARR 組み込み関数を使用してください。

注 :

1. 配列を分類すると前の順序は保存されません。 例えば、別のオーバーレイ配列を使用して配列を 2 回分類すると、最終順序 は最後の分類の順序となります。 分類の順序が同じで 16 進数値が (例えば、代替照合順序または順序を決定 するオーバーレイ配列を使用したために) 異なる要素は、分類後に前と同じ順 序にならないことがあります。
2. 基底ポインターの配列を分類する場合には、配列のすべての値が同じ空間内 のアドレスであることを確認しなければなりません。 そうでない場合には、結果に整合性がなくなる場合があります。 詳しくは、567 ページの『比較命令』 を参照してください。
3. ヌル値可能配列が分類される場合、その分類では、ヌル・フラグの設定値は 考慮に入れられません。
4. すべての定義済み要素が割り振り済みでない動的割り振り配列に 対する分類によって、エラーが発生する可能性があります。 %SUBARR 組み込み関数を使用して、ソートを割り振り済み要素のみに限定してください。
5. DESCEND キーワードを指定して定義された配列のソートでは、命令拡張 'A' を使用できません。 また、ASCEND キーワードを指定して定義された配列のソートでは、命令拡張 'D' を使用できません。
6. キー付きデータ構造構文 *DS(*).SUBFIELD* を使用して配列データ構造をソートするとき:
   a. 修飾名で指標 (*) より前の部分は、配列を表す必要があります。 また、(*) より後の部分は、スカラー・サブフィールドまたは指標付きスカラー配列を表す必要があります。
   b. 複合修飾名に複数の配列サブフィールドがある場合には、1 つの配列サブフィールドしかソートできません。 修飾名にあるその他の配列ではすべて、指標の指定が必要です。 例えば、配列データ構造 FAMILY に配列サブフィールド CHILD があり、CHILD の要素に配列サブフィールド PET があり、PET サブフィールドにサブフィールド NAME がある場合、FAMILY、CHILD、および PET のいずれか 1 つの配列でしか、1 つの SORTA 命令でソートできません。 配列 CHILD をソートする場合には、FAMILY と PET の配列に明示的な指標が必要になります。 SORTA の有効なオペランドの例として、FAMILY(i).CHILD(*).PET(1).NAME があります。 この SORTA 命令では、FAMILY(i) の配列 CHILD を PET(1) のサブフィールド NAME でソートします。
   c. 配列データ構造は、命令拡張 'D' が指定されない限り、キーの昇順でソートされます。
   d. ソート・キーが順序配列の要素である場合、配列データ構造のソート時にその順序は考慮されません。

詳しくは、「561 ページの『配列命令』」を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
DARRY             S              1A   DIM(8) ASCEND
DARRY2            S              1A   DIM(8)
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  The SORTA operation sorts ARRY into ascending sequence because
 *  the ASCEND keyword is specified.
 *  If the unsorted ARRY contents were GT1BA2L0, the sorted ARRY
 *  contents would be ABGLT012.
C                   SORTA     ARRY

 *  The SORTA operation sorts ARRY2 into descending ascending sequence
 *  the (D) operation extender is specified.
 *  If the unsorted ARRY2 contents were GT1BA2L0, the sorted ARRY2
 *  contents would be 210TLGBA.
C                   SORTA(D)  ARRY2
```

図 *387. SORTA* 命令

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
 * In this example, the base array has the values aa44 bb33 cc22 dd11
 * so the overlaid array ARRO has the values 44 33 22 11.
D                 DS
D ARR                            4    DIM(4) ASCEND
D ARRO                           2    OVERLAY(ARR:3)
D
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C
 * After the SORTA operation, the base array has the values
 * dd11 cc22 bb33 aa44
C
C                   SORTA     ARRO
```

図 *388. OVERLAY* による *SORTA* 命令

```
 * The names array does not have a sequence keyword
 * (ASCEND or DESCEND) specified.
D info            DS                  QUALIFIED
D   names                       10A   DIM(2)

 /free

    // Initialize the array
    info.names(1) = 'Bart';
    info.names(2) = 'Lisa';

    // Sort the info.names in descending order
    SORTA(D) info.names;
    // info.names(1) = 'Lisa'
    // info.names(2) = 'Bart'

    // Sort the info.names in ascending order
    SORTA(A) info.names;
    // info.names(1) = 'Bart'
    // info.names(2) = 'Lisa'

    // With no operation extender, it defaults to ascending order
    SORTA info.names;
    // info.names(1) = 'Bart'
    // info.names(2) = 'Lisa'
```

図 *389.* 昇順または降順の *SORTA* 命令

```
D emp             DS                  QUALIFIED DIM(25)
D   name                         25A   VARYING
D   salary                        9P 2
D numEmp          S               10I 0

// Initialize the data structure
emp(1).name = 'Maria';
emp(1).salary = 1100;
emp(2).name = 'Pablo';
emp(2).salary = 1200;
emp(3).name = 'Bill';
emp(3).salary = 1000;
emp(4).name = 'Alex';
emp(4).salary = 1300;
numEmp = 4;

// Sort the EMP array using the NAME subfield as a key
SORTA %subarr(emp(*).name : 1 : numEmp);
// emp(1).name = 'Alex'       <-----
// emp(1).salary = 1300
// emp(2).name = 'Bill'       <-----
// emp(2).salary = 1000
// emp(3).name = 'Maria'      <-----
// emp(3).salary = 1100
// emp(4).name = 'Pablo'      <-----
// emp(4).salary = 1200

// Sort the EMP array using the SALARY subfield as a key
SORTA %subarr(emp(*).salary : 1 : numEmp);
// emp(1).name = 'Bill'
// emp(1).salary = 1000       <-----
// emp(2).name = 'Maria'
// emp(2).salary = 1100       <-----
// emp(3).name = 'Pablo'
// emp(3).salary = 1200       <-----
// emp(4).name = 'Alex'
// emp(4).salary = 1300       <-----

// Sort the EMP array descending using the SALARY subfield
SORTA(D) %subarr(emp(*).salary : 1 : numEmp);
// emp(1).name = 'Alex'
// emp(1).salary = 1300       <-----
// emp(2).name = 'Pablo'
// emp(2).salary = 1200       <-----
// emp(3).name = 'Maria'
// emp(3).salary = 1100       <-----
// emp(4).name = 'Bill'
// emp(4).salary = 1000       <-----
```

図 *390.* キー付き配列データ構造構文を使用した配列データ構造での *SORTA* 命令

```
D emp             DS                  QUALIFIED DIM(25)
D   name                         25A   VARYING
D   salary                        9P 2
D numEmp          S               10I 0

// Initialize the data structure
emp(1).name = 'Maria';
emp(1).salary = 1300;
emp(2).name = 'Pablo';
emp(2).salary = 1200;
emp(3).name = 'Bill';
emp(3).salary = 1100;
emp(4).name = 'Alex';
emp(4).salary = 1200;
numEmp = 4;

// Sort the EMP array using the SALARY and NAME subfields
SORTA %SUBARR(emp : 1 : numEmp) %FIELDS(salary : name);
// emp(1).name = 'Bill'
// emp(1).salary = 1100       <-----
// emp(2).name = 'Alex'       <-----
// emp(2).salary = 1200       <-----
// emp(3).name = 'Pablo'      <-----
// emp(3).salary = 1200       <-----
// emp(4).name = 'Maria'
// emp(4).salary = 1300       <-----
```

図 *391. %FIELDS* を使用して複数のサブフィールドでソートする配列データ構造での *SORTA* 命令

```
D emp_t           DS                  QUALIFIED TEMPLATE
D   name                        25A   VARYING
D teams           DS                  QUALIFIED DIM(2)
D   manager                     25A   VARYING
D   emps                            LIKEDS(emp_t) DIM(2)

// Initialize the data structure
teams(1).manager = 'Jack';
teams(1).emps(1).name = 'Yvonne';
teams(1).emps(2).name = 'Mary';
teams(2).manager = 'Ann';
teams(2).emps(1).name = 'Wendy';
teams(2).emps(2).name = 'Thomas';

// Sort the TEAMS array using the MANAGER subfield as a key
SORTA teams(*).manager;
//    teams(1).manager = 'Ann'           <-----
//    teams(1).emps(1).name = 'Wendy'
//    teams(1).emps(2).name = 'Thomas'
//    teams(2).manager = 'Jack'          <-----
//    teams(2).emps(1).name = 'Yvonne'
//    teams(2).emps(2).name = 'Mary'

// Sort the TEAMS array using the EMPS(2).NAME subfield as a key
SORTA teams(*).emps(2).name;
//    teams(1).manager = 'Jack'
//    teams(1).emps(1).name = 'Yvonne'
//    teams(1).emps(2).name = 'Mary'     <-----
//    teams(2).manager = 'Ann'
//    teams(2).emps(1).name = 'Wendy'
//    teams(2).emps(2).name = 'Thomas'   <-----

// Sort the TEAMS(1).EMPS array using the NAME subfield as a key
SORTA teams(1).emps(*).name;
//    teams(1).manager = 'Jack'
//    teams(1).emps(1).name = 'Mary'     <-----
//    teams(1).emps(2).name = 'Yvonne'   <-----
//    teams(2).manager = 'Ann'
//    teams(2).emps(1).name = 'Wendy'
//    teams(2).emps(2).name = 'Thomas'

// Sort the TEAMS array first by the MANAGER subfield
// and then by the EMPS.NAME subfields
SORTA teams(*).manager;
for i = 1 to %ELEM(TEAMS);
   SORTA teams(i).emps(*).name;
endfor;
// After the first sort, by MANAGER:
//    teams(1).manager = 'Ann'           <-----
//    teams(1).emps(1).name = 'Wendy'
//    teams(1).emps(2).name = 'Thomas'
//    teams(2).manager = 'Jack'          <-----
//    teams(2).emps(1).name = 'Mary'
//    teams(2).emps(2).name = 'Yvonne'
// After loop with the second sort, by EMPS.NAME:
//    teams(1).manager = 'Ann'
//    teams(1).emps(1).name = 'Thomas'   <----- 1
//    teams(1).emps(2).name = 'Wendy'    <----- 1
//    teams(2).manager = 'Jack'
//    teams(2).emps(1).name = 'Mary'     <----- 2
//    teams(2).emps(2).name = 'Yvonne'   <----- 2
```

図 *392.* 複合配列データ構造での *SORTA* 命令

SQRT (平方根)

自由形式構文	(許可されていない - %SQRT 組み込み関数を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
SQRT (H)		値	平方根			

SQRT 命令では、演算項目 2 に指定されたフィールドの平方根が求められま す。 演算項目 2 の平方根が結果のフィールドに入れられます。

演算項目 2 は数値でなければならず、配列、配列要素、フィールド、形象定 数、リテラル、名前付き定数、サブフィールド、またはテーブル名の いずれかを入れることができます。

結果フィールドは数値でなければならず、配列、配列要素、サブフィールド、 またはテーブル要素のいずれかを入れることができます。

演算項目 2 および結果フィールドに配列名が入っている場合には、配列全体 を SQRT 命令で使用することができます。

結果フィールドの小数点以下の桁数は、演算項目 2 の小数点以下の桁数より小さくても大きくてもかまいません。 しかし、結果フィールドが演算項目 2 の小数点以下の桁数の半分より小さくなってはなりません。

演算項目 2 のフィールドの値がゼロの場合には、結果フィールドの値もゼロ になります。 演算項目 2 のフィールドの値が負の場合には、RPG IV 例外/エラー処理ルーチンに制御が渡されます。

SQRT 命令に関する規則について詳しくは、557 ページの『算術演算』を参照し てください。

SQRT 命令の例については、560 ページの図 181 を参照してください。

SUB (減算)

自由形式構文	(許可されていない - - または -= 演算子を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
SUB (H)	被減数	減数	差	+	-	Z

演算項目 1 が指定されている場合には、演算項目 1 から演算項目 2 が引かれ て、その差が結果フィールドに入れられます。 演算項目 1 が指定されていない 場合には、演算項目 2 の内容が結果フィールドの内容から引かれます。

演算項目 1 および演算項目 2 は数値でなければならず、それぞれ配列、配列要 素、フィールド、形象定数、リテラル、名前付き定数、サブフィール ド、またはテーブル名のいずれかを入れることができます。

結果フィールドは数値でなければならず、配列、配列要素、サブフィールド、 またはテーブル名のいずれかを入れることができます。

SUB 命令に関する規則については、557 ページの『算術演算』を参照してく ださい。

SUB 命令の例については、560 ページの図 181 を参照してください。

SUBDUR (期間減算)

自由形式構文	許可されていない - - または -= 演算子と %YEARS および %MONTHS などの 期間関数、 または %DIFF 組み込み関数を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
SUBDUR (E) (期間)	日付/時刻/タイム・スタンプ	日付/時刻/タイム・スタンプ	期間: 期間コード	_	ER	_
SUBDUR (E) (新しい日付)	日付/時刻/タイム・スタンプ	期間: 期間コード	日付/時刻/タイム・スタンプ	_	ER	_

SUBDUR 命令には次の機能があります。

- 新しい日付、時刻、またはタイム・スタンプを設定するための 888 ページの『期間の減算』
- 888 ページの『期間の計算』

期間の減算

SUBDUR 命令は、演算項目 1 に指定されたフィールドまたは定数から演算項目 2 に指定された期間を引いて、その結果を結果フィールドに指定されたフィールドの結果の日付、時刻、またはタイム・スタンプに入れるために使用すること ができます。

演算項目 1 は任意指定で、日付、時刻、またはタイム・スタンプ・フィールド、 配列、配列要素、リテラル、または定数を入れることができます。 演算項目 1 にフィールド名、配列、または配列要素が入っている場合には、 そのデータ・タイプは結果フィールドに指定されたフィールドと同じタイプ でなければなりません。 演算項目 1 が指定されていない場合には、結果フィールドに指定されたフィー ルドから期間が引かれます。

演算項目 2 は必須で、2 つの副演算項目が入ります。 最初の副演算項目は、 小数点以下の桁数がゼロの数値フィールド、配列、または定数です。 フィールドが負の場合には、期間がフィールドに加えられます。 2 番目の副演算項目は、期間のタイプを示す有効な期間コードでなければなりません。 期間コードは、結果フィールドのデータ・タイプと同じでなければなりません。 例えば、年、月、または日の期間を引くことはできますが、分の期間を日付 フィールドから引くことはできません。期間コードとその短縮形のリストについては、572 ページの『日付命令』を参照してください。

結果フィールドは、日付、時刻、またはタイム・スタンプ・データ・タイプ・フ ィールド、配列、または配列要素でなければなりません。 演算項目 1 がブランクの場合には、結果フィールドの値から期間が引かれれま す。 結果フィールドが配列の場合には、演算項目 2 の値が配列のそれぞれの要素か ら引かれます。 結果フィールドが時刻フィールドの場合には、結果は常に有効な時刻となりま す。 例えば、00:58:59 から 59 分を引くと、-00:00:01 になります。 この時刻は有効でないので、コンパイラーはこれを 23:59:59 に調整します。

月の期間を日付から引く場合には、一般的な規則では月の部分が期間の月数 だけ引かれて、日の部分は変わりません。 この例外は、結果の日の部分が結果の月の実際の日数を超える場合です。 この場合には、結果の日の部分が実際の月の最終日付に合わせて調整されます。 次の例 (*YMD 形式と想定している) はこの点について示しています。

- `'95/05/30' SUBDUR 1:*MONTHS` は '95/04/30' になる。

  結果の月の部分が 1 だけ小さくなって、日の部分は変わりません。
- `'95/05/31' SUBDUR 1:*MONTHS` は '95/04/30' になる。

  結果の月の部分は 1 だけ小さくなって (4 月は 30 日しかないので) 結果 の日の部分が調整されています。

年の期間を減算する場合にも同様の結果になります。 例えば、'92/02/29' から 1 年を引くと (結果の年は閏年でないので) '91/02/28' になります。

月の期間と年の期間の減算について詳しくは、574 ページの『予期しない結果』を参照してください。

注 : システムは期間を 15 桁に制限します。 有効桁数が 15 桁を超える期間の減算 を行うと、エラーまたは切り捨てが起こります。 これらの問題は、演算項目 2 の最初の副演算項目を 15 桁に制限することによって回避することができます。

期間の計算

SUBDUR 命令は、次の期間の計算にも使用することができます。

1. 2 つの日付
2. 日付とタイム・スタンプ
3. 2 つの時刻
4. 時刻とタイム・スタンプ
5. 2 つのタイム・スタンプ

演算項目 1 と演算項目 2 のデータ・タイプは、上記に指定した通り、矛盾しないものでなければなりません。

演算項目 1 は必須で、日付、時刻、またはタイム・スタンプ・フィールド、 サブフィールド、配列、配列要素、定数、またはリテラルを入れることが できます。

演算項目 2 も必須で、日付、時刻、またはタイム・スタンプ・フィールド、配列 、配列要素、リテラル、または定数を入れることができます。

次の期間コードが有効です。

- 2 つの日付または日付とタイム・スタンプの場合: *DAYS (*D)、*MONTHS (*M)、および *YEARS (*Y)
- 2 つの時刻または時刻とタイム・スタンプの場合: *SECONDS (*S)、*MINUTES (*MN)、および *HOURS (*H)
- 2 つのタイム・スタンプの場合: *MSECONDS (*MS)、*SECONDS (*S)、 *MINUTES (*MN)、*HOURS (*H)、*DAYS (*D)、*MONTHS (*M)、および *YEARS (*Y)

結果は整数の数字で、剰余は廃棄されます。 例えば、61 分は 1 時間と等しく、59 分は 0 時間に等しくなります。

結果フィールドは 2 つの副演算項目から構成されます。 最初の副演算項目は、 この命令の結果が入れられる、小数点以下の桁数がゼロの数値フィールド、 配列、または配列要素の名前です。 2 番目の副演算項目には、期間のタイプを 示す期間コードが入ります。 演算項目 1 の日付が演算項目 2 の日付より速い場 合には、結果フィールドは負になります。

日付時刻フィールドの処理について詳しくは、572 ページの『日付命令』を参照してください。

注 : マイクロ秒の期間 (*mseconds) の計算は、期間に対する 15 桁のシステム の制限を超えることがあって、エラーまたは切り捨ての原因になります。 このよ うな状況は、演算項目 1 と演算項目 2 の指定に 32 年と 9 か月を超える差があ る時に起こります。

起こり得るエラー状況

1. 期間の減算の場合。
   - 演算項目 1 の日付、時刻、またはタイム・スタンプ・フィールドの値が正しく ない場合。
   - 演算項目 1 がブランクで、 演算の前の結果フィールドの値が正しくない場合。
   - あるいは命令の結果が *HIVAL より大きいか、 または *LOVAL より小さい場合。
2. 期間の計算の場合。
   - 演算項目 1 または演算項目 2 の日付、時刻、またはタイム・スタンプ・フィー ルドの値が正しくない場合。
   - あるいは結果フィールドが小さ過ぎて結果の期間が入らない場合。

このような場合には、エラーが通知されます。

エラーが検出されると、次のプログラム状況コードのいずれかでエラーが 生成されます。

- 00103: 結果フィールドが結果を入れるだけ大きくない
- 00112: 日付、時刻、またはタイム・スタンプの値が正しくない
- 00113: 日付オーバーフローまたはアンダーフローが起こった (すなわち、結果 の日付が *HIVAL より大きいか、または *LOVAL より小さい)

結果フィールドの値は変わりません。 プログラム状況コードが 103、112 または 113 である例外を処理するために、命令コード拡張 'E' または エラー標識 ER を指定できますが、両方を指定することはできません。エラー処理について詳しくは、163 ページの『プログラム例外/エラー』を参照してください。

SUBDUR の例

```
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 * Determine a LOANDATE which is xx years, yy months, zz days prior to
 * the DUEDATE.
C     DUEDATE       SUBDUR    XX:*YEARS     LOANDATE
C                   SUBDUR    YY:*MONTHS    LOANDATE
C                   SUBDUR    ZZ:*DAYS      LOANDATE
 * Add 30 days to a loan due date
 *
C                   SUBDUR    -30:*D        LOANDUE
 * Calculate the number of days between LOANDATE and DUEDATE.
 * If DUEDATE is after LOANDATE, the value of NUM_DAYS will be positive.
C     DUEDATE       SUBDUR    LOANDATE      NUM_DAYS:*D       5 0
 * Determine the number of months between LOANDATE and DUEDATE.
C     DUEDATE       SUBDUR    LOANDATE      NUM_MONTHS:*M     5 0
```

図 *393. SUBDUR* 命令

SUBST (サブストリング)

自由形式構文	(許可されていない - %SUBST を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
SUBST (E P)	取り出す長さ	基本ストリング: 開始	ターゲット・ストリング	_	ER	_

SUBST 命令は、演算項目 2 に指定された位置から始まる演算項目 1 に指定 された長さのサブストリングを演算項目 2 から戻して、このサブストリングを 結果フィールドに入れます。 演算項目 1 が指定されていない場合には、開始位置からのストリングの長さが 使用されます。 図形ストリングまたは UCS-2 ストリングの場合には、開始位置は 2 バイト単位で計算されます。 基本ストリングとターゲット・ストリングは両方とも同じタイプ (両方とも文字、両方とも図形、または両方とも UCS-2) でなければなりません。

演算項目 1 には、演算項目 2 に指定されたストリングから抜き出すストリングの長さの値を入れることができます。 これは小数点以下の桁数がゼロの数値である必要があり、フィールド名、配列要素、テーブル名、リテラル、または名前付き定数のいずれかを入れることができます。

演算項目 2 には、基本ストリングか、または基本ストリングに ':' を付け開始位置を指定したものを入れなければなりません。 基本ストリング部分には、フィールド名、配列要素、名前のついた定数、 データ構造名、テーブル名、またはリテラルのいずれかを入れることができま す。 開始位置は小数点以下の桁数がゼロの数値でなければならず、フィールド名、 配列要素、テーブル名、リテラル、または名前のついた定数のいずれかを 入れることができます。 開始位置が指定されていない場合には、SUBST は基本ストリングの 1 桁目から 開始します。 図形ストリングまたは UCS-2 ストリングの場合には、開始位置は 2 バイト単位で計算されます。

抜き出されるサブストリングの開始位置および長さは、正の整数でなければ なりません。 開始位置は基本ストリングの長さを超えてはならず、長さは開始位置からの基 本ストリングの長さを超えてはなりません。 これらの条件のいずれかまたは両方が 満たされない場合には、この命令は実行されません。

SUBST 例外 (プログラム状況コード 100) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、163 ページの『プログラム例外/エラー』を参照してください。

結果フィールドは文字、図形、または UCS-2 でなければならず、フィールド名、配列要素、データ構造、またはテーブル名のいずれかを入れることができます。 結果は左寄せされます。 結果フィールドの長さは、少なくとも演算項目 1 に指定された長さと同じである必要があります。 サブストリングが結果フィールドに指定されたフィールドより長い場合には、そのサブストリングの右側で切り捨てが行われます。 結果のフィールドが可変長の場合、その長さは変わりません。

詳しくは、「589 ページの『ストリング命令』」を参照してください。

注 : 演算項目 1、演算項目 2、または結果フィールドに形象定数を使用することは できません。 演算項目 1 と結果フィールドまたは演算項目 2 と結果フィールドにオーバーラ ップがあってもかまいません。 演算項目 1 が結果フィールドの長さより短い場合には、命令拡張桁に指定され た P によって、サブストリング処理の後で結果フィールドの右側にブランクを 埋め込むように指示されます。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 * The SUBST operation extracts the substring from factor 2 starting
 * at position 3 for a length of 2.  The value 'CD' is placed in the
 * result field TARGET.  Indicator 90 is not set on because no error
 * occurred.
C
C                   Z-ADD     3             T                2 0
C                   MOVEL     'ABCDEF'      String          10
C     2             SUBST     String:T      Target               90
 *
 * In this SUBST operation, the length is greater than the length
 * of the string minus the start position plus 1.  As a result,
 * indicator 90 is set on and the result field is not changed.
C
C                   MOVE      'ABCDEF'      String           6
C                   Z-ADD     4             T                1 0
C     5             SUBST     String:T      Result               90
C
 * In this SUBST operation, 3 characters are substringed starting
 * at the fifth position of the base string.  Because P is not
 * specified, only the first 3 characters of TARGET are
 * changed.  TARGET contains '123XXXXX'.
C
C                   Z-ADD     3             Length           2 0
C                   Z-ADD     5             T                2 0
C                   MOVE      'TEST123'     String           8
C                   MOVE      *ALL'X'       Target
C     Length        SUBST     String:T      Target           8
```

図 *394. SUBST* 命令

```
 *
 * This example is the same as the previous one except P
 * specified, and the result is padded with blanks.
 * TARGET equals '123bbbbb'.
C
C                   Z-ADD     3             Length            2 0
C                   Z-ADD5                  T                 2 0
C                   MOVE      'TEST123'     String            8
C                   MOVE      *ALL'X'       Target
C     Length        SUBST(P)  String:T      Target            8
C
C
 *
 * In the following example, CITY contains the string
 * 'Toronto, Ontario'.  The SCAN operation is used to locate the
 * separating blank, position 9 in this illustration.  SUBST
 * without factor 1 places the string starting at position 10 and
 * continuing for the length of the string in field TCNTRE.
 * TCNTRE contains 'Ontario'.
C      ' '          SCAN      City          C
C                   ADD       1             C
C                   SUBST     City:C        TCntre
 *
 * Before the operations STRING='bbbJohnbbb&
 * RESULT is a 10 character field which contains 'ABCDEFGHIJ'.
 * The CHECK operation locates the first nonblank character
 * and sets on indicator 10 if such a character exists.  If *IN10
 * is on, the SUBST operation substrings STRING starting from the
 * first non-blank to the end of STRING.  Padding is used to ensure
 * that nothing is left from the previous contents of the result
 * field.  If STRING contains the value '   HELLO ' then RESULT
 * will contain the value 'HELLO     ' after the SUBST(P) operation.
 * After the operations RESULT='Johnbbbbbb'.
C
C      ' '          CHECK     STRING        ST                      10
C   10              SUBST(P)  STRING:ST     RESULT
```

TAG (タグ)

自由形式構文	(許可されていない - LEAVE、ITER、 および RETURN などの他の命令コードを使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
TAG	ラベル					

宣言命令の TAG は、788 ページの『GOTO (演算命令のスキップ)』 または 717 ページの『CABxx (比較および分岐)』 命令の行き先を識別するラベルを指定します。 この命令は、演算内 (合計演算内を含む) の任意の場所に指定することができ ます。

サイクル・メイン・プロシージャーのサブルーチン内の GOTO は、同じサブルーチン、明細演算、または合計演算の中の TAG に出すことができます。 サブプロシージャーのサブ ルーチン内の GOTO は、同じサブルーチン内またはサブプロシージャーの本体 内の TAG に出すことができます。

制御レベルの指定 (7 から 8 桁目) はブランクにするか、 あるいはプログラムの適切なセクション内のステートメントをグループにまとめる ために L1 から L9 標識、LR 標識、または L0 標識を入れることができます。 条件付け標識の指定 (9 から 11 桁目) は使用できません。

演算項目 1 には、GOTO または CABxx 命令の行き先の名前を入れなければな りません。 この名前は、GOTO 命令の演算項目 2 に指定されるか、または CABxx 命令の結果フィールドに指定される固有の記号名でなければなりません。 この名前は、複数の GOTO または CABxx 命令の共通の行き先として使用することができます。

RPG IV 論理サイクルの別の部分から TAG に分岐すると、無限ループに入ることがあります。 例えば、明細演算行に合計演算 TAG 命令への GOTO 命令を指定すると、無限 ループが発生する場合があります。

TAG 命令の例については、789 ページの図 330 を参照してください。

詳しくは、562 ページの『分岐命令』または575 ページの『宣言命令』を参照してください。

TEST (日付/時刻/タイム・スタンプのテスト)

自由形式構文	TEST {(EDTZ) }{*dtz* 形式 } フィールド名

コード	演算項目 1 (*dtz* 形式)	演算項目 2	結果フィールド (フィールド名)	標識		
TEST (E)			日付/時刻またはタイムスタンプ・フィールド	_	ER	_
TEST (D E)	日付の形式		文字または数値フィールド	_	ER	_
TEST (E T)	時刻の形式		文字または数値フィールド	_	ER	_
TEST (E Z)	タイム・スタンプ形式		文字または数値フィールド	_	ER	_

TEST 命令コードでは、日付、時刻、またはタイム・スタンプ・フィールドを 使用する前にそれらの妥当性を検査することができます。

使用可能な形式については、273 ページの『日付データ・タイプ』、275 ページの『時刻データ・タイプ』、および 277 ページの『タイム・スタンプ・データ・タイプ』を参照してください。

- フィールド名 オペランドが日付、時刻、またはタイム・スタンプとして宣言される場合、次のようになります。
  - *dtz* 形式 オペランドは指定できません。
  - 命令コード拡張 'D'、'T'、および 'Z' は使用できません。
- フィールド名 オペランド文字または数値として宣言されているフィールドの場合には、命令コード拡張 'D'、'T'、または 'Z' のいずれかを指定しなければなりません。

  注 : フィールド名 オペランドは区切り文字のない文字フィールドで、*dtz* 形式オペランドには、後ろにゼロが付いた日付、時刻、またはタイム・スタンプ形式が入っている必要があります。
  - 命令コード拡張に 'D' (日付のテスト) が 含まれている場合。
    - *dtz* 形式 は任意指定で、有効な日付形式のいずれかになります (273 ページの『日付データ・タイプ』を参照)。
    - *dtz* 形式が指定されていない場合は、制御仕様書に DATFMT キーワードで指定された形式と見なされます。 このキーワードが指定されていない場合には、 *ISO と見なされます。
  - 命令コード拡張に 'T' (時刻のテスト) が 含まれている場合。
    - *dtz* 形式 は任意指定で、有効な時刻形式のいずれかになります (275 ページの『時刻データ・タイプ』を参照)。
    - *dtz* 形式が指定されていない場合は、制御仕様書に TIMFMT キーワードで指定された形式と見なされます。 このキーワードが指定されていない場合には、 *ISO と見なされます。

    注 : 命令コード拡張 (T) で *USA の日付の形式を使用することはできません。 *USA の日付の形式には、数値の結果のフィールドが使用された場合に数値に変 換できない AM/PM の制約があります。
  - 命令コード拡張に 'Z' (タイム・スタンプのテスト) が 含まれている場合。
    - *dtz* 形式 はオプションであり、*ISO または *ISO0 (277 ページの『タイム・スタンプ・データ・タイプ』を参照) にすることができます。

区切り記号のない数値フィールドおよび文字フィールドは、日付、時刻、またはタイム・スタンプの値の数字部分の妥当性についてテストされます。 文字フィールドは、数字および区切り記号の両方の 妥当性についてテストされます。

フィールド名・オペランドとして指定された文字または数字フィールドが、 テスト対象の形式で必要とされるよりも長い場合は、余分なデータは無視されます。 文字データの場合、左端のデータのみが使用され、数値データの場合、右端のデータのみが使用されます。 例えば、*dtz* 形式 オペランドが数値の日付のテストのための *MDY である 場合、フィールド名 オペランドの右端の 6 桁が検査されます。

テスト命令では、命令コード拡張 'E' またはエラー標識 ER を指定する必要がありますが、 両方を指定することはできません。 フィールド名 オペランドが正しくない場合、プログラム状況コード 112 が通知されます。 次に、指定されたエラー処理方法に基づいて、エラー標識がオンに設定されるか、 または %ERROR 組み込み関数が '1' を戻すように設定されます。エラー処理について詳しくは、163 ページの『プログラム例外/エラー』を参照してください。

数値フィールドまたは文字フィールド に 'Z' 命令コード拡張が指定されている場合、 秒の小数部が正確に 6 桁の値であると想定されます。

詳しくは、572 ページの『日付命令』または 595 ページの『テスト命令』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++
D
D Datefield       S               D   DATFMT(*JIS)
D Num_Date        S              6P 0 INZ(910921)
D Char_Time       S              8    INZ('13:05 PM')
D Char_Date       S              6    INZ('041596')
D Char_Tstmp      S             20    INZ('19960723140856834000')
D Char_Date2      S              9A   INZ('402/10/66')
D Char_Date3      S              8A   INZ('2120/115')
D
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  Indicator 18 will not be set on, since the character field is a
 *  valid *ISO timestamp field, without separators.
C     *ISO0         TEST (Z)                Char_Tstmp             18
 *  Indicator 19 will not be set on, since the character field is a
 *  valid *MDY date, without separators.
C     *MDY0         TEST (D)                Char_Date              19
 *
 *  %ERROR will return '1', since Num_Date is not *DMY.
 *
C     *DMY          TEST (DE)               Num_Date
 *
 *  No Factor 1 since result is a D data type field
 *  %ERROR will return '0', since the field
 *  contains a valid date
C
C                   TEST (E)                Datefield
C
 * In the following test, %ERROR will return '1' since the
 * Timefield does not contain a valid USA time.
C
C     *USA          TEST (ET)               Char_Time
C
 *  In the following test, indicator 20 will be set on since the
 *  character field is a valid *CMDY, but there are separators.
C
C     *CMDY0        TEST (D)                char_date2             20
C
 *  In the following test, %ERROR will return '0' since
 *  the character field is a valid *LONGJUL date.
C
C     *LONGJUL      TEST (DE)               char_date3
```

図 *395. TEST (E D/T/Z)* の例

TESTB (ビットのテスト)

自由形式構文	(許可されていない - %BITAND 組み込み関数を使用。619 ページの図 204 を参照。)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
TESTB		ビット数	文字フィールド	OF	ON	EQ

TESTB は、演算項目 2 で示されたビットを、結果フィールドとして指定さ れたフィールド内の対応するビットと比較します。 結果フィールドは 1 桁の文 字フィールドでなければなりません。 71 から 76 桁目の結果の 標識は、結果フィールドのビットの状況を反映します。 演算項目 2 は、常に結果フィールドの元となるビットです。

演算項目 2 には以下のものを入れることができます。

- ビット番号 *0* から *7:* 1 つの 命令で 1 から 8 ビットをテストすることができます。 テストされるビットは 0 から 7 の番号で識別されます (0 が左端ビットです)。 ビット番号はアポストロフィで囲まなければなりません。例えば、ビット 0、2、および 5 をテストするには、演算項目 2 に「025」を入れます。
- フィールド名*:* 演算項目 2 に、1 桁の文字フィールドの名前、テーブル名、または配列要素を指定することができます。 このフィールド、テーブル名、または配列要素でオンのビットが、結果フィールドの対応するビットと比較されます。オフのビットは対象外です。 配列のそれぞれの要素が 1 桁の文字フィールドである場合には、結果フィール ドに指定されるフィールドは配列要素とすることができます。
- *16* 進数リテラルまたは名前のついた定数*:* 1 バイトの 16 進数リテラルまたは 16 進数の名前のついた定数を指定する ことができます。 演算項目 2 でオンのビットが、結果フィールドの対応するビットと比較されま す。オフのビットは対象外です。

896 ページの図 396 に、TESTB 命令の使用法を示します。

71 から 76 桁目に割り当てられた標識は、結果フィールドのビットの状況を反映 します。 1 つの命令に少なくとも 1 つの標識を割り当てなければならず、最大 3 つまで割り当ていることができます。 TESTB 命令では、結果の標識は次のように設定されます。

- *71* 桁目と *72* 桁目*:* これらの桁の標識は、演算項目 2 に指定されたビット番号または演算項目 2 のフィールドでオンになっているそ れぞれのビットが結果フィールドでオフになっている場合にオンに設定されます。 すなわち、指定されたすべてのビットがオフに等しくなります。
- *73* 桁目と *74* 桁目*:* これらの桁の標識は、演算項目 2 に指定されたビット番号または演算項目 2 のフィールドでオンになっているそ れぞれのビットが、結果フィールドで混在する状況 (あるものはオンで、あるものはオフ) の場合にオンに設定されます。 すなわち、指定された少なくとも 1 つのビットがオンになります。

  注 : 1 つのビットだけをテストする場合には、これらの桁はブランクでなけ ればなりません。 フィールド名が演算項目 2 に指定されていて、1 つのビットだけがオンになっ ている場合には、73 桁目と 74 桁目の標識はオンに設定されません。
- *75* 桁目と *76* 桁目*:* これらの桁の標識は、演算項目 2 に指定されたビット番号または演算項目 2 のフィールドでオンになっているそ れぞれのビットが結果フィールドでオンになっている場合にオンに設定されます。 すなわち、指定されたすべてのビットがオンに等しくなります。

  注 : 演算項目 2 のフィールドにオンのビットがない場合には、どの標識も オンに設定されません。

詳しくは、562 ページの『ビット操作』または 595 ページの『テスト命令』を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  The field bit settings are FieldF = 00000001, and FieldG = 11110001.
 *
 *  Indicator 16 is set on because bit 3 is off (0) in FieldF.
 *  Indicator 17 is set off.
C                   TESTB     '3'           FieldF               16  17
 *
 *  Indicator 16 is set on because both bits 3 and 6 are off (0) in
 *  FieldF.  Indicators 17 and 18 are set off.
C                   TESTB     '36'          FieldF               161718
 *
 *  Indicator 17 is set on because bit 3 is off (0) and bit 7 is on
 *  (1) in FLDF.  Indicators 16 and 18 are set off.
C                   TESTB     '37'          FieldF               161718
 *
 *  Indicator 17 is set on because bit 7 is on (1) in FLDF.
 *  Indicator 16 is set off.
C                   TESTB     '7'            FieldF              16  17
 *
 *  Indicator 17 is set on because bits 0,1,2, and 3 are off (0) and
 *  bit 7 is on (1).  Indicators 16 and 18 are set off.
C                   TESTB     FieldG         FieldF              161718
 *
 *  The hexadecimal literal X'88' (10001000) is used in factor 2.
 *  Indicator 17 is set on because at least one bit (bit 0) is on
 *  Indicators 16 and 18 are set off.
C                   TESTB     X'88'          FieldG              161718
```

図 *396. TESTB* 命令

TESTN (数字のテスト)

自由形式構文	(許可されていない - 使用する前に変数をテストするのではなく、 変数の使用法を MONITOR グループ内でコーディングし、エラーが発生した場合は ON-ERROR で処理する。 『エラー処理命令』を参照。)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
TESTN			文字フィールド	NU	BN	BL

TESTN 命令では、文字の結果フィールドにゾーン 10 数字およびブランク があるかどうかがテストされます。 結果フィールドは文字フィールドでなければなりません。 数値と見なされるためには、フィールドの最下位の文字以外のそれぞれの文字 に、16 進数の F ゾーンおよび数字 (0 から 9) が入っていなければなりません。 最下位の文字は、16 進数の C、D、または F ゾーンおよび数字 (0 から 9) が入っ ていれば数値となります。 英字の J から R は、フィールドの最下位にある時に TESTN で負数として扱われる ことに注意してください。 テストの結果として、結果の標識は次の場合にオンに設定されます。

- *71* 桁目と *72* 桁目*:* 結果フィールドに数字が入っている場合。 低位の文字は A から R の文字である可能性があります。これらの文字は C、D、または F のゾーンを持ち、0 から 9 の 数字を持つためです。
- *73* 桁目と *74* 桁目*:* 結果フィールドに数字と少なくと も 1 個のブランクが入っている場合。 例えば、値 b123 または bb123 でこの標識がオンに設定されます。 しかし、値 b1b23 は、組み込みブランクがあるために有効な数値フ ィールドとはならず、この値ではこの標識はオンに設定されません。

  注 : 文字フィールドには最低 1 桁の数字と 1 個の先行ブランクが入っていな ければならないので、長さが 1 桁のフィールドをテストする場合には、73 桁 目と 74 桁目に標識を指定することはできません。
- *75* 桁目と *76* 桁目*:* 結果フィールドにすべてブランク が入っている場合。

同じ標識を複数の条件に使用することができます。 いずれかの条件が存在す れば、標識がオンに設定されます。

TESTN 命令は、使用すると好ましくない結果や例外が生ずる命令 (例えば、 算術演算) を使用する前に、フィールドの妥当性検査を行うために使用 することができます。

詳しくは、「595 ページの『テスト命令』」を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  The field values are FieldA = 123, FieldB = 1X4, FieldC = 004,
 *  FieldD = ␣␣␣, FieldE = 1␣3, and FieldF = ␣12.
 *
 *  Indicator 21 is set on because FieldA contains all numeric
 *  characters.
C                   TESTN                   FieldA               21
 *  Indicator 22 is set on because FieldA contains all numeric
 *  characters.  Indicators 23 and 24 remain off.
C                   TESTN                   FieldA               222324
 *  All indicators are off because FieldB does not contain valid
 *  numeric data.
C                   TESTN                   FieldB               252627
 *  Indicator 28 is set on because FieldC contains valid numeric data.
 *  Indicators 29 and 30 remain off.
C                   TESTN                   FieldC               282930
 *  Indicator 33 is set on because FieldD contains all blanks.
 *  Indicators 31 and 32 remain off.
C                   TESTN                   FieldD               313233
 *  Indicators 34, 35, and 36 remain off.  Indicator 35 remains off
 *  off because FieldE contains a blank after a digit.
C                   TESTN                   FieldE               343536
 *  Indicator 38 is set on because FieldF contains leading blanks and
 *  valid numeric characters.  Indicators 37 and 39 remain off.
C                   TESTN                   FieldF               373839
```

図 *397. TESTN* 命令

TESTZ (ゾーンのテスト)

自由形式構文	(許可されていない - %BITAND 組み込み関数で X'F0' を使用し、文字のゾーン部分を分離します)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
TESTZ			文字フィールド	AI	JR	XX

TESTZ 命令では、結果フィールドの左端の文字のゾーンがテストされます。 結果フィールドは文字フィールドでなければなりません。 結果の標識は、このテストの結果に応じてオンに設定されます。 最低 1 つの結果の標識 (71 から 76 桁目) を指定しなければなりません。 71 桁目と 72 桁目の標識は、 文字 &、A から I、および文字 A と同じゾーンのいずれかの文字でオンに設定されます。 73 桁目と 74 桁目の標識は、文字 - (マイナス)、 J から R、および文字 J と同じゾーンのいずれかの文字でオンに設定されます。 75 桁目と 76 桁目の標識は、その他のゾーンの文字でオンに設定されます。

詳しくは、「595 ページの『テスト命令』」を参照してください。

TIME (時刻と日付の検索)

自由形式構文	(許可されていない ― %DATE、%TIME、および %TIMESTAMP 組み込み関数を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
TIME			ターゲット・フィールド			

TIME 命令は、プログラムの処理中の任意の時点でシステム時刻 またはシステム日付 (あるいはその両方) にアクセスします。 システム時刻は 24 時間時計に基づいています。

結果フィールドには次のいずれかを指定することができます。そこには、 時刻、または時刻とシステム日付が書き込まれます。

結果フィールド	戻り値	様式
6 桁の数値	時刻	hhmmss
12 桁の数値	時刻および日付	hhmmssDDDDDD
14 桁の数値	時刻および日付	hhmmssDDDDDDDD
時刻	時刻	結果の形式
日付	日付	結果の形式
タイム・スタンプ	タイム・スタンプ	*ISO

結果フィールドが数値フィールドである場合、時刻にだけアクセスする には、結果フィールドを 6 桁の 数値フィールドとして指定します。 時刻とシステム日付の両方をアクセスする場合には、結 果フィールドを 12 桁 (年の部分が 2 桁) または 14 桁 (年の部分が 4 桁) の数値フィールドとして指定します。 時刻は、常に結果フィールドの最初の 6 桁に次の形式で入れられます。

- hhmmss (hh = 時、mm = 分、および ss = 秒)

結果フィールドが数値フィールドで、システム日付が含まれる 場合には、結果フィールドの 7-12 桁目または 7-14 桁目に入れられます。 日付の形式は、日付の形式のジョブ属性 DATFMT によって異なり、mmddyy、ddmmyy、yymmdd、または年間通算日にすることがで きます。 年の部分が 2 桁の年間通算日形式では、7 から 8 桁目に年、9-11 桁目に日 (1-366、右寄せされて、未使用の高位桁には 0 が入る)、12 桁目に 0 が入ります。 年の部分が 4 桁の場合には、7-10 桁目に年、11-13 桁目に日 (1-366、右寄せ されて、未使用の高位桁にはゼロが入る)、14 桁目に 0 が入ります。

結果フィールドがタイム・スタンプである場合、マイクロ秒部分の 最後の 3 桁は常に 000 です。

注 : 特殊なフィールド UDATE および *DATE にはジョブ日付が入ります。 これらの値は、深夜を超えたりプログラムの実行中にジョブ日付が変わっても 更新されません。

詳しくは、「580 ページの『情報命令』」を参照してください。

```
D   Timeres       S               T   TIMFMT(*EUR)
D   Dateres       S               D   DATFMT(*USA)
D   Tstmpres      S               Z
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  When the TIME operation is processed (with a 6-digit numeric
 *  field), the current time (in the form hhmmss) is placed in the
 *  result field CLOCK.  The TIME operation is based on the 24-hour
 *  clock, for example, 132710.  (In the 12-hour time system, 132710
 *  is 1:27:10 p.m.)
C                   TIME                    Clock             6 0
 *  When the TIME operation is processed (with a 12-digit numeric
 *  field), the current time and day is placed in the result field
 *  TIMSTP.  The first 6 digits are the time, and the last 6 digits
 *  are the date; for example, 093315121579 is 9:33:15 a.m. on
 *  December 15, 1979.
C                   TIME                    TimStp           12 0
C                   MOVEL     TimStp        Time              6 0
C                   MOVE      TimStp        SysDat            6 0
 *  This example duplicates the 12-digit example above but uses a
 *  14-digit field.  The first 6 digits are the time, and the last
 *  8 digits are the date;   for example, 13120001101992
 *  is 1:12:00 p.m. on January 10, 1992.
C                   TIME                    TimStp           14 0
C                   MOVEL     TimStp        Time              6 0
C                   MOVE      TimStp        SysDat            8 0
 *  When the TIME operation is processed with a date field,
 *  the current date is placed in the result field DATERES.
 *  It will have the format of the date field.  In this case
 *  it would be in *USA format ie: D'mm/dd/yyyy'.
C                   TIME                    Dateres
 *  When the TIME operation is processed with a time field,
 *  the current time is placed in the result field TIMERES.
 *  It will have the format of the time field.  In this case
 *  it would be in *EUR format ie: T'hh.mm.ss'.
C                   TIME                    Timeres
 *  When the TIME operation is processed with a timestamp field,
 *  the current timestamp is placed in the result field TSTMPRES.
 *  It will be in *ISO format.
 *  ie: Z'yyyy-mm-dd-hh.mm.ss.mmmmmm'
C                   TIME                    Tstmpres
```

図 *398. TIME* 命令

UNLOCK (データ域のアンロックまたはレコードの解放)

自由形式構文	UNLOCK{(E)} 名前

コード	演算項目 1	演算項目 2	結果フィールド	標識		
UNLOCK (E)		名前 (ファイルまたはデータ域)		_	ER	_

UNLOCK 命令は、データ域をアンロック し、レコードのロックを解放するために使用します。

UNLOCK 例外 (プログラム状況コード 401 から 421、431、および 432) を処理するために、命令コード拡張 'E' またはエラー標識 ER を 指定できますが、両方を指定することはできません。エラー処理について詳しくは、163 ページの『プログラム例外/エラー』を参照してください。

71 桁目、72 桁目、75 桁目、および 76 桁目はブランクでなければなりません。

UNLOCK 命令に関する規則について詳しくは、571 ページの『データ域命令』を参照してください。

データ域のアンロック

名前 オペランドは、アンロックするデータ域の名前、または予約語 *DTAARA でなければなりません。

*DTAARA が指定されている場合には、プログラム中でロックされているすべてのデータ域がアンロックされます。

データ域は、既に *DTAARA DEFINE ステートメントの結果フィールドに指定されているか、定義仕様書の DTAARA キーワードで指定されている必要があります。名前 は内部データ域またはプログラム初期化パラメーター (PIP) データ域を参照してはなりません。 すでにアンロックされているデータ域に UNLOCK 命令を指定してもエラーには なりません。

詳しくは、「576 ページの『ファイル操作』」を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  TOTAMT, TOTGRS, and TOTNET are defined as data areas in the
 *  system.  The IN operation retrieves all the data areas defined in
 *  the program.  The program processes calculations, and
 *  then unlocks the data areas. The data areas can them be used
 *  by other programs.
 *
C     *LOCK         IN        *DTAARA
C                   :
C                   :
C                   UNLOCK    *DTAARA
C     *DTAARA       DEFINE                TOTAMT            8 2
C     *DTAARA       DEFINE                TOTGRS           10 2
C     *DTAARA       DEFINE                TOTNET           10 2
```

図 *399.* データ域のアンロック命令

レコード・ロックの解放

UNLOCK 命令によって、更新ディスク・ファイルの最後にロックされたレコ ードのロックを解放することもできます。

名前 は、UPDATE ディスク・ファイルの名前でなければなりません。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
FFilename++IPEASFRlen+LKlen+AIDevice+.Keywords+++++++++++++++++++++++++++++
 *
FUPDATA    UF   E             DISK
 *
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  Assume that the file UPDATA contains record format vendor.
 *  A record is read from UPDATA. Since the file is an update
 *  file, the record is locked. *IN50 is set somewhere else in
 *  the program to control whether an UPDATE should take place.
 *  otherwise the record is unlocked using the UNLOCK operation.
 *  Note that factor 2 of the UNLOCK operation is the file name,
 *  UPDATA, not the record format, VENDOR
 *
C                   READ      VENDOR                                  12
C                   :
C     *IN50         IFEQ      *ON
C                   UPDATE    VENDOR
C                   ELSE
C                   UNLOCK    UPDATA                               99
C                   ENDIF
```

図 *400.* レコードのアンロック操作

UPDATE (既存のレコードの変更)

自由形式構文	UPDATE{(E) } 名前 {データ構造 | %FIELDS(名前*{:*名前*...* *}*) }

コード	演算項目 1	演算項目 2	結果フィールド	標識		
UPDATE (E)		名前 (ファイルまたはレコード様式)	データ構造	_	ER	_

UPDATE 命令は、更新ディスク・ファイルまたはサブファイルから処理用に 検索された、最後にロックされたレコードを修正します。 レコードを検索する入力操作から UPDATE 命令までの間に、他の操作を実行し てはいけません。

名前 オペランドは、更新するファイルまたはレコード様式の名前でなければなりません。 外部記述ファイルの場合には、レコード様式名が必要です。 レコード様式名は、ファイルから読み取られる最後のレコードの名前でなけれ ばなりません。そうでない場合にはエラーが起こります。 プログラム記述ファイルの場合には、名前 オペランドとしてファイル名が必要です。

データ構造オペランドが指定されている場合、レコードはデータ構造から直接更新されます。 データ構造は、以下の規則に準拠する必要があります。

1. データ構造 オペランドが指定されている場合、 レコードはデータ構造から直接更新されます。
2. 名前 が プログラム記述ファイルを参照する場合、データ構造は、 宣言されたファイルのレコード長と同じ長さの任意のデータ構造にできます。
3. 名前 が外部記述ファイルまたは外部記述データベース・ファイルのレコード様式を参照する場合、データ構造は、同じファイルまたはレコード様式で、LIKEREC または EXTNAME キーワードの 2 番目のパラメーターとして *INPUT、*OUTPUT、または *ALL を指定して定義されたデータ構造、または LIKEREC キーワードに 2 番目のパラメーターが指定されていないデータ構造にする必要があります。
4. 名前 が外部記述ディスプレイ・ファイルのサブファイル・レコード様式を参照する場合、データ構造は、同じファイルまたはレコード様式で、LIKEREC または EXTNAME キーワードの 2 番目のパラメーターとして *OUTPUT または *ALL を指定して定義されたデータ構造にする必要があります。
5. データ構造の定義方法、およびデータ構造とファイルの間のデータ転送方法については、576 ページの『ファイル操作』を参照してください。

更新するフィールドのリストは %FIELDS を使用して指定できます。 %FIELDS のパラメーターは、更新するフィールド名のリストです。 フィールドの更新の図については、642 ページの『%FIELDS (更新するフィールド)』 の終わりにある例を参照してください。

UPDATE 例外 (ファイル状況コードが 1000 より大きい) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、146 ページの『ファイル例外/エラー』を参照してください。

UPDATE 命令を使用する場合には、次のことに留意してください。

- 名前 がレコード様式名の場合、レコード定義の中のこのフィールドのプログラムにおける現行値を使用して、レコードが修正されます。
- レコードの (すべてのフィールドでなく) 一部のフィールドを更新する場 合には、UPDATE 命令を使用しないで出力仕様を使用するか、あるいは %FIELDS を使用して 更新対象のフィールドを識別してください。
- ファイルまたはレコードに UPDATE を出す前に、同じファイルまたはレコ ードに、ロックを伴う有効な入力命令 (READ、READC、READE、READP、 READPE、CHAIN、または 1 次/2 次ファイル) を出す必要があります。 読み取り操作でエラー条件が戻された場合、またはロックなしで読み取られて場 合には、そのレコードはロックされないので、UPDATE を出すことはできません。 このレコードは、ロック要求を指定するために、命令拡張をデフォルトの値のブラ ンクにしてもう一度読み取らなければなりません。
- 同じファイルまたはレコードに連続して UPDATE 命令を使用することはで きません。 更新するレコードを位置決めしてロックするためには、間に正常な読み取り命令 を出さなければなりません。
- 合計演算中に 1 次または 2 次ファイルに UPDATE 命令を使用する場合には 注意してください。 RPG IV サイクルのこの段階では、現行レコード (処理しようとしているレコード) のフィールドは、まだ処理域に転送されていません。 したがって、UPDATE 命令では現行レコードが前のレコードのフィールドで更新 されます。 また、現行レコードのフィールドが処理域に転送される場合にも、 それらは前のレコードから更新されたフィールドになります。

- 複数装置ファイルの場合、名前 オペランドとしてサブファイル・レコード様式を指定します。 この命令は、ファイル仕様書に DEVID キーワードを使用して指定されたフィールド名で識別されるプログラム装置で処理されます。 プログラム装置が指定されていない場合には、最後の正常な入力命令で使用さ れた装置が使用されます。 この装置は、入力命令 (UPDATE 命令に先行する必要がある) で指定された装置 と同じでなければなりません。 入力命令から UPDATE 命令までの間に、他の装 置に対する入出力命令を処理してはいけません。 そのようなことをすると、 UPDATE 命令は正常に実行されなくなります。
- 複数のサブファイル・レコード様式を持つ表示装置ファイルの場合には、 1 つのサブファイル・レコードに対する更新のための読み取り命令を、同じ表示 装置ファイルの別のサブファイルに対する入力命令から UPDATE 命令までの間 に処理してはいけません。 そのようなことをすると、UPDATE 命令は正常に実行されなくなります。
- サブファイル・レコード様式に対する UPDATE 命令は、サブファイル・ レコード様式名に対する正常な入力命令 (READC、CHAIN) が少なくとも 1 回 (中間に別の様式名に対する入力命令を入れないで) 行われた場合に有効となります。 更新レコードは、最後の正常な入力操作で検索されたレコードとなります。 このことは、1 つのレコードが正常に読み取られて、次に同じ形式が正常に読 み取られなかった場合に、更新は行われるが、最初のレコードが更新される ことを意味しています。 これは DISK ファイルの場合とは異なります。

  正しくないレコードの更新を避けるために、結果の標識またはレコード識別 標識を調べて、正常な入力操作が行われた後で更新命令が実行されることを 確認してください。

ヌル値を含むヌル値可能フィールドを持つレコードの更新については、286 ページの『データベースのヌル値サポート』を参照してください。

詳しくは、「576 ページの『ファイル操作』」を参照してください。

WHEN (真の場合に選択)

自由形式構文	WHEN{(MR)} 標識式

コード	演算項目 1	拡張演算項目 2
WHEN (M/R)		標識式

WHEN 命令コードは、SELECT 命令の中の行の処理を制御する点で WHENxx と似 ています。 条件が、標識式 オペランドの論理式によって指定される点では異なっています。 WHEN 命令で制御される命令は、標識式 オペランドの式が真になった時に実行されます。 式について詳しくは、596 ページの『式』を参照してください。命令拡張 M および R の使用法については、608 ページの『数値演算の精度の規則』を参照してください。

詳しくは、567 ページの『比較命令』または 591 ページの『構造化プログラミング命令』を参照してください。

```
CL0N01Factor1+++++++Opcode(E)+Extended-factor2++++++++++++++++++++++++++..
 *
C                   SELECT
C                   WHEN    *INKA
C                   :
C                   :
C                   :
C                   WHEN    NOT(*IN01) AND (DAY = 'FRIDAY')
C                   :
C                   :
C                   :
C                   WHEN    %SUBST(A:(X+4):3) = 'ABC'
C                   :
C                   :
C                   :
C                   OTHER
C                   :
C                   :
C                   :
C                   ENDSL
```

図 *401. WHEN* 命令

WHENxx (真の場合に選択)

自由形式構文	(許可されていない - WHEN 命令コードを使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
WHENxx	被比較値	被比較値				

選択グループの WHENxx 命令は、872 ページの『SELECT (選択グループの始め)』 命令が処理された後でどこに制御を渡すかを決定します。

WHENxx 条件付き命令は、演算項目 1 と演算項目 2 に xx に よって指定された関係がある場合に真となります。条件が真の場合には、WHENxx の 後の命令が、 次の WHENxx、OTHER、ENDSL、 または END 命令まで処理されます。

WHENxx 命令を実行する場合には、次のことに留意してください。

- この命令グループが処理された後で、ENDSL 命令の後のステートメントに 制御が渡されます。
- 複雑な WHENxx 条件は、ANDxx および ORxx を使用してコーディングすることができます。 演算は、WHENxx、ANDxx、および ORxx 命令の組み合わせによって指定された条件が 真である場合に処理されます。
- WHENxx は空にすることができます。
- 合計演算の中では、制御レベルの指定 (7 から 8 桁目) は ブランクにするか、あるいはプログラムの該当するセクション内のステートメントを グループにまとめる L1-L9 標識、LR 標識、または L0 の指定を入れることができます。 制御レベルの指定は文書化のためだけのものです。 条件付け標識の指定 (9 から 11 桁目) は使用できません。

xx に使用できる値については、567 ページの『比較命令』を参照してください。

詳しくは、「591 ページの『構造化プログラミング命令』」を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 * The following example shows nested SELECT groups.  The employee
 * type can be one of 'C' for casual, 'T' for retired, 'R' for
 * regular, and 'S' for student.  Depending on the employee type
 * (EmpTyp), the number of days off per year (Days) will vary.
 *
C                   SELECT
C     EmpTyp        WHENEQ    'C'
C     EmpTyp        OREQ      'T'
C                   Z-ADD     0             Days
C     EmpTyp        WHENEQ    'R'
 *
 * When the employee type is 'R', the days off depend also on the
 * number of years of employment.  The base number of days is 14.
 * For less than 2 years, no extra days are added.  Between 2 and
 * 5 years, 5 extra days are added.  Between 6 and 10 years, 10
 * extra days are added, and over 10 years, 20 extra days are added.
 *
C                   Z-ADD     14            Days
 * Nested select group.
C                   SELECT
C     Years         WHENLT    2
C     Years         WHENLE    5
C                   ADD       5             Days
C     Years         WHENLE    10
C                   ADD       10            Days
C                   OTHER
C                   ADD       20            Days
C                   ENDSL
 * End of nested select group.
C     EmpTyp        WHENEQ    'S'
C                   Z-ADD     5             Days
C                   ENDSL
```

図 *402. WHENxx* 命令

```
 *----------------------------------------------------------------
 * Example of a SELECT group with complex WHENxx expressions.  Assume
 * that a record and an action code have been entered by a user.
 * Select one of the following:
 *   - When F3 has been pressed, do subroutine QUIT.
 *   - When action code(Acode) A (add) was entered and the record
 *     does not exist (*IN50=1), write the record.
 *   - When action code A is entered, the record exists, and the
 *     active record code for the record is D (deleted); update
 *     the record with active rec code=A.  When action code D is
 *     entered, the record exists, and the action code in the
 *     record (AcRec) code is A; mark the record as deleted.
 *   - When action code is C (change), the record exists, and the
 *     action code in the record (AcRec) code is A; update the record.
 *   - Otherwise, do error processing.
 *----------------------------------------------------------------
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
C     RSCDE         CHAIN     FILE                             50
C                   SELECT
C     *INKC         WHENEQ    *ON
C                   EXSR      QUIT
C     Acode         WHENEQ    'A'
C     *IN50         ANDEQ     *ON
C                   WRITE     REC
C     Acode         WHENEQ    'A'
C     *IN50         ANDEQ     *OFF
C     AcRec         ANDEQ     'D'
C     Acode         OREQ      'D'
C     *IN50         ANDEQ     *OFF
C     AcRec         ANDEQ     'A'
C                   MOVE      Acode         AcRec
C                   UPDATE    REC
C     Acode         WHENEQ    'C'
C     *IN50         ANDEQ     *OFF
C     AcRec         ANDEQ     'A'
C                   UPDATE    REC
C                   OTHER
C                   EXSR      ERROR
C                   ENDSL
```

WRITE (新しいレコードの作成)

自由形式構文	WRITE{(E) } 名前 {データ構造 }

コード	演算項目 1	演算項目 2	結果フィールド	標識		
WRITE (E)		名前 (ファイルまたはレコード様式)	データ構造	_	ER	EOF

WRITE 命令は新しいレコードをファイルに書き出します。

名前オペランドは、プログラム記述ファイルの名前か、または外部記述ファイルから取得されるレコード様式の名前である必要があります。

データ構造オペランドが指定されている場合、レコードはデータ構造から直接ファイルに書き出されます。名前が プログラム記述ファイルを参照している場合、データ構造は、 宣言されたファイルのレコード長と同じ長さのデータ構造である 必要があります。 名前 が外部記述ファイルのレコード様式を参照する場合、データ構造はタイプ EXTNAME(...:*OUTPUT or *ALL) または LIKEREC(...:*OUTPUT or *ALL) で定義されているデータ構造にする必要があります。 DISK ファイルでは、データ構造が LIKEREC で定義され、フィールドのタイプが指定されない場合、出力バッファー・レイアウトが入力バッファー・レイアウトと完全に一致すれば、そのデータ構造を WRITE 命令に使用できます。 例については、906 ページの図 404 を参照してください。 データ構造の定義方法、およびファイルとデータ構造の間のデータ転送方法については、576 ページの『ファイル操作』を参照してください。

WRITE 例外 (ファイル状況コードが 1000 より大きい) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。 オーバーフローが外部記述印刷ファイ

ルに達して、 ファイル仕様書にオーバーフロー標識が指定されていない場合には、エラーが発生します。 エラー処理について詳しくは、146 ページの『ファイル例外/エラー』を参照してください。

75-76 桁目に、WRITE 命令でファイルの終わりになった (サブファイルがいっぱいに なった) かどうかを知らせる標識を指定することができます。 この標識は WRITE 命令 が実行されるたびにオン (EOF 条件) またはオフに設定されます。 この情報は %EOF 組み込み関数からも入手することができます。この関数 は、EOF 条件が発生した場合に '1' を戻します。それ以外の場合には '0' を戻します。

WRITE 命令を使用する場合には、次のことに留意してください。

- 名前 がレコード様式名の場合、レコード定義の中のすべてのフィールドのプログラムにおける現行値を使用して、レコードが構成されます。
- 相対レコード番号を使用するレコードがファイルに書き出される場合には、RECNO ファイル仕様書キーワード (相対レコード番号) で指定されたフィールド 名を、書き出されるレコードの相対レコード番号が入るように更新しなければ なりません。
- 入力可能または更新可能な DISK ファイル (自由形式ファイル定義の USAGE キーワード または固定形式ファイル定義の 17 桁目を参照) にレコードを追加するために WRITE 命令 を使用する場合、そのファイルが出力可能 (自由形式定義の USAGE キーワードに *OUTPUT を指定するか、 または、ファイル記述仕様書の 20 桁目に A を指定します) でもあるように定義する 必要があります。
- 装置従属の機能は制限されます。 例えば、"PRINTER" 装置に "WRITE" が出されると、ファイル仕様書にキーワード PRTCTL が指定されていないと (通常は、行送りやスキップ情報は出力仕様の 41-51 桁目に指定される)、印刷後スペースは 1 に設定されます。 ファイルが外部記述の場合には、これらの機能は外部記述の一部です。
- 複数装置ファイルの場合には、データはファイル仕様書の DEVID キーワードで 指定されたフィールド名に指定されたプログラム装置 に書き出されます(369 ページの『DEVID(フィールド名)』を参照してください。) DEVID キーワードが指定されていない場合には、データは、最後の正常な入力 操作が処理されたプログラム装置に書き出されます。

ヌル値を含むヌル値可能フィールドを持つレコードの追加については、286 ページの『データベースのヌル値サポート』を参照してください。

詳しくは、「576 ページの『ファイル操作』」を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 *  The WRITE operation writes the fields in the data structure
 *  DS1 to the file, FILE1.
 *
C                   WRITE     FILE1         DS1
```

図 *403.* プログラム記述ファイルを使用した *WRITE* 命令

```
    dcl-f FILE1 DISK USAGE(*OUTPUT);

    dcl-ds likerec_default LIKEREC(FMT1);
    dcl-ds likerec_output LIKEREC(FMT1 : *OUTPUT);
    dcl-ds likerec_all LIKEREC(FMT1 : *ALL);

    WRITE FMT1 likerec_default;
    WRITE FMT1 likerec_output;
    WRITE FMT1 likerec_all;
```

図 *404. LIKEREC* データ構造を含む *DISK* ファイルに対する *WRITE* 命令

XFOOT (配列要素の合計)

自由形式構文	(許可されていない - %XFOOT 組み込み関数を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
XFOOT (H)		配列名	和	+	-	Z

XFOOT は、配列の要素を加算してその合計を結果フィールドとして指定されたフィールドに入れます。 演算項目 2 には、配列の名前が入ります。

四捨五入が指定されている場合には、すべての要素が合計された後で、 結果フィールドに結果が転送される前に丸めが行われます。 結果フィールドが演算項目 2 に指定された配列の要素である場合には、配列の合計の計算に XFOOT 命令の前の要素の値が使用されます。

配列が浮動の場合、XFOOT は次のように実行されます。すなわち、 配列が降順の場合、要素は逆順でまとめて追加されます。 降順でない場合、要素は、配列の最初の要素からまとめて追加されます。

XFOOT 命令に関する規則について詳しくは、557 ページの『算術演算』 または 561 ページの『配列命令』 を参照してください。

XFOOT 命令の例については、560 ページの図 181 を参照してください。

XLATE (変換)

自由形式構文	(許可されていない - %XLATE 組み込み関数を使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
XLATE (E P)	変換元:変換先	ソース・ストリング:開始	ターゲット・ストリング	_	ER	_

ソース・ストリング (演算項目 2) 内の文字が、変換元および変換先ストリング (両方とも演算項目 1 に入っている) に従って変換されて、受け入れフィールド (結果フィールド) に入れられます。 変換元ストリング中の一致するソース・ストリングの文字は、変換先ストリングの 対応する文字に変換されます。 変換元、変換先、ソース、およびターゲットのストリングは、同じタイプ (すべて文字、すべて図形、またはすべて UCS-2) でなければなりません。 同様に、それらの CCSID は、それらの CCSID の 1 つが 65535 でない限り、 または、図形フィールドの場合には、CCSID(*GRAPH: *IGNORE) が制御仕様書で指定されたのでない限り、同じでなければなりません。

XLATE では、ソース・ストリングの変換は演算項目 2 に指定された位置から開始さ れて、左から右へ 1 文字ずつ続けられます。 ソース・ストリングの文字が変換元ストリングの中に存在する場合には、変換先ストリングの対応する文字が結果フィールドに入れられます。 ソース・フィールドの開始位置より前の文字は、変更されないで結果フィールドに 入れられます。

演算項目 1 には、変換元ストリングの後にコロンを付けその後に 変換先ストリングを入れなければなりません。 変換元ストリングと変換 先ストリングには、フィールド名、配列要素、名前のついた定数、データ構造名、リテラル、またはテーブル名のいずれかを入れなければなりません。

演算項目 2 には、ソース・ストリングかまたはソース・ストリングにコロンと開始位置 を指定したもののいずれかを入れなければなりません。 演算項目 2 の ソース・ストリング部分は、フィールド名、配列要素、名前の付いた 定数、データ構造名、データ構造サブフィールド、リテラル、またはテーブル名 のいずれかを入れることができます。 この命令で図形データまたは UCS-2 データが使用される場合には、開始位置は 2 バイト文字によって参照されます。 演算項目 2 の開始位置部分は、小数点以下の桁数がゼロの数値でなければなら ず、名前のついた定数、配列要素、フィールド名、リテラル、またはテー ブル名とすることができます。 開始位置が指定されていない場合には、値 1 が使用されます。

結果フィールドは、フィールド、配列要素、データ構造、またはテーブルのいずれでもかまいません。 結果フィールドの長さは、演算項目 2 に指定されたソース・ストリングと同じである必要があります。 結果フィールドがソース・ストリングより大きい場合には、結果は左寄せされます。 結果フィールドがソース・ストリングより短い場合には、結 果フィールドに、変換済みのソース・ストリングの左端部分が入ります。 結果のフィールドが可変長の場合、その長さは変わりません。

変換元ストリングの文字が重複している場合には、最初に現れたもの (左端) が使用されます。

注 : 演算項目 1、演算項目 2、または結果フィールドに形象定数を使用することは できません。 演算項目 1 と結果のフィールドまたは演算項目 2 と結果のフィールドに、デー タ構造のオーバーラップがあってはなりません。

変換元ストリングが変換先ストリングより長い場合、変換元ストリングにある余分な文字は無視されます。

7 から 11 桁目には、有効な標識を指定することができます。

演算項目 2 が結果フィールドより短い場合には、命令拡張桁に指定された P によって、変換後に結果フィールドの右側にブランクが埋め込むように指示 されます。 結果フィールドが図形で P が指定されている場合には、図形のブランクが使用 されます。 結果フィールドが UCS-2 で P が指定されている場合には、UCS-2 のブランクが使用されます。

XLATE 例外 (プログラム状況コード 100) を 処理するために、命令コード拡張 'E' またはエラー標識 ER を指定できますが、 両方を指定することはできません。エラー処理について詳しくは、163 ページの『プログラム例外/エラー』を参照してください。

75-76 桁目はブランクでなければなりません。

詳しくは、「589 ページの『ストリング命令』」を参照してください。

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq....
 *
 * The following translates the blank in NUMBER to '-'.  The result
 * in RESULT will be '999-9999'.
 *
C                   MOVE      '999 9999'    Number            8
C     ' ':'-'       XLATE     Number        Result            8
```

図 *405. XLATE* 命令

```
*...1....+....2....+....3....+....4....+....5....+....6....+....7...+....
DName++++++++++ETDsFrom+++To/L+++IDc.Keywords++++++++++++++++++++++++++++++
D Up              C                   'ABCDEFGHIJKLMNOPQRS-
D                                     TUVWXYZ'
D Lo              C                   'abcdefghijklmnopqrs-
D                                     tuvwxyz'
CL0N01Factor1+++++++Opcode(E)+Factor2+++++++Result++++++++Len++D+HiLoEq..
 *
 * In the following example, all values in STRING are translated to
 * lowercase.  As a result, RESULT='rpg dept'.
 *
C                   MOVE      'RPG DEPT'    String            8
C     Up:Lo         XLATE     String        Result
 *
 * In the following example only part of the string is translated
 * to lowercase.  As a result, RESULT='RPG Dept'.
 *
C     Up:Lo         XLATE     String:6      Result
```

図 *406.* 名前のついた定数を使用した *XLATE* 命令

XML-INTO (XML 文書の変数への構文解析)

自由形式構文	XML-INTO{(EH) } 受信者 %XML(*xmlDoc* {: オプション }); XML-INTO{(EH) } %HANDLER(*handlerProc* : *commArea*) %XML(*xmlDoc* {: オプション });

コード	演算項目 1	拡張演算項目 2
XML-INTO		受信者 %XML(*xmlDoc* {: オプション })

コード	演算項目 1	拡張演算項目 2
XML-INTO		%HANDLER(*handlerProc* : *commArea*) %XML(*xmlDoc* {: オプション })

ヒント: XML および XML 文書の処理の基本概念について十分理解していない場合は、以下のセクションを読み進める前に、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」のセクション『XML 文書の処理』を参照することをお勧めします。

XML-INTO は、以下の 2 つの方法で作動します。

- RPG 変数に直接 XML データを読み取る
- %HANDLER(handlerProc) により指定されたプロシージャーに渡される配列パラメーターに対して段階的に XML データを読み取る。

第 1 オペランドには、構文解析されるデータのターゲットを指定します。 変数名または %HANDLER 組み込み関数を使用できます。

第 2 オペランドは %XML 組み込み関数にする必要があり、構文解析される XML 文書およびその構文解析方法を制御するオプションを識別します。 %XML について詳しくは、708 ページの『%XML (xmlDocument {:options})』 を参照してください。

第 1 オペランドが変数名である場合は、以下のようになります。

- 構文解析は、変数に対して直接実行される。
- 変数の名前は、構文解析する XML 要素の名前を設定するために使用される。 これは path オプションを使用してオーバーライドできます。
- 変数がデータ構造である場合は、その命令がエラーで終了した場合でも、一部のサブフィールドはその命令により設定される。
- 変数が配列である場合、構文解析は配列に適合する量のデータに対してのみ検索を行う。 PSDS の 372 から 379 桁目にある「XML 要素の数」サブフィールドには、命令により正常に設定された要素の数が設定されます。 データ構造の配列の場合、その要素のサブフィールドのデータの構文解析中に構文解析エラーが発生したときは、この値には設定中の要素が含まれません。 ただし、この配列要素には、命令により設定されたそのサブフィールドの一部が含まれています。

第 1 オペランドが %HANDLER 組み込み関数である場合は、以下のようになります。

- %HANDLER の第 1 オペランドとして指定されたプロシージャーは、そ のプロシージャーにより処理される RPG 配列要素の指定された数を充てんす るのに十分な数の XML データがパーサーにより構文解析される と呼び出される。 ハンドラーから戻ると、配列要素の指定された数が再度充てんされ、処理プロシージャーを呼び出すために十分な数の XML データが構文解析されるまで、パーサーは XML データの構文解析を継続します。 これは、その文書が完全に構文解析されるまで、または構文解析が停止されたことを示す戻りコードをそのプロシージャーが戻すまで継続されます。

  処理プロシージャーの最後の呼び出しでは、RPG 配列要素数が処理プロシージャーが処理する数より少ない場合があります。 処理プロシージャーは常に「要素の数」パラメーターを参照し、XML データがない配列要素にアクセスしないようにする必要があります。

  %HANDLER の第 2 オペランドとして指定される通信域変数は、処理プロシ ージャーへの最初のパラメーターとしてパーサーにより受け渡されます。 これにより、XML-INTO 命令をコーディングするプロシージャーは処理プロ シージャーと通信でき、さらに処理プロシージャーはある呼び出しから次の呼び出しに情報を保管できます。
- プロシージャーの配列パラメーターの保持に使用される一時変数の各要素は、そのデフォルト値に初期化された後に XML データからロードされる。
- 検索対象の XML 要素の名前を指定するには、*path* オプションを使用する必要がある。 *path* オプションについて詳しくは、920 ページの『DATA-GEN、DATA-INTO、および XML-INTO の %DATA および %XML オプション』および 912 ページの『期待される XML データの形式』を参照してください。
- 配列処理プロシージャーは、XML-INTO 命令の実行中に複数回呼び出される場合がある。 パーサーが第 2 パラメーターの DIM キーワードにより指定された要素の数を検出すると、そのプロシージャーが呼び出

されます。 そのプロシージャーが呼び出される最後の回では、要素の数が DIM キーワードで指定された数より少ない場合があります。 要素がない場合、そのプロシージャーは呼び出されません。

処理プロシージャーでは、以下のパラメーター数および戻り値の型を指定 する必要があります。

パラメーター数または戻り値	データ型および引き渡しモード	記述
戻り値	4 バイト整数 (10I 0)	戻り値がゼロの場合、構文解析が継続することを示します。 戻り値がその他の値の場合、構文解析が終了することを示します。
1	任意の型、参照による受け渡し	XML-INTO 命令とハンドラーの間、およびハンドラーの連続呼び出しの間の通信に使用される。
2	配列またはデータ構造の配列、読み取り専用の参照による受け渡し (CONST キーワード)	配列要素には、*path* オプションにより指定された XML 要素からのデータが含まれています。
3	4 バイト符号なし整数 (10U 0)、値による受け渡し	XML データを表す第 2 パラメーター内の配列要素の数。

- %HANDLER について詳しくは、648 ページの『%HANDLER (handlingProcedure : communicationArea)』を参照してください。

データ構造のサブフィールドは、XML 文書内にある順序で設定されます。 その順序は、データ構造内でサブフィールドのオーバーラップがある場合に重要になることがあります。

%NULLIND は、XML-INTO 命令の実行中にはフィールドまたはサブフィールドに対して更新されません。

命令拡張 H を指定すると、数値データを四捨五入して代入できます。 命令拡張 E を指定すると、以下の状況コードを処理できます。

00351
XML 構文解析時のエラー。

00352
無効な XML オプション。

00353
XML 文書が RPG 変数に一致しない。

00354
XML 構文解析の準備時エラー。

注 : 命令拡張は、自由形式構文が使用された場合にのみ指定できます。

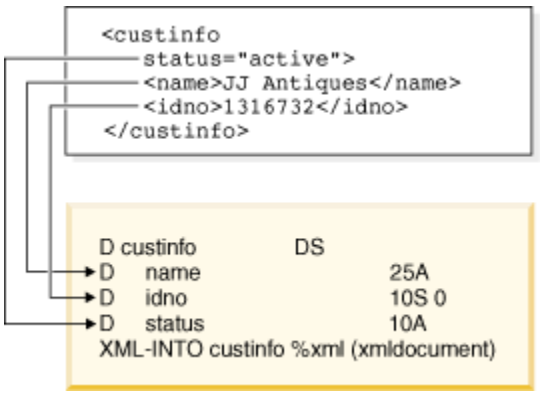
状況 00351 の場合、パーサーからの戻りコードは、PSDS の 368 から 371 桁目のサブフィールド「外部戻りコード」に配置されます。 このサブフィールドは、命令の開始時にゼロに設定され、命令の終了時にはパーサーにより戻された値に設定されます。

%XML 組み込み関数のオプション・パラメーターにおいて不明、無効、または無関係のオプションが検出された場合、その命令は状況コード 00352 (XML オプション・エラー) で失敗します。 PSDS の外部戻りコード・サブフィールドは、命令の開始時に設定された初期値のゼロから更新されません。

XML 文書は、XML の要素および属性の各名前に関して RPG 変数と一致することが期待されます。

- RPG データ構造の XML データには、そのデータ構造と同じ名前の XML 要素、および RPG サブフィールドと同じ名前の子要素および/または属性があることが期待されます。
- RPG 配列の XML データには、RPG 配列と同じ名前の一連の要素があることが期待されます。

path オプションを使用すると、指定された変数の名前と一致する XML 要素の名前を設定できます。 ただし、指定された変数のサブフィールドに一致する XML 要素および/または属性の名前の設定には使用できません。 例えば、変数 DS1 にサブフィールド SF1 がある場合、DS1 の XML 要素には任意の名前を使用できますが、SF1 の XML 要素または属性には名前「sf1」(または *case* オプションに応じて「SF1」、「Sf1」など) を使用する必要があります。



RPG 変数が配列またはデータ構造の配列である場合、または %HANDLER 組み込み関数が指定された場合、配列要素に対応する XML 要素は別の XML 要素に含まれることが期待されます。 デフォルトで、各 XML 要素は、文書の最外部の XML 要素の子要素であることが期待されます。 *path* オプションを使用すると、配列要素に対応する XML 要素への実際のパスを指定できます。 例えば、最外部の XML 要素の名前が「transaction」であり、それに「parts」という名前の子要素があり、さらにその中に「part」という名前の複数の子要素がある場合、配列に「part」XML 要素を読み込むには、オプション「path=transaction/parts/part」を指定します。

```
<transaction>
  <parts>
     <part type = "bracket" size="15" num="100"/>
     <part type="frame" size="2" num="500"/>
  </parts>
<transaction>
```

XML 文書が RPG 変数に一致しない場合、例えば XML 文書にデフォルト のパスまたは指定されたパスがない場合、または RPG データ構造のサブフィ ールドに一致する一部の XML 要素または属性が欠落している場合、XML-INTO 命令は状況 00353 で失敗します。 *allowextra* および *allowmissing* オプションを使用すると、XML 要素のデータを RPG 変数を完 全に設定するために必要なデータよりも多くするかまたは少なくするかを指定できます。

一部の RPG データ型では、XML 属性を指定して、XML データを RPG 変数に代入する方法を制御できます。これらの属性について詳しくは、916 ページの『XML-INTO および DATA-INTO の RPG 変数にデータを転送する場合の規則』を参照してください。

事前定義の参照 &amp、&apos、&lt、&gt、&quot、 または 16 進数ユニコード参照 &#xxxx 以外の XML 参照が検出された場 合、結果にはその参照自身が「&refname;」の形式で含まれます。 この値がデータ型に対して無効である場合、その命令は失敗します。 例えば、XML 要素に値 <data>1&decpoint;50/data> がある場合、「1」、「&decpoint;」、および「0」の 3 つの部分からストリング「1&decpoint;50」が作成されます。 このデータは、文字または UCS-2 変数に対して有効ですが、他の型に変換された場合はエラーが発生します。

ヒント: XML データにそのような参照が含まれていることが分かっている場合は、XML-INTO 命令の完了後に、文字データおよび UCS-2 データを検査して、参照が存在するか、%SCANRPL、 または %SCAN と %REPLACE などのストリング命令を使用して置換した参照の値が正しいかを検査する必要があります。

XML データがそれに一致する RPG 変数の型に対して無効である場合は、その命令は状況コード 0353 で失敗します。 代入エラーに固有の状況コードがメッセージ RNX0353 の置換テキストに表示されます。

ヒント: 日付や数値などの型の RPG フィール ドにデータを正常に代入できなかったことで失敗する XML-INTO 命令を防止するには、文字または UCS-2 の型のすべてのサブフィールドに受信側変数を定義します。それにより、データは、変換組み込み関数 %DATE や %INTT などを使用して RPG プログラムにより別のデータ型に変換されます。

XML-INTO 命令は、DOCTYPE 宣言を無視します。 DOCTYPE 宣言には、プログラムにより手動で処理する必要があるエンティティー参照の値が含まれている場合があります。 XML 文書の DOCTYPE 宣言を使用する場合は、XML-SAX 命令を使用します。 XML-SAX 処理プロシージャーでは、DOCTYPE 宣言値が検出されるか、または DOCTYPE 宣言が使用されていないと判断されると、構文解析を停止できます。

XML-INTO について詳しくは、以下のリンクを参照してください。

- 920 ページの『DATA-GEN、DATA-INTO、および XML-INTO の %DATA および %XML オプション』
- 912 ページの『期待される XML データの形式』
- 916 ページの『XML-INTO および DATA-INTO の RPG 変数にデータを転送する場合の規則』
- 917 ページの『XML-INTO 命令の例』

期待される XML データの形式

XML 要素の構造は、RPG 変数の構造と一致することが必要です。

- RPG 変数に一致する XML 要素のネスティング・レベルは、XML 文書の任意のレベルに設定可能ですが、その XML 要素が文書の想定されたネスティング・レベルにない場合は *path* オプションを指定する必要があります。 *path* オプションが指定されない場合は、以下の前提条件が適用されます。
  - 非配列の変数 (テーブル名および複数回繰り返しデータ構造など) の 場合、文書の要素 (最外部の XML 要素) は RPG 変数に一致する XML 要素であることが前提です。 最外部の XML 要素の名前が RPG 変数の名前と同じではない場合は、*path* オプションを指定して、使用される XML 要素を指定する必要があります。
  - 配列変数の場合、文書の要素 (最外部の XML 要素) の直接の子は RPG 変数に一致する XML 要素であることが前提です。
- RPG サブフィールドに一致する XML 要素は以下の場合があります。
  - RPG サブフィールドの親データ構造に一致する XML 要素の XML 属性 (それ自体はデータ構造ではないサブフィールドの場合のみ)
  - サブフィールドが含まれているデータ構造に一致する XML 要素の直接の子 XML 要素
- RPG 配列に一致する XML 要素は、同じ親 XML の子である必要があります。 それらの子要素が XML 文書に一緒に存在する必要はなく、別の要素を挿入することも可能です。

注 : XML 処理命令は、XML-INTO では無視されます。 処理命令の形式は以下のとおりです。

```
<?targetname data value ?>
```

スカラー変数

```
D libname      S             10A
 /free
      XML-INTO libname %XML(xmldoc : option)
```

XML-INTO libname の XML の例	***path* オプション**
<libname>*data*</libname>	
<library>*data*</library>	'path=library'
<info><library>*data*</library></info>	'path=info/library'

配列要素

```
D sites        S             25A   DIM(3)
 /free
      XML-INTO sites(n) %XML(xmldoc : option)
```

XML-INTO sites の XML の例	***path* オプション**
<sites>*data*</sites>	ブランク
<custsites>*data*</custsites>	'path=custsites'
<info><sites>*data*</sites></info>	'path=info/sites'

テーブル名

```
D tabname      S             10A   DIM(5)
 /free
      XML-INTO tabname %XML(xmldoc : opts)
```

XML-INTO tabname の XML の例	path オプション
<tabname>*data*</tabname>	ブランク
<library>*data*</library>	'path=library'
<info><library>*data*</library></info>	'path=info/library'

単純なデータ構造または複数回繰り返しデータ構造

注 : 例における XML データでは、判読しやすいように改行およびインデントが使用されています。 XML データは、任意の便利な方式で形式化できます。

```
D pgm           DS
D   name                  10A
D   lib                   10A

OR

D pgm           DS                  OCCURS(5)
D   name                  10A
D   lib                   10A
 /free
      XML-INTO pgm %XML(xmldoc : option)
```

XML-INTO pgm の XML の例	path オプション
<pgm> <name>*data*</name> <lib>*data*</lib> </pgm>	ブランク
<program> <name>*data*</name> <lib>*data*</lib> </program>	'path=program'
<api> <program> <name>*data*</name> <lib>*data*</lib> </program> </api>	'path=api/program'

注: サブフィールド情報は、XML 要素または XML 属性により提供されます。 データ構造のサブフィールドに XML を指定するその他の有効な方法を以下に示します。 スカラー・サブフィールドの XML データを作成する場合には、属性または要素のいずれかを自由に使用して XML 文書を設計できます。

```
 <pgm name="data" lib="data"/>
OR
 <pgm name="data">
   <lib>data</lib>
 </pgm>
```

スカラー型の配列

```
D sites         S             25A   DIM(3)
 /free
      XML-INTO sites %XML(xmldoc : option)
```

XML-INTO sites の XML の例	path オプション
<anything> <sites>data</sites> <sites>data</sites> <sites>data</sites> </anything>	ブランク
<info> <custsites>data</custsites> <custsites>data</custsites> <custsites>data</custsites> </info>	'path=info/custsites'

データ構造の配列

```
D pgm          DS                  DIM(3) QUALIFIED
D   name                   10A
D   lib                    10A
 /free
      XML-INTO pgm %XML(xmldoc : option)
```

XML-INTO pgm の XML の例	path オプション
<anything> <pgm name="name1" lib="lib1"/> <pgm><name>name2</name> <lib>lib2</lib></pgm> <pgm lib="lib3"><name>name3</pgm> </anything>	ブランク
<programs> <pgm name="name1" lib="lib1"/> <pgm><name>name2</name> <lib>lib2</lib></pgm> <pgm lib="lib3"><name>name3</pgm> </programs>	'path=programs/pgm'

注: 3 つの "pgm" XML 要素には、 XML 要素および XML 属性のさまざまな組み合わせで指定された名前およびライブラリー情報があります。 スカラー・サブフィールドの XML データを作成する場合には、属性または要素のいずれかを自由に使用して XML 文書を設計できます。

複雑なデータ構造

```
D qualname     DS                  QUALIFIED
D   name                   10A
D   lib                    10A
D dtaaraInfo   DS                  QUALIFIED
D   dtaara                         LIKEDS(qualname)
D   type                   10I 0
D   value                 100a
 /free
      XML-INTO dtaaraInfo %XML(xmldoc : option)
```

XML-INTO dtaaraInfo の XML の例	path オプション
<dtaarainfo> <dtaara> <name>data</name> <lib>data</lib> </dtaara> <type>data</type> <value>data</value> </dtaarainfo>	ブランク

XML-INTO dtaaraInfo の XML の例	path オプション
<sys> <obj> <dta> <dtaara> <name>data</name> <lib>data</lib> </dtaara> <type>data</type> <value>data</value> </dta> </obj> </sys>	'path=sys/obj/dta'

データ構造の配列があるハンドラー・プロシージャー

```
D myCommArea   DS
D   total                 20u 0
D custType     DS                 qualified
D   name                  50a    varying
D   id_no                 10i 0
D   city                  20a
D custHdlr     PR
D   commArea                     likeds(myCommArea)
D   custinfo                     likeds(custType) dim(5)
D   numElems              10u 0  const
 /free
      XML-INTO %HANDLER(custHdlr : myCommArea) %XML(xmldoc : option)
```

注: %HANDLER が指定された場合には *path* オプションが必要です。

XML-INTO %HANDLER(custHdlr:x) の XML の例	path オプション
<info> <cust> <name>data</name> <id_no>data</id_no> <city>data</city> </cust> <cust> <name>data</name> <id_no>data</id_no> <city>data</city> </cust> : : <cust> <name>data</name> <id_no>data</id_no> <city>data</city> </cust> </info>	'path=info/cust'

スカラー型の配列があるハンドラー・プロシージャー

```
D myCommArea   DS
D   total                 20u 0
D nameHdlr     PR
D   commArea                     likeds(myCommArea)
D   names                 10a    dim(5)
D   numNames              10u 0  const
 /free
      XML-INTO %HANDLER(nameHdlr : myCommArea) %XML(xmldoc : option)
```

注: %HANDLER が指定された場合には *path* オプションが必要です。

XML-INTO %HANDLER(nameHdlr:x) の XML の例	path オプション
<info> <name>data</name> <name>data</name> <name>data</name> <name>data</name> : : <name>data</name> <name>data</name> </info>	'path=info/names'

XML-INTO および DATA-INTO の RPG 変数にデータを転送する場合の規則

- 整数、符号なし整数、10 進数 (パック、ゾーン、バイナリー) および浮動小数点の各フィールドの場合、RPG が %INT、%UNS、%DEC、%FLOAT を使用するそれぞれの場合と同様の規則を使用してデータが転送されます。DATA-INTO 命令コードまたは XML-INTO 命令コードで四捨五入命令拡張が指定された場合は、%INTH、%UNSH および %DECH が使用されます。 制御キーワード EXPROPTS のオプション *ALWBLANKNUM および *USEDECEDIT がデータの処理方法に与える影響について詳しくは、569 ページの『組み込み関数を使用して文字値を数値に変換するための規則』を参照してください。
- 日付、時刻、およびタイムスタンプの各フィールドの場合、RPG が %DATE、%TIME および %TIMESTAMP を使用するそれぞれの場合と同じ規則を使用してデータが転送されます。 形式のデフォルトは、区切り記号を使用した *ISO です。

  形式は、要素内の属性 fmt によって指定できます。 属性の値は、それぞれの組み込み関数に対して有効な形式にする必要があり、先導するアスタリスクはオプションです。 RPG において複数の区切り記号が許可される形式の場合、デフォルトの区切り記号は、その形式に対する RPG のデフォルト区切り記号です。 例えば、日付フィールドの場合、以下の XML のフラグメントは有効です。

```
<myDate fmt="DMY/">25/12/04</myDate>   <!-- 2004-12-25 -->
<myDate fmt="Dmy">25.12.04</myDate>    <!-- 2004-12-25 -->
<myDate fmt="*cymd0">0971123</myDate>  <!-- 1997-11-23 -->
```

- 標識、文字、および UCS-2 の各フィールドの場合、必要に応じて適 切な CCSID 変換が実行されてデータが転送されます。 固定長フィールドは、デフォルトで左寄せで代入されます。

  要素内の属性 adjust に「left」または「right」のいずれかの値を使用することにより、調整を指定できます。 例えば、RPG 変数 data が 10 バイトの長さである場合、以下の XML データでは DATA の値が「␣␣␣␣␣abcde」 に設定されます。

```
<data adjust="right">abcde</data>
```

- グラフィック・フィールドの場合、必要に応じて適切な CCSID 変換が実行され、%GRAPH 組み込み関数と同様の規則を使用してデータが転送されます。 固定長フィールドは、デフォルトで左寄せで代入されます。 要素内の属性 adjust に「left」または「right」のいずれかの値を使用することにより、調整を指定できます。
- ポインターおよびプロシージャー・ポインターの各サブフィールドはサポートされていないため、DATA-INTO 命令と XML-INTO 命令では無視されます。
- 特別な属性 fmt および adjust は、一致した変数の代入に関係ない場合、または属性の値が無効である場合には通常の属性として扱われます。 例えば、以下の属性は通常の属性として扱われます。

  'fmt="abc"'
  「abc」は無効な形式です。

  'adjust=yes'
  「yes」は adjust 属性に対して無効な値です。

  'fmt="mdy/"'、数値フィールドで指定された場合

  「adjust=right」、可変長フィールドで指定された場合。

- 属性 fmt および adjust、およびその値は、case オプションにより指定された大/小文字で指定する必要があります。 以下の表には、case オプションの各値に対して有効な属性の例が示されています。

***case* オプション**	***fmt*、例 "*MDY/"**	***adjust*、例 "right"**
指定なし	`fmt="mdy/"` `fmt="*mdy/"`	adjust="right"
'case=lower'	`fmt="mdy/"` `fmt="*mdy/"`	adjust="right"
'case=upper'	`fmt="MDY/"` `fmt="*MDY/"`	ADJUST="RIGHT"
'case=any'	`Fmt="Mdy/"` `FMT="*mDY/"` `...`	`Adjust="Right"` `adjust="RIGHT"` `...`

XML-INTO 命令の例

```
D qualName        DS                  QUALIFIED
D   name                        10A
D   lib                         10A

D copyInfo        DS                  QUALIFIED
D   from                              LIKEDS(qualName)
D   to                                LIKEDS(qualName)

D toName          S             10A   VARYING

 //  Assume file cpyA.xml contains the following lines:
 //    <copyinfo>
 //     <to><name>MYFILE</name><lib>*LIBL</lib></to>
 //     <from name="MASTFILE" lib="CUSTLIB"></from>
 //    </copyinfo>
 /free
       // Data structure "copyInfo" has two subfields, "from"
       // and "to".  Each of these subfields has two subfields
       // "name" and "lib".
       xml-into copyInfo %XML('cpyA.xml' : 'doc=file');
       // copyInfo.from  .name = 'MASTFILE  ' .lib = 'CUSTLIB   '
       // copyInfo.to    .name = 'MYFILE    ' .lib = '*LIBL     '

       // Parse the "copyinfo/to/name" information into variable
       // "toName".  Use the "path" option to specify the location
       // of this information in the XML document.
       xml-into toName %XML('cpyA.xml'
                            : 'doc=file path=copyinfo/to/name';
       // toName = 'MYFILE'
```

図 *407.* ファイルから変数への直接の構文解析

```
D info            DS
D   name                          10A
D   val                            5I 0  DIM(2)
D xmlFragment     S             1000A    VARYING
D opts            S               20A    INZ('doc=string')
D dateVal         S               10A    INZ('12/25/04')
D format          S                4A    INZ('mdy/')
D mydate          S                 D    DATFMT(*ISO)

 /free

  // 1. Parsing into a data structure containing an array
  xmlFragment = '<info><name>Jill</name>'
              + '<val>10</val><val>-5</val></info>';
  xml-into info %XML(xmlFragment);
  // info now has the value
  //    name = 'Jill'
  //    val(1) = 10
  //    val(2) = -5

  // 2. Parsing into a date.  The "fmt" XML attribute indicates the
  //    format of the XML date.
  xmlFragment = '<mydate fmt="' + format + '">'
              + dateVal + '</mydate>';
  xml-into mydate %XML(xmlFragment);
  // xmlFragment = '<mydate fmt="mdy">12/25/04</mydate>'
  // mydate = 2004-12-25
```

図 *408.* ストリング変数から変数への直接の構文解析

```
 // DDS for "MYFILE"
 // A          R PARTREC
 // A            ID            10P 0
 // A            QTY           10P 0
 // A            COST           7P 2

 // XML data in "partData.xml"
 //  <parts>
 //   <part><qty>100</qty><id>13</id><cost>12.03</cost></part>
 //   <part><qty>9</qty><id>14</id><cost>3.50</cost></part>
 //   ...
 //   <part><qty>0</qty><id>254</id><cost>1.98</cost></part>
 //  </records>
Fmyfile    o    e             disk
D options         S            100A
D allOk           S               N

D partHandler     PR            10I 0
D   ok                            N
D   parts                           LIKEREC(partrec) DIM(10) CONST
D   numRecs                     10U 0 VALUE

:
:
 /free
   // Initiating the parsing
   options = 'doc=file path=parts/part';
   allOk = *ON;
   xml-into %HANDLER(partHandler : allOk)
            %XML('partData.xml' : options);
   // Check if the operation wrote the data
   // successfully
   if not allOk;
      // some output error occurred
   endif;
 /end-free

:
:
 // The procedure to receive the data from up to 10
 // XML elements at a time.  The first call to the
 // this procedure would be passed the following data
 // in the "parts" parameter:
 //   parts(1) .id = 13   .qty = 100  .cost = 12.03
 //   parts(2) .id = 14   .qty = 9    .cost = 3.50
 //   ...
 // If there were more than 10 "part" child elements in
 // the XML file, this procedure would be called more
 // than once.
P partHandler     B
D                 PI            10I 0
D   ok                           1N
D   parts                           LIKEREC(partrec) DIM(10) CONST
D   numRecs                     10U 0 VALUE

D i               S             10I 0
 * xmlRecNum is a static variable, so it will hold its
 * value across calls to this procedure.
 * Note: Another way of storing this information would be to
 * pass it as part of the first parameter; in that
 * case the first parameter would be a data structure
 * with two subfields: ok and xmlRecNum
```

図 *409.* 処理プロシージャーを使用した不明な数の *XML* 要素の構文解析

```
D xmlRecNum       S             10I 0 STATIC INZ(0)
 /free
   for i = 1 to numRecs;
      xmlRecNum = xmlRecNum + 1;
      write(e) partRec parts(i);
      // Parameter "ok" was passed as the second parameter
      // for the %HANDLER built-in function for the XML-INTO
      // operation.  The procedure doing the XML-INTO
      // operation can check this after the operation to
      // see if all the data was written successfully.
      if %error;
        // log information about the error
        logOutputError (xmlRecNum : parts(i));
        ok = *OFF;
      endif;
   endfor;

   // continue parsing
   return 0;
 /end-free
P                 E
```

XML 命令について詳しくは、596 ページの『XML 命令』を参照してください。

DATA-GEN、DATA-INTO、および XML-INTO の %DATA および %XML オプション

オプションを使用して、DATA-GEN、DATA-INTO、および XML-INTO の各命令をカスタマイズできます。 オプションは、%DATA または %XML 組み込み関数の 2 番目のパラメーターとして指定します。 パラメーターには定数または変数の式を使用できます。 オプションは、「`opt1=val1 opt2=val2`」の形式で指定します。

オプションの指定方法について詳しくは、708 ページの『%XML (xmlDocument {:options})』 および 626 ページの『%DATA (文書 {:オプション})』 を参照してください。

- *allow extra* オプションは、RPG 変数の設定に必要ない追加情報が文書にある場合に RPG ランタイムが状態を処理する方法を指定します。

  このオプションは、DATA-INTO および XML-INTO で使用されます。
- *allow missing* オプションは、データ構造のすべての RPG サブフィールドのデータを提供するために必要な情報が文書にない場合に RPG ランタイムが状態を処理する方法を指定します。

  このオプションは、DATA-INTO および XML-INTO で使用されます。
- *case* オプションは、RPG フィールド名および path オプション内の名前に一致する名前を検索する際に文書内の要素名および属性名を解釈する方法を指定します。

  このオプションは、DATA-INTO および XML-INTO で使用されます。
- *ccsid* オプション は、文書の構文解析に使用される CCSID を指定します。

  このオプションは、DATA-INTO および XML-INTO で使用されます。
- *count prefix* オプションは、DATA-INTO 命令または XML-INTO 命令で設定された RPG 配列要素の数を受け取る追加サブフィールドの名前の接頭部を指定します。 これは、DATA-GEN 命令によって生成される RPG 配列要素の数を示す追加のサブフィールドの名前の接頭部を指定します。

  このオプションは、DATA-GEN、DATA-INTO、および XML-INTO で使用されます。
- *data subfield* オプションは、RPG データ構造のテキスト・データが存在する状態を処理するために使用される追加のサブフィールドの名前を指定します。 (通常、データ構造のデータは、データ構造のサブフィールドに対してのみ提供されます。)

  このオプションは、DATA-INTO および XML-INTO で使用されます。
- *doc* オプションは、%DATA 組み込み関数または %XML 組み込み関数の最初のパラメーターが文書の内容であるか、文書の内容が含まれるファイルの名前であるかを指定します。

  このオプションは、DATA-GEN、DATA-INTO、および XML-INTO で使用されます。

- *output* オプションは、DATA-GEN 命令の出力ファイルが存在しない場合に使用される CCSID を指定します。

  このオプションは、DATA-GEN で使用されます。
- *namee* オプションは、生成される文書の上位レベルで使用される名前を指定します。

  このオプションは、DATA-GEN で使用されます。
- *output* オプションは、データが生成される前に出力変数またはファイルを消去するか、新規データを既存のデータに付加するかを指定します。

  このオプションは、DATA-GEN で使用されます。
- 名前空間 オプションは、 XML-INTO が、path オプション内の名前またはデータ構造のサブフィールド名と付き合わせる際に、 名前空間を伴う XML 名をどのように扱うのかを制御します。

  このオプションは、XML-INTO で使用されます。
- 名前空間接頭部 オプションは、 名前から名前空間を除去するために名前空間オプションが使用された場合に XML 名から除去された 名前空間の値を、RPG プログラムが検出することを可能にします。

  このオプションは、XML-INTO で使用されます。
- *path* オプションは、目的の要素を配置する文書内の場所を指定します。

  このオプションは、DATA-INTO および XML-INTO で使用されます。
- *count prefix* オプションは、サブフィールドの名前そのものではなく、サブフィールドに生成する名前を指定する追加サブフィールドの名前の接頭部を指定します。

  このオプションは、DATA-GEN で使用されます。
- *trim* オプションは、DATA-INTO 命令または XML-INTO 命令の RPG 変数にデータを割り当てる前に、ブランク、タブ、および行末の文字をデータから削除するかどうかを指定します。 DATA-GEN 命令の生成プログラムにデータを渡す前に、データからブランクを削除するかどうかを指定します。

  このオプションは、DATA-GEN、DATA-INTO、および XML-INTO で使用されます。

allowextra (デフォルト no)

allowextra オプションは、DATA-GEN、DATA-INTO、および XML-INTO で使用されます。

RPG データ構造のサブフィールドへの代入に不必要なデータが文書にある状態の場合は、*allowextra* オプションを使用して、この状態をエラーとみなすかどうかを指示できます。 以下の状況では、データは余分であるとみなされます。

- RPG データ構造と一致するデータにおいて、非空 白文字のテキスト・コンテンツが検出された場合。
- RPG データ構造の配列サブフィールドと一致するデータ において、文書内の項目の数が RPG サブフィールド配列の次元より大きい場合。
- RPG スカラー変数 (データ構造でもなく指標なし配列でもない) と一致するデータにおいて、XML-INTO 命令の一部のデータ・タイプで許可される特別な形式の属性以外の子要素が項目に含まれている場合 (916 ページの『XML-INTO および DATA-INTO の RPG 変数にデータを転送する場合の規則』 を参照)。

予期しないデータが検出され、'allowextra=yes' が指定されていない場合、その命令は状況コード 00353 (XML が RPG 変数と一致しない) または 00356 (DATA-INTO の文書が RPG 変数と一致しない) で失敗します。

注意: XML-INTO 命令の場合、非データ構造 XML 要素の XML 属性は、任意の時点で RPG ランタイムによる変換処理の対象になる場合があります。 現在、"fmt" および "adjust" は、一部のターゲット・データ型の場合 に RPG ランタイムによりすでに変換処理されています。 PTF であっても、任意の時点でその他の属性のサポートが追加される場合があります。 現在、ある属性がオプション 'allowextra=yes' で無視されていて、その属性が RPG ランタイムに対して意味がある存在になった場合、それがデータの処理に影響を及ぼすことがあります。

- *no* は、RPG 変数または配列 要素の設定に使用される文書内の項目に、その変数の設定に必要なデータのみが 含まれている必要があることを指示します。
- *yes* は、文書内の余分なデータを無視するよう指示します。

サブフィールド配列に対して余分な要素がある場合の *allowextra* オプションの例

以下の例の中では次の定義が使用されています。

```
D employee        DS                  QUALIFIED
D   name                        10A   VARYING
D   type                        10A

D empInfo2        DS                  QUALIFIED
D   emp                               LIKEDS(employee)
D                                     DIM(2)

D empInfoAway     DS                  QUALIFIED
D   emp                               LIKEDS(employee)
D                                     DIM(2)
D   away                        10A   DIM(2)
```

ファイル emp.xml に以下の行が含まれていると想定します。

```
<employees>
 <emp><name>Jack</name><type>Normal</type></emp>
 <emp><name>Mary</name><type>Manager</type></emp>
 <emp><name>Sally</name><type>Normal</type></emp>
</employees>
```

1. XML 文書には 3 つの *emp* XML 要素 があり、RPG *emp* 配列には 2 つしか要素がないため、データ構造 *empInfo2* には オプション *allowextra=yes* を指定する必要があります。

   ```
   xml-into empInfo2
            %XML('emp.xml'
                 : 'doc=file allowextra=yes path=employees');
   // empInfo2.emp(1)    .name = 'Jack'    .type = 'Normal'
   // empInfo2.emp(2)    .name = 'Mary'    .type = 'Manager'
   ```

2. データ構造 *empInfo2* に対してオプション *allowextra* が指定されていません。 XML 文書に含まれる "emp" 要素の数が RPG 配列には多すぎるため、 XML-INTO 命令は状況コード 00353 で失敗します。

   ```
   xml-into(e) empInfo2
            %XML('emp.xml' : 'doc=file path=employees');
   // %error = *on
   // %status = 353
   ```

3. 構造 *empInfoAway* には 2 つの *emp* 要素 と 2 つの *away* 要素が必要です。 XML 文書に含まれているのは 3 つの *emp* 要素 とゼロ個の *away* 要素です。 オプション *allowextra=yes allowmissing=yes* が 指定されているため、*emp* および *away* XML 要素 の数がどのような数でも、命令は成功します。 余分な *emp* 要素および欠落している *away* 要素 は無視されます。

   ```
   xml-into empInfoAway
            %XML('emp.xml' : 'allowextra=yes ' +
                             'allowmissing=yes ' +
                             'path=employees ' +
                             'doc=file');
   // empInfoSite.emp(1)  .name = 'Jack'    .type = 'Normal'
   // empInfoSite.emp(2)  .name = 'Mary'    .type = 'Manager'
   // empInfoSite.away(1) = ' '
   // empInfoSite.away(2) = ' '
   ```

RPG サブフィールドに対応しない *XML* データがある場合の *allowextra* オプションの例

以下の例の中では次の定義が使用されています。

```
D qualName        DS                  QUALIFIED
D   name                        10A
D   lib                         10A

D copyInfo        DS                  QUALIFIED
D   from                              LIKEDS(qualName)
D   to                                LIKEDS(qualName)

D copyInfo3       DS                  QUALIFIED
D   from                              LIKEDS(qualName)
```

```
D   to                                  LIKEDS(qualName)
D   create                        1N
```

ファイル cpyA.xml に以下の行が含まれていると想定します。

```
<copyInfo>
 <to><name>MYFILE</name><lib>*LIBL</lib></to>
 <from name="MASTFILE" lib="CUSTLIB"></from>
</copyInfo>
```

ファイル cpyC.xml に以下の行が含まれていると想定します。

```
<copyinfo errors="tolerate">
 <to><name>MYFILE</name><lib>MYLIB</lib></to>
 <from><name>MASTFILE</name><lib>CUSTLIB</lib></from>
 <to><name>MYFILE2</name></to>
</copyinfo>
```

ファイル cpyD.xml に以下の行が含まれていると想定します。

```
<copyinfo to="MYLIB/MYFILE">
 <from><name>MASTFILE</name><lib>CUSTLIB</lib></from>
</copyinfo>
```

1. データ構造 *copyInfo* には *from* と *to* という 2 つのサブフィールドがあります。 これらの各サブフィールドには *name* と *lib* という 2 つのサブフィールドがあります。ファイル *cpyA.xml* は *copyInfo* 構造と完全に一致しています。そのため、*allowextra* はデフォルトで yes に設定されるため *allowextra* オプションは不要です。

   ```
   xml-into copyInfo %XML('cpyA.xml' : 'doc=file');
   // copyInfo.from  .name = 'MASTFILE  ' .lib = 'CUSTLIB   '
   // copyInfo.to    .name = 'MYFILE    ' .lib = '*LIBL     '
   ```

2. ファイル *cpyC.xml* の XML 要素 *copyinfo* には、一致する RPG サブフィールドがない XML 属性があります。 また、 *to* サブフィールドが複数回指定されています。 XML 文書内の余分なサブフィールドを許容するためには、オプション *allowextra=yes* の 指定が必要です。 余分な XML データは無視されます。

   ```
   xml-into copyInfo
            %XML('cpyC.xml' : 'doc=file allowextra=yes');
   // copyInfo.from  .name = 'MASTFILE  ' .lib = 'CUSTLIB   '
   // copyInfo.to    .name = 'MYFILE    ' .lib = 'MYLIB     '
   ```

3. データ構造 *copyInfo3* には、ファイル cpyC.xml にはないサブフィールド *create* があります。*cpyC.xml* には、データ構造 *copyInfo3* に対して、欠落しているサブフィールドと余分なサブフィールドの両方が含まれています。 *allowextra=yes allowmissing=yes* の両方のオプションを指定する必要があります。 余分なサブフィールドは無視され、欠落しているサブフィールドは元の値を保持します。

   ```
   clear copyInfo3;
   xml-into copyInfo3
            %XML('cpyC.xml' : 'allowextra=yes ' +
                              'allowmissing=yes ' +
                              'doc=file' +
                              'path=copyinfo');
   // copyInfo3.from  .name = 'MASTFILE  ' .lib = 'CUSTLIB   '
   // copyInfo3.to    .name = 'MYFILE    ' .lib = 'MYLIB     '
   // copyInfo3.create = '0' (from the CLEAR operation)
   ```

4. ファイル cpyD.xml にある XML 要素 *copyInfo* には、属性 *to* があります。 属性でサブフィールドを指定できるのは、サブフィールドが配列でもデータ構造でもない場合のみです。 *to* 属性は予期されていないため、 および、*to* XML 要素が見つからないため、XML-INTO 命令は状況コード 00353 で失敗します。

   ```
   xml-into(e) copyInfo %XML('cpyC.xml' : 'doc=file');
   // %error = *on
   // %status = 353
   ```

5. オプション *allowextra=yes allowmissing=yes* が指定されることによって、 余分な *to* 属性は無視され、欠落している *to* 要素 は許容されます。 *to* サブフィールドは XML-INTO 命令によって変更されません。

```
copyInfo.to.name = '*UNSET*';
copyInfo.to.lib = '*UNSET*';
xml-into copyInfo %XML('cpyD.xml' : 'doc=file ' +
                  'allowextra=yes allowmissing=yes');
// copyInfo.from  .name = 'MASTFILE  ' .lib = 'CUSTLIB   '
// copyInfo.to    .name = '*UNSET*   ' .lib = '*UNSET*   '
```

スカラー変数またはサブフィールドに対して予期しない非テキスト・コンテンツがある場合の *allowextra* オプションの例

以下の例の中では次の定義が使用されています。

```
D text            S            200A    VARYING

D order           DS                   QUALIFIED
D   part                        25A    VARYING
D   quantity                    10I 0
```

ファイル `txt.xml` に以下の行が含まれていると想定します。

```
<?xml version='1.0' ?>
<text><word>Hello</word><word>World</word></text>
```

ファイル `ord.xml` に以下の行が含まれていると想定します。

```
<?xml version='1.0' ?>
<order>
 <part>Jack in a box<discount>yes</discount></part>
 <quantity multiplier="10">2</quantity>
</order>
```

1. RPG 変数 *text* は、独立フィールドです。 XML ファイル `txt.xml` には *text* という名前の要素があり、*word* という名前の子要素が 2 つあります。 *text* XML 要素 に子要素があり、*allowextra* オプション はデフォルトで no に設定されているため、XML-INTO 命令は状況コード 00353 で失敗します。

```
xml-into(e) text %XML('txt.xml' : 'doc=file');
// %error = *on
// %status = 353
```

2. オプション *allowextra=yes* が指定されます。 子 XML 要素 は無視されます。 XML-INTO 命令は成功しますが、 *text* XML 要素の内容は子 XML 要素のみであるため、 RPG フィールド *text* で使用可能なデータはありません。

```
xml-into text %XML('txt.xml' : 'allowextra=yes doc=file';
text = '';
```

3. RPG 変数 *order* は 2 つのサブフィールドがあるデータ構造 であり、サブフィールド自体はデータ構造ではありません。 これらのサブフィールドを表す XML 要素 には子要素や属性があるべきではありませんが、*part* XML 要素 には *discount* という 1 つの子があり、*quantity* XML 要素 には属性 *multiplier* があります。 オプション *allowextra=yes* が 指定されているため、*discount* 要素および *multiplier* 属性 は無視されます。

```
xml-into order %XML('ord.xml'
                    : 'doc=file allowextra=yes');
// order.part = "Jack in a box"
// order.quantity = 2
```

allowmissing (デフォルト no)

allowmissing オプションは、DATA-INTO および XML-INTO で使用されます。

RPG データ構造のサブフィールドに対して十分な数の項目が文書にない状態の場合は、*allowmissing* オプションを使用して、この状態をエラーとみなすかどうかを指示できます。 以下の状況では、データが欠落しているとみなされます。

- RPG データ構造 (データ構造サブフィールドも含む) と一致する項目において、すべての RPG サブフィールドに対する子項目が文書内の項目にない場合。
- RPG データ構造の配列サブフィールドと一致するデータ において、文書内の項目の数が RPG サブフィールド配列の次元より小さい場合。

予期したデータが検出されず、'allowmissing=yes' が指定されていない場合、その命令は状況コード 00353 (XML が RPG 変数と一致しない) または状況コード 00356 (DATA-INTO の文書が RPG 変数と一致しない) で失敗します。

ヒント: *countprefix* オプションを使用して、 文書でデータ構造のすべてのサブフィールドに十分なデータがない状態を処理することもできます。

XML-INTO 命令または DATA-INTO 命令で指定された配列に対し少ない数の配列要素を使用する場合には、'allowmissing=yes' の指定は必要ありません。 文書に含まれる要素の数が RPG 配列より少ない場合、その命令は失敗しません。 PSDS の 372 から 379 桁目にある「要素の数」サブフィールドを使用して、命令により正常に設定される要素の数を判別できます。

- *no* は、データ構造のすべての サブフィールド (データ構造サブフィールドのサブフィールドを含む) に対しデータが存在する必要があり、すべてのサブフィールド配列のすべての要素に対してもデータが存在する必要があることを示します。
- *yes* は、サブフィールドおよびサブフィールド配列のすべての要素に対してデータが存在しない場合に、その命令が失敗しないことを示します。 変数が XML-INTO または DATA-INTO の第 1 オペランドとして指定された場合、未設定のサブフィールドには命令の実行前と同じ値が保持されます。 %HANDLER が XML-INTO または DATA-INTO の第 1 オペランドとして指定された場合、処理プロシージャーに渡された配列の未設定のサブフィールドにはそのタイプのデフォルト値 (数値の場合はゼロ、データ値の場合は *LOVAL など) が設定されます。

サブフィールド配列に対してデータ数が不十分な場合の *allowmissing* オプションの例

以下の例の中では次の定義が使用されています。

```
D employee        DS                  QUALIFIED
D   name                        10A   VARYING
D   type                        10A

D empInfo3        DS                  QUALIFIED
D   emp                               LIKEDS(employee)
D                                     DIM(3)

D empInfo2        DS                  QUALIFIED
D   emp                               LIKEDS(employee)
D                                     DIM(2)

D empInfo4        DS                  QUALIFIED
D   emp                               LIKEDS(employee)
D                                     DIM(4)
```

ファイル `emp.xml` に以下の行が含まれていると想定します。

```
<employees>
  <emp><name>Jack</name><type>Normal</type></emp>
  <emp><name>Mary</name><type>Manager</type></emp>
  <emp><name>Sally</name><type>Normal</type></emp>
</employees>
```

1. *empInfo3* データ構造には、3 つの要素を持つ配列 *emp* が あります。 XML 文書にも 3 つの *emp* XML 要素があるため、 *allowmissing* オプションは必要ありません。 XML 文書がデータ構造と完全に一致するため、デフォルト の *allowmissing=no* を使用できます。

   ```
   xml-into empInfo3 %XML('emp.xml' :
                          'doc=file path=employees');
   // empInfo3.emp(1)    .name = 'Jack'    .type = 'Normal'
   // empInfo3.emp(2)    .name = 'Mary'    .type = 'Manager'
   // empInfo3.emp(3)    .name = 'Sally'   .type = 'Normal'
   ```

2. それでも、オプション *allowmissing=no* を指定してもかまいません。

```
xml-into empInfo3 %XML('emp.xml' :
                       'doc=file ' +
                       'allowmissing=no path=employees');
// empInfo3.emp(1)    .name = 'Jack'    .type = 'Normal'
// empInfo3.emp(2)    .name = 'Mary'    .type = 'Manager'
// empInfo3.emp(3)    .name = 'Sally'   .type = 'Normal'
```

3. XML 文書には 3 つの *emp* XML 要素しかなく、 RPG *emp* 配列には 4 つの要素があるため、 データ構造 *empInfo4* にはオプション *allowmissing=yes* を指定する必要があります。

```
xml-into empInfo4
         %XML('emp.xml' : 'doc=file ' +
                       'allowmissing=yes path=employees');
// empInfo4.emp(1)    .name = 'Jack'    .type = 'Normal    '
// empInfo4.emp(2)    .name = 'Mary'    .type = 'Manager   '
// empInfo4.emp(3)    .name = 'Sally'   .type = 'Normal    '
// empInfo4.emp(4)    .name = ''        .type = '          '
```

4. データ構造 *empInfo4* に対してオプション *allowmissing* が指定されていません。 RPG 配列に対して十分な数の *emp* XML 要素が XML 文書に 含まれていないため、XML-INTO 命令は状況コード 00353 で失敗します。

```
xml-into(e) empInfo4 %XML('emp.xml' :
                       'doc=file path=employees');
// %error = *on
// %status = 353
```

すべてのサブフィールドに対してデータ数が不十分な場合の *allowmissing* オプションの例

以下の例の中では次の定義が使用されています。

```
D qualName        DS                  QUALIFIED
D   name                        10A
D   lib                         10A

D copyInfo        DS                  QUALIFIED
D   from                            LIKEDS(qualName)
D   to                              LIKEDS(qualName)
```

ファイル `cpyA.xml` に以下の行が含まれていると想定します。

```
 <?xml version='1.0' ?>
  <copyInfo>
  <to><name>MYFILE</name><lib>*LIBL</lib></to>
  <from name="MASTFILE" lib="CUSTLIB"></from>
 </copyInfo>
```

ファイル `cpyB.xml` に以下の行が含まれていると想定します。

```
<copyInfo>
 <from><name>MASTER</name><lib>PRODLIB</lib></from>
 <to><name>MYCOPY</name></to>
</copyInfo>
```

1. データ構造 *copyInfo* には *from* と *to* という 2 つのサブフィールドがあります。 これらの各サブフィールドには *name* と *lib* という 2 つのサブフィールドがあります。 ファイル `cpyA.xml` は *copyInfo* 構造と 完全に一致しています。したがって、*allowmissing* オプションは不要です。

```
xml-into copyInfo %XML('cpyA.xml' : 'doc=file');
// copyInfo.from  .name = 'MASTFILE  ' .lib = 'CUSTLIB   '
// copyInfo.to    .name = 'MYFILE    ' .lib = '*LIBL     '
```

2. ファイル `cpyB.xml` では、XML 要素 *copyinfo.to* に *lib* サブフィールドが欠落しています。 XML 文書でのサブフィールドの欠落を許容するには、オプション *allowmissing=yes* の指定が必要です。 *copyInfo*

構造は命令の前にクリアされるため、プログラムはどのサブフィールドにデータが割り当てられなかったのかを判別できます。

```
clear copyInfo;
xml-into copyInfo %XML('cpyB.xml'
                  : 'doc=file allowmissing=yes');
// copyInfo.from  .name = 'MASTER    ' .lib = 'PRODLIB   '
// copyInfo.to    .name = 'MYCOPY    ' .lib = '          '
if copyInfo.from.lib = *blanks;
  copyInfo.from.lib = '*LIBL';
endif;
if copyInfo.to.lib = *blanks;
  copyInfo.to.lib = '*LIBL';
endif;
```

case (デフォルト lower)

case オプションは、DATA-INTO および XML-INTO で使用されます。

case オプションは、 RPG フィールド名および *path* オプション内の名前に 一致する名前を検索する際に XML-INTO および DATA-INTO が文書内の 要素名および属性名を解釈する方法を指定します。 文書内の項目が正しく解釈されないと、 サブフィールド名およびパス内の名前に正しく一致しなくなり、 命令は状況コード 00353 (XML-INTO の場合) または 00356 (DATA-INTO の場合) で失敗します。

- *lower* は、RPG 変数名に一致す る文書内の項目の名前が小文字であることを示します。
- *upper* は、RPG 変数名に一致す る文書内の項目の名前が大文字であることを示します。
- *any* は、RPG 変数名に一致す る文書内の項目の名前が大/小文字混合または不明であることを示します。 文書内の項目の名前は、大文字に変換された後、大文字の RPG 変数名と比較されます。
- *convert* は、文書内の項目の名前が、 RPG 名とのマッチングの前に、有効な RPG 名に変換されることを示します。 名前の変換は 以下の手順で実行されます。
  1. ジョブの *LANGIDSHR 変換テーブルを使用して、 名前に含まれている英字が大文字 A-Z 文字に変換されます。 例えば、 名前「èñ-Àúb」は、このステップで「EN-AUB」に変換されます。
  2. このステップでは、名前に含まれている A-Z でも 0-9 でもないすべての文字 (名前に含まれている 2 バイト文字シーケンスも含む) が 下線文字に変換されます。 例えば、名前「EN-AUB#」はこのステップ で「EN_AUB_」に変換されます。
  3. 残っている下線はすべてマージされて単一の下線になります。 これには、元の名前にある下線と、前のステップで追加された下線の 両方が含まれます。 例えば、 名前「EN-$_AUB」は前のステップで「EN___AUB」に変換され、 このステップで「EN_AUB」に変換されます。
  4. 結果としてできた名前の先頭文字が下線文字の場合、 その下線文字は名前から削除されます。 例えば、名前「_EN_AUB」はこのステップ で「EN_AUB」に変換されます。
  5. 注意: 一部の英字は A-Z 文字 に変換されない場合があります。 例えば、文字「Ä」はスウェーデン語文字セットでは A とは別の文字なので、 *LANGIDSHR 変換テーブルの使用によって文字「A」にマップされることはありません。 スウェーデン語ジョブ では、名前「ABÄC」は変換の最初のステップでは変更されないため、「Ä」文字は最初のステップの後も名前の中に 残ります。 「Ä」文字は 2 番目のステップで _ に変換され、 結果としてできる名前は、予期されていた「ABAC」ではなく「AB_C」になります。

値 *upper*、*lower*、および *any* が指定された *case* オプションの例

以下の例の中では次の定義が使用されています。

```
D info            DS                  QUALIFIED
D   name                        10A
D   id_no                        5A
D xmlDoc          S             1000A    VARYING
```

1. XML 文書中の要素名および属性に小文字が使用されています。 *case* オプションはデフォルトで小文字に設定されるため、指定する必要はありません。

```
xmlDoc = '<info><name>Jim</name><id_no>103</id_no></info>';
xml-into info %XML(xmlDoc);
```

```
// info.name = 'Jim       '
// info.id_no = '103'
```

2. XML 文書中の要素名および属性に大文字が使用されています。 オプション case=upper を指定する必要があります。

```
xmlDoc = '<INFO><NAME>Bill</NAME><ID_NO>104</ID_NO></INFO>';
xml-into info %XML(xmlDoc : 'case=upper');
// info.name = 'Bill      '
// info.id_no = '104'
```

3. XML 文書中の要素名および属性は大/小文字混合です。 オプション case=any を指定する必要があります。

```
xmlDoc = '<INFO><name>Tom</name>'
           + '<ID_NO>105</ID_NO></INFO>';
xml-into info %XML(xmlDoc : 'case=any');
// info.name = 'Tom       '
// info.id_no = '104'
```

4. XML 文書中の要素名および属性は大/小文字混合ですが、 *case* オプションは指定されていません。 XML-INTO 命令は、XML 要素の名前が小文字であると 想定するため、状況コード 00353 で失敗します。

```
xmlDoc = '<INFO><name>Tom</name>'
           + '<ID_NO>105</ID_NO></INFO>';
xml-into(e) info %XML(xmlDoc);
// %error = *on
// %status = 353
```

case=convert オプションの例

1. XML 文書に、RPG サブフィールドには有効でない英字を使用した名前が含まれています。

   以下の例の中では次のデータ構造が使用されています。

```
D etudiant        ds                  qualified
D  age                          3p 0
D  nom                         25a   varying
D  ecole                       50a   varying

D student         ds                  likeds(etudiant)
```

   ファイル info.xml に以下の行が含まれていると想定します。

```
<Étudiant Nom="Élise" Âge="12">
   <École>Collège Saint-Merri</École>
</Étudiant>
```

   a. オプション *case=convert ccsid=ucs2* が指定されます。 オプション *case=convert* は、 XML 文書中の名前が、パス内およびサブフィールドのリスト内の RPG 名とマッチングされる前に、 ジョブの *LANGIDSHR 変換テーブルを使用して変換されることを 指定します。 名前 *Étudiant*、*Âge*、 および *École* は、*ETUDIANT*、*AGE*、 および *ECOLE* に変換されます。 XML データ自体は変換されないため、サブフィールド *ecole* は、 値「Collège Saint-Merri」を XML 文書中にあるとおりに受け取ります。

      デフォルトのパスは RPG 変数 *ETUDIANT* の名前であり、これは 実際の XML 名 *Étudiant* の変換後の形と一致するため、path オプションは必要ありません。

```
xml-into etudiant %xml('info.xml'
                       : 'doc=file case=convert '
                       + 'ccsid=ucs2');
// etudiant.nom = 'Élise'
// etudiant.age = 12
// etudiant.ecole = 'Collège Saint-Merri'
```

b. RPG データ構造は *student* という名前です。 *Étudiant* XML 要素 が *student* データ構造と一致することを示すために、path オプションを指定する必要があります。 path オプション は、変換後の XML 名と一致するように *path=etudiant* と指定されます。

```
    xml-into student %xml('info.xml'
                        : 'doc=file case=convert '
                        + 'ccsid=ucs2 path=etudiant');
    // student.nom = 'Élise'
    // student.age = 12
    // student.ecole = 'Collège Saint-Merri'
```

2. XML 文書に、XML ではサポートされているが RPG 名では使用できない非英数字を使用した名前が含まれています。

   以下の例の中では次のデータ構造が使用されています。

```
D employee_info   ds                  qualified
D  last_name                    25a   varying
D  first_name                   25a   varying
D  is_manager                    1a

D emp             ds                  likeds(employee_info)
```

   ファイル data.xml に以下の行が含まれていると想定します。

```
<employee-info is-manager="y">
   <last-name>Smith</last-name>
   <first-name>John</first-name>
</employee-info>
```

   a. オプション *case=convert* が指定されます。 ジョブ の *LANGIDSHR テーブルを使用して英字の変換が行われた後、 その次のステップでは、 A-Z でも 0-9 でもない残りの文字が下線文字 に変換されます。 XML 名 *employee-info*、*is-manager*、*last-name*、 および *first-name* は、*EMPLOYEE_INFO*、*IS_MANAGER*、*LAST_NAME*、 および *FIRST_NAME* に変換されます。

      RPG データ構造名 *employee_info* は、 XML 名 *employee-info* の変換後の形 *EMPLOYEE_INFO* に 一致するため、path オプションは必要ありません。

```
    xml-into employee_info %xml('data.xml'
                         : 'doc=file case=convert ');
    // employee_info.last_name = 'Smith'
    // employee_info.first_name = 'John'
    // employee_info.is_manager = 'y'
```

   b. RPG データ構造は *emp* という名前です。 *employee-info* XML 要素 が *emp* データ構造と一致することを示すために、path オプションを指定する必要があります。 path オプション は、変換後の XML 名と一致するように *path=employee_info* と指定されます。

```
    xml-into emp %xml('data.xml'
                    : 'doc=file case=convert '
                    + 'ccsid=ucs2 path=employee_info' );
    // emp.last_name = 'Smith'
    // emp.first_name = 'John'
    // emp.is_manager = 'y'
```

3. XML 文書には、2 バイト・データを使用した名前が含まれています。

   以下の例の中では次の定義が使用されています。

```
D employee_info_  ds                  qualified
D  last_name_                   25a   varying
D  first_name_                  25a   varying
D  is_manager_                   1a
```

   ファイル data.xml に以下の行が含まれていると想定します。ここで、「DBCS」は 2 バイト・データを表します。

```
<employee_info_DBCS is_manager_DBCS="y">
   <last_name_DBCS>Smith</last_name_DBCS>
```

```
   <first_name_DBCS>John</first_name_DBCS>
</employee_info_DBCS>
```

オプション *case=convert* が指定されます。 ジョブ の *LANGIDSHR テーブルを使用して英字の変換が行われた後、 その次のステップでは、DBCS データおよび関連付けられたシフトアウトおよびシフトイン文字 を含めて、A-Z でも 0-9 でもない残りの文字が、下線文字に変換されます。 このステップの後、XML 名 *last_name_DBCS* は変換後は *LAST_NAME_____* になります。次のステップでは、元の名前に含まれている下線も含めて、残りの下線がすべてマージされて 1 つの下線になります。 結果としてできる名前は *LAST_NAME_* です。

```
xml-into employee_info_ %xml('data.xml'
                     : 'doc=file case=convert '
                     + 'ccsid=ucs2');
// employee_info_.last_name_ = 'Smith'
// employee_info_.first_name_ = 'John'
// employee_info_.is_manager_ = 'y'
```

4. XML 文書に、先頭が 2 バイト・データの名前が含まれています。

   以下の例の中では次の定義が使用されています。

```
D employee_info   ds                  qualified
D  last_name                    25a   varying
D  first_name                   25a   varying
D  is_manager                    1a
```

   ファイル data.xml に以下の行が含まれていると想定します。ここで、「DBCS」は 2 バイト・データを表します。

```
<DBCS_employee_info DBCS_is_manager="y">
   <DBCS_last_name>Smith</DBCS_last_name>
   <DBCS_first_name>John</DBCS_first_name>
</DBCS_employee_info>
```

   オプション *case=convert* が指定されます。 非英数字から単一の下線文字への変換が行われた後、 名前 *DBCS_last_name* は 変換後は *_LAST_NAME* になります。 RPG は下線で始まる名前をサポートしていないため、 最初の下線は削除されます。 変換後の最終的な名前は *LAST_NAME* です。

```
xml-into employee_info %xml('data.xml'
                        : 'doc=file case=convert '
                        + 'ccsid=ucs2');
// employee_info.last_name = 'Smith'
// employee_info.first_name = 'John'
// employee_info.is_manager = 'y'
```

ccsid (デフォルト best)

ccsid オプションは、DATA-INTO および XML-INTO で使用されます。

ccsid オプションは、文書の処理に使用される CCSID を指定します。 XML-INTO 命令または DATA-INTO 命令の実行時に、以下のように何らかの CCSID 変換が実行される場合があります。

- 文書の CCSID が構文解析に使用される CCSID と異なる場合は、文書から文書の一時コピーに CCSID 変換が必要なことがあります。
- 構文解析に使用される CCSID が RPG 変数の CCSID と異なる場合は、データの RPG 変数への代入時に CCSID 変換が必要なことがあります。

実際の文書の CCSID がその文書の処理に使用される CCSID と異なる場合は、構文解析の開始前に文書全体で CCSID 変換が実行されます。 文書の処理に使用される CCSID が RPG 変数の CCSID と異なる場合は、RPG 変数への代入時にデータに対して CCSID 変換が実行されます。

- *best* は、文書内のデータを最適に保持する CCSID でその文書が処理されることを示します。 文書がジョブ CCSID またはジョブ CCSID に関連する ASCII CCSID にある場合、その文書はジョブ CCSID で処理されます。 その他の場合、文書は UCS-2 で処理され、データはジョブ CCSID に変換されてから UCS-2 以外のデータ型で変数に代入されます。

  注 : ccsid=best は XML-INTO 命令でのみサポートされています。

- *job* は、その文書がジョブ CCSID で処理されることを示します。 データは、UCS-2 変数への代入時には UCS-2 に変換されます。
- *ucs2* は、その文書が UCS-2 で処理されることを示します。 データは、UCS-2 以外のデータ型で変数に代入される場合にはジョブ CCSID に変換されます。

文書がファイルにある場合は、ファイル全体のコンテンツが構文解析の開始前に別の CCSID に変換される場合があります。

以下の表には、いくつかのファイルとその CCSID がリストされています。

ファイル	ファイル CCSID	関連する EBCDIC CCSID
file1.xml	37	37
file2.xml	1252	37
file3.xml	874	838
file4.xml	13488	(該当なし、UCS-2)
file5.xml	1208	(該当なし、UTF-8)

以下の表には、*ccsid* オプションの各値に対してファイルの処理に使用される CCSID が示されています。ここでは、ジョブ CCSID が 37 であると想定しています。 アスタリスクは、ファイルが処理前に別の CCSID に変換されることを示しています。

注 : 値「best」は DATA-INTO 命令でサポートされていません。

ファイル	CCSID オプション値		
	best	***job***	***ucs2***
file1.txt	37	37	13488*
file2.txt	37*	37*	13488*
file3.txt	13488*	37*	13488*
file4.txt	13488	37*	13488
file5.txt	13488*	37*	13488*

文書が変数にある場合、構文解析の開始前に文書全体が別の CCSID に変換される場合があります。

以下の変数定義を想定します。

```
D chrData         S            100A
D ucs2Data        S            100C
```

以下の表には、「ccsid」オプションの各値に対して変数の処理に使用される CCSID が示されています。ここでは、ジョブ CCSID が 37 であると想定しています。 アスタリスクは、変数内のデータが処理前に別の CCSID に変換されることを示しています。

変数	CCSID オプション値		
	best	***job***	***ucs2***
chrData	37	37	13488
ucs2Data	13488	37*	13488

countprefix

countprefix オプションは、DATA-GEN、DATA-INTO、および XML-INTO で使用されます。

XML-INTO 命令と DATA-INTO 命令の *countprefix* オプションは、サブフィールド配列について XML-INTO 命令または DATA-INTO 命令で設定された要素の数を受け取ることができるサブフィールドの接頭部を指

定します。 カウント・サブフィールドの名前の形式は、countprefix の値に配列名を付加したものです。 例えば、データ構造のサブフィールド配列 *meeting.attendees* に 「countprefix=num」が指定された場合、XML-INTO 命令または DATA-INTO 命令は、命令によって設定された *meeting.attendees* 配列の要素の実際の数を *meeting.numattendees* に設定します。 以下の countprefix オプションの説明では、サブフィールド *meeting.numattendees* を *countprefix* サブフィールド、*meeting.attendees* を *counted* サブフィールド と呼びます。

countprefix オプションの処理は、データ構造またはデータ構造サブフィールドのデータが構文解析された後に行われます。

DATA-GEN 命令の *countprefix* オプションは、カウントされる配列サブフィールドの要素をどのくらい生成するか示すサブフィールドの接頭部、または非配列サブフィールドを生成するかどうかを指定します。 例えば、データ構造のサブフィールド配列 *meeting.attendees* に 「countprefix=num」が指定された場合、DATA-GEN 命令は *meeting.numattendees* の値を使用して、命令によって生成された *meeting.attendees* 配列の要素の数を判別します。

注 :

1. countprefix サブフィールドは数値でなければなりません。また、これはスカラーでなければなりません。 つまり、配列またはデータ構造にすることはできません。 サブフィールドが countprefix の名前を持つが、数値でもスカラーでもない場合、 そのサブフィールドには通常の処理が行われます。これは、countprefix サブフィールドと見なされません。
2. カウントされるサブフィールドには、任意のタイプのサブフィールドが可能です。配列である必要はありません。

   XML-INTO 命令および DATA-INTO 命令では、カウントされるサブフィールドが配列でない場合、その countprefix サブフィールドには、そのサブフィールドを設定する文書内のデータが存在しなければ 0 (ゼロ)、存在すれば 1 が設定されます。

   DATA-GEN 命令では、カウントされるサブフィールドが配列でない場合、その countprefix サブフィールドの値が 1 であれば生成され、0 (ゼロ) であれば生成されません。
3. countprefix サブフィールドは、カウント可能とみなされません。 例えば、countprefix=num_ が指定されていて、データ構造にサブフィールド arr、num_arr、および num_num_arr がある場合、 num_arr は配列 arr の countprefix サブフィールドと見なされますが、num_num_arr は num_arr の countprefix サブフィールドと見なされません。
4. XML-INTO 命令および DATA-INTO 命令の場合:
   - サブフィールドが countprefix サブフィールドによってカウントされる場合、そのサブフィールドについて allowmissing オプションは考慮されません。 countprefix サブフィールドによってカウントされるすべてのサブフィールドに、オプション allowmissing=yes が暗黙指定されます。
   - サブフィールドの文書内のデータが多すぎる場合、countprefix サブフィールドは、XML-INTO 命令または DATA-INTO 命令で実際に設定された配列要素の数のみを反映します。 例えば、配列 arr に 10 個の要素があって、11 個の要素のデータがある場合、 arr の countprefix サブフィールドの値は 10 になります。
   - XML-INTO 命令または DATA-INTO 命令がエラーで終了した場合、命令で更新された RPG サブフィールドの正確な数を countprefix サブフィールドが反映していない可能性があります。 countprefix の処理は、それぞれのデータ構造またはデータ構造サブフィールドのデータが構文解析された後に行われます。 構文解析中、または countprefix の処理中にエラーが発生した場合、countprefix の処理は完了されません。
   - countprefix サブフィールドを文書内のデータで明示的に設定することはできません。 countprefix サブフィールドを設定する、文書内の項目は余分と見なされます。
   - countprefix サブフィールドは、datasubf サブフィールドと同じにできません。 例えば、countprefix=num_ が指定されていて、データ構造にサブフィールド arr と num_arr がある場合、num_arr が countprefix サブフィールドです。 このデータ構造にオプション datasubf=num_arr も指定することはできません。

「countprefix」オプションの例については、933 ページの『countprefix オプションの例』を参照してください。

countprefix オプションの例

以下の例の中では次の定義が使用されています。

```
D attendee_type...
D                 DS                  qualified template
D   name                         20a   varying
D   phone                         4s 0

D meeting         DS                  qualified
D   location                     20a   varying
D   attendee                           likeds(attendee_type)
D                                      dim(100)
D   numAttendee...
D                                10i 0

D email           DS                  qualified
D   to                           40a   varying
D   cc                           40a   varying
D   from                         40a   varying
D   countCc                       5i 0
D   subject                     100a   varying
D   countSubject                  5i 0
D   body                       1000a   varying

D order1          DS                  qualified
D   numpart                      10i 0
D   part                         20a   varying dim(100)

D order2          DS                  qualified
D   numpart                      10i 0
D   part                         20a   varying dim(100)
D   countpart                    10i 0
```

1. ファイル meeting123.xml に以下が含まれていると想定します。

```
<meeting>
   <location>Room 7a</location>
   <attendee name="Jim" phone="1234"/>
   <attendee name="Mary" phone="2345"/>
   <attendee name="Abel" phone="6213"/>
</meeting>
```

   a. countprefix オプションは *num* 接頭部を指定します。

      この XML-INTO 命令は、countprefix サブフィールドである *numAttendee* を 3、つまり、この命令によって設定された *attendee* サブフィールドの数に設定します。 配列 attendee 用の countprefix サブフィールドがあることによって、この配列に対する XML データの欠落が暗黙的に許容されるため、オプション *allowmissing=yes* を指定する必要はありません。

```
xml-into meeting %xml('meeting123.xml'
                 : 'doc=file countprefix=num');
// meeting.attendee(1):  name='Jim'   phone=1234
// meeting.attendee(2):  name='Mary'  phone=2345
// meeting.attendee(3):  name='Abel'  phone=6213
// meeting.numAttendee = 3 for i = 1 to meeting.numAttendee;
   if meeting.attendee(i) ...
endfor;
```

   b. countprefix サブフィールドは指定されません。

      配列 *attendee* に対する XML データが不十分であるため、この XML-INTO 命令は 失敗し、*numAttendee* の XML データはまったくありません。

```
xml-into(e) meeting %xml('meeting123.xml'
                 : 'doc=file');
// %error = *on
```

2. ファイル email456.txt に以下が含まれていると想定します。

```
<email to="jack@anywhere.com" from="jill@anywhere.com">
   <subject>The hill</subject>
   <body>How are you feeling after your fall?</body>
</email>
```

countprefix=count オプションが指定され、 *countprefix* サブフィールドの 接頭部が *count* であると指示しています。

cc サブフィールドに対する XML データはありませんが、countprefix サブフィールドである *countCc* があって、*cc* サブフィールドが XML から設定されなかったという情報を受け取る ことができるため、この XML-INTO 命令は成功します。

```
xml-into email %xml('email456.xml'
                    : 'doc=file countprefix=count');
// email.to = 'jack@anywhere.com'
// email.from = 'jill@anywhere.com'
// email.cc = ??  (not set by XML-INTO)
// email.countCc = 0
// email.subject = 'The hill'
// email.countSubject = 1
// email.body = 'How are you feeling after your fall?'
```

プログラムは *countCc* および *countSubject* サブフィールドの 値を使用して、*cc* および *subject* サブフィールド が XML-INTO 命令によって設定されたかどうかを判別します。

```
if email.countCc = 1;
   cc = email.cc;
else;
   cc = '';
endif;
if email.countSubject = 1;
   subj = email.subject;
else;
   subj = "NO SUBJECT";
endif;
```

3. ファイル File order789.txt に以下が含まれていると想定します。

```
<order numpart="2">
   <part>hammer</part>
   <part>saw</part>
</order>
```

XML 文書には、属性 *numpart* が 含まれていて、文書中にある *part* 要素の数を示しています。

a. オプション *countprefix=num* が指定され、*numpart* が 配列 *part* 用の *countprefix* サブフィールドであると指示しようとしています。

この XML-INTO 命令は失敗します。 サブフィールド *numpart* は countprefix サブフィールド であるため、明示的に XML-INTO 命令で設定することはできません。

```
xml-into(e) order1 %xml('order789.xml'
            : 'doc=file countprefix=num path=order');
// %error is set on
```

b. オプション *countprefix=count* が指定され、*countpart* が 配列 *part* 用の *countprefix* サブフィールドであると指示しています。

この XML-INTO 命令は成功します。 サブフィールド *numpart* は XML 文書から 2 に 設定され、サブフィールド *countpart* は countprefix 処理によって 2 に 設定されます。 *part* 配列 は *countprefix* オプションによってカウントされるため、 配列全体を設定するための十分な XML データがないことはエラーにはなりません。

```
xml-into order2 %xml('order789.xml'
          : 'doc=file countprefix=count path=order');
// order2.numpart = 2
// order2.part(1) = 'hammer'
// order2.part(2) = 'saw'
// order2.countpart = 2
```

4. 以下の例では、*meeting.numAttendee* が 27 に設定されており、DATA-GEN 命令にオプション「countprefix=num」が指定されています。 *meeting.attendee* で生成される要素は 27 個のみです。 *meeting.numAttendee* は、その countprefix サブフィールドだからです。

```
meeting.numAttendee = 27;
DATA-GEN meeting %DATA('output.txt' : 'doc=file countprefix=num') %GEN('MYGENPGM');
```

5. 以下の例では、*email.countcc* はゼロに設定され、DATA-GEN 命令に対してオプション「countprefix=count」が指定されています。 サブフィールド *email.cc* は、その countprefix サブフィールド *email.countcc* がゼロであるため、生成されません。

```
email.cc = 0;
DATA-GEN email %DATA('output.txt' : 'doc=file countprefix=count') %GEN('MYGENPGM');
```

datasubf

datasubf オプションは、DATA-INTO および XML-INTO で使用されます。

datasubf オプションは、 RPG データ構造に一致する項目にテキスト・データがある状態を処理するために使用される、追加のスカラー・サブフィールドの名前を指定します。

例えば、このオプションに datasubf=txt が指定されていて、RPG データ構造にスカラー・サブフィールド *txt* があると、そのサブフィールドは、このデータ構造に一致する項目のテキスト・データを受け取ります。

デフォルト: *datasubf* オプションが指定されていない場合、RPG データ構造に一致する項目は、テキスト・データを含むことができません。 テキスト・データは、データ構造のサブフィールドとのみ関連付けることができます。

注 :

1. *datasubf* オプションで名前が指定されたスカラー・サブフィールドが RPG データ構造にある場合、以下の規則が適用されます。
   - 一致する項目にテキスト・データがある場合、そのテキスト・データがスカラー・サブフィールドに代入されます。
   - データ構造にあるその他のすべてのサブフィールドの値は、属性で設定される必要があります。 そのため、項目が子項目を持つことはできず、データ構造にあるその他のサブフィールドはすべてスカラー・サブフィールドでなければなりません。
   - データ構造に一致する項目は、*datasubf* オプションと同じ名前の属性または 子項目を持つことはできません。
   - 項目にテキスト・データがない場合、 *datasubf* サブフィールドには空値が設定されます。 サブフィールドのデータ・タイプで空値がサポートされていないと (例えば、数値および日付タイプなど)、サブフィールドへの代入で例外が発生します。
2. *datasubf* オプションで名前が指定されたスカラー・サブフィールドが RPG データ構造にない場合、 そのデータ構造について *datasubf* オプションは無視されます。 RPG データ構造に一致する項目は、テキスト・データを含むことができません。
3. datasubf オプションで指定された名前と同じ名前の配列またはデータ構造サブフィールドが RPG データ構造にある場合、 そのデータ構造について datasubf オプションは無視されます。 RPG データ構造に一致する XML 要素は、テキスト・データを含むことができません。
4. 複合 RPG データ構造には、多くのデータ構造サブフィールドが含まれることがあります。 *datasubf* オプションは、データ構造サブフィールドごとに別々に考慮されます。 XML-INTO 命令または DATA-INTO 命令が正常終了するために、あるデータ構造サブフィールドのデータでは datasubf オプションを必要とするが、 別のデータ構造サブフィールドでは必要としないことがあります。
5. datasubf サブフィールドは、countprefix サブフィールドと同じにできません。 例えば、countprefix=num_ が指定されていて、データ構造にサブフィールド arr と num_arr がある場合、num_arr が countprefix サブフィールドです。 このデータ構造にオプション datasubf=num_arr も指定することはできません。

datasubf オプションの例

以下の例の中では次の定義が使用されています。

```
D customer        ds                  qualified
D    id                          10a
D    value                      100a   varying

D order           ds                  qualified
D    id                          10a
D    type                        10a

D customers       ds                  qualified
D    customer                         likeds(customer) dim(2)

D orderinfo       ds                  qualified
D    customer                         likeds(customer)
D    order                            likeds(order)
```

1. *datasubf* オプションは *value* サブフィールドを指定します。

   ファイル customer1.xml に以下が含まれていると想定します。

   ```
   <customer id="A34R27K">John Smith</customer>
   ```

   XML-INTO が「John Smith」を検出するのは *customer* データ構造 を処理しているときです。 この命令は *customer* データ構造 に *value* という名前のサブフィールドがあることを検出し、したがって、そのサブフィールド を「John Smith」データ用に使用します。

   ```
     xml-into customer %xml('customer1.xml'
                            : 'doc=file datasubf=value');
     // customer.id = "A34R27K"
     // customer.value = "John Smith"
   ```

2. datasubf オプションは指定されません。

   ファイル customer2.xml に以下が含まれていると想定します。

   ```
   <customer id="A34R27K">John Smith</customer>
   ```

   XML-INTO が「John Smith」を検出するのは *customer* データ構造 を処理しているときです。 XML-INTO はデータ構造に対するデータを保持することを通常はサポートしないため、この XML-INTO 命令は、余分な XML データが原因で、状況コード 00353 で失敗します。

   ```
     xml-into(e) customer %xml('customer2.xml'
                               : 'doc=file');
     // %error = *on
     // %status = 353
   ```

3. XML 文書中に、datasubf オプションと同じ名前の通常の XML 要素があります。

   ファイル customer3.xml に以下が含まれていると想定します。

   ```
   <customer id="A34R27K">
      <value>John Smith</value>
   </customer>
   ```

   datasubf オプションは 指定されません。 XML 文書中に *value* という名前の通常の XML 要素 があるため、*customer* データ構造の *value* サブフィールド は通常の方法で埋められます。 datasubf オプションは不要です。

   ```
     xml-into customer %xml('customer3.xml' : 'doc=file');
     // customer.id = "A34R27K"
     // customer.value = "John Smith"
   ```

datasubf=value オプション が指定されます。 XML 文書中に、*value* という名前の通常の XML 要素があります。 datasubf オプションの名前を持つスカラー・サブフィールドを XML 属性 または XML 要素で埋めることはできないため、この XML-INTO 命令は状況コード 00353 で 失敗します。

```
xml-into(e) customer %xml('customer3.xml'
                          : 'doc=file datasubf=value');
// %error = *on
// %status = 353
```

4. 複雑なデータ構造の場合、*datasubf* オプションが必要な場合もあれば、 不要な場合もあります。

   ファイル `customer4.xml` に以下が含まれていると想定します。

```
<orderinfo>
  <customer id="A34R27K">John Smith</customer>
  <order id="P8H41"><type>telephone</type></order>
</orderinfo>
```

   datasubf=value オプション が指定されます。 *customer* データ構造サブフィールドには *value* サブフィールド があり、したがって *datasubf* オプションが使用されます。 *order* データ構造サブフィールド には *value* サブフィールドはないため、*datasubf* オプション は無視されます。

```
xml-into orderinfo %xml('customer4.xml'
                        : 'doc=file datasubf=value');
// orderinfo.customer.id = "A34R27K"
// orderinfo.customer.value = "John Smith"
// orderinfo.order.id = "P8H41"
// orderinfo.order.type = "telephone"
```

doc (デフォルト string)

doc オプションは、DATA-GEN、DATA-INTO、および XML-INTO で使用されます。

doc オプションは、%XML または %DATA の第 1 オペランドを解釈する方法を示します。

- *string* は、%XML または %DATA の第 1 オペランドが変数であることを示します。 XML-INTO および DATA-INTO の場合、第 1 オペランドには構文解析される文書が含まれます。 DATA-GEN の場合、第 1 オペランドは、生成される文書を受け取ります。
- *file* は、ソース・オペランドに統合ファイル・システム内のファイルの名前が含まれていることを示します。

doc オプションの例

1. 次の例では、%XML および %DATA の最初のパラメーターは ファイルの名前です。 オプション `doc=file` を指定する必要があります。

```
ifsfile1 = 'myfile.xml';
ifsfile2 = 'myfile.json';
ifsfile3 = 'myfile.csv';

opt = 'doc=file';
XML-INTO myfield %XML(ifsfile1 : opt);
DATA-INTO myfield %DATA(ifsfile2 : opt) %PARSER('MYJSONPARS');
DATA-GEN myfield %DATA(ifsfile3 : opt) %GEN('MYCSVGEN');
```

2. 次の例の %XML の最初のパラメーターは、XML-INTO 命令または DATA-INTO 命令の場合は文書を含んでいる文字変数、DATA-GEN 命令の場合は文書を受け取るための文字変数です。 doc オプション はデフォルトで「string」に設定されるため、オプションは不要です。 ただし、オプション `doc=string` を指定することもできます。 以下の例では、命令の各ペアが同等です。

```
xmldata = '<data><num>3</num></data>';
XML-INTO data %XML(xmldata);
XML-INTO data %XML(xmldata : 'doc=string');

jsondata = '{"data":{"num":"3"}}';
DATA-INTO data %DATA(jsondata) %PARSER('MYJSONPARS');
DATA-INTO data %DATA(jsondata : 'doc=string') %PARSER('MYJSONPARS');
```

```
chain (id) custRec rec custDs;
DATA-GEN custDs %DATA(custInfo) %GEN('MYCSVGEN');
DATA-GEN custDs %DATA(custInfo : 'doc=string') %GEN('MYCSVGEN');
```

fileccsid (デフォルト utf8)

fileccsid オプションは、DATA-GEN で使用されます。

fileccsid オプションは、DATA-GEN 命令の出力ファイルが存在しない場合に使用される CCSID を指定します。

fileccsid オプションが指定されておらず、出力ファイルが存在しない場合は、出力ファイルは CCSID UTF-8 (1208) で作成されます。

utf8
UTF-8 (1208)

utf16
UTF-16 (1200)

ジョブ
ジョブの CCSID。 ジョブの CCSID が 65535 の場合は、ジョブのデフォルト CCSID が使用されます。

数値
特定の CCSID

name (デフォルトなし)

name オプションは、DATA-GEN で使用されます。

name オプションは、DATA-GEN 命令で生成される文書の最上位に使用する名前を指定します。 名前は、オプションに指定された文字と同じ文字 (大/小文字)で生成プログラムに渡されます。

例えば、次の DATA-GEN 命令の場合、最上位項目の名前として生成プログラムに渡される名前は「Orders」です。

「name」オプションが指定されない場合、生成プログラムに渡される名前は、データ構造の定義に指定された名前である「Rec」であり、名前が指定された場合と同じ大/小文字混合形式で表示されます。

```
DCL-DS Rec LIKEDS(orders_t);
...
DATA-GEN rec %DATA(filename : 'doc=file name=Orders') %GEN('MYPGM');
```

ns (デフォルト keep)

ns オプションは、XML-INTO で使用されます。

ns オプションは、 名前空間付きの XML 名を、XML-INTO が *path* オプション内の 名前またはデータ構造のサブフィールド名との突き合わせでどのように扱うのかを制御します。 例えば、XML 名「cust:name」には名前空間「cust」が付いています。

- *keep* は、XML 名の名前空間とコロンが保持されることを 指示します。 名前空間付きの XML 名は、どの RPG 名とも一致しません。
- *remove* は、RPG 名と突き合わせるときに XML 名から 名前空間とコロンが除去されることを指示します。 例えば、XML 名 が *ABC:DEF* の場合、RPG 名と比較するときに名前 *DEF* が使用されます。
- *merge* は、RPG 名と突き合わせるときに XML 名に含まれるコロンが下線に置き換えられる ことを指示します。 例えば、XML 名 が *ABC:DEF* の場合、RPG 名と比較するときに名前 *ABC_DEF* が使用されます。

注 :

1. *ns* オプションは、*path* オプションを処理するときに適用されます。 path 内の名前は、ns オプションの処理が終わった後で XML 名と一致するように 指定される必要があります。 例えば、 XML パスが *abc:info/abc:cust* で、オプション *'ns=remove'* が 指定される場合、*path* オプションは 'path=info/cust'

と指定される必要があります。 オプション *ns=merge* が指定される場合、*path* オプション は 'path=abc_info/abc_cust' と指定される必要があります。

2. オプション *ns=remove* が指定される場合、*nsprefix* オプション を使用して、任意のサブフィールドの名前空間の値を取得できます。

ns オプションの例

1. オプション *ns=remove* が使用されます。

   以下の例の中では次の定義が使用されています。

```
D info            DS                  QUALIFIED
D   type                         25A   VARYING
D   qty                          10I 0
D   price                         7P 3
```

   ファイル `info1.xml` に以下が含まれていると想定します。

```
<abc:info xmlns:abc="http://www.abc.xyz">
 <abc:type>Chair</abc:type>
 <abc:qty>3</abc:qty>
 <abc:price>79.99</abc:price>
</abc:info>
```

   この XML 文書内の名前 (*abc:type* など) を RPG サブフィールド名に使用することはできません。

   オプション *ns=remove* は、 XML-INTO 命令が、一致するサブフィールドまたは *path* オプション内に 指定された名前を検索する前に、名前空間 (*abc*) とコロンが XML 名から除去 されることを指定します。

   XML 名 *abc:type* は、名前空間 *abc:* が除去された後は、RPG サブフィールド *TYPE* に 一致します。

```
   xml-into info %xml('info1.xml'
                  : 'doc=file ns=remove');
   // info.type = 'Chair'
   // info.qty = 3
   // info.price = 79.99
```

2. オプション *ns=merge* が使用されます。

   以下の例の中では次の定義が使用されています。

```
D info            DS                  QUALIFIED
D   abc_type                     25A   VARYING
D   def_type                     25A   VARYING
D   abc_qty                      10I 0
D   abc_price                     7P 3
```

   ファイル `info2.xml` に以下が含まれていると想定します。

```
<abc:info xmlns:abc="http://www.abc.xyz"
          xmlns:def="http://www.def.xyz">
 <abc:type>Chair</abc:type>
 <abc:qty>3</abc:qty>
 <def:type>Modern</def:type>
 <abc:price>79.99</abc:price>
</abc:info>
```

   この XML 文書には、名前空間 *abc* が付いている名前 *abc:type* が含まれています。

   また、この XML 文書には、名前空間 *def* が付いている名前 *def:type* も含まれています。 このケースで オプション *namespace=remove* を 使用することはできません。もし使用すると、2 つの異なる XML 要素の 名前が、1 つの RPG サブフィールド *TYPE* に一致することになるからです。

   オプション *ns=merge* が指定されます。これは、一致するサブフィールドを XML-INTO 命令が検索する前に、 名前空間 (*abc*) と XML 名の残りの部分を、名前の 2 つの部分を下線で 区切ってマージすることを指示します。

名前空間 *abc* と *type* がマージされた後、 名前 *abc_type* は RPG サブフィールド *ABC_TYPE* と 一致します。 名前 *def_type* と RPG サブフィールド *DEF_TYPE* は、XML 名の 2 つの部分がマージされた後に 一致します。

データ構造名 *info* は、 マージされた XML 名 *abc_info* と一致しないため、path オプション が指定される必要があります。 マージされた名前 *abc_info* が path オプション内で使用 されます。

このタイプの XML 文書を処理する別の方法については、名前空間接頭部オプションを 参照してください。

```
xml-into info %xml('info2.xml'
                   : 'doc=file ns=merge path=abc_info');
// info.abc_type = 'Chair'
// info.def_type = 'Modern'
// info.abc_qty = 3
// info.abc_price = 79.99
```

nsprefix

nsprefix オプションは、XML-INTO で使用されます。

nsprefix オプションは、 オプション *ns=remove* が指定された場合に XML 名から除去された名前空間の 値を RPG プログラムが判別することを可能にします。

nsprefix オプションは、名前空間の値を 受け取るサブフィールドの名前の接頭部を指定します。 *nsprefix* オプション は、オプション *ns=remove* が指定されていない場合は無視されます。

例えば、XML 要素 <abc:def>hello</abc:def> と、オプション *ns=remove* および *nsprefix=PFX_* が指定されている場合、RPG サブフィールド *DEF* は値「hello」を、RPG サブフィールド *PFX_DEF* は値「abc」を受け取ります。

nsprefix オプションに関する規則 :

1. *nsprefix* サブフィールドは英数字タイプまたは UCS-2 タイプでなければなりません。
2. XML データと突き合わせられるサブフィールドが配列である場合、*nsprefix* サブフィールド も、要素数が同じ配列でなければなりません。 XML データと突き合わせ られるサブフィールドが配列ではない場合、*nsprefix* サブフィールド は配列であってはなりません。
3. XML 要素に名前空間が付いていない場合、*nsprefix* サブフィールドには空ストリング '' が入れられます。
4. *nsprefix* サブフィールドとして正しい名前を持つサブフィールド があるが、*nsprefix* サブフィールドであるための基準が満たされない場合、それはエラーとは見なされません。 例えば、*nsprefix*=ns が 指定され、データ構造内に、要素が 2 つある配列サブフィールド *NAME* と、 要素が 3 つある英数字配列サブフィールド *NSNAME* がある場合、 サブフィールド *NSNAME* は *nsprefix* サブフィールドであるとは見なされず、 したがって、XML-INTO はその値を設定するための XML データが見つかることを予期します。
5. *case* オプションは、 *nsprefix* サブフィールドに入れられる名前空間値に影響しません。 例えば、*case=convert* オプション が指定されていて、XML 名が *a--b:name* の場合、値 "a--b" が *nsprefix* サブフィールドに入れられます。
6. *datasubf* サブフィールドには *nsprefix* オプションは考慮されません。

nsprefix オプションの例

1. 以下の例の中では次の定義が使用されています。

```
D info            DS                  QUALIFIED
D   type                         25A   VARYING DIM(2)
D   ns_type                      10A   VARYING DIM(2)
D   qty                          10I 0
D   price                         7P 3
D   ns_price                     10A   VARYING
```

ファイル `info3.xml` に以下が含まれていると想定します。

```
<abc:info xmlns:abc="http://www.abc.xyz"
          xmlns:def="http://www.def.xyz">
```

```
  <abc:type>Chair</abc:type>
  <abc:qty>3</abc:qty>
  <def:type>Modern</def:type>
  <abc:price>79.99</abc:price>
 </abc:info>
```

RPG プログラマーが、RPG サブフィールドのうちのいくつかと一致する XML 名に 使用される名前空間を取得できるように、XML-INTO のオプション *ns=remove nsprefix=ns_* が 指定されます。 オプション *nsprefix=ns_* は、 *NS_* で始まるサブフィールドが、名前空間値を保持する候補であることを 示します。

この XML 文書には、RPG サブフィールド *TYPE* にマップされる 要素が、*abc:type* と *def:type* の 2 つあります。

名前空間が除去された後、名前が *type* の XML 要素が 2 つあるため、*TYPE* サブフィールドは DIM(2) を指定して定義されます。 *NS_TYPE* サブフィールド も DIM(2) を指定して定義されていて、XML-INTO は、 *TYPE* サブフィールドに一致する XML 名の各オカレンスの名前空間値を入れることができます。

XML-INTO は、XML 名 *abc:type* を処理するときに、*TYPE(1)* サブフィールド を値 'Chair' に設定し、*NS_TYPE(1)* サブフィールド を値 'abc' に設定します。

XML-INTO は、XML 名 *def:type* を処理するときに、 *TYPE(2)* サブフィールドを値 'Modern' に設定し、*NS_TYPE(2)* サブフィールドを値 'def' に設定します。

XML-INTO は、XML 名 *abc:qty* を処理するときに、*QTY* サブフィールドを値 3 に設定します。 *NS_QTY* という名前のサブフィールドはないため、名前空間値がサブフィールドに保存されることはありません。

XML-INTO は、XML 名 *abc:price* を処理するときに、*PRICE* サブフィールド を値 79.99 に設定し、*NS_PRICE* サブフィールドを値 'abc' に設定します。

```
xml-into info %xml('info3.xml'
                   : 'doc=file ns=remove nsprefix=ns_');
// info.type(1) = 'Chair'
// info.ns_type(1) = 'abc'
// info.type(2) = 'Modern'
// info.ns_type(2) = 'def'
// info.qty = 3
// info.price = 79.99
// info.ns_price = 'abc'
```

出力 (デフォルトはコンテキストによって異なる)

output オプションは、DATA-GEN で使用されます。

output オプションは、DATA-GEN 命令の開始時に出力変数またはファイルを処理する方法を指定します。

- *append* は、命令の開始時に出力変数またはファイルが変更されないことを示します。 生成プログラムによって提供される情報は、変数またはファイルに追加されます。

  742 ページの『複数の DATA-GEN 命令を使用した単一文書の生成』を参照してください。
- *clear* は、命令が開始する前に変数またはファイルが消去されることを示します。

  これは、第 1 オペランドとして *END が指定されていない DATA-GEN 命令のデフォルトです。
- *continue* は、命令が DATA-GEN 命令のシーケンスの続きであることを示します。

  これは、第 1 オペランドとして *END が指定されている DATA-GEN 命令のデフォルトです。 742 ページの『複数の DATA-GEN 命令を使用した単一文書の生成』を参照してください。

path

path オプションは、DATA-INTO および XML-INTO で使用されます。

path オプションは、スラッシュで区切った項目を使用して、文書内にある項目へのパスを指定します。 例えば、このオプションが path=main/info/name である場合、パーサーは 、最外部の項目が「main」、「main」の子が「info」、さらに「info」の子が「name」であると解釈します。 項目を検出できない場合、その命令は状況コード 00353 (XML 文書が RPG 変数と一致しない) または 00356 (DATA-INTO の文書が RPG 変数と一致しない) で失敗します。

注 : 「allowmissing」オプションの値はこの状態には影響しません。

注 : *path* オプションは、配列処理プロシージャーの指定に %HANDLER が使用された場合に必要です。

デフォルト: *path* オプションが指定されない場合、RPG 変数に一致する項目の検索は、変数の型に依存します。

- 非配列の変数の場合、最外部の項目が RPG 変数と同じ名前であることが必要です。
- 命令コードが XML-INTO であるときに、配列変数の場合、最外部の XML 要素に RPG 配列変数と同じ名前の子要素があることが必要です。 最外部の XML 要素には任意の名前を使用できます。 命令コードが DATA-INTO であるときに、配列変数の場合、最外部の項目は配列と一致することが必要です。

注 :

1. 変数が修飾サブフィールドである場合、文書内の項目へのパスの判別にサブフィールドの名前のみが使用されます。 例えば、変数が DS.SUB1 である場合、デフォルトで文書の最外部の項目の名前は「sub1」であることが必要です。
2. このオプションで指定するパスは、大/小文字を区別します。 *case* オプションも指定されている場合を除いて、文書内の 一致する項目と大/小文字が同じである必要があります。

非配列の変数での *path* オプションの例

以下の例の中では次の定義が使用されています。

```
D info            DS
D   num                          5P 2

D xmlDoc          S             1000A    VARYING

D qualDs          DS                     QUALIFIED
D   subf                        10A
```

1. XML 名 *myinfo* が RPG 名 *info* と 異なっているため、XML 要素の名前を指定するために path オプションが使用されます。

```
 /free

  xmlDoc = '<myinfo><num>123.45</num></myinfo>';
  xml-into info %XML(xmlDoc : 'path=myinfo');
  // num = 123.45
```

2. path オプションは指定されませんが、RPG 名 *info* と XML 名 *myinfo* は異なっています。 XML 文書 に *info* 要素は含まれていないため、XML-INTO 命令は状況コード 00353 で 失敗します。

```
  xmlDoc = '<myinfo><num>456.1</num></myinfo>';
  xml-into(e) info %XML(xmlDoc');
  // %error = '1'
  // %status = 353
```

3. 必要な XML 要素は XML 文書中の最外部の要素ではないため、 必要な XML 要素を位置指定するために *path* オプションが使用されます。

```
  xmlDoc = '<data><info><num>-789</num></info></data>';
  xml-into info %XML(xmlDoc : 'path=data/info');
  // num = -789
```

4. XML-INTO 命令のターゲットは、データ構造ではなくサブフィールド です。 *path* オプションは、*num* RPG サブフィールドに 一致する *num* XML 要素へのパスを指定しています。

```
  xmlDoc = '<data><info><num>.3</num></info></data>';
  xml-into num %XML(xmlDoc :
                    'path=data/info/num');
  // num = .3
```

5. XML 文書はファイルに入っています。したがって、*doc* オプション と *path* オプションの両方の指定が必要です。

```
  //
  //    myfile.xml:
```

```
//      <data>
//       <val>17</val>
//      </data>
xml-into num %XML('myfile.xml' : 'doc=file path=data/val');
// num = 17
```

6. 修飾されたサブフィールドが XML-INTO 命令の ターゲットとして指定されます。 サブフィールド名 *subf* は XML 名 *subf* と 一致しているため、*path* オプションは不要です。

```
xmlDoc = '<subf>-987.65</subf>';
xml-into qualDs.subf %XML(xmlDoc);
// qualDs.subf = '-987.65'
```

7. XML 文書には、要素のレベルが *qualds* と *subf* の 2 つあります。 この XML 文書は RPG qualds データ構造に一致していますが、RPG プログラム は XML 命令のターゲットとして *qualds.subf* と指定しています。 デフォルトのパスはサブフィールドの名前なので、*path* オプション を *path=qualds/subf* と指定する必要があります。 これには、 変数を設定するためのデータが入っている XML 要素も含めて、必要な XML 要素へのパスにあるすべての XML 要素の名前が含まれていなければ なりません。

```
xmlDoc = '<qualds><subf>-987.65</subf></qualds>';
xml-into qualDs.subf %XML(xmlDoc :
                          'path=qualds/subf);
// qualDs.subf = '-987.65'
```

配列変数での *path* オプションの例

以下の例の中では次の定義が使用されています。

```
D loc             DS                  DIM(2)
D   city                        20A   VARYING
D   prov                         2A
D arr             S              5I 0  DIM(3)
D xmlDoc          S            1000A   VARYING
```

1. XML 文書には、反復する *arr* 要素があり、 これらは最外部の XML 要素 *outer* の子です。 これらの反復 XML 要素のデータを 受け取るための XML-INTO 命令のターゲットとして、RPG 配列が指定されています。 RPG 配列 *arr* は、 反復する XML 要素 *arr* の名前 と一致しているため、*path* オプションは不要です。

```
xmlDoc = '<outer>'
            + '<arr>3</arr>'
            + '<arr>4</arr>'
            + '<arr>-2</arr>'
            + '</outer> ;
xml-into arr %XML(xmlDoc);
// arr(1) = 3
// arr(2) = 4
// arr(3) = -2
```

2. myarray.xml に以下が含まれていると想定します。

```
<locations>
  <loc><city>Saskatoon</city><prov>SK</prov></loc>
  <loc><city>Regina</city><prov>SK</prov></loc>
</locations>
```

   XML-INTO 命令のターゲットは、 データ構造からなる配列です。 XML 文書には、コンテナー XML 要素 *locations* 内に、 *loc* という名前の反復 XML 要素が含まれています。 RPG データ構造配列の名前は *loc* であるため、*path* オプション は不要です。 最外部の XML 要素の名前は考慮されません。

```
xml-into loc %XML('myarray.xml' : 'doc=file');
// loc(1).city = 'Saskatoon'  loc(2).city = 'Regina'
// loc(1).prov = 'SK'         loc(2).prov = 'SK'
```

3. mydata.xml に以下が含まれていると想定します。

```
<data>
  <where><city>Edmonton</city><prov>AB</prov></where>
```

```
    <where><city>Toronto</city><prov>ON</prov></where>
  </data>
```

この例は、前の例と似ていますが、 RPG データ構造 *loc* の名前は、 反復 XML 要素 *where* の名前と異なります。 コンテナー XML 要素 *data* の名前と 反復 XML 要素 *where* の名前を使用して、*path* オプションを *path=data/where* と指定する必要があります。

```
xmlfile = 'mydata.xml';
xml-into loc %XML(xmlfile : 'path=data/where doc=file');
// loc(1).city = 'Edmonton'   loc(2).city = 'Toronto'
// loc(1).prov = 'AB'         loc(2).prov = 'ON'
```

renameprefix

renameprefix オプションは、DATA-GEN で使用されます。

renameprefix オプションは、サブフィールドの名前そのものではなく、サブフィールドに生成する名前を指定するための接頭部を指定します。 名前変更サブフィールドの名前は、renameprefix の値にサブフィールド名を付加することによって形成されます。 例えば、データ構造 *meeting* にサブフィールド *meeting.attendees* があり、オプション 「renameprefix=name_」が DATA-GEN 命令の %DATA オプションに指定されているとします。 *meeting.name_attendees* サブフィールドには「会議の出席者」という値があります。 DATA-GEN 命令は *meeting.attendees* の情報を使用して生成プログラムを呼び出すときに、「出席者」という名前ではなく「会議の参加者」を生成プログラムに渡します。 DATA-GEN 命令は、*meeting.name_attendees* に関するいかなる情報も生成プログラムに渡しません。

これ以降の「renameprefix」オプションの説明では、サブフィールド *meeting.name_attendees* を *renameprefix subfield*、*meeting.attendees* を *renamed subfield* と呼びます。

注 :

1. renameprefix サブフィールドは、文字または UCS-2 である必要がありません。 データ構造にすることはできません。
2. オプション「trim=all」が有効な場合、renameprefix サブフィールドの値は先行ブランクおよび後書きブランクから削除されます。
3. renameprefix サブフィールドをデータ構造や配列にすることはできません。
4. サブフィールドに renameprefix の名前があるが、renameprefix サブフィールドの規則を満たしていない場合､そのサブフィールドは正常に処理されます。これは renameprefix サブフィールドであるとは見なされません。
5. 名前変更されたサブフィールドは、任意のタイプのサブフィールドにすることができます。

「renameprefix」オプションの例については、944 ページの『renameprefix オプションの例』を参照してください。

renameprefix オプションの例

この例では、以下の定義を使用します。

1. データ構造 *order_info* は、順序に関する情報を保持するために RPG プログラムによって使用されます。
2. データ構造 *order_info_gen* には *order_info_gen* と同じサブフィールドがあり、DATA-GEN 命令で使用するため、サブフィールドの一部を名前変更するため、およびサブフィールドの一部にカウンターを提供するための追加サブフィールドがあります。
3. The subfields marked with 3 で示されているサブフィールドには、*rename_* で始まり、データ構造の同じレベルにある別のサブフィールドの名前で終わる名前があります。 例えば、*rename_item* と *item* はどちらもデータ構造 *order_info_gen* のサブフィールドであり、*rename_price* と *price* はどちらもデータ構造 *order_info_gen.item* のサブフィールドです。 %DATA 組み込み関数の第 2 オペランドにオプション 「renameprefix=rename-」を指定することにより、DATA-GEN では *rename_* で始まる名前のサブフィールドが renameprefix サブフィールドとして使用されます。 サブフィールド *order_info_gen.item* が生成されるとき、生成プログラムはサブフィールドの名前 (「品目」) ではなく *order_info_gen.rename_item* の値 (「注文した品目」) を受け取ります。
4. EVAL-CORR 命令は、データ構造 *order_info* から、サブフィールドが同じ名前のデータ構造 *order_info_gen* に割り当てるために使用されます。

5. サブフィールド *order_info_gen.num_item* は、getOrder への呼び出しによって戻される要素の数に設定されます。 オプション「countprefix=num_」が DATA-GEN 命令に指定されているため、サブフィールド *order_info_gen.num_item* は *order_info_gen.item* の「countprefix」サブフィールドになります。

   「countprefix」オプションについて詳しくは、「931 ページの『countprefix』」を参照してください。
6. オプション「name=Order」が %DATA のオプション・パラメーターに指定されています。 「name」オプションが指定されない場合、データ構造の生成プログラムに渡される最初の名前は「order_info_gen」です。 オプション「name=Order」が指定される場合、生成プログラムに渡される最初の名前は「注文」です。
7. オプション「countprefix=num_」が %DATA のオプション・パラメーターに指定されています。 このオプションは、カウントされるサブフィールド *item* および *itemName* が生成される数を示すために *num_item* および *num_itemName* が使用されることを示します。
8. %DATA のオプション・パラメーターにオプション「renameprefix=num_」が指定されています。 このオプションは、名前変更されるフィールド *item* および *id* の生成プログラムに渡される名前を示すために renameprefix サブフィールド *rename_id* および *rename_item* が使用されることを示します。

```
DCL-DS order_info QUALIFIED;                      // 1
   dueDate DATE(*ISO);
   item LIKEREC(orders) DIM(20);
END-DS;
DCL-DS order_info_gen QUALIFIED INZ;              // 2
   rename_item VARCHAR(10) INZ('Item ordered');   // 3
   num_item INT(10);
   DCL-DS item DIM(20);
      rename_itemId VARCHAR(20) INZ('Item ID');   // 3
      rename_price VARCHAR(100);                  // 3
      rename_quantity VARCHAR(100);               // 3
      itemId VARCHAR(100);
      price PACKED(7:2);
      quantity INT(10);
   END-DS;
END-DS;
DCL-S num_items INT(10);

num_items = getOrder (order_info);

EVAL-CORR order_info_gen = order_info;            // 4
order_info_gen.num_item = numItems;               // 5

DATA-GEN order_info_gen %DATA('myOrder.txt'
                              : 'doc=file '
                              + 'name=Order '           // 6
                              + 'countprefix=num_ '     // 7
                              + 'renameprefix=rename_') // 8
                        %GEN('MYGENPGM');
```

numItems の値が 2 の場合、*order_info_gen.rename_item(1)* の値は 「ドアの取っ手 X25-E」であり、*order_info_gen.rename_item(2)* の値が「メタルラック X42-B」の場合、以下の名前が生成プログラムに渡されます。

- 「注文」。これは、データ構造 *order_info_gen* の「name」オプションによって指定されます。
- 「品目 ID」。これは、サブフィールド *id* のサブフィールド *rename_id* によって指定されます。
- 「ドアの取っ手 X25-E」。これは、サブフィールド *order_info_gen.Item(1)* の *order_info_gen.rename_item(1)* によって指定されます。
- 「価格」。これは、定義されたときのものと同じ大/小文字混合で定義されている *order_info_gen.Item(1).Price* の名前です。
- 「数量」。これは、定義されたときのものと同じ大/小文字混合で定義されている *order_info_gen.Item(1).Quantity* の名前です。
- 「メタルラック X42-B」。これは、サブフィールド *order_info_gen.Item(2)* の *order_info_gen.rename_item(2)* によって指定されます。

- 「価格」。これは、定義されたときのものと同じ大/小文字混合で定義されている *order_info_gen.Item(2).Price* の名前です。
- 「数量」。これは、定義されたときのものと同じ大/小文字混合で定義されている *order_info_gen.Item(2).Quantity* の名前です。

「name」オプションについて詳しくは、938 ページの『name (デフォルトなし)』を参照してください。

trim (デフォルト all)

trim オプションは、DATA-GEN、DATA-INTO、および XML-INTO で使用されます。

trim オプションは、データを XML-INTO 命令および DATA-INTO 命令の RPG 変数に代入する前に、テキスト・データから 空白文字 (ブランク、改行、タブなど) を削除するかどうか、データを DATA-GEN 命令の生成プログラムに渡す前に、文字、UCS-2、およびグラフィックの RPG 変数のデータからブランクを削除するかどうかを指定します。

- *all* はすべてのデータが削除されることを示します。
- *none* は、データが削除されないことを示します。

XML-INTO および DATA-INTO の「trim」オプションの詳細

- 「trim=all」が指定されている場合、テキスト・コンテンツが RPG 文字または UCS-2 変数に割り当てられる前に、以下のステップが実行されます。
  1. 前後の空白文字が、テキスト・コンテンツから完全にトリムされる
  2. テキスト・コンテンツの内部にある空白文字のストリングが、単一のブランクに削減される
- 「trim=none」が指定されている場合、空白文字はテキスト・コンテンツから削除されません。 このオプションではパフォーマンスが最高になりますが、その使用は、空 白文字が必要な場合、データに不必要な空白文字がないことが既知であ る場合、または RPG プログラムにより空白文字自体の削除が処理される予定の場合のみにする必要があります。

注 :

1. 空白文字には、ブランク、タブ、行の終わり、復帰、改行が含まれます。
2. このオプションは、文字および UCS-2 の RPG 変数に代入されるデータに対してのみ適用されます。 他のデータ型の場合は、空白文字のトリミングが常に実行されます。
3. このオプションは、主にファイルからのデータ用に提供されていますが、変数からのデータにも適用されます。
4. XML-INTO 命令の場合、XML 要素間の空白文字は、常に無視されます。 trim オプションにより、要素および属性のテキスト・コンテンツにおける空白文字が制御されます。

trim オプションの例

以下の例の中では次の定義が使用されます。

```
D data            S            100A    VARYING
```

ファイル `data.xml` に以下の行が含まれていると想定します。

```
<text>
    line1
    line2
</text>
```

この同じファイルの別の表示は以下のようになります。ここでは次の表現が使われています。

'_'
ブランクを表します

'T'
タブを表します

'F'
改行を表します

```
<text>____F
Tline1F
____line2F
</text>F
```

1. デフォルト *trim=all* が使用されます。 先頭と末尾の空白文字は 除去されます。 内部にある空白文字のストリングは単一のブランクに変換されます。

   ```
   xml-into data %XML('data.xml' : 'doc=file');
   // data = 'line1 line2'
   ```

2. オプション *trim=none* が指定されます。 空白文字がテキスト・データから 切り取られることはありません。 結果の 2 つの表示を示します。

   a. 改行文字およびタブ文字は '?' で示されています。

   b. 空白文字、改行文字、およびタブ文字は、上記の文書の 2 番目の表示 と同じように示されていて、次の表現が使われています。

   '_'
   ブランクを表します

   'T'
   タブを表します

   'F'
   改行を表します

   ```
   xml-into data %XML('data.xml' : 'doc=file trim=none');
   // data = '    ??line1?    line2?'
   // data = '____FTline1F____line2F'
   ```

XML-SAX (XML 文書の構文解析)

自由形式構文	XML-SAX{(E) } %HANDLER(*handlerProc* : *commArea*) %XML(*xmlDoc* {: オプション });

コード	演算項目 1	拡張演算項目 2
XML-SAX{(E)}		%HANDLER(*handlerProc* : *commArea*) %XML(*xmlDoc* {: オプション })

ヒント: XML および XML 文書の処理の基本概念について十分理解していない場合は、以下のセクションを読み進める前に、「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」のセクション『XML 文書の処理』を参照することをお勧めします。

XML-SAX は XML 文書の SAX 構文解析を開始します。 XML-SAX 命令コードは、文書の構文解析を開始する XML パーサーを呼び出すことにより開始します。 パーサーによる要素の開始の検出、属性名の検出、および要素の終了の検出などのイベントが発生した場合、パーサーはそのイベントを記述したパラメーターで処理プロシージャー *handlerProc* を呼び出します。 処理プロシージャーが戻ると、パーサーは次のイベントが検出されるまで構文解析を継続し、再度処理プロシージャーを呼び出します。 パーサーがその文書の構文解析を完了すると、XML-SAX 命令の後のステートメントに制御が渡されます。

第 1 オペランドは %HANDLER 組み込み関数にする必要があります。 *handlerProc* は SAX イベントを処理するために呼び出されるプロシージャーを指定するプロトタイプ名であり、*commArea* はパーサーから処理プロシージャーに受け渡される通信域パラメーターです。 通信域パラメーターは、処理プロシージャーの第 1 プロトタイプ・パラメーターを同じタイプである必要があります。 それにより、XML-SAX 命令コードを指定するプロシージャーが処理プロシ ージャーと通信する方法、および処理プロシージャーがあるイベントから次 のイベントに構文解析に関連する情報を保管する方法が提供されます。 %HANDLER について詳しくは、648 ページの『%HANDLER (handlingProcedure : communicationArea)』 を参照してください。

第 2 オペランドは %XML 組み込み関数にする必要があり、構文解析される XML 文書およびその構文解析方法を制御するオプションを識別します。 %XML について詳しくは、708 ページの『%XML (xmlDocument {:options})』 を参照してください。

命令拡張 E を指定すると、以下の状況コードを処理できます。

00351
XML 構文解析時のエラー。

00352
無効な XML オプション。

00354
XML 構文解析の準備時エラー。

状況 00351 の場合、パーサーからの戻りコードは、PSDS の 368 から 371 桁目のサブフィールド「外部戻りコード」に配置されます。 このサブフィールドは、命令の開始時にゼロに設定され、命令の終了時にはパーサーにより戻された値に設定されます。 このサブフィールドは、XML-SAX 命令のあるモジュール内にのみ関係します。 SAX イベント処理プロシージャーは、パーサーからの情報をパラメーターとして受け取ります。

構文解析の開始前に例外が発生した場合には、イベント処理プロシージャーは呼び出されません。 例えば、指定されたファイルが検出されなかった場合、その命令は状況コード 00354 で即時に終了し、イベント処理プロシージャーに制御が移されることはありません。

構文解析時にエラーが発生した場合は *XML_EXCEPTION イベントを使用して処理プロシージャーが呼び出され、処理プロシージャーが戻ったときに構文処理が終了し、XML-SAX 命令は状況コード 00351 で失敗します。 パーサーからの戻りコードは、PSDS の 368 から 371 桁目の「外部戻りコード」サブフィールドに配置されます。

%XML オプション・ストリングにおいて不明、無効、または無関係のオプションが検出された場合、XML-SAX は状況コード 00352 で失敗します。 PSDS の 368 から 371 桁目の外部戻りコード・サブフィールドは、命令の開始時に設定された初期値のゼロから更新されません。

XML-SAX 命令コードの %XML オプション

doc (デフォルト *string*)
doc オプションは、%XML のソース・オペランドに含まれるものを示しています。

- *string* は、ソース・オペランドに XML データが含まれていることを表しています
- *file* は、ソース・オペランドに IFS ファイル名が含まれていることを示しています

```
// In the following example, the first parameter
// of %XML is the name of a file.  Option
// "doc=file" must be specified.
ifsfile = 'myfile.xml';
opt = 'doc=file';
XML-SAX %handler(hdlr:comm) %XML(ifsfile : opt);

// In the following example, the first parameter
// of %XML is an XML document.  Since the "doc"
// option defaults to "string", no options are
// necessary.
xmldata = '<data><num>3</num></data>';
XML-SAX %handler(hdlr:comm) %XML(xmldata);
```

図 *410. doc* オプションの例

ccsid (デフォルト *job*)
ccsid オプションは、XML データが戻される CCSID を指定します。

- *job* は、XML パーサーがジョブ CCSID にデータを戻すことを指示します。 この CCSID は、RPG コンパイラーがプログラム内の文字データ用に使用する CCSID です。
- *ucs2* は、XML パーサーがモジュールの UCS-2 CCSID にデータを戻すことを指示します。

- 数値 は、XML パーサーが指定の CCSID にデータを戻すことを指示します。 この場合、RPG プログラムで正常にデータが処理され るようにプログラムする必要があります。 RPG コンパイラーでは、文字データがジョブ CCSID であると想定します。

```
// In the following example, the data is to be
// returned in the job ccsid.  Even though the
// default for the "ccsid" option is "job", it
// is valid to specify it explicitly.
XML-SAX %handler(hdlr:comm) %XML(xmlString : 'ccsid=job');

// In the following example, the data is to be
// returned in UCS-2.
opt = 'ccsid=ucs2';
XML-SAX %handler(hdlr:comm) %XML(xmldata : opt);

// In the following example, the data is to be
// returned in UTF-8.  The handling procedure must
// exercise caution to convert the data to some CCSID
// that the program can handle, if the data is to be
// used within the handling procedure.
XML-SAX %handler(hdlr:comm) %XML(xmldata : 'ccsid=1208');
```

図 *411. ccsid* オプションの例

注: *XML_UCS2_REF および *XML_ATTR_UCS2_REF イベントの場合、*ccsid* オプションとは無関係に、データは常に UCS-2 の値として戻されます。

XML-SAX イベント処理プロシージャー

イベント処理プロシージャーは、ユーザー作成のプロトタイプ・プロシージャーです。 以下の戻りの型およびパラメーターが必要です。

パラメーター数または戻り値	データ型および引き渡しモード	説明
戻り値	4 バイト整数 (10I 0)	戻り値がゼロの場合、構文解析が継続することを示します。 戻り値がその他の値の場合、構文解析が終了することを示します。
1 – 通信域	任意の型、参照による受け渡し	XML-SAX 命令とハンドラーの間、およびハンドラーの連続呼び出しの間の通信に使用される。
2 – イベント	4 バイト整数 (10I 0)、値による受け渡し	パーサーにより発見された XML イベント。 *XML_START_ELEMENT などの特殊語を使用して、処理プロシージャー内のイベントを識別できます。 950 ページの『XML イベント』を参照してください。
3 – データ	ポインター (*)、値による受け渡し	パラメーターがイベントに関係ない場合、値は *NULL になります。 それ以外は、イベントのデータへのポインターになります。 *XML_UCS2_REF および *XML_ATTR_UCS2_REF イベントの場合、データは常に UCS-2 データです。 その他のすべてのイベントの場合、データは %XML 組み込み関数の "ccsid" オプションにより指定された CCSID のデータです。

パラメーター数または戻り値	データ型および引き渡しモード	説明
4 – 長さ	8 バイト整数 (20I 0)、値による受け渡し	ほとんどのイベントでは、この値は第 3 パラメーターによりポイ ントされたデータの長さ (バイト単位) です。 このパラメーターが特定のイベントに関係ない場合、値は -1 になります。 %XML 組み込み関数の "ccsid" オプションによりデータが UCS-2 で戻されている場合は、この値を 2 で除算し、UCS-2 文字の数を取得する必要があります。 *XML_EXCEPTION イベントの場合、このパラメーターはエラー発生時に構文解析されていた文書の長さになります。
5 – 例外 ID	4 バイト整数 (10I 0)、値による受け渡し	例外 ID。 *XML_EXCEPTION 以外のすべてのイベントの場合、このパラメーターの値はゼロになります。「*Rational Development Studio for i: ILE RPG* プログラマーの手引き」の XML 戻りコードのセクションを参照してください。

%HANDLER について詳しくは、648 ページの『%HANDLER (handlingProcedure : communicationArea)』を参照してください。

```
D saxHandler      pr            10i 0
D   commArea                          likeds(myCommArea)
D   event                       10i 0 value
D   string                       *   value
D   stringlen                   20i 0 value
D   exceptionId                 10i 0 value
```

図 *412. XML-SAX* 処理プロシージャーのプロトタイプの例

XML イベント

XML 文書の SAX 構文解析時には、いくつかの XML イベントがユーザーの XML-SAX 処理プロシージャーに受け渡されます。 ユーザーのプロシージャーにあるイベントを識別するには、*XML で始まる特殊名 (例えば *XML_START_ELEMENT) を使用します。

ほとんどのイベントの場合、処理プロシージャーにはそのイベントに関連付けられた値が渡されます。 例えば *XML_START_ELEMENT イベントの場合、その値は XML 要素の名前です。

表 *134. XML* イベント

イベント	値
1. 最初の XML 要素の前に発見されたイベント	
*XML_START_DOCUMENT	構文解析が開始されたことを示します
*XML_VERSION_INFO	XML 宣言からの "version" 値
*XML_ENCODING_DECL	XML 宣言からの "encoding" 値
*XML_STANDALONE_DECL	XML 宣言からの "standalone" 値
*XML_DOCTYPE_DECL	文書タイプ宣言の値
2. XML 要素に関連したイベント	
*XML_START_ELEMENT	開始中の XML 要素の名前
*XML_CHARS	XML 要素の値
*XML_PREDEF_REF	事前定義参照の値

表 134. XML イベント (続き)	
イベント	**値**
*XML_UCS2_REF	UCS-2 参照の値
*XML_UNKNOWN_REF	不明なエンティティー参照の名前
*XML_END_ELEMENT	終了する XML 要素の名前
3. XML 属性に関連したイベント	
*XML_ATTR_NAME	属性の名前
*XML_ATTR_CHARS	属性の値
*XML_ATTR_PREDEF_REF	事前定義参照の値
*XML_ATTR_UCS2_REF	UCS-2 参照の値
*XML_UNKNOWN_ATTR_REF	不明なエンティティー参照の名前
*XML_END_ATTR	属性の終了を示す
4. XML 処理命令に関連したイベント	
*XML_PI_TARGET	ターゲットの名前
*XML_PI_DATA	データの値
5. XML CDATA セクションに関連したイベント	
*XML_START_CDATA	CDATA セクションの開始
*XML_CHARS	CDATA セクションの値
*XML_END_CDATA	CDATA セクションの終了
6. その他のイベント	
*XML_COMMENT	XML コメントの値
*XML_EXCEPTION	パーサーによるエラーの発見を示す
*XML_END_DOCUMENT	構文解析が終了したことを示す

この XML 文書の例は、XML イベントの記述で参照されます。

```
<?xml version="1.0" encoding="ibm-1140" standalone="yes" ?>
<!DOCTYPE page [
  <!ENTITY abc "ABC Inc">
]>
<!-- This document is just an example  -->
<sandwich>
  <bread type="baker's best" supplier="&abc;" />
  <?spread   please use real mayonnaise ?>
  <spices attr="&#x2B;">Salt &amp; pepper</spices>
  <filling>Cheese, lettuce,
           tomato, &#0061; &xyz;
  </filling>
  <![CDATA[We should add a <relish> element in future!]]>
</sandwich>junk
```

図 413. XML イベントの記述で参照される XML 文書の例

***XML_START_DOCUMENT**
このイベントは、その文書の構文解析の開始時に一度発生します。 先頭の 2 つのパラメーターのみがこのイベントに関係しています。 ストリング・パラメーターにアクセスすると、ポインターがセットされていないというエラーが発生します。

***XML_VERSION_INFO**
このイベントは、XML 宣言にバージョン情報が含まれている場合に発生します。 ストリング・パラメーターの値は、XML 宣言からのバージョン値です。

例:
'1.0'

***XML_ENCODING_DECL**
このイベントは、XML 宣言にエンコード情報が含まれている場合に発生します。 ストリング・パラメーターの値は、XML 宣言からのエンコード値です。

例:
'ibm-1140'

***XML_STANDALONE_DECL**
このイベントは、XML 宣言にスタンドアロン情報が含まれている場合に発生します。 ストリング・パラメーターの値は、XML 宣言からのスタンドアロン値です。

例:
'yes'

***XML_DOCTYPE_DECL**
このイベントは、XML 宣言に DTD (文書タイプ宣言) が含まれている場合に発生します。 文書タイプ宣言は、文字シーケンス「<!DOCTYPE」で始まり、「>」の文字で終わります。

注: これは、XML テキストに区切り文字が含まれる場合の唯一のイベントで す。

ストリング・パラメーターの値は、開始および終了の文字列を含む DOCTYPE の値全体です。

例:

```
'<!DOCTYPE page [LF  <!ENTITY abc "ABC Inc">LF]>'
```

(*LF* は改行文字を表します。)

***XML_START_ELEMENT**
このイベントは、それぞれの要素タグまたは空の要素タグごとに一度発生します。 ストリング・パラメーターの値は、要素名です。

発生する順序での例:

1. 'sandwich'
2. 'bread'
3. 'spices'
4. 'filling'

***XML_CHARS**
このイベントは、コンテンツのそれぞれのフラグメントごとに発生します。 コンテンツは通常、そのテキストが複数の行であっても、単一のストリングで構成されています。 それに参照が含まれる場合、複数のイベントに分割されます。 ストリング・パラメーターの値は、コンテンツのフラグメントです。

例:

1. 'Salt '
2. ' pepper'
3. 'Cheese, lettuce,WWWtomato, '、ここで WWW は複数の「空白文字」 を表します。 注のセクションを参照してください。
4. 「今後、<relish> 要素を追加する必要があります。」

注 :

1. コンテンツのフラグメント '&' では、*XML_PREDEF_REF イベントが発生し、フラグメント '=' では *XML_UCS2_REF イベントが発生します。
2. 値が XML 文書の複数の行にまたがる場合、それには行末文字が含まれ、また、不必要な一連のブランクが含まれていることもあります。 例において、"lettuce," および "tomato" は改行文字および複数のブランクで分離されています。 これらの文字は空白文字 と呼ばれています。 空白文字は XML 要素の間に現れた場合には無視されますが、要素の中に現れた場合にはデータとみなされます。 XML データに不必要な空白文字が含まれている可能性がある場合は、データをトリムしてから使用する必要があります。 不必要な前後の空白文字をトリムするには、以下のコードを使用します。 961 ページの図 417 の例を参照してください。

```
* x'15'=newline  x'05'=tab     x'0D'=carriage-return
* x'25'=linefeed x'40'=blank
D whitespaceChr   C                   x'15050D2540'
/free
    temp = %trim(value : whitespaceChr);
```

***XML_PREDEF_REF**
このイベントは、コンテンツに事前定義の単一文字参照「&」、「'」、「>', '<」、および「"」のいずれかがある場合に発生します。 ストリング・パラメーターの値は、以下のような 1 バイト文字です。

&	&
'	'
>	<
<	>
"	"

注 : 構文解析が UCS-2 で実行されている場合、ストリングは UCS-2 文字です。

例:
'&'、"spices" 要素のコンテンツから。

***XML_UCS2_REF**
このイベントは、コンテンツに形式 '&#dd..;' または '&#xhh..' の参照がある場合に発生します。 ここで 'd' および 'h' は 10 進数および 16 進数字をそれぞれ表します。 ストリング・パラメーターの値は、参照の UCS-2 値です。

注 : このパラメーターは、構文解析が 1 バイト文字で実行されている場合でも、UCS-2 文字 (タイプ C) です。

例:
UCS-2 値 '=' ("=" として表記)、"filling" 要素の最後のフラグメントから。

***XML_UNKNOWN_REF**
このイベントは、コンテンツに現れる、上記の *XML_PREDEF_REF で示された 5 つの事前定義のエンティティー参照以外のエンティティー参照に対して発生します。 ストリング・パラメーターの値は参照の名前であり、開始の '&' と終了の ';' の間にあるデータです。

例:
'xyz'

***XML_END_ELEMENT**
このイベントは、パーサーが要素の終了タグ、または空要素の終了の不等号括弧を検出した場合に発生します。 ストリング・パラメーターの値は、要素名です。

発生する順序での例:

1. 'bread'
2. 'spices'
3. 'filling'

4. 'sandwich'

***XML_ATTR_NAME**
このイベントは、要素タグまたは空の要素タグ内のそれぞれの属性ごとに一度、有効な名前を認識した後に発生します。 ストリング・パラメーターの値は、属性名です。

発生する順序での例:

1. 'type'
2. 'supplier'
3. 'attr'

***XML_ATTR_CHARS**
このイベントは、属性値のそれぞれのフラグメントごとに発生します。 属性値は通常、そのテキストが複数の行であっても、単一のストリングで構成されています。 それに参照が含まれる場合、複数のイベントに分割されます。 ストリング・パラメーターの値は、属性値のフラグメントです。

発生する順序での例:

1. 'baker'
2. 's best'

注 :

1. フラグメント ''' により *XML_ATTR_PREDEF_REF イベントが発生します。
2. 不必要な行末文字および不必要なブランクの処理に関する推奨事項については、*XML_CHARS の説明を参照してください。

***XML_ATTR_PREDEF_REF**
このイベントは、属性値に事前定義された単一文字参照「&」、「&」、「>」、「<」、および「'"」のいずれかが含まれている場合に発生します。 ストリング・パラメーターの値は、以下のような 1 バイト文字です。

&	&
'	'
>	>
<	<
"	"

注: 構文解析が UCS-2 で実行されている場合、ストリングは UCS-2 文字です。

"type" 属性の値の例:
' (アポストロフィ文字、"&apos")

***XML_ATTR_UCS2_REF**
このイベントは、属性値に形式の参照 '&#dd..;' または '&#xhh..;' がある場合に発生します。 ここで 'd' および 'h' は 10 進数および 16 進数字をそれぞれ表します。 ストリング・パラメーターの値は、その参照の UCS-2 値です。

注: このパラメーターは、構文解析が 1 バイト文字で実行されている場合でも、UCS-2 文字 (タイプ C) です。

"attr" 属性の値からの例:
UCS-2 値 '+'、文書内では "+" で表記。

***XML_UNKNOWN_ATTR_REF**
このイベントは、属性に現れる、上記の *XML_ATTR_PREDEF_REF で示された 5 つの事前定義のエンティティー参照以外のエンティティー参照に対して発生します。 ストリング・パラメーターの値は参照の名前であり、開始の '&' と終了の ';' の間にあるデータです。

例:
'abc'

注 : パーサーは DOCTYPE 宣言を構文解析しません。 そのため、エンティティー "abc" が DOCTYPE 宣言で定義されていても、パーサーには未定義であると認識されます。

***XML_END_ATTR**
このイベントは、パーサーが属性値の終了に到達した場合に発生します。 ストリング・パラメーターはこのイベントでは関係ありません。 ストリング・パラメーターにアクセスすると、ポインターがセットされていないというエラーが発生します。

例:
属性 type="baker's best" の場合、属性値の 3 つの部分 ("baker"、' および "s best") がすべて処理された後、*XML_END_ATTR イベントが発生します。

***XML_PI_TARGET**
このイベントは、パーサーが処理命令 (PI) の開始文字列 '<?' に続く名前を認識した場合に発生します。 処理命令により、XML 文書にアプリケーション用の特別な命令を含めることができます。 ストリング・パラメーターの値は、処理命令の名前です。

例:
'spread'

***XML_PI_DATA**
このイベントは、処理命令の終了文字列「?>」までの (終了文字シーケンスは含まない) 処理命令のデータ部分に対して発生します。ストリング・パラメーターの値は、末尾の空白文字を含むが先頭の空白文字は含まない、処理命令データです。

例:
'please use real mayonnaise '

注 : 不必要な行末文字および不必要なブランクの処理の推奨については、*XML_CHARS の説明を参照してください。

***XML_START_CDATA**
このイベントは、CDATA セクションが開始した場合に発生します。 CDATA セクションは、ストリング「<![CDATA[」で開始され、ストリング「]]>」で終了します。このようなセクションは、本来 XML マークアップとして認識される文字が含まれるテキストのブロックを「回避」するために使用されます。 パーサーは単一の *XML_CHARS イベントとして、これらの区切り文字の間の CDATA セクションのコンテンツをパスします。 ストリング・パラメーターの値は、常に開始シーケンス「<![CDATA[」です。

例:

```
'<![CDATA['
```

***XML_END_CDATA**
このイベントは、CDATA セクションが終了した場合に発生します。ストリング・パラメーターの値は、常に終了文字シーケンス「]]>」です。

例:
']]>'

***XML_COMMENT**
このイベントは、XML 文書内のすべてのコメントに対して発生します。 ストリング・パラメーターの値は、前後の空白文字を含む、開始区切り文字「<!--」および終了区切り文字「-->」の間のデータです。

例:
' This document is just an example '

***XML_EXCEPTION**
このイベントは、パーサーがエラーを検出した場合に発生します。 「ストリング」パラメーターは、このイベントで は関係ありません。 ストリング・パラメーターにアクセスすると、ポインターがセットされていないというエラーが発生します。 ストリングの長さのパラメーターの値は、例外が発生した部分を含む、構 文解析された部分までの文書の長さです。 例外 ID パラメーターの値は、パーサーにより割り当てられた例外 ID です。 これらの例外の意味は、*Rational Development Studio for i: ILE RPG* プログラマーの手引きの XML 戻りコードのセクションに記載されています。

例:
パーサーが単語 "junk" を検出し、それが XML 文書の終了後に出現した非空白文字のデータである場合、この例外イベントが発生します。 (XML 文書は、"sandwich" 要素では要素の終了のタグで終了しています。)

***XML_END_DOCUMENT**
このイベントは、構文解析の完了時に発生します。 先頭の 2 つのパラメーターのみがこのイベントに関係しています。 ストリング・パラメーターにアクセスすると、ポインターがセットされていないというエラーが発生します。

注 : XML-SAX 処理プロシージャーのデバッグを補助するため、制御仕様キーワード DEBUG(*XMLSAX) を指定できます。 このキーワードについて詳しくは、331 ページの『DEBUG{(*DUMP | *INPUT | *RETVAL | *XMLSAX | *NO | *YES)}』、および *Rational Development Studio for i: ILE RPG* プログラマーの手引きのデバッグの章を参照してください。 RPG により使用される XML パーサーの制限など、XML の構文解析について詳しくは、*Rational Development Studio for i: ILE RPG* プログラマーの手引きの XML についての章を参照してください。

XML-SAX 命令の例

```
D xmlString       S               C    '<?xml version="1.0"> +
D                                      <elem>data</elem>'
D psds            DS
D   xmlRc                       10I 0  OVERLAY(psds:368)
 /free
   // The XML is in an IFS file.  The "option" operand of %XML specifies
   // that the document operand is the name of an IFS file.
   XML-SAX %HANDLER(mySaxHandler : myHandlerInfo)
           %XML('/home/myuserid/myxml.xml' : 'doc=file');

   // The XML is in a string.  The "option" operand of %XML is not specified.
   XML-SAX %HANDLER(mySaxHandler : myHandlerInfo) %XML(xmlString);
```

図 *414.* 自由形式演算での *XML-SAX* 命令

```
CL0N01Factor1+++++++Opcode&ExtExtended-Factor2++++++++++++++++++++++++++
C                   XML-SAX   %HANDLER(mySaxHandler : myHandlerInfo)
C                             %XML('/home/myuserid/myxml.xml' : 'doc=file')

C                   XML-SAX   %HANDLER(mySaxHandler : myHandlerInfo)
C                             %XML(xmlString)
```

図 *415.* 固定形式演算での *XML-SAX* 命令

```
H DEBUG(*XMLSAX)
Fqsysprt   o    f  132        printer

 * The xmlRc subfield will be set to a non-zero value
 * if the XML-SAX operation fails because of an error
 * discovered by the parser

D psds           SDS
D   xmlRc                      10I 0  OVERLAY(psds:368) [1]

D qsysprtDs       DS           132

 * This data structure defines the type for the parameter
 * passed to the SAX handling procedure.
 [2]
D value_t         S             50A    VARYING
D handlerInfo_t   DS                   QUALIFIED
D                                      BASED(dummy)
D   pValue                      *
D   numAttendees               5P 0
D   name                               LIKE(value_t)
D   company                            LIKE(value_t)
D   alwExtraAttr               1N
D   handlingAttrs...
D                                N

 * Define a specific instance of the handlerInfo_t data
 * structure and the prototype for the handler
D myHandlerInfo   DS                   LIKEDS(handlerInfo_t)
D mySaxHandler    PR            10I 0
D   info                               LIKEDS(handlerInfo_t)
D   event                      10I 0  VALUE
D   stringPtr                    *    VALUE
D   stringLen                  20I 0  VALUE
D   exceptionId                10I 0  VALUE


 /free
     monitor;
        // Start XML parsing
        // Indicate that the handler should not allow
        // any unexpected attributes in the XML elements.
        myHandlerInfo.alwExtraAttr = *OFF; [3]
        XML-SAX %HANDLER(mySaxHandler : myHandlerInfo)
                %XML('/home/myuserid/myxml.xml' : 'doc=file');
        // The XML parse completed normally
        // Results are passed back in the communication
        // area specified by the %HANDLER built-in function
        qsysprtDs = 'There are '
                  + %CHAR(myHandlerInfo.numAttendees)
                  + ' attendees.';
     on-error 00351;
        // The XML parse failed with a parser error.
        // The return code from the parser is in the PSDS.
```

図 *416. XML-SAX* 処理プロシージャーを説明する、完全な作動プログラム

```
        qsysprtDs = 'XML parser error: rc='
                  + %CHAR(xmlRc)
                  + '.';
     endmon;

     write qsysprt qsysprtDs;
     *inlr = '1';
 /end-free

P mySaxHandler    B
D                 PI            10I 0
D   info                              LIKEDS(handlerInfo_t)
D   event                       10I 0 VALUE
D   stringPtr                     *   VALUE
D   stringLen                   20I 0 VALUE
D   exceptionId                 10I 0 VALUE

D value           S                   LIKE(value_t)
D                                     BASED(info.pValue)

D chars           S          65535A   BASED(stringPtr)
D ucs2            S          16383C   BASED(stringPtr)
D ucs2Len         S             10I 0

 /free

  select;
```

```
    // start parsing
    when event = *XML_START_DOCUMENT; [4]
    clear info;

    // start processing an attendee, by indicating
    // that subsequent calls to this procedure should
    // handle XML-attribute events.
    when event = *XML_START_ELEMENT;
      if %subst(chars : 1 : stringLen) = 'attendee';
         info.handlingAttrs = *ON; [5]
         info.name = '';
         info.company = '';
         info.numAttendees += 1;
      endif;

    // display information about the attendee
    when event = *XML_END_ELEMENT;
      if %subst(chars : 1 : stringLen) = 'attendee';
         info.handlingAttrs = *OFF;
         qsysprtDs = 'Attendee '
                   + info.name
                   + ' is from company '
                   + info.company;
         write qsysprt qsysprtDs;
      endif;

    // prepare to get an attribute value by setting
    // a basing pointer to the address of the correct
    // variable to receive the value
    when event = *XML_ATTR_NAME;
      if info.handlingAttrs;
        if %subst(chars : 1 : stringLen) = 'name';
           info.pValue = %addr(info.name);
        elseif %subst(chars : 1 : stringLen) = 'company';
           info.pValue = %addr(info.company);
        else;
           // If the XML element is not expected to have
           // extra attributes, halt the parsing by
           // returning -1.
           if not info.alwExtraAttr;
              qsysprtDs = 'Unexpected attribute '
                        + %subst(chars : 1 : stringLen)
                        + ' found.';
              write qsysprt qsysprtDs;
              return -1; [6]
           endif;
           info.pValue = *NULL;
        endif;
     endif;
```

```
    // handle an exception
    when event = *XML_EXCEPTION;
        qsysprtDs = 'Exception '
                  + %char(exceptionId)
                  + ' occurred.';
        write qsysprt qsysprtDs;
        return exceptionId;

    other;

      // If this is an attribute we are interested
      // in, the basing pointer for "value" has been
      // set to point to either "name" or "company"

      // Append each fragment of the value to the
      // current data
      if  info.handlingAttrs
      and info.pValue <> *NULL;
        if event = *XML_ATTR_CHARS
        or event = *XML_ATTR_PREDEF_REF;
           value += %subst(chars : 1 : stringLen);
        elseif event = *XML_ATTR_UCS2_REF;
           ucs2Len = stringLen / 2; [7]
           value += %char(%subst(ucs2 : 1 : ucs2Len));
        endif;
      endif;
    endsl;

    return 0; [8]
 /end-free
P mySaxHandler    E
```

この例では、SAX 構文解析のいくつかの機能を説明しています。

1. これは、xmlRc という名前の PSDS の「外部戻りコード」サブフィールド。
2. 通信域データ構造、XML-SAX 命令と SAX イベント処理プロシージャーの間の通信に使用される。
3. XML-SAX 命令が XML 文書の構文解析を開始。
4. SAX イベント処理プロシージャーが、イベント・パラメーターを特殊名 *XML_START_DOCUMENT などと比較。
5. 通信域は、イベント処理プロシージャーが自身と呼び出しの間に通信するためにも使用される。
6. イベント処理プロシージャーがエラーを発見し、-1 を戻して構文解析を停止。
7. *XML_ATTR_UCS2_REF イベントには、通常はこの XML-SAX 命令がデータを戻す場合に使用する CCSID とは無関係に、UCS-2 データがある。 長さはデータのバイト数を表すため、UCS-2 文字の数を取得するには、それを 2 で除算する必要があります。
8. イベント処理プロシージャーは、エラーを発見しなかった場合には 0 を戻し、構文解析が継続可能であることを示す。

この例では、以下の XML 文書のサンプルを使用できます。

```
<meeting>
  <attendee name="Jack" company="A&amp;amp;B Electronics"/>
  <attendee company="City&amp;#x2B; Waterworks" name="Jill"/>
  <attendee name="Bill" company="Ace Movers" extra="yes"/>
</meeting>
```

```
 // The following procedure returns a string that is the same
 // as the input string except that strings of whitespace are
 // converted to a single blank.
P rmvWhiteSpace   b
D rmvWhiteSpace   pi         65535a   varying
D   input                    65535a   varying const
D output          s                   like(input) inz('')

 * x'15'=newline  x'05'=tab     x'0D'=carriage-return
 * x'25'=linefeed x'40'=blank
D whitespaceChr   C                   x'15050D2540'
D c               s              1A
D i               s             10I 0
D inWhitespace    s               N   INZ(*OFF)
 /free
      // copy all non-whitespace characters to the return value
      for i = 1 to %len(input);
         c = %subst(input : i : 1);
         if %scan(c : whitespaceChr) > 0;
            // If this is a new set of whitespace, add one blank
            if inWhitespace = *OFF;
               inWhitespace = *ON;
               output += ' ';
            endif;
         else;
            // Not handling whitespace now.  Add character to output
            inWhitespace = *OFF;
            output += c;
         endif;
      endfor;
      return output;
 /end-free
P rmvWhiteSpace   e
```

図 *417. XML* データからの内部空白文字の除去

XML 命令について詳しくは、596 ページの『XML 命令』を参照してください。

Z-ADD (ゼロにして加算)

自由形式構文	(許可されていない - EVAL 命令コードを使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
Z-ADD (H)		加数	和	+	-	Z

演算項目 2 がゼロのフィールドに加算されます。 合計が結果フィールドに入れられます。 演算項目 1 は使用されません。 演算項目 2 は数値でなければならず、配列、配列要素、フィールド、形象定数、 リテラル、名前のついた定数、サブフィールド、またはテーブル名のいず れかを入れることができます。

結果フィールドは数値でなければならず、配列、配列要素、フィールド、 サブフィールド、またはテーブル名のいずれかを入れることができます。

四捨五入を指定することができます。

Z-ADD 命令の規則については、557 ページの『算術演算』 を参照してください。

Z-ADD 命令の例については、560 ページの図 181 を参照してください。

Z-SUB (ゼロにして減算)

自由形式構文	(許可されていない - EVAL 命令コードを使用)

コード	演算項目 1	演算項目 2	結果フィールド	標識		
Z-SUB (H)		減数	差	+	-	Z

演算項目 2 がゼロのフィールドから引かれます。 差 (演算項目 2 の負数) が 結果フィールドに入れられます。 この命令を使用して、フィールドの符号を変更することができます。 演算項目 1 は使用されません。 演算項目 2 は数値でなければならず、配列、配列要素、フィールド、形象定数、 リテラル、名前のついた定数、サブフィールド、またはテーブル名のいず れかを入れることができます。

結果フィールドは数値でなければならず、配列、配列要素、フィールド、 サブフィールド、またはテーブル名のいずれかを入れることができます。

四捨五入を指定することができます。

Z-SUB 命令の規則については、557 ページの『算術演算』を参照してください。

Z-SUB 命令の例については、560 ページの図 181 を参照してください。

付録

付録 A. RPG IV 制約事項

機能	制約事項
コンパイル時の配列/テーブルの入力レコード長	最大長は 100
文字フィールド長	固定長文字フィールドの最大長は、16773104 です。 可変長文字フィールドの最大長は、16773100 です。
図形フィールドまたは UCS-2 フィールドの長さ	固定長の図形フィールドまたは UCS-2 フィールドの最大長は、8386552 です。 可変長の図形フィールドまたは UCS-2 フィールドの最大長は、8386550 です。
制御フィールド (入力仕様の 63 桁目と 64 桁目) の長さ	最大長は 256
名前のついたデータ構造の長さ	最大 16773104
名前のないデータ構造の長さ	最大 16773104
データ構造のオカレンス数	データ構造当たり最大 16773104、最大合計サイズ 16773104
ネストされたデータ構造サブフィールドのネスト・レベル	グローバル・データ構造の場合は最大 198、サブプロシージャーで定義されているデータ構造の場合は 197。
編集語	最大長 115
配列/テーブルの要素 (定義仕様書の DIM キーワード)	配列当たり最大 16773104、最大合計サイズ 16773104
構造化グループのネスト・レベル	最大 100
式のネスト・レベル	最大 100
先読み	1 つのファイルに 1 回だけ指定できる。 指定できるのは 1 次ファイルと 2 次ファイルだけである。
名前付き定数またはリテラル	文字リテラルまたは 16 進リテラルの場合は、最大長 16380 文字。 図形リテラルの場合は、最大長 16379 文字 (DBCS)。 UCS-2 リテラルの場合は、最大長 8190 文字 (UCS-2)。 数値リテラルの場合は、最大長 63 桁と小数点以下の桁数 63。
オーバーフロー標識	1 つの印刷出力ファイルについて固有のオーバーフロー標識を 1 つだけ 指定することができる。
プログラムに対するパラメーター	最大 255
プロシージャーに対するパラメーター	最大 399
1 次ファイル (ファイル仕様書の 18 桁目に P)	1 つのプログラムにつき最大 1 つ

機能	制約事項
グローバル・プリンター・ファイル (メイン・ソース・セクションに定義される)	1 つのプログラムにつき最大 8
1 ページ当たりの印刷行数	最小 2、最大 255
プログラム状況データ構造	1 つのプログラムにつき 1 つだけ
レコード・アドレス・ファイル (ファイル仕様書の 18 桁目に R)	1 つのプログラムにつき 1 つだけ
ファイルのレコード長	最大長は 99999 [1]
構造化グループ (ネストのレベルを参照)	
記憶域の割り振り	最大長は 16776704 [2]
記号名	最大長は 4096

注:

1. 装置レコード・サイズの制約があれば、それがこの値に優先します。
2. 実際の最大値は、普通これよりかなり小さくなります。

付録 B. EBCDIC 照合順序

表 *135. EBCDIC* 照合順序

序数 番号	Symbol	意味	10 進数 表記	16 進表記
65	␣	スペース	64	40
. . .				
75	¢	セント記号	74	4A
76	.	ピリオド、小数点	75	4B
77	<	より小記号	76	4C
78	(	左括弧	77	4D
79	+	プラス符号	78	4E
80	|	縦線、論理 OR	79	4F
81	&	アンパーサンド	80	50
. . .				
91	!	感嘆符	90	5A
92	$	ドル記号	91	5B
93	*	アスタリスク	92	5C

表 *135. EBCDIC* 照合順序 *(*続き*)*

序数 番号	Symbol	意味	10 進数 表記	16 進表記
94	)	右括弧	93	5D
95	;	セミコロン	94	5E
96	¬	論理 NOT	95	5F
97	-	マイナス、ハイフン	96	60
98	/	スラッシュ	97	61
. . .				
107	ª	縦の分割線	106	6A
108	,	コンマ	107	6B
109	%	パーセント記号	108	6C
110	_	下線	109	6D
111	>	より大記号	110	6E
112	?	疑問符	111	6F
. . .				
122	`	アクサングラーブ	121	79
123	:	コロン	122	7A
124	#	番号記号、ポンド記号	123	7B
125	@	単価記号	124	7C
126	'	アポストロフィ、プライム符号	125	7D
127	=	等号	126	7E
128	"	引用符	127	7F
. . .				
130	a		129	81
131	b		130	82
132	c		131	83
133	d		132	84
134	e		133	85
135	f		134	86

表 *135. EBCDIC* 照合順序 *(*続き*)*				
序数 番号	**Symbol**	**意味**	**10 進数 表記**	**16 進表記**
136	g		135	87
137	h		136	88
138	i		137	89
. . .				
146	j		145	91
147	k		146	92
148	l		147	93
149	m		148	94
150	n		149	95
151	o		150	96
152	p		151	97
153	q		152	98
154	r		153	99
. . .				
162	~	波形記号	161	A1
163	s		162	A2
164	t		163	A3
165	u		164	A4
166	v		165	A5
167	w		166	A6
168	x		167	A7
169	y		168	A8
170	z		169	A9
. . .				
193	{	左中括弧	192	C0
194	A		193	C1
195	B		194	C2

表 *135. EBCDIC* 照合順序 *(*続き*)*

序数 番号	Symbol	意味	10 進数 表記	16 進表記
196	C		195	C3
197	D		196	C4
198	E		197	C5
199	F		198	C6
200	G		199	C7
201	H		200	C8
202	I		201	C9
. . .				
209	}	右中括弧	208	D0
210	J		209	D1
211	K		210	D2
212	L		211	D3
213	M		212	D4
214	N		213	D5
215	O		214	D6
216	P		215	D7
217	Q		216	D8
218	R		217	D9
. . .				
225	\	逆斜線	224	E0
. . .				
227	S		226	E2
228	T		227	E3
229	U		228	E4
230	V		229	E5
231	W		230	E6
232	X		231	E7

表 *135. EBCDIC* 照合順序 *(*続き*)*

序数 番号	Symbol	意味	10 進数 表記	16 進表記
233	Y		232	E8
234	Z		233	E9
. . .				
241	0		240	F0
242	1		241	F1
243	2		242	F2
244	3		243	F3
245	4		244	F4
246	5		245	F5
247	6		246	F6
248	7		247	F7
249	8		248	F8
250	9		249	F9

注 : これらの記号は、すべてのコード・ページで同じでない場合があります。 コード・ページは、 様々な言語ごとに、各記号に異なる 16 進値を指定することがあります。 詳しくは、IBM i Information Center のグローバリゼーションに関するトピックを参照してください。

参照文献

ILE RPG プログラミングに関するトピックについて詳しくは、以下の資料を参照してください。

- 「*CL プログラミング*」*(SD88-5038)*では、オブジェクトとライブラリー、CL プログラミング、プログラム間の制御の流れと連絡、CL プログラムでのオブジェクトの処理、および CL プログラムの作成についての概要説明など、プログラミングに関するトピックが広範囲にわたって説明されています。 その他のトピックとしては、事前定義メッセージと即時メッセージおよび メッセージの処理、ユーザー定義のコマンドとメニューの定義と作成、デバッグ・ モード、ブレークポイント、追跡、および表示機能を含むアプリケーションのテストなどが入って います。

  IBM i 制御言語 (CL) およびそのコマンドの説明については、IBM i Information Center の「プログラミング」のカテゴリー (URL http://www.ibm.com/systems/i/infocenter/) を参照してください。
- 「*Communications Management*」*(SC41-5406)* では、通信環境における実行管理機能、通信状況、通信障害の追跡および診断、エラーの処理と回復手順、パフォーマンス、特定の回線速度、およびサブシステム記憶域情報について説明されています。
- アプリケーション・プログラムにおけるファイルの使用、データベース編成、データ記述仕様 (DDS) および DDS キーワード、分散データ管理 (DDM)、組み込み SQL プログラミング、およびアプリケーション・プログラミング・インターフェースなどの、データベース・プログラミングに関連するトピックについては、IBM i Information Center の「データベースとファイル・システム」の カテゴリーを参照してください。
- 「*Experience RPG IV Multimedia Tutorial*」*(GK2T-9882-00)* は、RPG III と RPG IV の相違点および新しい ILE 環境内での作業方法について説明している対話式の自習プログラムです。 付属のワークブックには追加の練習問題が載せられていて、学習終了時には解説書として役立ちます。 このチュートリアルと一緒に出荷されている ILE RPG コードの例は、オペレーティング・システムで直接実行することができます。
- 「*ILE 概念*」*(SC41-5606)* では、統合言語環境 (ILE) アーキテクチャーに関する概念および用語が説明されています。 扱われているトピックとしては、モジュールの作成、プログラムのバインディング、プログラムの実行、 プログラムのデバッグ、および例外の処理があります。
- *Rational Development Studio for i: ILE RPG プログラマーの手引き, (SD88-5042)* では、ILE RPG プログラミング言語 (統合言語環境 (ILE) で実行される RPG IV 言語の実装) について説明されています。 プログラムの作成および実行に関する情報と、プロシージャー呼び出しおよび言語間プログラミングに関する考慮事項が記載されています。 また、この手引きにはデバッグおよび例外処理についての説明があり、RPG プ ログラムのファイルおよび装置を使用する方法についての説明もあ ります。 付録には、RPG IV へのマイグレーションに関する情報およびサンプル・コンパイラー・リストが記載されています。 この資料は、データ処理の概念および RPG プログラミング言語の基礎を理解している方を対象としています。
- 「*Who Knew You Could Do That with RPG IV? A Sorcerer's Guide to System Access and More*」(SG24-5402) には、RPG IV および 統合言語環境 (ILE) の利点を最大に利用するシステム・プログラマー向けのヒントが記載されています。

現行の IBM i および IBM i 情報および資料については、以下の Web サイトの IBM i Information Center から入手することができます。

```
http://www.ibm.com/systems/i/infocenter/
```

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合が IBM 日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。 本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。 これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の 製品、プログラム、またはサービスを使用することができます。 ただし、IBM 以外の製品、プログラム、またはサービスの操作を評価および検証するのは、ユーザーの責任です。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。 本書の提供は、お客様にこれらの特許権について 実施権を許諾することを意味するものではありません。 実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

IBM ライセンス交付のディレクター
IBM Corporation
日本アイ・ビー・エム株式会社
法務・知的財産
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to: -->

Intellectual Property Licensing
Legal and Intellectual Property Law
日本 IBM 会社
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、 商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含む すべての明示もしくは黙示の保証責任を負わないものとします。 国または地域によっては、法律の強行規定により、保証責任の制限が 禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。 本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、 改良または変更を行うことがあります。

IBM 以外の Web サイトへのこの情報の参照は、便宜のために提供されており、それらの Web サイトの推奨事項としてはいかなる方法でも提供されません。 それらの Web サイトにある資料は、この IBM 製品の資料の一部では ありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、 自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Corporation
Software Interoperability Coordinator, Department YBWA
2800 丁目 NW
ロチェスター、 MN 55901-4441
U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用すること ができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項 IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で 決定されたものです。 そのため、他の操作環境で得られた結果は、異なる可能性があります。 一部の測定が、開発レベルのシステムで行われた可能性がありますが、 その測定値が、一般に利用可能なシステムのものと同じである保証はありません。 さらに、一部の測定値が、推定値である可能性があります。 実際の結果は、異なる可能性があります。 お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、それらの製品の供給者、公開された発表、またはその他の公に利用可能なソースから入手したものです。 IBM はこれらの製品をテストしていないため、IBM 以外の製品に関連するパフォーマンス、互換性、またはその他のクレームの正確性を確認できません。 IBM 以外の製品の機能に関する質問は、それらの製品の供給者にお願いする必要があります。

IBM の今後の方向性または意図に関するすべての記述は、予告なく変更または撤回される場合があります。これらは、目標および目的を提示するのみです

本書はプランニング目的としてのみ記述されています。 記述内容は 製品が使用可能になる前に変更になる場合があります。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。 より具体性を与えるために、それらの例には、個人、企業、ブランド、 あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、 名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。 お客様は、サンプル・プログラムが書かれているオペレーティング・ プラットフォームのアプリケーション・プログラミング・インターフェースに 準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、 いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、 配布することができます。 このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。 従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性が あることをほのめかしたり、保証することはできません。 これらのサンプル・プログラムは特定物として現存するままの状態で提供されるものであり、いかなる保証も提供されません。 IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次の ように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (年). このコードの一部は、IBM Corp. から派生したものです。 サンプル・プログラム。

© Copyright IBM Corp. _年を入れる_.

プログラミング・インターフェース情報

この「ILE RPG 解説書」資料には、プログラムを作成するユーザーが IBM i のサービスを使用するためのプログラミング・インターフェースが記述されています。

商標

IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corporation の商標です。 他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。 現時点での IBM の商標リストについては、"www.ibm.com/legal/copytrade.shtml" をご覧ください。

Adobe、Adobe ロゴ、PostScript、PostScript ロゴは、Adobe Systems Incorporated の米国およびその他の国における登録商標または商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

・ UNIX は、 The Open Group の米国およびその他の国における登録商標です。

ジャワ およびすべての ジャワ製品およびロゴは Oracle, Inc. の米国およびその他の国における商標または登録商標です。

他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。

使用条件

これらの資料は、以下の条件に同意していただける場合に限りご使用いただけます。

個人使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。 ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、 二次的著作物を作成したり、配布 (頒布、送信を含む) または表示 (上映を含む) することは できません。

商業的使用: これらの資料は、すべての著作権表示その他の所有権表示を していただくことを条件に、お客様の企業内に限り、複製、配布、および表示することが できます。 ただし、IBM の明示的な承諾をえずにこれらの資料の二次的著作物を作成したり、 お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。

資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと 判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものと させていただきます。

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入 関連法規を含む、すべての関連法規を遵守するものとします。

IBM は、これらの資料の内容 についていかなる保証もしません。 これらの資料は、特定物として現存するままの状態で提供され、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任なしで提供されます。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

[タ行]

[ナ行]

[ハ行]

[マ行]

[ヤ行]

[ラ行]

[ワ行]

[数字]

A

B

C

D

E

F

G

I

J

T

U

V

W

X

Y

Z

[特殊文字]

プログラム番号: 5770-WDS

SC09-2508-12