Enterprise COBOL for z/OS 6.3

カスタマイズ・ガイド

SC43-3366-02

注記

本書および本書で紹介する製品をご使用になる前に、89 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。

本書は、IBM® Enterprise COBOL for z/OS® バージョン 6 リリース 3 (プログラム番号 5655-EC6) および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。 製品のレベルに合った正しい版をご使用ください。

Enterprise COBOL for z/OS ライブラリー においてソフトコピー資料を無償で表示またはダウンロードできます。 Enterprise COBOL for z/OS が継続的デリバリー (CD) モデルをサポートしていて、その CD モデルでデリバリーされる機能を文書化するために資料が更新されるため、更新がないかを 2 カ月ごとに確認することは良い考えです。

資料番号を更新することなく、本リリースの製品資料を定期的に更新するのが当社の意図です。 製品資料のバージョンを一意的に参照する必要がある場合は、更新日とともに資料番号を参照してください。

目次

図

表

前書き

本書について

本書は、IBM Enterprise COBOL for z/OS をそれぞれの設置場所でカスタマイズする責任のあるシステム・プログラマーを対象に書かれています。 本書は、z/OS のもとで Enterprise COBOL のカスタマイズの計画と実行を行う際に必要となる情報について説明しています。 また、組織にとっての Enterprise COBOL の価値を査定するために役立つ情報も含まれています。

本書で使用している「オペレーティング・システム」という総称用語 は、z/OS を指します。

本書を使用し、正常にカスタマイズを行うためには、 Enterprise COBOL およびシステム稼働環境を 理解している必要があります。

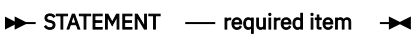
構文図の読み方

このセクションでは、本書中の構文図の読み方を説明します。

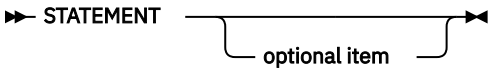
- 構文図は、左から右、上から下へと線をたどってください。下の表は、構文図の線の始まりと終わりにある記号の意味を示しています。

記号	意味
▶▶──	構文図の始まりを示しています。
──▶	構文図が次の線に続くことを示しています。
▶──	構文図が前の線から続いていることを示しています。
──▶◀	構文図の終わりを示しています。

  完全なステートメント以外の構文単位の図は、▶── 記号で始まり、──▶ 記号で終わっています。
- 必須項目は、横線 (幹線) 上に示されています。

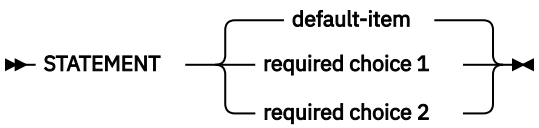


- 任意指定項目は、幹線の下に示されています。

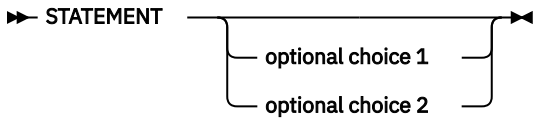


- 2 つ以上の項目から選択できる場合は、それらの項目が縦に重ねられています。

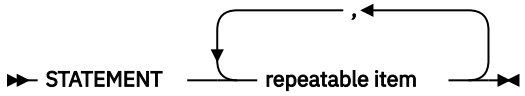
  いずれか 1 つの項目を選択しなければならない 場合は、重ねられた項目のうちの 1 つが幹線上に示されています。デフォルトがあれば、それが幹線上に示されます。 ユーザーが別の選択項目を指定しない場合は、IGYCOPT マクロはこのデフォルトを選択します。デフォルトは、プログラムが実行されているシステム によって異なる場合があります。



  いずれか 1 つの項目の選択が任意である場合は、重ねられた項目全体が幹線の下に示されています。



- 幹線から分岐して左へ戻る矢印は、反復可能な項目を示しています。



  重ねられた項目の上に反復矢印がある場合は、重ねられた項目から 2 つ以上の 項目を選択するか、または 1 つの項目を繰り返すことができます。
- キーワードは大文字で示されています (例えば、PRINT)。 キーワードは、示されているとおりに入力しなければなりません。変数はイタリック体で 示されています (例えば、*item*)。 それらの変数は、ユーザーが指定する名前や値を表します。
- 句読記号、括弧、算術演算子などの記号が示されている場合は、それらを構文の一部として入力しなければなりません。
- パラメーターを区切るには、1 つ以上のブランクまたはコンマを 使用してください。

構文図におけるアスタリスク (*) の意味と詳しい説明については、13 ページの『コンパイラー・オプションの構文および説明』を参照してください。

マクロ計画ワークシートの使用

本書に記載されている計画ワークシート (4 ページの『コンパイラー・オプション用の IGYCDOPT ワークシート』) を、Enterprise COBOL のカスタマイズを準備するために使用できます。 これらのワークシートに記入することにより、IBM 提供のデフォルトから変更する必要のある値を容易に判別することができます。記入後のワークシートを、IBM 提供のデフォルト値をカスタマイズする際の基礎として使用することもできます。

各ワークシートのヘッディングは、互いに少し異なります。 コンパイラー・オプション用ワークシートの各欄のヘッディングの説明については、以下の定義リストを参照してください。

コンパイラー・オプション
: 特定のインストール・マクロに含まれるオプション。 この欄には、マクロに指定するとおりの形でオプションが示されています。

固定の場合 * を記入
: アプリケーション・プログラマーがオーバーライドできないオプション。 固定するオプションについてのみ、アスタリスク (*) を記入してください。

選択を記入
: 各オプションに関連付けられる値。 用意されているスペースに、各オプションに割り当てたい値を記入してください。 適切な値を選択できるよう支援するため、「**構文の説明**」欄に参照情報が示されています。

IBM 提供のデフォルト
: オプションを変更しなかった場合に 指定のインストール・マクロに渡される値。IBM 提供のデフォルトが、設定する値と同じである場合は、その特定のマクロ内のオプションを変更する必要はありません。

構文の説明
: 構文図およびオプションについての具体的情報が含まれている トピック。

ワークシートが完成したら、IBM 提供のデフォルトと異なるオプションを確認してください。それらの項目をインストール・マクロにコーディングする必要があります。ワークシートの各項目は、それらの項目の順序が実際のコーディングのセマンティクスと整合するような順序で並べられています。

変更の要約

このセクションでは、Enterprise COBOL for z/OS バージョン 6 リリース 3 以降で本書に対して行われた重要な変更を示します。 最新の技術的な変更は、HTML 版では >| と |< で囲まれ、PDF 版では左側の余白にある縦棒 (|) で示されます。

バージョン 6 リリース 3 (PTF インストール済み)

コンパイラー・オプションの変更

- 新しいコンパイラー・オプション:
  - PH34804: TUNE: 新しい TUNE オプションで、実行可能プログラムの最適化の対象となるアーキテクチャーが指定されるようになりました。(69 ページの『TUNE』)
  - PH37328: INVDATA: 非推奨の ZONEDATA コンパイラー・オプションが新しい INVDATA コンパイラー・オプションに置き換えられ、無効なデータが含まれた USAGE DISPLAY および USAGE PACKED-DECIMAL のデータ項目を処理するためのコードをコンパイラーで生成する方法を、ユーザーがきめ細かく制御できるようになりました。(36 ページの『INVDATA』)
- 変更されたコンパイラー・オプション:
  - PH22581: INITCHECK: INITCHECK オプションに新しいサブオプション LAX | STRICT が追加されました。このサブオプションは、データ項目がステートメントへの論理パスのうち 1 つ以上で初期化されているという条件と、ステートメントへのすべての論理パスで初期化されているという条件の、どちらの条件に当てはまらない場合にコンパイラーでデータ項目に関する警告メッセージを出すかを制御します。(33 ページの『INITCHECK』)
  - PH27536: NUMCHECK=(ZON): NUMCHECK=(ZON) オプションに新しいサブオプション LAXREDEF | STRICTREDEF が追加されました。このサブオプションは、再定義された項目をコンパイラーで検査してそれについての警告メッセージを出すかどうかを制御します。(45 ページの『NUMCHECK』)
  - PH29542: NUMCHECK=(BIN): NUMCHECK=(BIN) オプションに新しいサブオプション TRUNCBIN | NOTRUNCBIN が追加されました。このサブオプションは、バイナリー・データ項目を検査するコードをコンパイラーで生成するかどうかを制御します。(45 ページの『NUMCHECK』)
  - PTF UI71591 (APAR 番号なし): 数値の受信側に移動されているコンテンツの所有者である英数字の送信側を検査する新しい機能が NUMCHECK に追加されました。数値の受信側に移動されているコンテンツの所有者である英数字の送信側については、コンパイラーはこの送信側を整数の数値として扱うため、NUMCHECK は英数字の送信側ごとに暗黙的な数値のクラス・テストを生成します。(45 ページの『NUMCHECK』)
  - PH33122: RULES: RULES オプションに新しいサブオプション LAXREDEF | NOLAXREDEF が追加されました。このサブオプションは、長さが一致しない再定義済みの項目をユーザーに通知します。(57 ページの『RULES』)
  - PH35643: SOURCE: SOURCE オプションに新しいサブオプション DEC | HEX が追加されました。このサブオプションは、ソースのリストのシーケンス番号を 10 進形式と 16 進形式のどちらで生成するかを制御します。(59 ページの『SOURCE』)
  - PH35652: OFFSET: OFFSET オプションの動作が変更されました。 COBOL コードの 1 行に対して複数の命令ブロックがある場合は、それらの命令に対して OFFSET テーブル内に複数の項目が生成されます。(51 ページの『OFFSET』)
- 以下のコンパイラー・オプションは非推奨です。
  - PH37328: INVDATA: このコンパイラー・オプションは非推奨ですが、使用が許容されており、同等の形式の新しい INVDATA コンパイラー・オプションに自動的にマップされます。(75 ページの『ZONEDATA』)

IGYCDOPT デフォルト・オプションの処理の動作の変更

- PH37331: COBOL カスタマイズ・マクロ内で誤ってコーディングされているオプションや OPTION= ではなく OPTION() としてコーディングされているオプションの診断のためのサポートが追加されました。 (79 ページの『コンパイラー・オプションのデフォルトの変更』)

バージョン 6 リリース 3

コンパイラー・オプションの変更

- 新しいコンパイラー・オプションは以下のとおりです。
  - LP: 新規 LP コンパイラー・オプションを使用すれば、関連言語機能を有効にした状態で生成するのが AMODE 31 (31 ビット) プログラムなのか AMODE 64 (64 ビット) プログラムなのかを指示できます。 LP(32) がデフォルトです。 (41 ページの『LP』)
- 変更されたコンパイラー・オプション:
  - ARCH: ARCH(7) は現在受け入れられません。 新しい上位レベルの ARCH(13) が受け入れられます。 ARCH(8) がデフォルトです。 (17 ページの『ARCH』)
  - NUMCHECK: コンパイル時に無効なデータが見つかった場合、NUMCHECK(MSG) と NUMCHECK(ABD) のどちらが指定されていても、エラー・レベル・メッセージが生成され、検査は除去されます。 (45 ページの『NUMCHECK』)

共有ストレージ変更におけるコンパイラー・フェーズ

- 共有ストレージにコンパイラー・フェーズを配置するためのインストール・カスタマイズが除去されました。

第 1 章 Enterprise COBOL のカスタマイズの計画

Enterprise COBOL のカスタマイズを計画する ときには、コンパイラー・オプションのデフォルト値を変更するかどうか、 追加の予約語テーブルを作成するかどうか を検討する必要があります。

カスタマイズを計画する際、以下の説明が役立ちます。

- 1 ページの『インストール後の変更: なぜカスタマイズが必要か』
- 1 ページの『コンパイラー・オプションのデフォルト値を変更する計画』
- 7 ページの『追加の予約語テーブルを作成する計画』
- 9 ページの『製品登録を使用して Enterprise COBOL を使用可能/使用不可にする』

IBM Debug for z/OS (以前は IBM Debug for z Systems® および IBM Debug Tool for z/OS) をインストールしようとしている場合は、そのモジュールを共有ストレージに置くかどうか、また、デバッガーと連携して機能するように CICS® 環境をセットアップするかどうかを決めることができます。

実際のカスタマイズ手順については、79 ページの『第 3 章 Enterprise COBOL のカスタマイズ』を参照してください。

本書には、マクロ内の IBM 提供のデフォルト値の変更を計画する際に役立つワークシートも含まれています。計画シートについては、xii ページの『マクロ計画ワークシートの使用』を参照してください。

重要 : Enterprise COBOL のカスタマイズを計画する際には、この製品を使用するアプリケーション・プログラマーと相談してください。 これにより、カスタマイズ によって適用される変更が、アプリケーション・プログラマーの要件を満たすと同時に、 開発されるアプリケーションを サポートするようになります。

インストール後の変更: なぜカスタマイズが必要か

Enterprise COBOL のインストール時には、コンパイラー・オプション、および予約語テーブルに関して IBM 提供のデフォルトを受け取ります。アプリケーション・プログラマーのニーズにより合うように、Enterprise COBOL をカスタマイズしたい場合があります。

Enterprise COBOL をインストールした後、以下を行うことができます。

- コンパイラー・オプションのデフォルト値を変更する: 1 ページの『コンパイラー・オプションのデフォルト値を変更する計画』を参照してください。
- コンパイラー・オプションを固定する: 2 ページの『コンパイラー・オプションの固定』を参照してください。
- 追加の予約語テーブルを作成する: 7 ページの『追加の予約語テーブルを作成する計画』を参照してください。

コンパイラー・オプションのデフォルト値を変更する計画

コンパイラー・オプションのデフォルト値 IGYCDOPT プログラムで設定されます。

IGYCDOPT は、インストール時に AMODE 31 および RMODE ANY でリンク・エディットされます。

IGYCDOPT プログラムで設定されるコンパイラー・オプションのデフォルト値については、4 ページの表 1 を参照してください。

IGYCDOPT のような COBOL カスタマイズ・パーツをアセンブルするときには、システム MACLIB にアクセスする必要があります。通常、MACLIB は SYS1.MACLIB にあります。COBOL MACLIB IGY.V6R3M0.SIGYMAC へのアクセスも必要です。

IGYCDOPT プログラムにより、コンパイラー・オプションのデフォルトを選択して固定することができます。 Enterprise COBOL のインストール時の IBM 提供のコンパイラー・オプション値をそのまま受け入れることも、お客様のシステムを担当するプログラマーの要件により適合するように値を変更することもでき

ます。また、アプリケーション・プログラマーがこれらのオプションを オーバーライドできるようにするかどうかを選択することもできます。

注 : 高位修飾子 IGY.V6R3M0 は、 Enterprise COBOL がインストールされたときに変更されている場合があります。

コンパイラー・オプションの固定

以下のセクションでは、コンパイラー・オプションを固定する理由、コンパイラー・オプションを固定する方法、および固定されたオプションを迂回する方法を説明します。

Enterprise COBOL では、お客様に固有のプログラミング標準をセットアップ できます。例えば、多くのお客様では RENT を優先コンパイラー・オプションとして 設定し、強制的に使用する必要がある場合があります。

Enterprise COBOL では、IGYCDOPT プログラムを使用して オプションを固定し、コンパイル時にそのオプションが変更またはオーバーライドされないように指定します。これにより、コンパイル時に、固定されたオプションをオーバーライドしようと すると、ゼロ以外のコンパイラー戻りコードで診断メッセージが出されます。

一貫して使用するために特定のオプションを固定した場合、特別な条件のために、固定されたオプションをう回する必要が 生じることがあります。このような変更を行うには、別のパラメーターを 指定して、IGYCDOPT プログラムの一時コピーをアセンブルします。プログラマーはコンパイル時に、必要な IGYCDOPT モジュールを含む JOBLIB または STEPLIB を使用して、固定されたオプションを迂回することができます。

例えば、0PT=1 オプションを固定する (つまり、COBOL コンパイラーで常に最適化されたオブジェクト・コードを生成する) ことを選択し、あるアプリケーションをこの要件から除外する必要がある場合は、このオプションからアスタリスク・パラメーターを除去後に IGYCDOPT プログラムを再アセンブルしなければなりません。 次に、 アセンブルした IGYCDOPT モジュールを一時ライブラリーに入れ、コンパイル時に JOBLIB または STEPLIB としてアクセスされるようにします。

サンプル・インストール・ジョブ

Enterprise COBOL には 2 つのサンプル・インストール・ジョブ があり、それらを修正して使用すれば、コンパイラー・オプションのデフォルト値を変更することができます。 サンプル・ジョブ IGYWDOPT は、コンパイラーの IBM 提供デフォルトを変更する例です。 もう 1 つのサンプル・ジョブ IGYWUOPT は、固定されているコンパイラー・オプションをオーバーライドする例です。 これらのジョブは COBOL サンプル・データ・セット IGY.V6R3M0.SIGYSAMP に入っています。

IGYWDOPT

このサンプル・インストール・ジョブを使用すれば、SMP/E を使用して IBM 提供のデフォルトを変更することができます。

初めて IGYWDOPT を実行する場合、以下の手順に従ってください。

1. システム要件に合わせてジョブ・カードを変更します。
2. 以下の項目を変更します。
   - #globalcsi (これをインストール場所の CSI 名にする)
   - #tzone (これをインストール場所の TARGET ZONE 名にする)
3. メンバー SIGYSAMP(IGYCDOPT) を SIGYSAMP(IGYWDOPT) にコピーして、ステップ DOPT の ++ SRC ステートメントに続くコメント行を置き換えます。
4. コピーした IGYCDOPT テキストを変更し、オーバーライドする必要があるコンパイラー・オプションのリストが入るようにします。 例えば、次のように指定します。

```
COPY
IGYCDOPT

           IGYCDOPT
CSECT

           IGYCDOPT AMODE
```

```
ANY
          IGYCDOPT RMODE
ANY
                   IGYCOPT
ARCH=10,                                                              X

OPTIMIZE=*2,                                                          X

NUMCHECK=(ZON,PAC,BIN,MSG),                                           X

INVDATA=NO
                   END   IGYCDOPT
```

必要に応じて、継続文字「X」を桁 72 に使用してください。

5. IGYWDOPT を実行し、usermod を受信および適用し、カスタマイズ済みバージョンの MOD(IGYCDOPT) を作成します。
6. **重要:** 今後の参照のために、変更した SIGYSAMP(IGYWDOPT) のコピーを保存しておいてください。

   注意: usermod に対して ACCEPT を実行しないでください。 usermod を受け入れると、必要なときに、SMP/E でその usermod に対して RESTORE を実行することができなくなります。

MOD(IGYCDOPT) が IBM PTF によって変更され、SIGYSAMP(IGYWDOPT) ジョブの再実行を必要とする場合、以下の手順に従ってください。

1. IGYWDOPT ジョブによって作成された usermod に対して RESTORE を実行します。 これは、SMP/E コマンド RESTORE SELECT (IGYWDOP) で行います。 これで、MOD(IGYCDOPT) が以前の IBM PTF レベルに復元され、新しい IBM PTF を正しく適用できるようになります。
2. IBM PTF を適用します。
3. ++ USERMOD ステートメントの REWORK パラメーターを変更することによって、修正日を、変更が加えられる日付に変更します。
4. 適切な「PRE( )」ステートメントを「FMID( )」ステートメントの後に追加します。 これは一般に、ここで適用された PTF 番号です。 エラーが発生するときに、どの PRE ステートメントを追加するかを調べるには、技術情報を参照してください。
5. SIGYSAMP(IGYWDOPT) バックアップ・メンバーを参照用として使用し、追加または変更されている可能性があるオプションを SIGYMAC(IGYCOPT) で指し示すことによって、IGYWDOPT を更新し、オーバーライドする必要があるコンパイラー・オプションのリストが入るようにします。
6. IGYWDOPT を再実行し、usermod を受信および適用し、カスタマイズ済みバージョンの MOD(IGYCDOPT) を再作成します。
7. **重要:** 今後の参照のために、変更した SIGYSAMP(IGYWDOPT) のコピーを保存しておいてください。

IGYWDOPT は、条件コード 0 で実行されるはずです。

ASSEMBLER SYSPRINT データ・セットにある IGYNNNN 通知メッセージで、新しい IGYCDOPT モジュールが使用されるときに有効になるオプションを確認します。

IGYWUOPT
IGYCDOPT プログラムによって固定されたコンパイラー・オプションをオーバーライドする必要が生じた場合、このサンプル・インストール・ジョブを使用すれば、SMP/E の外側にモジュールを作成し、そのモジュール内で異なるデフォルトを 指定することができます。

コンパイラー・オプションの変更

コンパイラー・オプションの値を変更する場合は、IGYCOPT の構文形式を使用してください。

IBM 提供のデフォルト値は、計画ワークシートに示されているほか、各構文図のすぐ後にも示されています。構文図には、xi ページの『構文図の読み方』で説明しているようにデフォルトも示しています。

コンパイラー・オプションとそのデフォルトについては、以下に説明があります。 これらのオプション、およびデフォルト値をよく検討して、アプリケーションに最も適する値を決定してください。

IGYCOPT の形式

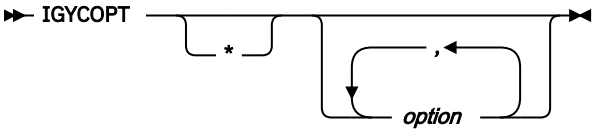


図 *1. IGYCOPT* コンパイラー・オプション・マクロの構文形式

コンパイラー・オプション用の IGYCDOPT ワークシート

IGYCDOPT ワークシートは、IGYCDOPT プログラムのコンパイラー・オプション部分を 計画し、コーディングする際に役立ちます。

「固定の場合 * を記入」欄と「選択を記入」欄に記入して、このワークシートを完成させてください。

注 :

- コンパイラー・オプションのデフォルト値を変更するときに アスタリスク [*] をコーディングすると、このオプションは固定され、 アプリケーション・プログラマーがこのオプションをオーバーライド できなくなります。
- ALOWCBL、DBCSXREF、および NUMCLS オプションは、コンパイル時にオーバーライドすることはできません。 したがって、ワークシートでは、これらのオプションについては、「固定の 場合 * を記入」欄はブランクになります。
- ADEXIT、INEXIT、LIBEXIT、MSGEXIT、および PRTEXIT の IBM 提供のデフォルト値はヌルです。したがって、これらのオプションについては、「IBM 提供のデフォルト」欄はブランクになっています。
- DUMP コンパイラー・オプションは、IGYCDOPT プログラムでは 設定できません。コンパイル時に変更しない限り、DUMP は必ず NODUMP に設定されます。
- OPTFILE コンパイラー・オプションは、IGYCDOPT プログラムでは 設定できません。

表 *1.* コンパイラー・オプション用の *IGYCDOPT* ワークシート

コンパイラー オプション	固定の 場合 * 固定	選択を記入	IBM 提供の デフォルト	構文 説明
ADATA=	____	__________	NO	13 ページの『ADATA』
ADEXIT=	____	__________		14 ページの『ADEXIT』
ADV=	____	__________	YES	14 ページの『ADV』
AFP=	____	__________	NOVOLATILE	15 ページの『AFP』
ALOWCBL=	____	__________	YES	15 ページの『ALOWCBL』
ALOWCOPYLOC=	____	__________	YES	16 ページの『ALOWCOPYLOC』
ALOWDEFINE=	____	__________	YES	16 ページの『ALOWDEFINE 』
ARCH=	____	__________	8	17 ページの『ARCH』
ARITH=	____	__________	COMPAT	18 ページの『ARITH』
AWO=	____	__________	NO	19 ページの『AWO』
BLOCK0=	____	__________	NO	19 ページの『BLOCK0』
BUF=	____	__________	4K	20 ページの『BUF』
CICS=	____	__________	NO	20 ページの『CICS』
CODEPAGE=	____	__________	1140	21 ページの『CODEPAGE』

表 1. コンパイラー・オプション用の *IGYCDOPT* ワークシート (続き)

コンパイラー・オプション	固定の場合 * 固定	選択を記入	IBM 提供のデフォルト	構文説明
COMPILE=	____	_____	NOC(S)	21 ページの『COMPILE』
COPYRIGHT=	____	_____	NO	22 ページの『COPYRIGHT』
CURRENCY=	____	_____	NO	22 ページの『CURRENCY』
DATA=	____	_____	31	23 ページの『DATA』
DBCS=	____	_____	Yes	24 ページの『DBCS』
DBCSXREF=	____	_____	NO	24 ページの『DBCSXREF』
DECK=	____	_____	NO	25 ページの『DECK』
DIAGTRUNC=	____	_____	NO	26 ページの『DIAGTRUNC』
DISPSIGN=	____	_____	COMPAT	26 ページの『DISPSIGN』
DLL=	____	_____	NO	27 ページの『DLL』
DYNAM=	____	_____	NO	28 ページの『DYNAM』
EXPORTALL=	____	_____	NO	28 ページの『EXPORTALL』
FASTSRT=	____	_____	NO	29 ページの『FASTSRT』
FLAG=	____	_____	(I,I)	29 ページの『FLAG』
FLAGSTD=	____	_____	NO	30 ページの『FLAGSTD』
HGPR=	____	_____	PRESERVE	32 ページの『HGPR』
INEXIT=	____	_____		32 ページの『INEXIT』
INITCHECK=	____	_____	NO	33 ページの『INITCHECK』
INITIAL=	____	_____	NO	34 ページの『INITIAL』
INLINE=	____	_____	YES	34 ページの『INLINE』
INTDATE=	____	_____	ANSI	35 ページの『INTDATE』
INVDATA=	____	_____	NO	36 ページの『INVDATA』
LANGUAGE=	____	_____	EN	38 ページの『LANGUAGE』
LIBEXIT=	____	_____		39 ページの『LIBEXIT』
LINECNT=	____	_____	60	40 ページの『LINECNT』
LIST=	____	_____	NO	40 ページの『LIST』
LITCHAR=	____	_____	QUOTE	41 ページの『LITCHAR』
LP=	____	_____	32	LP
MAP=	____	_____	NO	42 ページの『MAP』
MAXPCF=	____	_____	100000	42 ページの『MAXPCF』
MDECK=	____	_____	NO	43 ページの『MDECK』
MSGEXIT=	____	_____		44 ページの『MSGEXIT』
NAME=	____	_____	NO	44 ページの『NAME』
NSYMBOL=	____	_____	NATIONAL	45 ページの『NSYMBOL』

表 1. コンパイラー・オプション用の *IGYCDOPT* ワークシート (続き)

コンパイラー オプション	固定の 場合 * 固定	選択を記入	IBM 提供の デフォルト	構文 説明
NUM=	____	_____	NO	45 ページの『NUM』
NUMCHECK=	____	_____	(NO)	45 ページの『NUMCHECK』
NUMCLS=	____	_____	PRIM	49 ページの『NUMCLS』
NUMPROC=	____	_____	NOPFD	50 ページの『NUMPROC』
OBJECT=	____	_____	YES	51 ページの『OBJECT』
OFFSET=	____	_____	NO	51 ページの『OFFSET』
OPTIMIZE=	____	_____	0	52 ページの『OPTIMIZE』
OUTDD=	____	_____	SYSOUT	52 ページの『OUTDD』
PARMCHECK=	____	_____	(NO)	53 ページの『PARMCHECK』
PGMNAME=	____	_____	COMPAT	53 ページの『PGMNAME』
PRTEXIT=	____	_____		54 ページの『PRTEXIT』
QUALIFY=	____	_____	COMPAT	54 ページの『QUALIFY』
RENT=	____	_____	YES	55 ページの『RENT』
RMODE=	____	_____	AUTO	56 ページの『RMODE』
RULES=	____	_____	(NO)	57 ページの『RULES』
SEQ=	____	_____	YES	59 ページの『SEQ』
SERVICE=	____	_____	NO	59 ページの『SERVICE』
SOURCE=	____	_____	YES	59 ページの『SOURCE』
SPACE=	____	_____	1	60 ページの『SPACE』
SQL=	____	_____	NO	60 ページの『SQL』
SQLCCSID=	____	_____	YES	61 ページの『SQLCCSID』
SQLIMS=	____	_____	NO	61 ページの『SQLIMS』
SSRANGE=	____	_____	NO	62 ページの『SSRANGE』
STGOPT=	____	_____	NO	63 ページの『STGOPT』
SUPPRESS=	____	_____	YES	64 ページの『SUPPRESS』
TERM=	____	_____	NO	64 ページの『TERM』
TEST=	____	_____	(NO, NODWARF)	64 ページの『TEST』
THREAD=	____	_____	NO	67 ページの『THREAD』
TRUNC=	____	_____	STD	68 ページの『TRUNC』

表 1. コンパイラー・オプション用の *IGYCDOPT* ワークシート (続き)

コンパイラー・オプション	固定の場合 * 固定	選択を記入	IBM 提供のデフォルト	構文説明
TUNE=	____	__________	ARCH が指定されていない場合は 8。ARCH を指定する場合は、デフォルトの TUNE レベルと ARCH レベルが一致している必要があります。	69 ページの『TUNE』
VBREF=	____	__________	NO	70 ページの『VBREF』
VLR=	____	__________	STANDARD	70 ページの『VLR』
VSAMOPENFS=	____	__________	COMPAT	72 ページの『VSAMOPENFS』
WORD=	____	__________	NO	72 ページの『WORD』
XMLPARSE=	____	__________	XMLSS	73 ページの『XMLPARSE』
XREFOPT=	____	__________	FULL	74 ページの『XREFOPT』
ZWB=	____	__________	YES	76 ページの『ZWB』

追加の予約語テーブルを作成する計画

以下のセクションでは、追加の予約語テーブルを作成する理由と、予約語テーブルを変更することで、ネストされたプログラムの使用を制限する方法について説明します。 また、Enterprise COBOL で提供される予約語テーブルを掲載しています。

インストール後に、追加の予約語テーブルを作成することができます。コンパイル時に、WORD コンパイラー・オプションの値で、どの予約語テーブルを 使用するかを決定します。

追加の予約語テーブルを作成する目的

このセクションでは、追加の予約語テーブルを作成する利点について説明します。

以下の目的のために、追加の予約語テーブルを作成することができます。

- 予約語を別の言語 (フランス語またはドイツ語など) に変換する。
- アプリケーション・プログラマーが Enterprise COBOL の特定の命令 (GO TO など) を 使用するのを防ぐ。
- ネストされたプログラムの使用を制御する。
- CICS でサポートされていない語 (READ、WRITE など) にフラグを立てる。

ネストされたプログラムの使用の制御

他の COBOL 言語機能を制限せずに、ネストされたプログラムの 使用を制限するには、予約語テーブルを変更します。

これは、INFO および RSTR 制御ステートメントを使用して 行います。これらの変更を加える方法については、82 ページの『予約語テーブルの作成または変更』を参照してください。

Enterprise COBOL と一緒に提供される予約語テーブル

Enterprise COBOL では、インストール・メディアで予約語テーブルが提供されます。

提供される予約語テーブルは以下のとおりです。

- デフォルト予約語テーブル
- CICS 予約語テーブル

デフォルト予約語テーブル (IGYCRWT)

機能全体に対して提供されるデフォルト予約語テーブルについては、「*Enterprise COBOL* 言語解説書」で『予約語』を参照してください。

CICS 予約語テーブル (IGYCCICS)

Enterprise COBOL では、CICS アプリケーション・プログラム用の代替予約語テーブルが提供されるため、CICS ではサポートされていない COBOL 語にはコンパイラーによってフラグが立てられます。

CICS 予約語テーブルは、以下の COBOL 語が制限付き (RSTR) としてマークされている以外は、デフォルト予約語テーブルと同じです。

- CLOSE
- DELETE[3]
- FACTORY
- FD
- FILE[1]
- FILE-CONTROL[1]
- INPUT-OUTPUT[1]
- INVOKE
- I-O-CONTROL
- MERGE
- METHOD
- OBJECT
- OPEN
- READ
- RERUN
- REWRITE
- SD[1、2]
- SELF
- START
- SUPER
- WRITE

注:

1. CICS のもとで SORT ステートメントを使用したい場合は (COBOL は CICS のもとで SORT ステートメントのインターフェースをサポートします)、制限付きとマークされたワードのリストからワードを除去するよう CICS 予約語テーブルを変更する必要があります。
2. SORT キーワードは制限されませんが、SD キーワードは制限されます。 SD キーワードの場合、フォーマット 2 (テーブル) の SORT ステートメントは使用可能ですが、フォーマット 1 (ファイル) の SORT ステートメントは使用不可です。

3. DELETE キーワードを制限しても、BASIS 処理の DELETE 機能は使用できます。

テーブルの使用

CICS 予約語テーブルを使用するには、WORD(CICS) コンパイラー・オプションを指定してください。

CICS 予約語テーブルをデフォルトとして使用するには、WORD コンパイラー・オプションのデフォルト値を WORD=CICS に 設定してください。

テーブルの場所

CICS 予約語テーブルの作成に使用されるデータは、IGY.V6R3M0.SIGYSAMP 内のメンバー IGY8CICS に入っています。

注 : 高位修飾子 IGY.V6R3M0 は、 Enterprise COBOL がインストールされたときに変更されている場合があります。

製品登録を使用して Enterprise COBOL を使用可能/使用不可にする

Enterprise COBOL V6 はデフォルトの動作では各 z/OS システムで実行されますが、SYSx.PARMLIB の IFAPRDxx メンバーを使用すれば、選択した z/OS システムで COBOL V6 が実行されないようにすることができます。

COBOL V6 を使用不可にしたい場合は、以下のコードをアクティブな IFAPRDxx メンバーに追加します。

```
PRODUCT OWNER('IBM CORP')
        NAME('ENTERPRISE COBOL')
        ID(5655-EC6)
        VERSION(06) RELEASE(*) MOD(*)
        FEATURENAME(*)
        STATE(DISABLED)
```

この場合、コンパイラーは RC=16 で停止し、オペレーター宛メッセージを出力します。

COBOL V6 を明示的に使用可能にしたい場合は、以下のコードをアクティブな IFAPRDxx メンバーに追加します。

```
PRODUCT OWNER('IBM CORP')
        NAME('ENTERPRISE COBOL')
        ID(5655-EC6)
        VERSION(06) RELEASE(*) MOD(*)
        FEATURENAME(*)
        STATE(ENABLED)
```

ただし、COBOL V6 はデフォルトで使用可能になっているため、上記のステートメントをアクティブな IFAPRDxx メンバーに追加して COBOL V6 を明示的に使用可能にする必要はありません。

第 2 章 Enterprise COBOL コンパイラー・オプション

ここでは、デフォルト値を変更できるコンパイラー・オプションについて説明します。

いくつかの説明に付記されている注は、コンパイル時の他のオプションとの 相互影響など、これらのオプションについての追加情報です。

これらの情報は、ご使用のシステムに適切なデフォルト値を決定するために役立ちます。

コンパイラー・オプションについて詳しくは、「*Enterprise COBOL* プログラミング・ガイド」で『コンパイラー・オプション』を参照してください。

重要:

Enterprise COBOL のカスタマイズを計画する際には、この製品を使用するアプリケーション・プログラマーと相談してください。 そうすることにより、 アプリケーション・プログラマーのニーズを満たし、開発されるアプリケーションを サポートするような変更を行うことができます。

COBOL コンパイラー・オプションの指定

IGYCOPT マクロにコンパイラー・オプションを指定するときは、オプション名とその値の両方を大文字で指定する必要があります。

オプション名を大文字で指定しなければ、オプション名とその値の両方が 無視され、デフォルト値が使用されます。エラー・メッセージは出されません。オプション値だけが大文字でない場合は、無効なオプション値が指定されている ことを示すエラー・メッセージが出されます。

矛盾するコンパイラー・オプション

特定のコンパイラー・オプション値を指定すると、他のコンパイラー・オプションとの矛盾が生じる場合があります。このトピックでは、コンパイラー・オプション間に生じる可能性がある矛盾について説明します。

表 2. 矛盾するコンパイラー・オプション

コンパイラー・オプション	矛盾するオプション
AFP=VOLATILE	LP=64
CICS=YES	DYNAM=YES LP=64 RENT=NO
DATA=24	LP=64
DBCS=NO	NSYMBOL=NATIONAL
DBCSXREF=(NO 以外)	XREFOPT=NO
DLL=NO	EXPORTALL=YES
DLL=YES	DYNAM=YES RENT=NO

表 2. 矛盾するコンパイラー・オプション (続き)

コンパイラー・オプション	矛盾するオプション
DYNAM=YES	CICS=YES DLL=YES EXPORTALL=YES
EXPORTALL=YES	DLL=NO DYNAM=YES RENT=NO
FLAGSTD=(NO 以外)	WORD=*xxxx*
HGPR=NOPRESERVE	LP=64
LIST=YES	OFFSET=YES
LP=64	AFP=VOLATILE CICS=YES DATA=24 HGPR=NOPRESERVE RENT=NO RMODE=24 SQLIMS=YES THREAD=YES
NSYMBOL=NATIONAL	DBCS=NO
OBJECT=NO	TEST=(NO 以外)
OFFSET=YES	LIST=YES
RENT=NO	CICS=YES DLL=YES EXPORTALL=YES LP=64 THREAD=YES
RMODE=24	LP=64
SQLIMS=YES	LP=64
THREAD=YES	INITIAL=YES LP=64 RENT=NO
WORD=*xxxx*	FLAGSTD=(NO 以外)
XREFOPT=NO	DBCSXREF=(NO 以外)

標準準拠のためのコンパイラー・オプション

85 COBOL 標準に準拠するには、いくつかのコンパイラー・オプションが必要です。

詳しくは、「*Enterprise COBOL* プログラミング・ガイド」で『*85 COBOL* 標準に準拠するオプション設定』を参照してください。

コンパイラー・オプションの構文および説明

以下のトピックにある構文図は、変更可能な各コンパイラー・オプション を説明しています。各構文図の下にあるテキストは、特定のパラメーターを選択した場合の結果を説明しています。

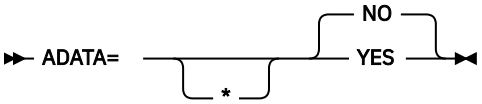
注 :

- DUMP オプションはこのリストに含まれていません。コンパイル時に DUMP を変更 しない限り、常に NODUMP に設定されます。このオプションは、通常は使用しません。IBM 担当員から要求されたときにのみ使用してください。
- OPTFILE オプションはこのリストに含まれていません。これはコンパイラー 呼び出し PARM オプションとして、または COBOL ソース・プログラム内の PROCESS または CBL ステートメントでのみ指定できます。
- コンパイラー・オプションのデフォルト値を変更するときにアスタリスク (*) を コーディングすると、このオプションは固定され、 アプリケーション・プログラマーがこのオプションをオーバーライド できなくなります。

ADATA

ADATA は、コンパイル時に関連データ・ファイルを作成するかどうかに作用します。

構文



デフォルト
ADATA=NO

YES
適切なレコードを使用して関連データ・ファイルを作成します。

NO
関連データ・ファイルを作成しません。

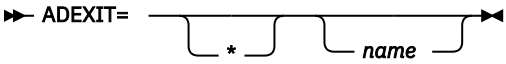
注 :

- ADATA オプションの指定は、呼び出し時にオプション・リストを介してか、JCL の PARM フィールド上か、コマンド・オプションとしてか、またはインストール・システム・デフォルトとしてか、のいずれかの場合のみ可能です。
- 日本語オプションを選択すると、関連データ・ファイル内の レコードに DBCS 文字が書き込まれることがあります。
- NOCOMPILE(W|E|S) を指定した場合、コンパイルが途中で停止し、特定の関連データ・レコードが消失することがあります。
- INEXIT オプションを使用 すると、コンパイル・ソース・モジュールは SYSADATA (関連データ・ファイル) 情報内で識別されません。

ADEXIT

ADEXIT は、SYSADATA ファイルに書き込まれるレコードごとに呼び出すモジュールを指定します。

構文



デフォルト

出口は指定されません。EXIT コンパイラー・オプションの NOADEXIT サブオプションを 指定するのと同等です。 ADEXIT=* が *name* パラメーターなしでコーディングされた場合、NOADEXIT をオーバーライド することはできません。

name

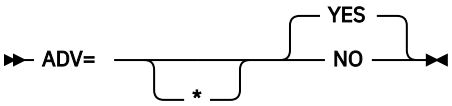
EXIT コンパイラー・オプションで使用するモジュールを識別します。このユーザー出口の サブオプションが指定されている場合、コンパイラーは指定されたモジュールを ロードし、SYSADATA ファイルに書き込まれるレコードごとにそのモジュールを呼び出します。

EXIT コンパイラー・オプションについて詳しくは、「*Enterprise COBOL* プログラミング・ガイド」で『*EXIT* コンパイラー・オプション』を参照してください。

ADV

ADV は、WRITE ... ADVANCING ステートメントに作用し、プリンター制御文字のレコード長に 1 バイト追加するかどうかを決定します。

構文



デフォルト

ADV=YES

YES

印刷制御文字用の 1 バイトをレコード長に追加します。このオプションは、ソース・ファイル内で WRITE ... ADVANCING を使用する プログラマーに有用です。レコードの先頭文字をプログラマーが明示的に予約する 必要はありません。

NO

レコード長を WRITE ... ADVANCING 用に調整しません。コンパイラーは、指定されたレコード域の先頭文字を用いて印刷制御文字を 置きます。アプリケーション・プログラマーは、レコード記述にこの追加のバイトが見込まれて いることを確認する必要があります。

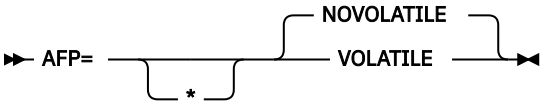
注 :

- ADV=YES を指定した場合、物理装置上のレコード長は、ソース・プログラムにおける レコード記述長よりも 1 バイト大きくなります。
- 出力ファイルのレコード長がソース・コードで定義されていない場合は、COBOL は、DCB パラメーターが適切に設定されていることを確認します。
- ADV=YES が指定され、かつ出力ファイルのレコード長がソース・コードで 定義された場合は、プログラマーは、レコード記述長をソース・プログラムの レコード記述よりも 1 バイト大きいものとして指定する必要があります。さらに、プログラマーは、1 バイト大きいレコード・サイズの正確な倍数で ブロック・サイズを指定する必要があります。
- ファイル記述 (FD) に LINAGE 節が指定されると、コンパイラーは、ADV=YES が指定されているものとしてそのファイルを処理します。

AFP

AFP オプションは、z/Architecture プロセッサーに備わっている追加浮動小数点 (AFP) レジスターのコンパイラーによる使用を制御します。

構文



Enterprise COBOL コンパイラーは、z/Architecture プロセッサーに備わっている 16 個の浮動小数点レジスター (FPR) をすべて使用するコードを生成します。 これらの FPR には以下があります。

- オリジナル FPR。 0、2、4、および 6 の番号が付けられています。
- AFP レジスター。 1、3、5、7、および 8 から 15 の番号が付けられています。

デフォルト
AFP=NOVOLATILE

VOLATILE
AFP=VOLATILE を指定した場合、8 から 15 までの AFP レジスターは揮発性のレジスターであるとみなされます。 つまり、呼び出されたサブプログラムがこれらのレジスターを変更する可能性があります。 このため、COBOL コンパイラーはこれらのレジスターの値を保護するために、追加コードを生成します。

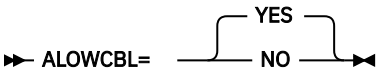
NOVOLATILE
AFP=NOVOLATILE を指定した場合、8 から 15 までの AFP レジスターは不揮発性のレジスターであるとみなされます。 つまり、これらのレジスターは、呼び出された各サブプログラムによって変更されないもの、または保持されるものとみなされます。 このため、コンパイラーは浮動小数点演算が含まれるプログラムに対して、より効率的なコード・シーケンスを生成できます。 これは通常の z/OS アーキテクチャーの規則です。

64 ビットの考慮事項: LP=64 コンパイラー・オプションが有効な場合、AFP=VOLATILE オプションはサポートされません。 このオプションがユーザーによって明示的に指定されている場合は、通知メッセージが出されてその設定は無視されます。

ALOWCBL

ALOWCBL は、COBOL プログラムで PROCESS (または CBL) ステートメントを使用できるかどうかに作用します。

構文



デフォルト
ALOWCBL=YES

YES
COBOL プログラムでの PROCESS (または CBL) ステートメントの 使用を許可します。

NO
プログラムで PROCESS (または CBL) ステートメントが 使用された場合は、エラーと診断します。

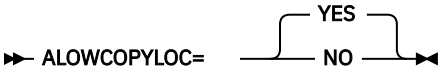
注 :

- ALOWCBL は、コンパイル時にオーバーライドすることはできません。ALOWCBL を PROCESS (または CBL) ステートメントに組み込むことはできないからです。
- PROCESS (または CBL) ステートメントは、ソース・プログラム内で コンパイラー・オプション・パラメーターを指定するために使用されます。インストール要件により、ソース・プログラムでコンパイラー・オプションを指定する ことを許可しない場合は、ALOWCBL=NO を指定してください。

ALOWCOPYLOC

ALOWCOPYLOC は、COBOL プログラムのコンパイル時に COPYLOC オプションを使用できるかどうかに影響します。

構文



デフォルト
ALOWCOPYLOC=YES

YES
COBOL プログラムのコンパイル時に COPYLOC オプションを指定できます。

NO
COPYLOC オプションの指定がエラーとして診断されます。

注 :

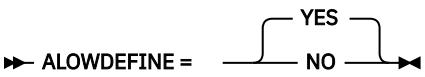
- ALOWCOPYLOC は、コンパイル時にオーバーライドすることはできません。ALOWCOPYLOC を PROCESS (または CBL) ステートメントに組み込むことはできないからです。
- インストール要件により、ソース・プログラムで COPYLOC コンパイラー・オプションを指定する ことを許可しない場合は、ALOWCOPYLOC=NO を指定してください。

関連参照
COPYLOC (*Enterprise COBOL* プログラミング・ガイド)

ALOWDEFINE

ALOWDEFINE は、COBOL プログラムのコンパイル時に DEFINE オプションを使用できるかどうかに影響します。

構文



デフォルト
ALOWDEFINE=YES

YES
COBOL プログラムのコンパイル時に DEFINE オプションを指定できます。

NO
DEFINE オプションの指定がエラーとして診断されます。

注 :

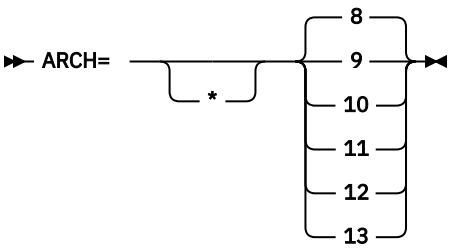
- ALOWDEFINE は、コンパイル時にオーバーライドすることはできません。ALOWDEFINE を PROCESS (または CBL) ステートメントに組み込むことはできないからです。
- インストール要件により、ソース・プログラムで DEFINE コンパイラー・オプションを指定する ことを許可しない場合は、ALOWDEFINE=NO を指定してください。

関連参照
DEFINE (*Enterprise COBOL* プログラミング・ガイド)

ARCH

ARCH オプションは、実行可能プログラム命令の生成対象となるマシン・アーキテクチャーを指定します。

構文



より高い ARCH レベルを指定すると、コンパイラーは、新しいより高速な命令を使用するコードを生成します。 そのようなコードからなるアプリケーションは、ARCH オプションで指定されたアーキテクチャー・レベルより低いアーキテクチャー・レベルの プロセッサー上で実行されると異常終了する可能性があります。 アプリケーションを実行する最も低いレベルのマシン・アーキテクチャーに合致する ARCH レベルを使用してください。

現在サポートされているアーキテクチャー・レベルおよびモデルのグループは以下のとおりです。

デフォルト

ARCH=8

8

z/Architecture モードの 2097-xxx (IBM System z10® EC) モデルおよび 2098-xxx (IBM System z10 BC) モデルで有効な命令を使用するコードを生成します。

具体的には、このような ARCH=8 マシンとその後継マシンは、汎用命令拡張機能でサポートされる命令を追加します。

9

z/Architecture モードの 2817-xxx (IBM zEnterprise® 196) モデルおよび 2818-xxx (IBM zEnterprise 114) モデルで使用可能な命令を使用するコードを生成します。

具体的には、このような ARCH=9 マシンとその後継マシンは、以下の機能でサポートされる命令を追加します。

- 上位ワード機能
- インターロック・アクセス機能
- 条件付きロード/ストア機能
- 独立オペランド機能
- ポピュレーション・カウント機能

10

z/Architecture モードの 2827-xxx (IBM zEnterprise EC12) モデルおよび 2828-xxx (IBM zEnterprise BC12) モデルで有効な命令を使用するコードを生成します。

具体的には、このような ARCH=10 マシンとその後継マシンは、以下の機能でサポートされる命令を追加します。

- 実行ヒント機能
- ロード・アンド・トラップ機能
- 各種命令拡張機能
- トランザクション実行機能
- ゾーン 10 進数データ項目と 10 進浮動小数点データ項目間のより効率的な変換を可能にする、拡張 10 進浮動小数点機能。 算術計算を実行するためにゾーン 10 進数データ項目をパック 10 進数データ項目に変換する代わりに (これが許可される条件であり、最適化レベルが 0 より大きい場合) 、コン

パイラーはゾーン 10 進数データ項目を 10 進浮動小数点データ項目に直接変換し、計算の完了後に再度ゾーン 10 進数データ項目に戻します。

11
z/Architecture モードの 2964-xxx (IBM z13®) モデルおよび 2965-xxx (IBM z13s®®) モデルで有効な命令を使用するコードを生成します。

具体的には、これらの ARCH=11 マシンおよびその後継マシンには、以下の機能のサポートを使用する命令が追加されています。

- パック 10 進数データ項目と 10 進浮動小数点中間結果データ項目間のより効率的な変換を可能にする、拡張 10 進浮動小数点機能 (周囲条件が最適で、最適化レベルが 0 より大きい場合)。
- いくつかの `INSPECT REPLACING` ステートメントおよび `INSPECT TALLYING` ステートメントのための、ベクトル拡張機能 (SIMD) 命令の利用。

  ベクトル拡張機能 (SIMD) 命令を使用するには、APAR OA43803 および PI12412 の PTF がインストールされている z/OS V2.1、または z/OS V2.2 で稼働するマシンにおいてコードを実行する必要があります。

12
z/Architecture モードの 3906-xxx (IBM z14) モデルおよび 3907-xxx (IBM z14® ZR1) モデルで使用できる命令が使用されるコードが生成されます。

具体的には、ARCH=12 マシンとその後継マシンでは、ベクトル・パック 10 進数機能をサポートする命令が追加されます。ベクトル・パック 10 進数機能では、中間結果がメモリーではなくベクトル・レジスターに格納されるためパック 10 進数およびゾーン 10 進数の計算が高速になります。

13
z/Architecture モードの 8561-xxx (IBM z15™) モデルで使用できる命令を利用するコードが生成されます。

具体的には、このような ARCH=13 マシンとその後継マシンは、以下の機能でサポートされる命令を追加します。

- ベクトル・パック 10 進数拡張機能
- ベクトル拡張機能 2
- その他の命令拡張機能 3
- 調整済みベクトル・ロード/保管のヒント

注 : より高い ARCH レベルには、低い ARCH レベルの機能が含まれています。 例えば、ARCH=13 には、それよりも低い ARCH レベルの機能がすべて含まれています。

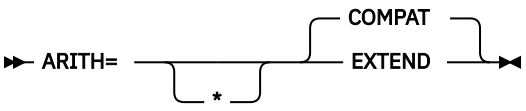
これらの機能について詳しくは、「*z/Architecture* 解説書」を参照してください。

ARCH オプションは、TUNE オプションと一緒に使用します。 ARCH と TUNE の間の相互作用について詳しくは、69 ページの『TUNE』を参照してください。

ARITH

ARITH は、整数にコーディングできる最大桁数、および固定小数点の中間結果で使用される桁数に作用します。

構文



デフォルト
ARITH=COMPAT

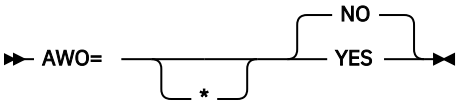
COMPAT
10 進データの最大精度として 18 桁を指定します。

EXTEND
10 進データの最大精度として 31 桁を指定します。

AWO

AWO は、可変長ブロック化フォーマットの物理順次ファイルに対して APPLY-WRITE-ONLY 節をアクティブにするかどうかに作用します。

構文



デフォルト
AWO=NO

YES
APPLY-WRITE-ONLY 節がプログラムで指定されているかどうかに関係なく、プログラム内の可変長ブロック形式のすべての物理順次ファイルについて、APPLY-WRITE-ONLY 節をアクティブにします。

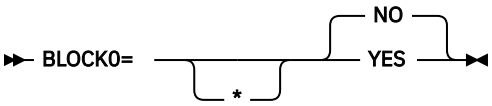
パフォーマンスの考慮: AWO=YES を使用すると、通常は、入出力処理時に実行時ファイルのために データ管理サービスを呼び出す回数が減ります。

NO
APPLY-WRITE-ONLY 節がプログラムで指定されていない場合には、プログラム内の可変長ブロック形式のすべての物理順次ファイルについて、APPLY-WRITE-ONLY 節をアクティブにしません。

BLOCK0

BLOCK0 は、QSAM ファイルのデフォルト・ブロック化指定を非ブロック化からブロック化に変更するかどうかに作用します。

構文



デフォルト
BLOCK0=NO

YES
ファイル記述項目に BLOCK CONTAINS も RECORDING MODE U も指定されていない QSAM ファイルに対するデフォルトのブロック化仕様を変更します。 BLOCK0=YES を指定すると、そのようなファイルに対して BLOCK CONTAINS 0 節がアクティブになり、 その結果として、実行時に、システムが決定するブロック・サイズになります。

パフォーマンスの考慮: BLOCK0=YES を指定すると、処理速度が速くなり、QSAM 出力ファイルのストレージ要件を最小化することができます。 ただし、以下の推奨事項を参照してください。

NO
どのようなファイルについても、デフォルトで BLOCK CONTAINS 0 節をアクティブにしません。

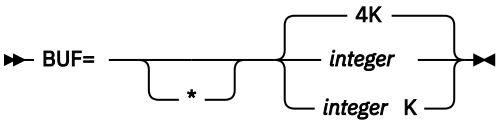
推奨: 既存プログラム内のファイル記述に BLOCK CONTAINS 0 節を追加 すると、INPUT としてオープンされるファイルに対して 望ましくない影響が起こるなど、プログラムの動作が変わる可能性があります。 この理由から、インストールのデフォルトとして BLOCK0=YES を設定しないことをお勧めします。

詳しくは、「*Enterprise COBOL プログラミング・ガイド*」で『*BLOCK0*』を参照してください。

BUF

BUF は、コンパイル時に使用する動的ストレージの量を指定します。

構文



デフォルト
BUF=4K

integer
各コンパイラー作業ファイル・バッファーに割り振られる動的ストレージの量を バイト単位で指定します。最小値は 256 バイトです。

パフォーマンスの考慮: 大容量のバッファーを使用すると、 コンパイラーの性能が向上します。

***integer*K**
バッファーに割り振られる動的ストレージの量を 1K (1024) バイト単位で 指定します。

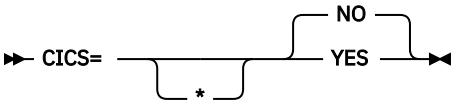
注 :

- BUF は、使用する装置のトラック容量を超えてはならず、また、データ管理サービスで許可される最大量を超えてもなりません。

CICS

CICS は、CICS ステートメントを含む COBOL ソース・プログラムが、組み込みの CICS 変換プログラムによって処理されるかどうかに作用します。

構文



デフォルト
CICS=NO

YES
COBOL ソース・プログラムに CICS ステートメントが含まれているが CICS 変換プログラムによってプリプロセスされていない場合は、YES オプションを指定する必要があります。

NO
NO オプションを指定すると、ソース・プログラムで検出された CICS ステートメントはすべて診断され廃棄されます。

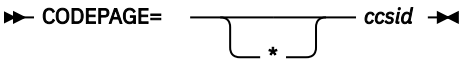
注 :

- CICS コンパイラー・オプションには CICS サブオプションを含めることができます。 CICS サブオプションの区切り文字として、引用符またはアポストロフィを使用することができます。CICS サブオプションを COBOL インストール時のデフォルトとして指定することはできません。
- CICS コンパイラー・オプションは、どのコンパイラー・オプション・ソースでも指定できます。すなわち、インストール時のデフォルト、コンパイラー呼び出し、PROCESS (CBL) ステートメントのいずれかに指定することができます。
- LP(64) コンパイラー・オプションが有効な場合、CICS オプションはサポートされません。 ユーザーがこのオプションを明示的に指定すると、診断メッセージが出力されます。

CODEPAGE

CODEPAGE は、文字エンコードに依存する、コンパイル時および実行時の COBOL 操作を処理するための、EBCDIC コード・ページのコード化文字セット ID (CCSID) に作用します。

構文



デフォルト
CODEPAGE=1140

ccsid
EBCDIC コード・ページを示す有効な文字セット ID (CCSID) 整数を 指定します。

デフォルトの CCSID 1140 は CCSID 37 (EBCDIC Latin-1、USA) と等価ですが、ユーロ記号を含みます。

推奨: COBOL および Db2 を使用するシステムでの必要のない変換および関連したパフォーマンスのオーバーヘッドを防ぐため、Db2 サブシステム・パラメーターとアプリケーション・プログラミングのデフォルト (DSNHDECP で指定) と同じ CODEPAGE コンパイラー・オプション設定を使用してください。

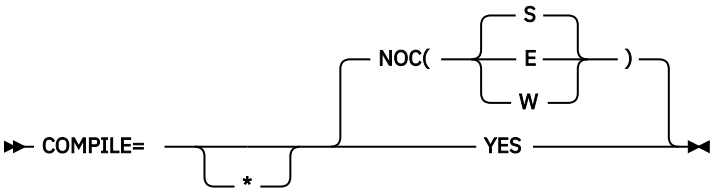
注 : TEST オプションを指定する場合は、CODEPAGE オプションを、COBOL ソース・プログラムで使用する CCSID に設定する必要があります。 特に、DBCS リテラルまたは DBCS ユーザー定義語で日本語文字を使用するプログラムは、CODEPAGE オプションを日本語コード・ページ CCSID に設定してコンパイルする必要があります。

詳しくは、「*Enterprise COBOL プログラミング・ガイド*」で『*CODEPAGE*』を参照してください。

COMPILE

COMPILE は、指定した重大度の診断メッセージが生成された場合にコンパイルを続行するかどうかを決定します。

構文



デフォルト
COMPILE=NOC(S)

NOC
構文検査のみを行うことを示します。

NOC(W)
NOC(E)
NOC(S)
エラー・メッセージ・レベルを指定します。W は警告、E はエラー、S は重大 です。指定したレベルまたはより重大なレベルのエラーが発生すると、コンパイルが 停止し、それ以降はコンパイルのバランスを取る構文検査のみが行われます。

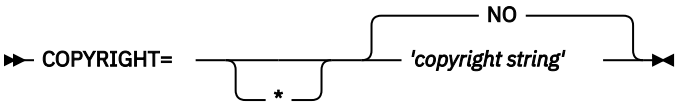
YES
診断およびオブジェクト・コードを含む、完全なコンパイルを行うことを 示します。

NOCOMPILE を指定した場合、コンパイルが途中で停止し、関連データ・ ファイルに影響を与え、特定のメッセージが消失することがあります。

COPYRIGHT

COPYRIGHT は、オブジェクト・モジュールが生成された場合に、オブジェクト・モジュール内にストリングを配置します。 オブジェクトがプログラム・オブジェクトにリンクされている場合、ストリングはこのプログラム・オブジェクトとともにメモリーにロードされます。

構文



デフォルト

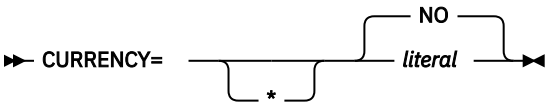
COPYRIGHT=NO

copyright string の長さは 64 文字に制限されています。

CURRENCY

CURRENCY は、代替通貨記号を使用するかどうかを決定します。 デフォルトはドル記号 ($) です。

構文



デフォルト

CURRENCY=NO

リテラル

プログラムで使用するデフォルト通貨記号を表します。

このリテラルは、以下の項目以外の 1 バイト EBCDIC 文字を表す英数字リテラル (場合によっては 16 進数リテラル) でなければなりません。

- 数値 0 から 9
- 大文字の英字: A B C D P R S V X Z
- 小文字の英字 a から z
- スペース
- 特殊文字: * + - / , . ; ( ) = '
- COBOL プログラムが DBCS 項目を PICTURE 記号 G で定義している場合は、大文字の英字 G。記号 G が PICTURE 節で通貨記号と見なされるため、PICTURE 節は DBCS 項目については無効となります。
- COBOL プログラムが DBCS 項目を PICTURE 記号 N で定義している場合は、大文字の英字 N。記号 N が PICTURE 節で通貨記号と見なされるため、PICTURE 節は DBCS 項目については無効となります。
- COBOL プログラムが外部浮動小数点項目を定義している場合は、大文字の英字 E。記号 E が PICTURE 節で通貨記号と見なされるため、PICTURE 節は外部浮動小数点項目については無効となります。

リテラル (16 進数リテラルを含む) の構文規則は、次のとおりです。

- リテラル区切り文字には、APOST オプションと QUOTE オプションのどちらが有効であるかに関係なく、引用符またはアポストロフィのいずれかを使用できます。
- アポストロフィ (') を通貨記号にする場合、組み込みアポストロフィは二重にする必要があります。つまり、リテラル内で 1 つのアポストロフィを表すために、2 つのアポストロフィをコーディングする必要があります。 例えば、次のように指定します。

```
'''' または "'"
```

- 16 進数リテラルの指定形式は、次のとおりです。

```
X'H1H2' または X"H1H2"
```

  H1H2 は、通貨記号リテラルに関する上記の規則に準拠した 1 バイトの EBCDIC 文字を表す、有効な 16 進値です。 16 進数リテラル内の英字は、大文字でなければなりません。

  注 : 16 進値 X'20' および X'21' は使用できません。

NO

CURRENCY オプションによって代替のデフォルト通貨記号を指定しないことを示します。また、コンパイル時に CURRENCY オプションを指定しない限り、ドル記号をプログラムのデフォルト通貨記号として使用することを示します。

値 NO を指定すると、COBOL ソース・プログラムで CURRENCY SIGN 節を省く 場合と同じ結果になります。

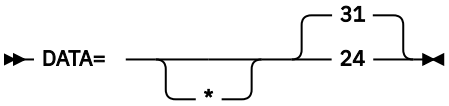
注 :

- COBOL プログラムの PICTURE 節で使用する通貨記号を選択する場合に、CURRENCY SIGN 節 (COBOL ソース・プログラムで指定する) の 代わりに CURRENCY オプションを使用することができます。
- CURRENCY オプションと CURRENCY SIGN 節の両方がプログラム内で使用される 場合、CURRENCY オプションが固定 (*) であっても、CURRENCY SIGN 節に指定されている記号が使用されると、その記号が PICTURE 節内の通貨記号です。

DATA

DATA オプションは、動的データ域のストレージおよび他の動的ランタイム・ストレージを 16 MB 境界の上下どちらから取得するかに作用します。

構文



デフォルト

DATA=31

24

ユーザー・データ域が、LOC=BELOW オプションを指定した GETMAIN によって獲得されたストレージの、16 MB より下の仮想アドレスで割り振られます。

データ・パラメーターを 24 ビット・モードのプログラムに渡すプログラム を RENT オプションでコンパイルする場合、DATA=24 を指定してください。これには、以下の場合が含まれます。

- COBOL プログラムが WORKING-STORAGE 内の項目を AMODE 24 プログラムに渡す場合。
- COBOL プログラムが、参照によって、その呼び出し元から受け取ったデータ項目を AMODE 24 プログラムに渡す場合。受け取られるデータが 16 MB 境界より下にある場合も、DATA=24 を指定する必要があります。

指定しなければ、呼び出し先プログラムでデータをアドレス指定することができません。

DATA は、LOCAL-STORAGE データの 位置に作用しません。 代わりに、STACK ランタイム・オプションがその位置をプログラムの AMODE とともに制御します。

31

ユーザー・データ域 (WORKING-STORAGE や FD レコード域など) が、制限のないストレージから割り振られるか、または LOC=ANY オプションを指定した GETMAIN によって獲得されたスペースで割り振られます。このオプションを指定した場合、ストレージは 16 MB 境界より上の仮想アドレスで 獲得されることもありますし、16MB 境界より下の仮想アドレスで獲得されることも あります。オペレーティング・システムは、通常は、16 MB より上の仮想アドレスの スペースが使用可能であれば、そこで要求を満たします。

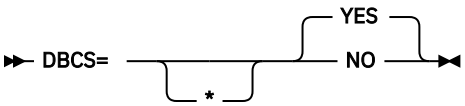
注 :

- プログラムを RENT オプションでコンパイルすると、WORKING-STORAGE およびパラメーター・リスト用のスペースの獲得方法を DATA オプションが制御します。
- DATA オプションは、NORENT オプションでコンパイルされるプログラムには 影響を与えません。
- DATA オプションは、LP(64) が有効な場合は無視されます。 ユーザーが明示的に DATA オプションを指定した場合は、通知メッセージが出されます。
- LOCAL-STORAGE セクションは、Language Environment によって管理されるスタック・ストレージから割り振られます。 LE は、64 ビット・エンクレーブにおいては 2 GB 境界より上のスタック・ストレージを割り振ります。

DBCS

DBCS は、コンパイラーが英数字リテラル内の X'0E' および X'0F' を認識し、それらを DBCS データを区切るためのシフトアウトおよびシフトイン制御文字として扱うかどうかに作用します。

構文



デフォルト
DBCS=YES

YES
英数字リテラル内の X'0E' および X'0F' を認識し、それらを DBCS データを区切るためのシフトアウト制御文字およびシフトイン制御文字として認識します。

NO
英数字リテラル内の X'0E' および X'0F' をシフトアウト制御文字およびシフトイン制御文字として認識しません。

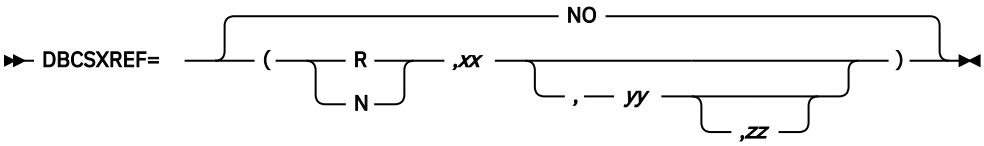
注 :

- 英数字リテラル内に DBCS データが存在すると、コンパイラーはそのリテラルの特定の使用を許可しないことがあります。 例えば、DBCS 文字はプログラム名または DDNAMES としては使用できません。
- DBCS=NO を指定すると、NSYMBOL(NATIONAL) と矛盾します。

DBCSXREF

DBCSXREF は、配列プログラムを使用して DBCS 名の相互参照を行うことを指示します。

構文



デフォルト
DBCSXREF=NO

R
DBCS 日本語配列プログラム (DBCSOS) をユーザー領域にロードすることを 指定します。

N
DBCS 日本語配列プログラム (DBCSOS) を共有システム域 (MLPA など) に ロードすることを指定します。

xx
これは、DBCS 相互参照を生成するための関連配列プログラムのプログラム・オブジェクトを指定します。 8 文字の長さにする必要があります。

yy
配列タイプを指定します。2 文字の長さにする必要があります。このパラメーターを省略すると、指定した配列プログラムが定義するデフォルトの 配列タイプが使用されます。

zz
指定した配列タイプが使用するエンコード・テーブルを指定します。8 文字の長さにする必要があります。このパラメーターを省略すると、特定の配列タイプに関連した デフォルトのエンコード・テーブルが使用されます。

NO
DBCS 名の相互参照に配列プログラムを使用しないことを指定します。XREF フェーズを指定すると、プログラム内の物理的な順序に基づいた DBCS 名相互参照リストが提供されます。

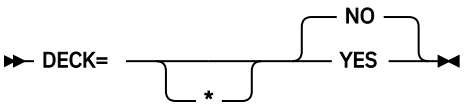
注 :

- DBCSXREF=NO 以外を指定するためには、DBCS 日本語配列プログラム (DBCSOS) が インストールされていなければなりません。
- R が指定されていて、場合は、ユーザー領域がコンパイラーと配列プログラムの両方を入れるために十分な大きさになるようにします。
- XREFOPT=NO と配列プログラムを指定した DBCSXREF の両方を指定すると、カスタマイズ・マクロのアセンブル時にゼロ以外の戻りコードが返されます。
- アセンブル処理は、妥当性検査で以下の状態が診断されると終了します。
  - パラメーター長が無効である
  - 「R」および「N」以外の文字が指定された
  - コンマの後ろのパラメーターが欠落している
  - *zz* が指定されているときに *yy* が欠落している

DECK

DECK は、SYSPUNCH DD ステートメントで定義されたファイル内に 80 桁のオブジェクト・コード・レコードを作成するかどうかを決定します。

構文



デフォルト
DECK=NO

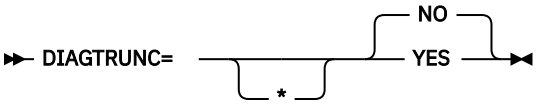
YES
生成されたオブジェクト・コードを、SYSPUNCH が定義するファイルに置きます。

NO
SYSPUNCH にオブジェクト・コードを送りません。

DIAGTRUNC

DIAGTRUNC は、受け取り側が数値である MOVE ステートメントで、受け取りデータ項目の整数桁数が送り出しデータ項目またはリテラルの整数桁数より少ない場合に、コンパイラーが重大度 4 (警告) の診断メッセージを出すかどうかに作用します。

構文



デフォルト
DIAGTRUNC=NO

YES
受け取り側が数値である MOVE ステートメントの場合、受け取りデータ項目の整数桁数が送り出しデータ項目またはリテラルの整数桁数より少ないと、コンパイラーは、重大度 4 (警告) の診断メッセージを出します。

NO
コンパイラーは重大度 4 メッセージを作成しません。

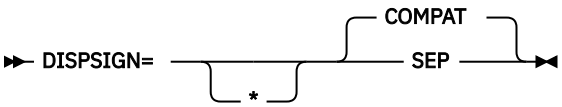
注 :

- 送り出し側が英数字データ名またはリテラルで、受け取り側が数値の場合の移動についても、診断メッセージが出されます (送り出しフィールドが参照変更される場合を除きます)。
- COMP-5 の受け取り側、または TRUNC(BIN) オプションが指定された場合の 2 進数の受け取り側については、診断メッセージは出されません。

DISPSIGN

DISPSIGN オプションは、符号付き数値項目の DISPLAY の出力形式設定を制御します。

構文



デフォルト
DISPSIGN=COMPAT

COMPAT
DISPSIGN=COMPAT を指定した場合、符号付き数値項目の表示値は、 旧バージョンの Enterprise COBOL と互換性がある形式になります。 オーバーパンチ符号が生成される場合もあります。

SEP
DISPSIGN=SEP を指定した場合、符号付き 2 進数、符号付きパック 10 進数、またはオーバーパンチ符号付きゾーン 10 進数の各項目の表示値は 常に、先頭に分離符号が付いた形式になります。

以下の例は、DISPSIGN=COMPAT オプションまたは DISPSIGN=SEP オプションが 指定された DISPLAY 出力を示しています。

表 *3. DISPSIGN=COMPAT* オプションまたは *DISPSIGN=SEP* オプションが指定された *DISPLAY* 出力*:*

データ項目	**DISPSIGN=COMPAT オプションが指定された DISPLAY 出力**	**DISPSIGN=SEP オプションが指定された DISPLAY 出力**
符号なし 2 進数	111	111
正の 2 進数	111	+111
負の 2 進数	11J	-111

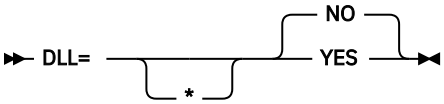
表 3. *DISPSIGN=COMPAT* オプションまたは *DISPSIGN=SEP* オプションが指定された *DISPLAY* 出力: (続き)

データ項目	DISPSIGN=COMPAT オプションが指定された DISPLAY 出力	DISPSIGN=SEP オプションが指定された DISPLAY 出力
符号なしパック 10 進数	222	222
正のパック 10 進数	222	+222
負のパック 10 進数	22K	-222
符号なしゾーン 10 進数	333	333
末尾が正のゾーン 10 進数	33C	+333
末尾が負のゾーン 10 進数	33L	-333
先頭が正のゾーン 10 進数	C33	+333
先頭が負のゾーン 10 進数	L33	-333

DLL

DLL は、コンパイラーによって生成されたオブジェクト・モジュールがダイナミック・リンク・ライブラリー (DLL) サポートで使用可能かどうかに作用します。

構文



デフォルト
DLL=NO

YES
ダイナミック・リンク・ライブラリー (DLL) サポートで使用可能な オブジェクト・モジュールを生成します。DLL 使用可能性が必要となるのは、プログラムが DLL の一部である場合、プログラムが DLL を参照する場合、またはプログラムがオブジェクト指向 の COBOL 構文 (例えば、INVOKE ステートメント、クラス定義) を含んでいる場合です。

DLL オプションを指定する場合は、NODYNAM オプションおよび RENT オプションも使用する必要があります。

NO
DLL 使用を可能にしないオブジェクト・モジュールを生成します。

注 : LP(64) コンパイラー・オプションが有効な場合は、以下の状態になります。

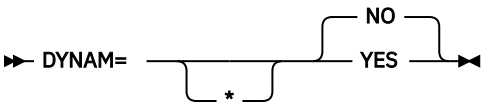
- DLL オプションは不要になります。 LP(64) を使用して生成されたオブジェクト・ファイルは DLL 対応です。 これらは DLL としても非 DLL としてもリンク可能です。
- EXPORALL オプションがサポートされます。 DLL のビルド時にプログラムからシンボルをエクスポートするには、このオプションを使用します。

関連参照
CALLINTERFACE (*Enterprise COBOL* 言語解説書)

DYNAM

DYNAM は、CALL *literal* ステートメントによって呼び出されるサブプログラムを、コンパイラーが動的にロードするかどうかに作用します。

構文



デフォルト
DYNAM=NO

YES
CALL *literal* ステートメントによって呼び出されるサブプログラムを 動的にロードします。

パフォーマンスの考慮: DYNAM=YES を使用すると、サブプログラムを変更した場合にアプリケーションを再リンク・エディットしないため、サブプログラムの保守が容易になります。ただし、CALL *literal* ステートメントのある個々のアプリケーションは、パスが長くなるためにパフォーマンスが多少低下する可能性があります。

NO
呼び出し側プログラムにおいて、CALL *literal* ステートメントによって呼び出されるサブプログラムのテキスト・ファイルが単一のプログラム・オブジェクトに組み込まれます。

注 :

- DYNAM オプションは、コンパイル時に CALL *identifier* ステートメントには 影響を与えません。CALL *identifier* ステートメントは常に、コンパイルされて動的呼び出しになります。
- CICS のもとで実行されるアプリケーションについては、DYNAM=YES を指定してはなりません。

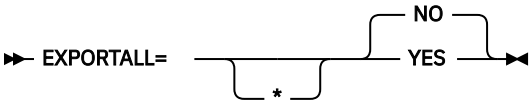
詳しくは、「*Enterprise COBOL* プログラミング・ガイド」で『*DYNAM*』を参照してください。

関連参照
CALLINTERFACE (*Enterprise COBOL* 言語解説書)

EXPORTALL

EXPORTALL は、オブジェクト・デックをリンク・エディットして DLL を作成するときに、コンパイラーが特定の記号を自動的にエクスポートするかどうかに作用します。

構文



デフォルト
EXPORTALL=NO

YES
オブジェクト・デックをリンク・エディットして DLL を作成するときに、プログラム名および代替エントリー・ポイント名を自動的にエクスポートします。

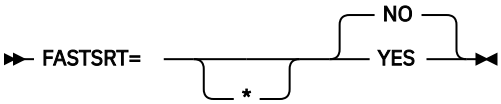
EXPORTALL を指定する場合は、DLL、RENT、および NODYNAM オプションも 使用する必要があります。

NO
どの記号もエクスポートしません。

FASTSRT

FASTSRT は、DFSORT または同等の製品によってソートおよびマージの入出力を実行するかどうか、または Enterprise COBOL によってそれらを実行するかどうかを決定します。 形式 1 SORT ステートメントを使用したファイルのソートにのみ適用されます。

構文



デフォルト
FASTSRT=NO

YES
USING または GIVING オプションの使用時に、 IBM DFSORT ライセンス・プログラムまたは同等の製品で入出力を実行することを指定します。

パフォーマンスの考慮: FASTSRT=YES を 使用すると、各レコードを処理した後で Enterprise COBOL に戻る 際のオーバーヘッドが軽減されます (CPU 時間の使用に関して)。しかし、このオプションを使用する場合には、従わなければならない制約事項が あります。(この制約事項について詳しくは、「*Enterprise COBOL* プログラミング・ガイド」で『*FASTSRT* を使用してのソートのパフォーマンスの向上』を参照してください。)

NO
Enterprise COBOL がソートおよびマージについて入出力を実行することを指定します。

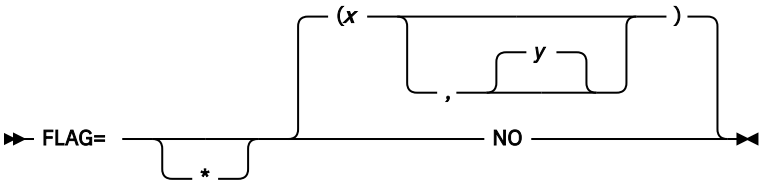
注 :

- コンパイル時に FASTSRT が有効である場合、コンパイラーは、以下の 2 つを除くすべての制約事項について FASTSRT インターフェースを使用できるかどうかを検査します。
  - ソート作業ファイル用に直接アクセス装置以外の装置を使用する必要が あること
  - 入力ファイルまたは出力ファイルの DD ステートメントの DCB パラメーターが、ファイルのファイル記述 (FD) と一致する必要があること
- FASTSRT が使用できない場合は、コンパイラーは、USING または GIVING オプションの使用時に診断メッセージを生成し、ソート・プログラムが入出力を実行できないようにします。このため、デフォルトとして YES を指定すると、有利な場合があります。

FLAG

FLAG は、指定した重大度レベル以上の診断メッセージをコンパイラーが生成するかどうかに作用します。

構文



デフォルト
FLAG=(I,I)

注 : この構文で使用する 2 番目の重大度レベルは、1 番目の重大度レベル以上でなければ なりません。

x
I|W|E|S|U

指定された重大度レベル以上のエラーにフラグを立てて、そのエラーをソース・リストの末尾に書き込むことを指定します。

ID	タイプ	戻りコード
I	通知	0
W	警告	4
E	エラー	8
S	重大エラー	12
U	回復不能エラー	16

y

I|W|E|S|U

任意指定の 2 番目の重大度レベルは、ソース・リストの終わりに書き込まれる だけでなくソース・リスト内に挿入される構文メッセージのレベルを指定します。

NO

エラー・メッセージにフラグを立てないことを示します。

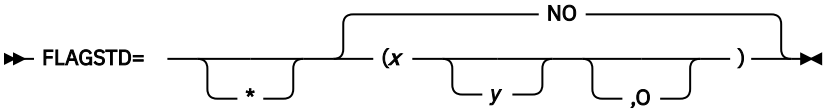
注 :

- ソース・リストにメッセージを挿入する場合は、コンパイル時に SOURCE を指定 する必要があります。こうすると、該当するソース・ステートメントの後に メッセージが置かれるので、生産性が向上します。
- FLAG(W|E|S) を指定すると、関連データ (SYSADATA) ファイル内のイベント・レコードから全クラスのメッセージが消失する場合があります。 詳しくは、「*Enterprise COBOL プログラミング・ガイド*」で『*FLAG*』を参照してください。

FLAGSTD

FLAGSTD は、準拠していると見なされる 85 COBOL 標準言語エレメントのサブセット、および使用される言語エレメントに関する情報メッセージのフラグ付けに作用します。

構文



デフォルト

FLAGSTD=NO

x

M、I、または H です。FIPS COBOL サブセットまたは規格についての フラグを立てることを指定します。

M =

標準 COBOL の ANS 最小サブセット

I =

ANS 最小サブセットの一部ではない別の 中間サブセット言語エレメントから構成される、ANS 中間サブセット

H =

ANS 中間サブセットの一部ではない別の 高サブセット言語エレメントから構成される、ANS 高サブセット

y

D、N、または S の 1 つまたは 2 つの組み合わせです。フラグ設定のレベルを より詳しく定義します。

D

ANS デバッグ・モジュール・レベル 1 を指定します

N

ANS 分割モジュール・レベル 1 を指定します

S
ANS 分割モジュール・レベル 2 を指定します (S は N のスーパーセットです)

O
上記のセットの中で生じる差し替え済みエレメントにフラグを 立てることを指定します。

NO
FIPS フラグ設定を行わないことを指定します

注 :

- 以下のエレメントは、85 COBOL 標準に対する規格適合外の非標準 IBM 拡張機能としてフラグが立てられます。
  - COBOL 自動日付処理機能で使用される言語構文
  - オブジェクト指向および Java™ との相互運用性の向上のための言語構文
  - PGMNAME=LONGMIXED コンパイラー・オプションの使用

  フラグが立てられる、規格適合外の非標準エレメントの完全なリストについては、「*Enterprise COBOL* 言語解説書」で『*IBM* 拡張機能』を参照してください。
- FIPS フラグ設定を指定した場合、ソース・プログラム・リスト内の通知メッセージは以下を示します。
  - 言語エレメントが廃止であるか、非準拠の標準であるか、または非準拠の非標準であるか (廃止かつ非準拠である言語エレメントは、単に廃止としてフラグが立てられます)
  - 非準拠または廃止である構文を含んでいる節、ステートメント、またはヘッダー
  - ソース・プログラム行、およびその行での開始桁の提示
  - 言語エレメントが属しているレベルまたは任意指定のモジュール
- E レベル以上のエラーとして診断されるエラーが発生すると、FIPS フラグ設定は 抑制されます。
- FLAGSTD とその他のコンパイラー・オプションの相互作用:
  - 以下のコンパイラー・オプションがプログラムで明示的または暗黙的に指定されている場合は、FLAGSTD=(NO 以外) を指定すると、コンパイラーの FIPS メッセージが出されます。
    - ADV=NO
    - BLOCK0=YES
    - CICS=YES
    - DLL=YES
    - DYNAM=NO
    - EXPORTALL=YES
    - FASTSRT=YES
    - LITCHAR=APOST
    - NAME=NO
    - NUMPROC=PFD
    - PGMNAME=LONGMIXED
    - QUALIFY=EXTEND
    - THREAD=YES
    - TRUNC=OPT または BIN
    - VLR=COMPAT
    - WORD=(NO 以外、または RWT 以外)
    - INVDATA=(NO 以外)
    - ZWB=NO
  - 以下のオプションを FLAGSTD=(NO 以外) とともに指定すると、カスタマイズ・マクロのアセンブル試行時にゼロ以外の戻りコードが返されます。

- ADV=NO
- DBCS=YES
- DYNAM=NO
- LITCHAR=APOST
- NUM=YES
- NUMPROC=PFD
- QUALIFY=EXTEND
- SEQ=YES
- TRUNC=OPT または BIN
- VLR=COMPAT
- WORD=(NO 以外、または RWT 以外)
- INVDATA=(NO 以外)
- ZWB=NO

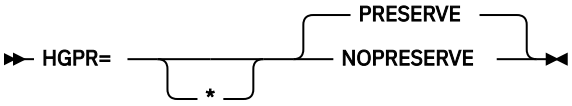
- FLAGSTD は、関連データ・ファイル内に、FIPS 規格合致メッセージの イベント・レコードを作成することがあります。エラー・メッセージは、ソース・レコード番号の順番になるとは限りません。

FLAGSTD メッセージを診断メッセージに変換するか、抑制することができます。 詳細については、44 ページの『MSGEXIT』を参照してください。

HGPR

HGPR オプションは、z/Architecture プロセッサーに備わっている 64 ビット・レジスターをコンパイラーが使用する方法を制御します。

構文



デフォルト: HGPR=PRESERVE

Enterprise COBOL コンパイラーは、64 ビット幅の z/Architecture 汎用レジスター (GPR) を使用します。HGPR は「High-halves of 64-bit GPRs (64 ビット GPR の上位半分)」の略語であり、ネイティブ 64 ビット命令を使用することを意味します。

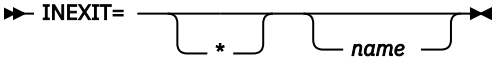
HGPR=PRESERVE
HGPR=PRESERVE を指定した場合、 コンパイラーは、プログラムが使用する 64 ビット GPR の高位半分を、関数のプロローグで保存してエピローグで復元することによって保持します。 PRESERVE サブオプションは、プログラムの呼び出し元が Enterprise COBOL、Enterprise PL/I、z/OS XL C/C++ のどのコンパイラー生成コードでもない場合にのみ必要です。

HGPR=NOPRESERVE
HGPR=NOPRESERVE を指定した場合、 コンパイラーは、プログラムが使用する 64 ビット GPR の高位半分を保持する処理を省略するため、パフォーマンスが向上します。

INEXIT

INEXIT は、SYSIN データ・セットを読み取る代わりに、ソース・ステートメントを取得するために呼び出すモジュールを指定します。

構文



デフォルト

出口は指定されません。EXIT コンパイラー・オプションの NOINEXIT サブオプションを 指定するのと同等です。 INEXIT=* が *name* パラメーターなしでコーディングされた場合、NOINEXIT を オーバーライドすることはできません。

name

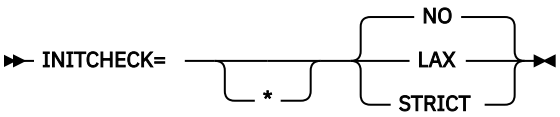
EXIT コンパイラー・オプションで使用するモジュールを識別します。このユーザー出口のサブオプションが指定されている場合、コンパイラーは SYSIN データ・セットを読み取らずに、指定されたモジュールを ロードし、それを呼び出して、ソース・ステートメントを取得します。このオプションを指定した場合は、SYSIN データ・セットはオープンされません。

EXIT コンパイラー・オプションについて詳しくは、「*Enterprise COBOL* プログラミング・ガイド」で『*EXIT* コンパイラー・オプション』を参照してください。

INITCHECK

INITCHECK オプションは、初期化されていないデータ項目をコンパイラーで検査したり、データ項目が初期化されずに使用されているときにコンパイラーに警告メッセージを発行させたりする場合に使用します。

構文



デフォルト

INITCHECK=NO

NO

コンパイラーは、初期化されていないデータ項目について警告メッセージをいっさい発行しません。

LAX

コンパイラーは、初期化されていないデータ項目について検査を行い、データ項目が初期化されずに使用されていると警告メッセージを発行します。 ただし、ステートメントへの少なくとも 1 つの論理パスでデータ項目が初期化されていれば、警告メッセージは発行されません。

STRICT

この場合も、コンパイラーは初期化されていないデータ項目について検査を行い、データ項目が初期化されずに使用されていると、警告メッセージを発行します。 ただし、INITCHECK=LAX とは違って、INITCHECK=STRICT は、ステートメントへのすべての論理パスでデータ項目が初期化されているのでないと、ステートメントで使用されているデータ項目が初期化されていないという警告メッセージを発行します。

以下のサンプル・プログラムでは、INITCHECK=LAX を指定した場合と INITCHECK=STRICT を指定した場合の動作の違いを示しています。 Y と Z は、値の節がない、とあるデータ項目を表しています。

```
    PROCEDURE DIVISION.
de
    IF Y > 5
      MOVE 2 TO Z
    END-IF
    DISPLAY Z
```

Z は、DISPLAY ステートメントへの 1 つのパスでは初期化されていますが、もう 1 つのパスでは初期化されていません。 そのため、INITCHECK=LAX が有効な場合は Y に対してのみ警告メッセージが発行され、 INITCHECK=STRICT の場合は Z に対しても警告メッセージが発行されます。

制限 :

- INITCHECK オプションでデータ項目が分析されるのは、WORKING-STORAGE SECTION および LOCAL-STORAGE SECTION においてのみです。 特に、LINKAGE SECTION や FILE SECTION ではデータ項目は分析されません。

- INITCHECK 分析では、外部データ項目およびグローバル・データ項目は追跡されません。
- INITCHECK 分析では、テーブルにある個別のエレメントは独立しては追跡されません。代わりに、テーブル・エレメントが 1 つ初期化されると、同様のテーブル・エレメントはすべて初期化されるとみなされます。 これは固定長テーブルにも可変長テーブルにも適用されます。
- INITCHECK 分析では、ポインターによって項目の初期化が行われる場合でも、項目の初期化は追跡されません。 例えば、初期化されていないデータ項目を指すポインターが ADDRESS-OF を使用して作成され、そのデータ項目がそのポインターによって初期化された場合は、INITCHECK 分析でも警告メッセージが発行される可能性があります。
- 初期化されていないデータ項目が BY REFERENCE で渡される場合、警告メッセージは発行されません。ただし、INITCHECK 分析では、BY CONTENT および BY VALUE で渡されるデータ項目が初期化されていないことに関しては警告が発せられます。

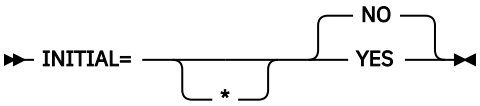
注:

- INITCHECK 分析はいずれもコンパイル時にのみ行われます。
- INITCHECK オプションは、プログラムのコンパイル後はプログラムの動作やパフォーマンスに影響しません。
- INITCHECK オプションを使用すると、コンパイル時間が長引いたりメモリー消費量が増えたりすることがあります。
- INITCHECK オプションは、グループ内の最初の初期化されていないデータ項目だけを報告および印刷します。 それに続く、同様に初期化されていないデータ項目は印刷されません。
- INITCHECK は、OPT=1 や OPT=2 と共に使用するとより正確になりますが、 OPT=0 と共に指定する場合にも役立ちます。

INITIAL

INITIAL コンパイラー・オプションによって、あるプログラムとそのネスト・プログラムが、IS INITIAL 節が PROGRAM-ID 段落に指定されたかのように動作します。

構文



デフォルト: INITIAL=NO

YES
INITIAL=YES によって、あるプログラムとそのネスト・プログラムが、IS INITIAL 節が PROGRAM-ID 段落に指定されたかのように動作します。

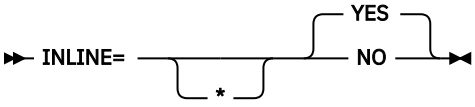
注: INITIAL=YES および IS INITIAL 節は、VALUE 節のないデータ項目には影響しません。

NO
INITIAL=NO は、そのソースにおいて既に IS INITIAL を PROGRAM-ID 段落に持っているプログラムには影響しません。

INLINE

INLINE オプションは、ソース・プログラムで PERFORM ステートメントによって参照されるプロシージャー (段落またはセクション) のインライン化を許可するかどうかを制御します。

構文



デフォルト: INLINE=YES

YES

INLINE=YES を指定すると、OPTIMIZE(1) または OPTIMIZE(2) が有効になっている場合、コンパイラーは、ソース・プログラムにおいて PERFORM ステートメントによって参照されるプロシージャーをインライン化できます。

NO

INLINE=NO を指定すると、有効になっている最適化レベル設定に関係なく、コンパイラーは、ソース・プログラムにおいて PERFORM ステートメントによって参照されるプロシージャーをインライン化[1] できなくなります。

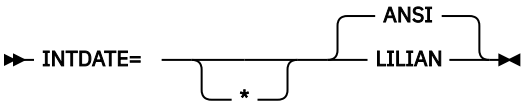
注:

1. ここで述べているインライン化 という単語は、コンパイラーがプロシージャー (段落またはセクション) の PERFORM をそのプロシージャーのコードのコピーで置き換えることを選択する可能性があることを暗示しています。 コンパイラーは、PERFORM の位置にプロシージャー・コードを挿入することで、ロジックをプロシージャーに/プロシージャーから分岐するオーバーヘッドを省きます。

INTDATE

INTDATE は、日付組み込み関数で使用される開始日に作用します。

構文



デフォルト

INTDATE=ANSI

ANSI

日付組み込み関数で使用される整数日付形式の日付に、ANSI COBOL 規格 の開始日付を使用します。 Day 1 = Jan 1, 1601。

INTDATE(ANSI) を指定すると、日付組み込み関数は COBOL/370 リリース 1 の場合と同じ結果を返します。

LILIAN

日付組み込み関数で使用される整数日付形式の日付に、言語環境リリアン開始日付を使用します。 Day 1 = Oct 15, 1582。

INTDATE(LILIAN) を指定すると、日付組み込み関数は、Language Environment の日付呼び出し可能サービスと互換性のある結果を返します。 これらの結果は、COBOL/370 リリース 1 での結果とは異なります。

注:

- INTDATE(LILIAN) が有効である場合は、組み込み関数または呼び出し可能 サービスのいずれかを使用して ANSI 整数を意味のある日付に変更することが できないので、CEECBLDY を使用することはできません。INTDATE(LILIAN) が有効である場合、呼び出しのターゲットとして CEECBLDY を 使用して CALL literal ステートメントをコーディングすると、コンパイラーはこれを診断し、この呼び出しターゲットを CEEDAYS に変換します。
- インストール・オプションを INTDATE(LILIAN) に設定した場合は、組み込み関数を使用するすべての COBOL/370 リリース 1 プログラムを 再コンパイルすることにより、すべてのコードがリリアン整数日付規格を 使用するようにしてください。整数の日付を保管し、プログラム間で渡したり、PL/I から COBOL および C プログラ ムへ問題なく渡すことができるので、このメソッドは最も安全な方式です。

INVDATA

INVDATA オプションは、USAGE DISPLAY データ項目および PACKED-DECIMAL データ項目におけるデータが有効であるかどうかをコンパイラーに伝え、そのデータが有効ではない場合、コンパイラーの動作はどうあるべきかをコンパイラーに指示します。

大半のユーザーは USAGE DISPLAY データ項目と USAGE PACKED-DECIMAL データ項目に有効なデータを入れているため、 NUMPROC(NOPFD) を使用する場合でも NOINVDATA を使用する必要があります。 実行時にプログラムが無効データを処理していることが NUMCHECK コンパイラー・オプションによって検出された場合にも、 無効データの処理を回避するためにプログラムを変更し、NOINVDATA を使用する必要があります。

注 : INVDATA オプションの目標は、無効な数値データがある場合に、 COBOL V4 以前のバージョンでコンパイルされたプログラムの動作になるべく対応した動作を提供することです。 矛盾が見つかった場合、このオプションは、COBOL V4 以前のバージョンの動作ともっとよく一致した動作になるように更新されます。

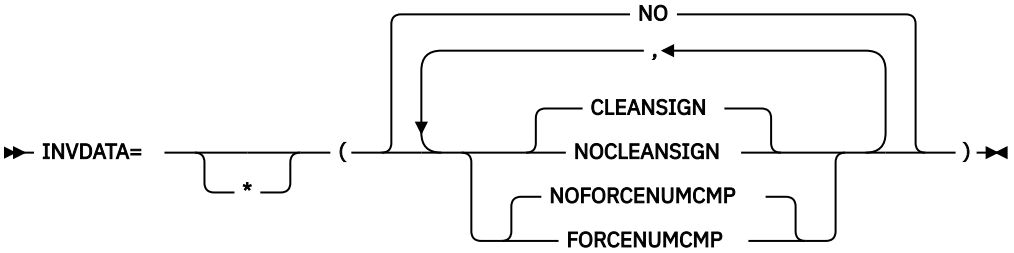
INVDATA オプションが有効な場合、ゾーン 10 進数またはパック 10 進数のデータ項目に無効な数字や無効な符号が含まれていたり、 ゾーン 10 進数データ項目に無効なゾーン・ビットが含まれていたりすると、コンパイラーは、COBOL V4 以前のバージョンとは異なる結果になる可能性がある既知の最適化を実行しません。

以下の表は、COBOL V4 以前のバージョンからマイグレーションする際に、そのバージョンで使用されている NUMPROC オプションの値と無効データの有無に応じて INVDATA オプションと NUMPROC オプションをどのように設定すればよいかを確認できるクイック・リファレンスです。

表 *4. INVDATA* オプションと *NUMPROC* オプションの設定

	V4 COBOL 以前のバージョンで NUMPROC=PFD	**V4 COBOL 以前のバージョンで NUMPROC=NOPFD**	**V4 COBOL 以前のバージョンで NUMPROC=MIG**
無効データはない	INVDATA=(NO),NUMPROC=(PFD)	INVDATA=(NO),NUMPROC=(NOPFD)	INVDATA=(NO),NUMPROC=(NOPFD)
無効データ	INVDATA=(NOFORCENUMCMP),NUMPROC=(PFD)	INVDATA=(NOFORCENUMCMP),NUMPROC=(NOPFD)	INVDATA=(FORCENUMCMP,NOCLEANSIGN),NUMPROC=(NOPFD)

構文



デフォルト

INVDATA=(NO)

サブオプションのデフォルトは以下のとおりです。

- INVDATA が指定されていて、FORCENUMCMP も NOFORCENUMCMP も指定されていない場合、デフォルトは NOFORCENUMCMP です。
- INVDATA が指定されていて、CLEANSIGN も NOCLEANSIGN も指定されていない場合、デフォルトは CLEANSIGN です。

(NO)

INVDATA=(NO) が有効な場合、コンパイラーは、USAGE DISPLAY データ項目および PACKED-DECIMAL データ項目にあるすべてのデータが有効であることを想定し、可能な限り最も効率的なコードを生成します。例えば、コンパイラーは数値変換を避けるためにストリング比較を生成する場合があります。

(FORCENUMCMP | NOFORCENUMCMP)

INVDATA=(FORCENUMCMP) が有効な場合、コンパイラーは、ゾーン 10 進数データ項目内の各桁のゾーン・ビットを無視する数値比較を行うための命令を生成します。例えば、ゾーン 10 進数値は比較前に PACK 命令によってパック 10 進数に変換されます。

INVDATA=(NOFORCENUMCMP) が有効な場合、コンパイラーは、COBOL V4 以前のバージョンで NUMPROC=(NOPFD | PFD) を使用した場合に COBOL V4 以前のバージョンが行うのと同じ方法でゾーン 10 進数データの数値比較または英数字比較を行うための命令を生成します。

- COBOL V4 以前のバージョンでゾーン・ビットが考慮されていた場合、コンパイラーは英数字比較を生成します。 この比較でも、ゾーン 10 進数データ項目内の各桁のゾーン・ビットが考慮されます。ゾーン 10 進数値はゾーン 10 進数のままです。
- COBOL V4 以前のバージョンでゾーン・ビットが無視されていた場合は、ゾーン 10 進数データ項目内の各桁のゾーン・ビットが無視される数値比較をコンパイラーが生成します。ゾーン 10 進数値は比較前に、PACK 命令によってパック 10 進数に変換されます。

COBOL V4 以前のバージョンでゾーン 10 進数データの比較の生成時に行われていた方法と同じ方法でコンパイラーがゾーン・ビットを処理するようにするには、 COBOL V6 で使用される NUMPROC サブオプションが、COBOL V4 以前のバージョンで使用される NUMPROC サブオプションと一致しなければなりません。

- COBOL V6 で COBOL V4 以前のバージョンの NUMPROC=(NOPFD) 動作になるようにするには、COBOL V6 で INVDATA=(NOFORCENUMCMP) および NUMPROC=(NOPFD) を使用します。
- COBOL V6 で COBOL V4 以前のバージョンの NUMPROC=(PFD) 動作になるようにするには、COBOL V6 で INVDATA=(NOFORCENUMCMP) および NUMPROC=(PFD) を使用します。

注 : 符号コードは、NUMPROC コンパイラー・オプション設定に準じた有効な符号コードでなければなりません。さらに、下位バイトには、SIGN IS LEADING または SIGN IS SEPARATE のいずれかによる符号解除および符号付けのために有効なゾーン (x'F') がなければなりません。

(CLEANSIGN | NOCLEANSIGN)

INVDATA が指定されていて、CLEANSIGN も NOCLEANSIGN も指定されていない場合、CLEANSIGN がデフォルトとなります。

INVDATA=(CLEANSIGN) オプションが有効な場合、コンパイラーは、比較、加算、減算、乗算、除算の各演算への入力において USAGE DISPLAY データ項目と USAGE PACKED-DECIMAL データ項目の符号ニブルを消去するコードを生成します。

注 : 符号付きのオーバーパンチと符号なしのデータ項目のみが影響を受けます。

INVDATA=(NOCLEANSIGN) オプションが有効な場合、コンパイラーは、比較、加算、減算、乗算、除算の各演算への入力において USAGE DISPLAY データ項目と USAGE PACKED-DECIMAL データ項目の符号ニブルを消去するコードを生成しません。これにより、演算のオペランドのいずれかに無効な符号ニブルが含まれるときに S0C7 アベンドが発生する可能性が高くなります。

注 : INVDATA オプションは、無効な数字、無効な符号コード、または無効なゾーン・ビットを含む可能性のある USAGE DISPLAY データ項目または PACKED-DECIMAL データ項目に対する、MOVE ステートメントの動作、比較、および計算に作用します。

以下の例は、データ項目 VALUE1 の中間にあるゾーン・ビットに、無効なゾーン・ビット 4 が REDEFINES によって強制的に入れられているデータ項目を示しています。

```
77  VALUE0    PIC X(4) VALUE '00 0'.              *>  x'F0F040F0'
 77  VALUE1    REDEFINES VALUE0 PIC 9(4).
 PROCEDURE DIVISION.
     IF VALUE1 = ZERO
        DISPLAY 'INVDATA(FORCENUMCMP) is in effect ' VALUE1
     ELSE
        DISPLAY 'INVDATA(NOFORCENUMCMP) is in effect '  VALUE1
     END-IFCopy code
```

この例では、次のようになっています。

- COBOL V4 以前のバージョンでは、NUMPROC=(MIG) オプションが使用されている場合はテストが true になり、NUMPROC=(NOPFD | PFD) の場合は false になります。
- COBOL V5 以降のバージョン:
  - INVDATA=(NO) が使用されている場合、テストは OPT=(0) で true であり OPT=(1 | 2) で false です。
  - INVDATA=(NOFORCENUMCMP) が使用されている場合、テストはいずれの OPT 設定でも false です。

どの値を指定した場合でも、COBOL V6 へのマイグレーションを容易にするには、以下のようにします。

- 数字、符号コード、およびゾーン・ビットが有効な場合は INVDATA=(NO) を使用します。 COBOL V4 以前のバージョンで NUMPROC=(PFD) または NUMPROC=(NOPFD) を使用していた場合は、COBOL V6 でも同じ NUMPROC 設定を使用します。 COBOL V4 以前のバージョンで NUMPROC=(MIG) を使用していた場合は、COBOL V6 の使用時は NUMPROC=(NOPFD) を使用します。
- データに無効な数字、無効な符号コード、または無効なゾーン・ビットがある場合は、プログラムが実行時に数値データ項目に無効なデータを持たないように、プログラムまたはシステムを変更してください。

  プログラムまたはシステムを修正したら、お好みの INVDATA=(NO) オプションを使用できます。この作業を含めることができず、無効データを使用した実行を続けなければならない場合にのみ、INVDATA に対して以下の選択項目を検討してください。
  - COBOL V4 以前のバージョンで NUMPROC=(MIG) を使用した場合は、INVDATA=(FORCENUMCMP,NOCLEANSIGN) と NUMPROC=(NOPFD) を COBOL V6 で使用します。
  - COBOL V4 以前のバージョンで NUMPROC=(NOPFD) を使用した場合は、INVDATA=(NOFORCENUMCMP,CLEANSIGN) と NUMPROC=(NOPFD) を COBOL V6 で使用します。
  - COBOL V4 以前のバージョンで NUMPROC=(PFD) を使用した場合は、INVDATA=(NOFORCENUMCMP,CLEANSIGN) と NUMPROC=(PFD) を COBOL V6 で使用します。

  注:
  - 過去に COBOL V4 以前のバージョンから COBOL V5 または V6 へのマイグレーションを完了し、COBOL V5 または V6 で非推奨の ZONEDATA=(MIG) オプションを使用していて、その動作に問題がなければ、 ZONEDATA=(MIG) の代わりに INVDATA=(FORCENUMCMP, CLEANSIGN) を使用してください。
  - 明らかに無効なデータが検出された場合、これらのオプションを使用したとしても、古いコンパイラーの動作を完全に一致させることは常に可能ではありません。例えば、 比較の場合でも、INVDATA=(NOFORCENUMCMP) により、すべてのケースで COBOL V4 と同じ結果が得られるわけではありません。

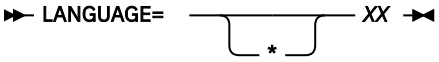
パフォーマンスの考慮: INVDATA=(NO) のランタイム・パフォーマンスは、 INVDATA(NOFORCENUMCMP | FORCENUMCMP,NOCLEANSIGN | CLEANSIGN) よりも優れています。 INVDATA(NOFORCENUMCMP | FORCENUMCMP,NOCLEANSIGN | CLEANSIGN) を使用すると、NUMPROC=(PFD) によって得られる最適化の一部が無効になります。

関連参照
50 ページの『NUMPROC』

LANGUAGE

LANGUAGE は、コンパイラー出力メッセージに使用される言語に作用します。

構文



デフォルト
LANGUAGE=EN

XX

コンパイラー出力メッセージ用の言語を指定します。このパラメーターの値は、以下のリストから選択してください。

表 *5. LANGUAGE* コンパイラー・オプションの値

値	言語
EN または **ENGLISH**	英語 (大 / 小文字混合)
JA、**JP**、または **JAPANESE**	日本語
UE または **UENGLISH**	英語 (大文字)

注 :

- LANGUAGE オプション名は、少なくとも最初の 2 文字を入力する必要があります。最初の 2 文字の後の文字も使用してかまいませんが、言語名の判別に 使用されるのは最初の 2 文字だけです。
- このコンパイラー・オプションは、ランタイム・メッセージが表示されるときの 言語には影響を与えません。ランタイム・オプションおよびメッセージについて詳しくは、「*z/OS* 言語環境プログラム プログラミング・ガイド」を参照してください。
- プリンターによっては、大文字のみを使用し、大 / 小文字混合 (LANGUAGE=ENGLISH) の出力を受け入れません。
- 日本語オプションを指定するためには、日本語機能がインストールされていなければ なりません。
- 英語オプション (英大/小文字混合) を指定するためには、英語機能が インストールされていなければなりません。
- ご使用のシステムに上記以外の言語が提供されており、それをご使用のシステムのデフォルトとして選択する場合は、その言語名の少なくとも最初の 2 文字を指定する必要があります。この 2 文字は英数字でなければなりません。
- ADATA オプションの指定と一緒に日本語を選択すると、 関連データ・ファイル内のエラー識別レコードに DBCS 文字が書き込まれる 結果になることがあります。
- コンパイラー・メッセージを大文字の英語に変更したり日本語に変更したりするには、LANGUAGE コンパイラー・オプションを使用するだけでなく、コンパイル時に Language Environment・ランタイム・オプション NATLANG も設定する必要があります。 コンパイル JCL では CEEOPTS DD を使用することをお勧めします。

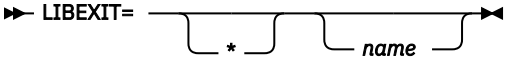
  例えば、メッセージを日本語に変更するには、LANGUAGE=JA を使用し、コンパイル時には NATLANG LE ランタイム・オプションも指定します。

```
//CEEOPTS DD *
          NATLANG(JPN)
/*
```

LIBEXIT

LIBEXIT は、SYSLIB または library-name データ・セットを読み取る代わりに、COPY ステートメントを取得するために呼び出すモジュールを指定します。

構文



デフォルト

出口は指定されません。EXIT コンパイラー・オプションの NOLIBEXIT サブオプションを 指定するのと同等です。 LIBEXIT=* が *name* パラメーターなしでコーディングされた場合、NOLIBEXIT を オーバーライドすることはできません。

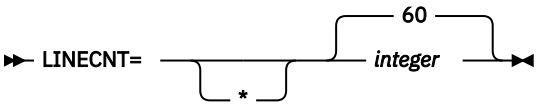
name
EXIT コンパイラー・オプションで使用するモジュールを識別します。このユーザー出口の サブオプションが指定されている場合、コンパイラーは SYSLIB または library-name データ・セットを 読み取らずに、指定されたモジュールをロードし、それを呼び出して COPY ステートメントを取得します。 このオプションを指定した場合は、SYSLIB および library-name データ・セットは オープンされません。

EXIT コンパイラー・オプションについて詳しくは、「*Enterprise COBOL* プログラミング・ガイド」で『*EXIT* コンパイラー・オプション』を参照してください。

LINECNT

LINECNT は、コンパイラー・ソース・リストの各ページに印刷される行数に作用します。

構文



デフォルト
LINECNT=60

integer
コンパイラー・ソース・コード・リストの各ページに印刷される行数を指定します。10 から 255 までの整数または 0 でなければなりません。見出しを生成するために 3 行が使用されます。 例えば、LINECNT=60 を指定した場合は、57 行のソース・コードが出力リストの 各ページに印刷され、3 行がヘッディング用に使用されます。

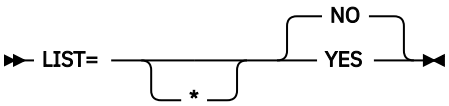
LINECNT=0 を指定すると、コンパイル・リストではページ替えが行われません。

LINECNT インストール・オプションは、LINECOUNT コンパイラー・オプションと同等です。

LIST

LIST は、ソース・リストにアセンブラー言語展開を作成するかどうかに作用します。

構文



デフォルト
LIST=NO

YES
以下を含むリストを作成します。

- ソース・コードのアセンブラー言語展開
- 定数域
- プログラム・プロローグ域 (PPA1、PPA2、PPA3、PPA4)
- タイム・スタンプ、コンパイラー・バージョン、およびビルド・レベル情報
- コンパイラー・オプションおよびプログラム情報
- ベース・ロケーター・テーブル
- 外部シンボル辞書
- 初期ヒープ・ストレージ・マップ
- スタック・ストレージ・マップ

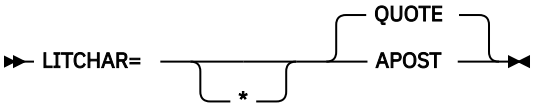
NO
このリストを抑止します。

LIST と OFFSET コンパイラー・オプションは、相互に排他的です。OFFSET=YES と LIST=YES を共に指定すると、カスタマイズ・マクロのアセンブル時に、ゼロ以外の戻りコードおよびエラー・メッセージが出されます。

LITCHAR

LITCHAR は、QUOTE 形象定数が引用符またはアポストロフィのどちらを表すかに作用します。

構文



デフォルト
LITCHAR=QUOTE

APOST
1 つ以上のアポストロフィ (') 文字を表すために形象定数 [ALL] QUOTE または [ALL] QUOTES が必要な場合は、APOST を使用します。

QUOTE
1 つ以上の引用符 (") 文字を表すために形象定数 [ALL] QUOTE または [ALL] QUOTES が必要な場合は、QUOTE を使用します。QUOTE は 85 COBOL 標準に準拠します。

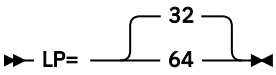
注 :

- APOST または QUOTE オプションが有効であるかどうかに関係なく、引用符またはアポストロフィのいずれかをリテラル区切り文字として使用できます。
- リテラルの左区切り文字として使用する区切り文字を、その同じ リテラルの右区切り文字として使用する必要があります。

LP

LP コンパイラー・オプションは、COBOL プログラムのコンパイル時に、関連する言語機能を有効にして AMODE 31 (31 ビット) または AMODE 64 (64 ビット) のどちらのプログラムを生成するかに作用します。

構文



LP オプションは、他のコンパイラー・オプションと同じ方法で指定できます。 ただし、CBL (PROCESS) ステートメントで LP オプションを指定する場合は、最初のプログラムに対してのみ指定できます。 バッチでは、後続のプログラムに対してオプションの値を変更することはできません。

デフォルト
LP=32

32
関連する言語機能を有効にして AMODE 31 (31 ビット) プログラムを生成することを指示します。

64
関連する言語機能を有効にして AMODE 64 (64 ビット) プログラムを生成することを指示します。

実行時の考慮事項: 現在、Language Environment は、同じアプリケーション内で AMODE 64 プログラムと AMODE 31 プログラムの混在をサポートしていません。 1 つのプログラムが LP(64) でコンパイルされた場合、アプリケーション内のすべてのプログラムも LP(64) でコンパイルする必要があります。 静的 CALL の場合、バインダーは、外部名の解決中にアドレッシング・モードの混在を検出するとメッセージを出します。 動的 CALL の場合、一方のプログラムが AMODE 64 で、もう一方のプログラムが AMODE 31 または 24 である場合、それらのプログラム間での CALL に対しては実行時エラーが出されます。

LP=64 コンパイラー・オプションを使用すると、コンパイル・プロセスには POSIX(ON) モードで実行されるコンポーネントが組み込まれます。 必然的に、このオプションを指定してコンパイラーを実行するユー

ザーごとに RACF® で OMVS セグメント (RACF 代替製品で同等のもの) が設定されていなければならないということになります。

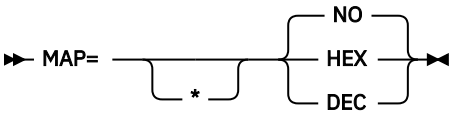
LP=64 を使用するプログラムの制限: LP=64 コンパイラー・オプションを使用してコンパイルされるプログラムには、XML GENERATE または XML PARSE ステートメント、JSON GENERATE または JSON PARSE ステートメント、オブジェクト指向 COBOL ステートメント、ALTER ステートメント、GO TO ステートメント、および DISPLAY ... UPON SYSPUNCH ステートメントを含めることはできません。さらに、AMODE 64 プログラムは CICS や IMS でも実行できません。

注 : LP(64) では、一部のコンパイラー・オプションは適用されません。 詳しくは、『AMODE 64 プログラムをコンパイルするコンパイラー・オプションの使用』 (*Enterprise COBOL* プログラミング・ガイド) を参照してください。

MAP

MAP オプションは、DATA DIVISION 項目に関するマップ情報と、暗黙的に宣言された項目をリストに表示するかどうかに作用します。 このオプションはまた、リストのマップ出力に 16 進数または 10 進数のどちらのオフセットを表示するかを制御します。

構文



デフォルト
MAP=NO

HEX または DEC
DATA DIVISION で宣言されている項目をマップします。 マップ出力には以下が含まれます。

- DATA DIVISION マップ
- ネストされたプログラム構造マップ、およびプログラム属性
- プログラムの WORKING-STORAGE と LOCAL-STORAGE の サイズ、およびプログラムが NORENT オプションを使用してコンパイルされた 場合には、オブジェクト・コード内でのその位置

MAP=HEX を指定した場合、グループ内のデータ項目オフセットは 16 進表記になります。

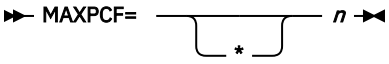
MAP=DEC を指定した場合、グループ内のデータ項目オフセットは 10 進表記になります。

NO
マッピングを実行しません。

MAXPCF

MAXPCF オプションは、最大プログラム複雑度係数値を指定するために使用します。 プログラム複雑度係数 (PCF) はコンパイラーによって計算され、その値はリスト・ファイルに収容されます。 プログラムの PCF が最大値を超えると、コンパイラーは自動的に最適化レベルを下げて、コンパイルを高速化し、ストレージ使用量を削減します。 そのため、一組のプログラムをコンパイルする際に、プログラムごとに OPTIMIZE オプション値を指定する必要はありません。

構文



デフォルト
MAXPCF=100000

n
0 から 999999 までの整数でなければなりません。

複雑度係数を計算する際には、プログラムについて以下を考慮してください。

- CICS、SQL または SQLIMS オプションから生成されたステートメント、および COPY や REPLACE ステートメントの拡張を含む、PROCEDURE DIVISION 内の COBOL ステートメント数
- value 節のある WORKING-STORAGE データ項目または LOCAL-STORAGE データ項目の初期化操作
- DATA DIVISION 内の可変長グループまたはサブグループのサイズを実行時に計算する操作

注 : PCF は、プログラムの複雑さを測定するための測定基準ではありません。 これは、COBOL 項目が多いときに最適化に関する問題を引き起こす可能性がある COBOL 項目の数にすぎません。 プログラムの複雑さを計測するには、IBM Developer for z/OS に備わっている測定基準機能のようなものを使用する必要があります。

巨大で複雑なプログラムの場合は、コンパイラーが最適化しようとするプログラムの複雑度に対してしきい値を MAXPCF オプションにより設定できます。 MAXPCF 値を下げると最適化のレベルが下がるため、コンパイラーに必要なメモリーおよびコンパイル時間は少なくなります。 コンパイル時間が長くなるのを承知の上でプログラムの最適化を試みるのであれば、MAXPCF 値を上げてください。

MAXPCF=0 を指定した場合は、 プログラムの複雑度に対して制限は課されず、MAXPCF オプションは何の効果もありません。

MAXPCF=*n* を指定し、*n* がゼロでない場合に、プログラム複雑度係数が *n* を超えると、OPTIMIZE(1) または OPTIMIZE(2) の指定はすべて OPTIMIZE(0) にリセットされ、警告メッセージが生成されます。

COBOL ソース・ファイルに一連のソース・プログラムが含まれている 場合 (バッチ・コンパイルの場合) は、MAXPCF 制限がプログラム単位で適用されます。

注 :

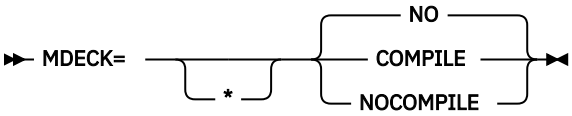
- OPT=1 オプションまたは OPT=2 オプションがインストール時にオーバーライド不可の固定オプションとして設定されている場合、MAXPCF=*n* (*n* はゼロ以外) は矛盾するオプションとなります。 この場合は、OPTIMIZE オプションが優先され、MAXPCF=0 オプションが強制的に適用されます。
- MAXPCF の値を *n* (*n* はデフォルトより大きい値) に上げることによって、または MAXPCF(0) を指定することによって、デフォルトしきい値より大きいプログラムを最適化しようとすると、コンパイラーでは、メモリー不足のために、成否にかかわらずコンパイルに過剰な時間がかかることがあります。

関連参照
52 ページの『OPTIMIZE』

MDECK

MDECK は、ライブラリー処理からの出力をファイルに書き込むかどうかに作用します。

構文



デフォルト
MDECK=NO

COMPILE
ライブラリーが処理され、MDECK 出力ファイルが生成された後、コンパイルは正常に継続します。

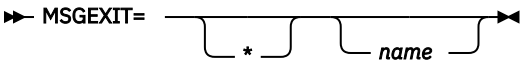
NOCOMPILE
ライブラリー処理が完了し、拡張ソース・プログラム・ファイルが書き込まれた後、コンパイルは終了します。

NO
MDECK 出力ファイルは作成されません。

MSGEXIT

MSGEXIT は、コンパイラー・メッセージのカスタマイズを可能にするために呼び出すモジュールを指定します。

構文



デフォルト

出口は指定されません。EXIT コンパイラー・オプションの NOMSGEXIT サブオプションを 指定するのと同等です。 MSGEXIT=* が *name* パラメーターなしでコーディングされた場合、NOMSGEXIT を オーバーライドすることはできません。

name

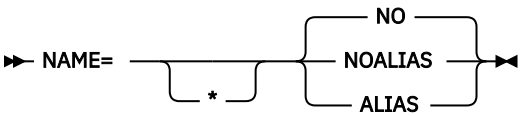
EXIT コンパイラー・オプションで使用するモジュールを識別します。このユーザー出口のサブオプション が指定されている場合、コンパイラーは指定されたモジュールをロード し、それを呼び出してコンパイラー・メッセージのカスタマイズを可能にします。 メッセージの重大度を変更でき、メッセージを抑制すること ができます。また、FLAGSTD コンパイラー・オプションを指定した場合に発生する FIPS メッセージ を診断メッセージに変換できます。

EXIT コンパイラー・オプションについて詳しくは、「*Enterprise COBOL* プログラミング・ガイド」で『*EXIT* コンパイラー・オプション』を参照してください。

NAME

NAME は、プログラム管理バインダー NAME ステートメントを各オブジェクト・モジュールに追加するかどうか、 および ENTRY ステートメントごとに ALIAS ステートメントを作成するかどうかに作用します。

構文



デフォルト

NAME=NO

ALIAS

これは、プログラム内の ENTRY ステートメントごとにプログラム管理バインダー ALIAS ステートメントを作成します。 PROGRAM-ID に対応する NAME ステートメントの前に ALIAS ステートメントが挿入されます。

NOALIAS

これは、バッチ・コンパイルで作成された各オブジェクト・モジュールに プログラム管理バインダー NAME ステートメント (NAME *modname*(R)) を付加します。 モジュール名 (*modname*) は、PROGRAM-ID から、外部モジュール名の形成に関する規則に従って導き出されます。

NO

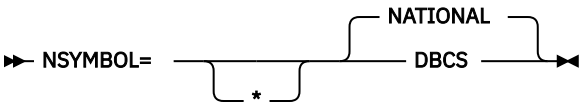
プログラム管理バインダー NAME ステートメントは付加されません。

NAME オプションを使用すると、単一のバッチ・コンパイルで、プログラム・ライブラリー内に複数のモジュールを作成することができ、動的呼び出しでの使用に役立ちます。

NSYMBOL

NSYMBOL は、PICTURE 節で使用される N 記号の解釈を制御し、国別処理または DBCS 処理のどちらを前提とするかを指示します。

構文



デフォルト
NSYMBOL=NATIONAL

DBCS
データ項目を PICTURE 記号 N のみで構成された PICTURE 節で定義し、USAGE 節を使用しない場合は、DBCS を使用します。このようなデータ項目は USAGE DISPLAY-1 節を指定した場合と同様に処理されます。形式が N"..." または N'...' のリテラルは、DBCS リテラルとして扱われます。

NATIONAL
データ項目を PICTURE 記号 N のみで構成された PICTURE 節で定義し、USAGE 節を使用しない場合は、NATIONAL を使用します。このようなデータ項目は USAGE NATIONAL 節を指定した場合と同様に処理されます。形式が N"..." または N'...' のリテラルは、国別リテラルとして扱われます。

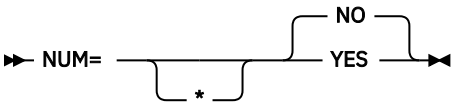
注 :

- NSYMBOL(DBCS) オプションは、IBM COBOL の前のリリースと互換性があります。
  NSYMBOL(NATIONAL) オプションは、N 記号を 2002 COBOL 標準に従って処理します。
- NSYMBOL(NATIONAL) を指定すると、DBCS オプションが強制的に実行されます。

NUM

NUM は、エラー・メッセージおよびプロシージャー・マップでソース・プログラム行番号を使用するかどうかに作用します。

構文



デフォルト
NUM=NO

YES
エラー・メッセージおよびプロシージャー・マップで、コンパイラーが生成した行番号ではなく、ソース・プログラムの行番号を使用します。

NO
エラー・メッセージおよびプロシージャー・マップで、コンパイラーが生成した行番号を使用します。

COBOL プログラマーは、COPY ステートメントを使用し、NUM=YES が 有効である場合には、ソース・プログラムの行番号と COPY メンバーの 行番号を調整する必要があります。

NUMCHECK

NUMCHECK コンパイラー・オプションは、データ項目が送信データ項目として使用されている場合にデータ項目を検証するために追加のコードを生成するかどうかをコンパイラーに指示します。 ゾーン 10 進数 (数値 USAGE DISPLAY) データ項目およびパック 10 進数 (COMP-3) データ項目の場合、コンパイラーは送信フィールドごとに暗黙的な数値クラス・テストを生成します。 数値の受信側に移動されているコンテンツの所有者である英数字の送信側については、コンパイラーはこの送信側を整数の数値として扱うため、NUMCHECK は英数字の送信側ごとに暗黙的な数値のクラス・テストを生成します。バイナリー・データ項

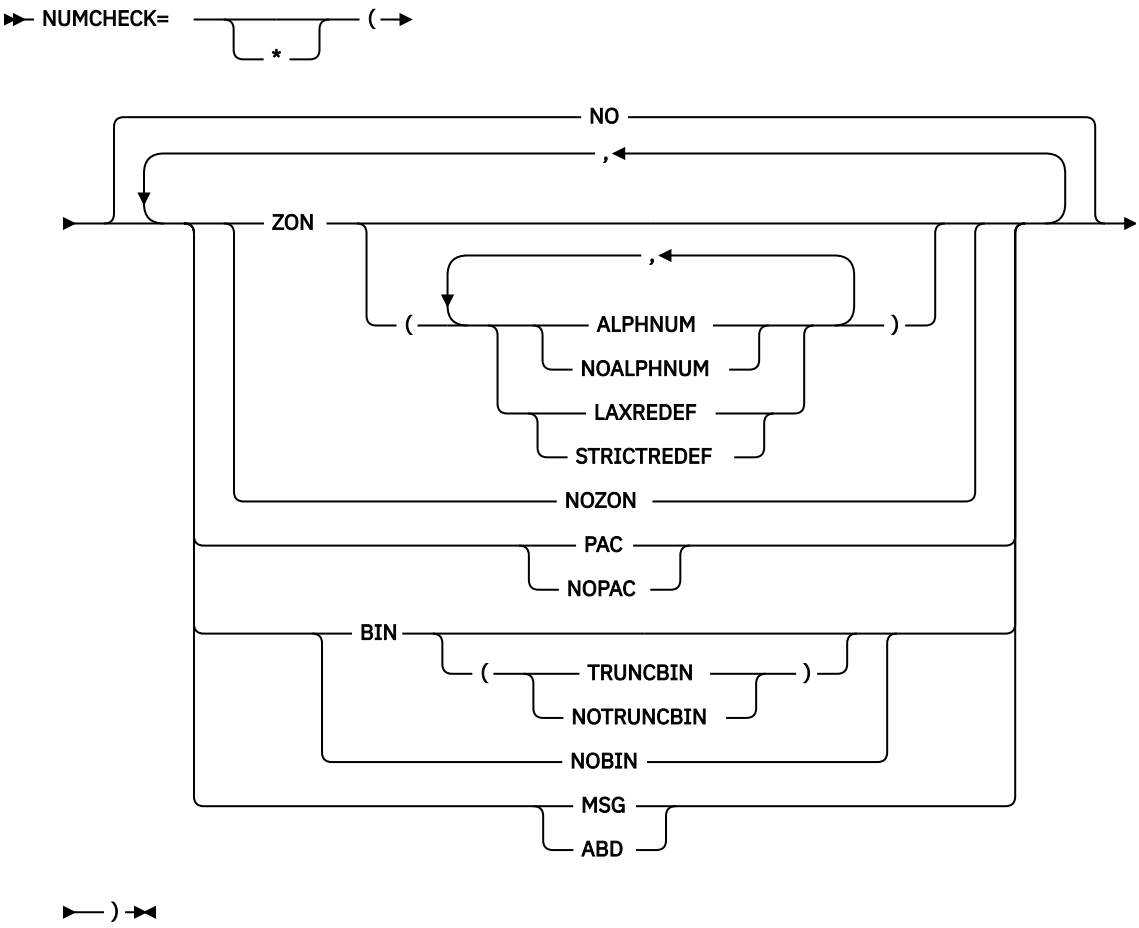
目の場合、コンパイラーは、データ項目の桁が、PICTURE 節で許可される桁を超えているかどうかを確認するために SIZE ERROR 検査を生成します。

NUMCHECK オプションが更新されました。 無効なデータに対する冗長性検査が除去されたため、ランタイム・パフォーマンスが向上しました。 以前よりもランタイム・メッセージが少なくなります。

冗長性検査を取り除くように行われた分析は、OPT(0) の場合よりも、OPT(1|2) で複雑になります。 OPT(0) では、より単純な形式の分析が行われるため、可能な限りコンパイル時間が短くなります。 より高い OPT レベルで、メッセージはより少なくなります。

コンパイル時に、検査によって常に無効データが見つかるものとコンパイラーが判断できる場合は、コンパイル時メッセージが生成され、実行時検査は除去されます。 (下の MSG|ABD を参照。)

構文



デフォルト

NUMCHECK=(NO)

サブオプションのデフォルトは以下のとおりです。

- サブオプションが指定されない場合のデフォルトは NUMCHECK=(ZON(ALPHNUM,STRICTREDEF),PAC,BIN(TRUNCBIN),MSG) です。
- データベース・サブオプションが指定されない場合のデフォルト・データ型サブオプションは ZON(ALPHNUM,STRICTREDEF)、PAC、および BIN(TRUNCBIN) です。例えば、NUMCHECK=(ABD) には NUMCHECK=(ZON(ALPHNUM,STRICTREDEF),PAC,BIN(TRUNCBIN),ABD) と同じ効果があります。
- データ型サブオプションが 1 つのみ指定されている場合のデフォルトは NOZON、NOPAC、NOBIN、および MSG です。例えば、NUMCHECK=(BIN) には NUMCHECK=(NOZON,NOPAC,BIN(TRUNCBIN),MSG) と同じ効果があります。

- NO が付いたデータ型サブオプションがすべて指定された場合、リストには NONUMCHECK が示されます。 例えば、NUMCHECK=(NOZON,NOPAC,NOBIN) には NUMCHECK=NO と同じ効果があります。

ZON(ALPHNUM|NOALPHNUM,LAXREDEF|STRICTREDEF) | NOZON

ZON(ALPHNUM) が指定されると、コンパイラーは、COBOL ステートメントで送信側データ項目として使用されるゾーン 10 進数 (数値 USAGE DISPLAY) データ項目の暗黙数値クラス・テスト用のコードを生成します。

ZON(NOALPHNUM) が指定されると、コンパイラーは、COBOL ステートメントで送信側データ項目として使用されるゾーン 10 進数 (数値 USAGE DISPLAY) データ項目に関する暗黙的な数値クラス・テスト用のコードを生成します (そのデータ項目が、英数字データ項目、英数字リテラル、または英数字形象定数との比較で使用されている場合を除く)。

以下のサンプル・ステートメント内のデータ項目 B のように、受信側が送信側と受信側の両方を兼ねていない限り、受信側はチェックされません。

```
ADD A TO B
```

```
DIVIDE A INTO B
```

```
COMPUTE B = A + B
```

```
INITIALIZE B REPLACING ALPHANUMERIC BY B
```

このチェックは、各ステートメント内でデータが使用される前に実行されます。

- データが NOT NUMERIC であれば、警告メッセージ (NUMCHECK=(ZON,MSG) の場合) または強制終了メッセージ (NUMCHECK=(ZON,ABD) の場合) が出されます。
- データが NUMERIC であれば、ステートメントの外部動作は、低速になる以外は NUMCHECK=(NOZON) と同じです。

ZON(LAXREDEF) を指定すると、ゾーン 10 進データ項目によって他のデータ項目が特定の方法で再定義される場合の、ゾーン 10 進データ項目内の無効データに対するコンパイラーの許容度が大きくなります。 コンパイラーによって検討される 2 つのケースを以下に示します。

- 符号なしのゾーン 10 進データ項目によって符号付きの末尾オーバーパンチのゾーン 10 進データ項目が再定義され、符号なしの項目の最終バイトが符号付きの項目の最終バイトとオーバーラップするようになっている。 このケースでは、NUMCHECK 検査のために、再定義する側である符号なしの項目が、符号付きのゾーン 10 進項目として扱われます。

  注:

  – 再定義される符号付きのゾーン 10 進項目は、level-01 項目または level-77 項目でなければなりません。 符号なしのゾーン 10 進項目は、level-01 項目か level-77 項目、あるいはグループ内の従属項目であることができます。

  – 符号なしのゾーン 10 進項目は、符号付きのゾーン 10 進項目全体とオーバーラップする必要はありません。 それぞれの項目の最終バイトのみがオーバーラップしている必要があります。 以下に例を示します。

  ```
  01 NUM1 PIC S9(8).
  01 NUM2 REDEFINES NUM1.
     03 NUM2-PART1 PIC 9(4).
     03 NUM2-PART2 PIC 9(2).
     03 NUM2-PART3 PIC 9(2).
  ```

  このケースでは、データ項目 NUM2-PART3 が、 NUMCHECK によって符号付きのゾーン 10 進データ項目として扱われます。 その最終バイトが、符号付きの末尾オーバーパンチのゾーン 10 進項目である NUM1 の最終バイトとオーバーラップしているためです。 したがって、以下の NUM2-PART3 の値はすべて有効と見なされます。

  - x'F1F2F3F4F5F6F7F8'
  - x'F1F2F3F4F5F6F7C8'

- x'F1F2F3F4F5F6F7D8'

.

- ゾーン 10 進データ項目により、(数字編集項目の PICTURE ストリングの Z 記号で示されるような) 先行スペースを含む可能性のある数字編集データ項目が再定義されていて、ゾーン 10 進データ項目の先行バイトが、数字編集項目の先行バイトの一部または全部と重なり合っている。 このケースでは、NUMCHECK は、スペースを許可する数字編集項目の先行バイトとオーバーラップする、 ゾーン 10 進データ項目の先行バイトに含まれるスペースを許容します。

  注:

  – 再定義される数字編集項目は、level-01 項目または level-77 項目でなければなりません。 ゾーン 10 進項目は、level-01 項目か level-77 項目、あるいはグループ内の従属項目であることができます。

  – ゾーン 10 進項目が符号付きである場合、それは符号付きの末尾オーバーパンチでなければなりません。

  – この処理に適格であると見なされるためには、ゾーン 10 進項目の最初のバイトが、数字編集項目の最初のバイトとオーバーラップしている必要があります。 ただし、ゾーン 10 進項目が数字編集項目全体とオーバーラップする必要はありません。 例えば、次のように指定します。

```
01 NUMED PIC ZZ99.99.
01 NUM REDEFINES NUMED.
   03 INTVAL PIC 9(4).
   03 FILLER PIC X.
   03 DECVAL PIC 9(2).
```

  このケースでは、NUMCHECK は、INTVAL の最初の 2 バイトに含まれるスペースを許容します。 それが、その PICTURE ストリング内の Z 記号で定義されている、NUMED の最初の 2 バイトとオーバーラップするためです。 したがって、以下の INTVAL の値はすべて有効と見なされます。

  - x'F1F2F3F4'

  - x'40F1F2F3'

  - x'4040F1F2'

  パフォーマンス上の理由から、先行バイト内ではスペースと非スペースの混在が許容されているため、'F140F1F2' も有効と見なされる、ということに注意してください。

ZON(STRICTREDEF) が指定されている場合、NUMCHECK は、 ゾーン 10 進データ項目によって再定義されている可能性のあるデータ項目を考慮せず、通常どおりゾーン 10 進データの厳格な検査が実行されます。

PAC | NOPAC

PAC が指定されると、コンパイラーは、COBOL ステートメントで送信データ項目として使用されるパック 10 進数 (COMP-3) データ項目に関する暗黙的な数値クラス・テスト用のコードを生成します。桁数が偶数であるパック 10 進数データ項目に関して、未使用ビットが 1 になっているかどうかが検査されます。

制約事項: CALL ステートメントに関しては、NUMCHECK=(ZON) および NUMCHECK=(PAC) によって、ゾーン 10 進数やパック 10 進数である BY CONTENT データ項目が検査されますが、BY REFERENCE パラメーターは検査されません。 (ゾーン 10 進数データ項目もパック 10 進数データ項目も BY VALUE 句では指定できません。)

BIN(TRUNCBIN|NOTRUNCBIN) | NOBIN

BIN が指定されると、コンパイラーは、バイナリー・データ項目が PICTURE 節の許可する桁数より多くの桁数を持っているかどうかをテストするために、ON SIZE ERROR に似たコードを生成します。この追加コードは、送信データ項目として使用されるバイナリー・データ項目に対してのみ生成されます。 COMP-5 データ項目の場合、この ON SIZE ERROR コードは生成されません。

BIN(TRUNCBIN) が有効になっている場合は、TRUNC=(BIN) コンパイラー・オプションが有効な場合でも、バイナリー・データ項目のための検査コードが生成されます。 BIN のサブオプションが指定されていない場合は、 BIN(TRUNCBIN) がデフォルトであることに注意してください。

BIN(NOTRUNCBIN) が有効になっている場合は、 TRUNC=(BIN) コンパイラー・オプションが有効でも、バイナリー項目のための検査コードは生成されません。

注: NUMCHECK=(..., BIN, ...) をデフォルト・オプションでの固定オプションにする一方で、TRUNC=(BIN) オプションを有効にしてコンパイルしたモジュールに対しては検査を行わないようにしたい場合は、BIN(NOTRUNCBIN) が役に立ちます。

MSG | ABD

これは、無効データに対して発行されるメッセージが警告レベル (処理は続行される) のメッセージなのか強制終了レベル (異常終了が引き起こされる) のメッセージなのかを判別します。

- MSG が有効になっている場合は、行番号、データ項目名、データ項目の内容、およびプログラム名が記されたランタイム警告メッセージが発行されます。
- ABD が有効になっている場合は、強制終了メッセージが発行され、異常終了が引き起こされます。

コンパイル時に、検査によって常に無効データが見つかるものとコンパイラーが判断できる場合は、コンパイル時エラー・レベル・メッセージが生成され、MSG または ABD が有効かどうかにかかわらず、検査は除去されます。

NO

データ項目が送信データ項目として使用されている場合、それらのデータ項目を検証するためのコードは生成されません。

パフォーマンスの考慮: COBOL プログラムで使用されるゾーン 10 進数 (数値 USAGE DISPLAY) データ項目、パック 10 進数 (COMP-3) データ項目、および 2 進データ項目の数によっては、NUMCHECK が指定されたときは、NUMCHECK=NO が指定されたときよりもはるかに時間がかかります。
COBOL V6.2 にサービスが適用されたため、NUMCHECK のパフォーマンスが向上しました。 ただし、NONUMCHECK を指定したときのパフォーマンスが最高であり、より高い OPT レベルで、ある程度向上します。

ZONECHECK は推奨されなくなり、IGYCDOPT に指定できなくなりました。 NUMCHECK=(ZON(ALPHNUM)) を使用すると、ZONECHECK と同じ結果を得られます。.

関連参照
50 ページの『NUMPROC』
68 ページの『TRUNC』
74 ページの『ZONECHECK』
36 ページの『INVDATA』

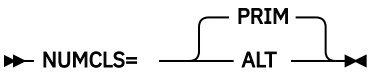
NUMCLS

NUMCLS は、NUMPROC とともに、コンパイラーが数値クラス・テストで有効として扱う数値符号に作用します。

NUMCLS は、以下のすべての条件を使用して定義されているデータ項目に対して数値クラス・テストで有効と認識される符号表記を指定します。

- 符号付きである (PICTURE 節で「S」によって指定)
- DISPLAY または COMPUTATIONAL-3 (パック 10 進) を使用する
- SIGN 節に SEPARATE 句を指定しない

構文



デフォルト
NUMCLS=PRIM

ALT
ALT を指定した処理では、16 進数 A から F が有効であるとして受諾されます。

PRIM
PRIM を指定した処理では、16 進数 C、D、および F が有効であるとして受諾されます。

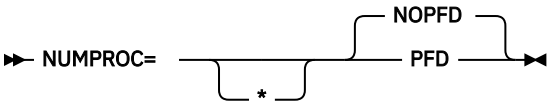
注 :

- NUMPROC オプションと NUMCLS オプションの指定内容が、数値クラス・テストに影響を 与えます。
- NUMCLS オプションは、NUMPROC=NOPFD の場合にのみ有効です。 NUMPROC=PFD は、有効な符号の構成に、より厳密な規則を指定します。

NUMPROC

NUMPROC は、内部 10 進数データおよびゾーン 10 進数データにおける符号の扱いや処理に作用します。

構文



デフォルト
NUMPROC=NOPFD

NOPFD
入力上の符号を修復します。修復が実行された後は、符号は NUMPROC=PFD に関する基準に適合します。

PFD
特に ゼロ以外の OPTIMIZE レベル (OPT=1 または OPT=2) が指定された場合に、生成コードが最適化されます。 明示的な符号の修復は実行されません。NUMPROC=PFD には、正しい結果を出すための厳密な基準があることに注意してください。NUMPROC=PFD を使用する場合は、以下の事項に従ってください。

- 符号なしの数値項目の符号桁は、X'F' にする必要があります。
- 符号付きの数値項目の符号桁は、正またはゼロの場合は X'C'、負の場合は X'D' にする必要があります。
- 別個に符号が付けられる数値項目の符号桁は、正またはゼロの場合は「+」、負の場合は「-」にする必要があります。

Enterprise COBOL における基本 MOVE ステートメントおよび算術ステートメント は、常にこれらの望ましい符号を伴う結果を生成しますが、グループ MOVE および再定義は、非準拠の結果を 生成することがあります。数値クラス・テストを検査のために使用することができます。NUMPROC=PFD を指定すると、符号が望ましい符号基準に適合しない場合は、数値項目の数値クラス・テストが失敗します。

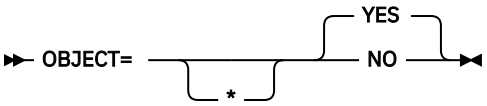
パフォーマンスの考慮: NUMPROC=PFD を使用すると、数値の比較に関してかなり効率的なコードが生成されます。COMP-3 および DISPLAY 数値データ項目を参照するほとんどの場合、NUMPROC=NOPFD を使用すると、符号の「修正」処理の ために余分なコードが生成されます。この余分なコードが原因で、他のいくつかのタイプの最適化が 禁止される場合もあります。このオプションを設定する前に、アプリケーション・プログラマーと相談して、アプリケーション・プログラムの出力に与える影響を判断してください。

NUMPROC と NUMCLS の両オプションは、数値クラス・テストに影響を与えます。 NUMPROC=NOPFD を指定した場合は、数値クラス・テストの 結果は、NUMCLS の設定によって制御されます。NUMPROC=PFD の場合、データ項目を望ましい符号基準と適合させて、数値と見なされるようにする必要があります。

OBJECT

OBJECT は、生成オブジェクト・コードをファイルに書き込むかどうかに作用します。

構文



デフォルト
OBJECT=YES

YES
生成されたオブジェクト・コードが、バインダーへの入力として使用されるファイル (SYSLIN DD ステートメントによって定義されたもの) に 配置されます。

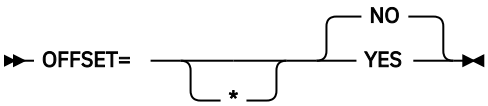
NO
オブジェクト・コードを SYSLIN に出力しません。

OBJECT=NO オプションは、TEST に指定された NO 以外のすべての値と矛盾します。

OFFSET

OFFSET は、圧縮された PROCEDURE DIVISION リストを生成するかどうかに作用します。

構文



デフォルト
OFFSET=NO

YES
圧縮された PROCEDURE DIVISION のリストを生成します。 これには、行番号とステートメント参照、およびステートメントに対して生成された各命令ブロックの位置が含まれます。 最適化プログラムは、段落をインライン化したりコードを移動したりすることがあります。実際に、最適化プログラムは、使用頻度が低いコード (エラー・メッセージ書式設定コードなど) をプログラム本体の後に置きます。その結果、 特定のステートメントの OFFSET テーブルに複数の項目が含まれる場合があります。

次の項目も出力リストに書き込まれます。

- 定数域
- プログラム・プロローグ域 (PPA1、PPA2、PPA3、PPA4)
- タイム・スタンプおよびコンパイラー・バージョン情報
- コンパイラー・オプションおよびプログラム情報
- ベース・ロケーター・テーブル
- 外部シンボル辞書
- 初期ヒープ・ストレージ・マップ
- スタック・ストレージ・マップ

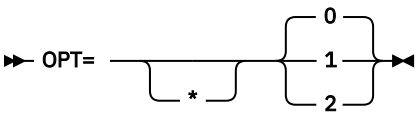
NO
リストを圧縮せず、上記のものを生成しません。

LIST と OFFSET コンパイラー・オプションは、相互に排他的です。OFFSET=YES と LIST=YES をともに指定すると、カスタマイズ・マクロのアセンブル試行時にゼロ以外の戻りコードが返されます。 競合解決について詳しくは、11 ページの『矛盾するコンパイラー・オプション』を参照してください。

OPTIMIZE

OPTIMIZE は、オブジェクト・コードに対して行われる最適化のレベルに作用し、その結果パフォーマンスが向上します。

構文



デフォルト
OPT=0

0
制限付きの最適化が指定されます。 結果として、コンパイル時間が最も短くなります。 TEST オプションを指定すると、全デバッグ機能が使用可能になります。

1
アプリケーション実行時パフォーマンスの向上につながる最適化が指定されます。 このレベルの最適化には、基本インライン化、強度の削減、複雑な演算から同等の単純な演算への単純化、一部の到達不能コードの除去、 ブロック再配置などがあります。 また、OPT=1 の場合は、共通する副次式の除去や値の伝搬など、いくつかのブロック内最適化が行われます。 TEST オプションを指定した場合は、大半のデバッグ機能を使用できます。

2
さらなる最適化が指定されます。 この最適化には、より積極的な単純化と命令スケジューリングが含まれます。 また、グローバル値の伝搬およびループ不変コードの動作など、いくつかのブロック間最適化も含まれます。 TEST オプションが指定されている場合は、いくつかのデバッグ機能が使用可能になります。

パフォーマンスの考慮: 通常、OPT=1 または OPT=2 を使用すると、より効率的なランタイム・コードが生成されます。

注 :

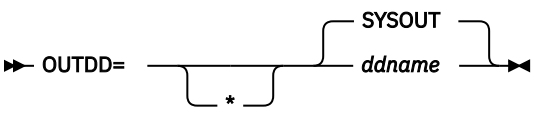
- OPTIMIZE コンパイラー・オプションは、Java インターオペラビリティー用の オブジェクト指向構文を使用するプログラムに対して完全にサポートされています。
- S レベル・エラーまたは U レベル・エラーが発生した場合、 あるいはプログラム複雑度係数が指定の MAXPCF 整数を超えた場合は、 最適化は 0 に設定されます。

詳しくは、「*Enterprise COBOL プログラミング・ガイド*」で『*OPTIMIZE*』を参照してください。

OUTDD

OUTDD は、DISPLAY 出力の送信先となる ddname を指定します。

構文



デフォルト
OUTDD=SYSOUT

ddname
実行時の DISPLAY 出力用に使用されるファイルの ddname を指定します。

実行時に、ddname として SYSOUT を必要とする別の製品と対立することが 予測される場合は、このオプションのデフォルトを変更してください。

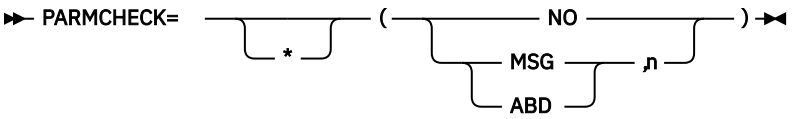
OUTDD が MSGFILE ランタイム・オプションとどのように相互作用するかについては、「*z/OS* 言語環境プログラム プログラミング・リファレンス」の MSGFILE の説明を参照してください。

PARMCHECK

PARMCHECK オプションは、WORKING-STORAGE における最後の項目の後に追加データ項目を生成するようにコンパイラーに指示します。 次にこのバッファー・データ項目は、呼び出されたサブプログラムが WORKING-STORAGE の端を超えてデータを破壊していないかどうかを検査するために実行時に使用されます。

呼び出し側プログラムが PARMCHECK でコンパイルされている場合、コンパイラーは、WORKING-STORAGE セクションにおいて最後にあるデータ項目の後にバッファーを生成します。 実行時に、各呼び出しの前にバッファーが ALL x'AA' に設定されます。 各呼び出しの後は、バッファーが検査されて、バッファーが変更されたかどうかが確認されます。 PARMCHECK オプションは、COBOL V4 以前のコンパイラーから COBOL V6 以降のコンパイラーに移行するときに役立つ可能性があります。 また、このオプションは、不正な COBOL データをクリーンアップしたり適正なプログラミング基準を検査したりするためにも使用できます。

構文



デフォルト
PARMCHECK=(NO)

MSG | ABD

これは、サブプログラムによるデータ破壊に対して発行されるメッセージが警告レベル (処理は続行される) のメッセージなのか強制終了レベル (異常終了が引き起こされる) のメッセージなのかを判別します。

- MSG が有効になっている場合は、パラメーターの名前、CALL ステートメントの行番号、およびプログラム名が記されたランタイム警告メッセージが発行されます。
- ABD が有効になっている場合は、同様のメッセージが発行されますが、そのメッセージは異常終了を引き起こす強制終了レベルのメッセージです。

n
WORKING-STORAGE における最後の項目の後に追加されるバッファーのサイズ (バイト)。 1 から 9999 までの範囲にある整数でなければなりません。

NO
WORKING-STORAGE にある最後の項目の後にバッファー・データ項目は生成されません。

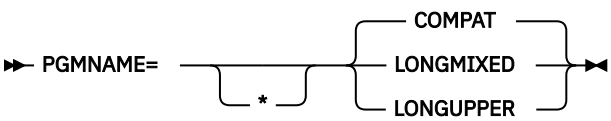
パフォーマンスの考慮事項: PARMCHECK が指定されると、コンパイラーは、CALL ステートメントを持つプログラムに対して低速のコードを生成します。 良好なパフォーマンスを得るには、NOPARMCHECK を指定する必要があります。

関連参照
CALL ステートメント (*Enterprise COBOL* 言語解説書)

PGMNAME

PGMNAME は、プログラム名および入り口点名の処理を制御します。

構文



デフォルト
PGMNAME=COMPAT

COMPAT
プログラム名は、COBOL/370 リリース 1 および VS COBOL II と 互換性のある方法で処理されます。

LONGMIXED
プログラム名は、切り捨てられたり、変換されたり、または大文字に変換されることなく、 現状のまま処理されます。

LONGUPPER
プログラム名はコンパイラーによって大文字に変換されるか、 あるいは、切り捨てられたり変換されることなく、現状のまま処理されます。

PGMNAME オプションは、以下のコンテキストで使用された名前の処理を 制御します。

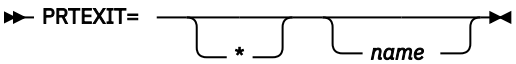
- PROGRAM-ID パラグラフで定義されたプログラム名
- ENTRY ステートメントのプログラム入り口点の名前
- 以下におけるプログラム名参照:
  - ネストされたプログラム、静的にリンクされたプログラム、または DLL を参照する、CALL ステートメント
  - 静的にリンクされたプログラムまたは DLL を参照する SET *procedure-pointer* または *function-pointer* ステートメント
  - ネストされたプログラムを参照する CANCEL ステートメント

詳しくは、「*Enterprise COBOL* プログラミング・ガイド」で『*PGMNAME*』を参照してください。

PRTEXIT

PRTEXIT は、SYSPRINT データ・セットに書き込む出力の代わりに、呼び出すモジュールを指定します。

構文



デフォルト
出口は指定されません。EXIT コンパイラー・オプションの NOPRTEXIT サブオプションを 指定するのと同等です。 PRTEXIT=* が *name* パラメーターなしでコーディングされた場合、NOPRTEXIT を オーバーライドすることはできません。

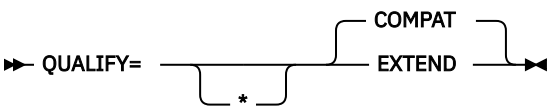
name
EXIT コンパイラー・オプションで使用するモジュールを識別します。このユーザー出口の サブオプションが指定されている場合、コンパイラーは SYSPRINT データ・セットに書き込む代わりに、 指定されたモジュールをロードし、それを呼び出します。 このオプションを指定した場合は、SYSPRINT データ・セットはオープンされません。

EXIT コンパイラー・オプションについて詳しくは、「*Enterprise COBOL* プログラミング・ガイド」で『*EXIT* コンパイラー・オプション』を参照してください。

QUALIFY

QUALIFY は、修飾の規則に作用し、COBOL 標準規則の下で参照できない一部のデータ項目を参照できるように修飾規則を拡張するかどうかを制御します。

構文



デフォルト
QUALIFY=COMPAT

COMPAT
QUALIFY=COMPAT が有効になっている場合、動作は以前の COBOL コンパイラーと同じです。 完全に一致する当該修飾子のセットを持つデータ項目が 1 つしか存在しない場合でも、参照は固有でなければなりません。

EXTEND
QUALIFY=EXTEND が有効になっている場合は、COBOL 標準規則では固有でない一部の参照を固有にできるように、修飾規則が拡張されます。 名前のグループを含む階層内のレベルがすべて修飾されている場合、その修飾子のセットは修飾子の完全セット と呼ばれます。 特定の修飾子の完全セットを持つデータ項目が 1 つしかない場合、参照はそのデータ項目に解決されます。 これは、同じ修飾子のセットが、修飾子の不完全セットとして別の参照に一致する場合も同様です。

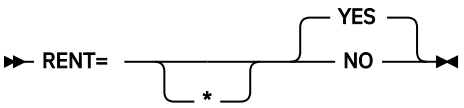
例

```
01 A.
  02 B.
    03 C PIC X.
  02 C PIC X.
.
.
.
Move space to C of A          *> Refers to 02 level C (unique only with QUALIFY(EXTEND))
Move space to C of B of A     *> Refers to 03 level C (unique by COBOL standard rules)
Move space to C of B          *> Refers to 03 level C (unique by COBOL standard rules)
```

RENT

RENT は、生成されるオブジェクト・コードが再入可能かどうかに作用します。

構文



デフォルト
RENT=YES

YES
生成されるオブジェクト・コードを再入可能にすることを指示します。 RENT=YES を使用すると、プログラムを 16 MB 境界より上で 実行するために、共有ストレージに置くことができます。しかし、このオプションを使用すると、コンパイラーはアプリケーション・ プログラムを再入可能にするための追加のコードを生成することになります。

NO
生成されるオブジェクト・コードを再入可能にしないことを指示します。

注 :

- プログラムが 16 MB 境界より上の仮想記憶アドレスで実行される場合は、 そのプログラムを、RENT を指定してコンパイルします。
- 16 MB より上で再入不可プログラムを実行することはサポートされていません。 NORENT を指定してコンパイルされたプログラムは RMODE 24 でなければなりません。
- CICS の下で実行されるプログラムの場合、RENT コンパイラー・オプションが必要です。
- LP(64) コンパイラー・オプションは RENT を暗黙指定します。 ユーザーが明示的に NORENT を指定すると、通知メッセージが出されてその設定は無視されます。
- プログラムに割り当てられる RMODE は、RENT|NORENT コンパイラー・オプションおよび RMODE コンパイラー・オプションによって決まります。 下の表に、有効な組み合わせを示します。

表 6. *RENT* および *RMODE* が常駐モードに与える影響

RENT|NORENT 設定	**RMODE 設定**	**割り当てられる 常駐モード**
RENT	AUTO	RMODE ANY
RENT	ANY	RMODE ANY
RENT	24	RMODE 24
NORENT	AUTO	RMODE 24
NORENT	ANY	コンパイラー・オプションの競合
NORENT	24	RMODE 24

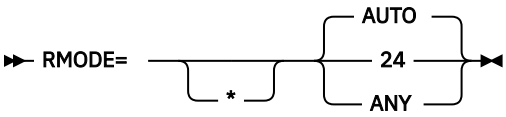
- THREAD コンパイラー・オプションを指定する場合、RENT コンパイラー・オプション も指定する必要があります。THREAD と NORENT を同じレベルの優先順位で指定した場合、RENT オプションが強制的に実行されます。

詳しくは、「*Enterprise COBOL* プログラミング・ガイド」の『*RENT*』を参照してください。

RMODE

RMODE は、生成されるオブジェクト・プログラムの常駐モードに作用します。

構文



デフォルト
RMODE=AUTO

24
NORENT または RENT のいずれが指定されているかには関係なく、プログラムが RMODE 24 となることを指定します。

ANY
これは、RENT が指定されている場合はプログラムで RMODE ANY が使用されるように指定し、NORENT が指定されている場合は プログラムがエラーを受け取るように指定します。

AUTO
プログラムが、NORENT が指定されている場合は RMODE 24 になり、RENT が指定されている場合は RMODE ANY になることを指定します。

注 :

- AMODE 24 で実行中のプログラムにデータを渡す Enterprise COBOL NORENT プログラム は、RMODE (24) オプションを指定してコンパイルするか、RMODE 24 を指定してリンク・エディットする必要があります。NORENT プログラムのデータ域が 16 MB 境界の上下どちらに置かれるかは、プログラムの RMODE によって異なります。 これは DATA(24) が指定されている場合でも同様です。 DATA(24) は、RENT オプションでコンパイルしたプログラムにのみ適用されます。
- Enterprise COBOL でコンパイルしたプログラムは、必ず AMODE ANY です。プログラムに割り当てられる RMODE は、 RMODE および RENT|NORENT コンパイラー・オプションによって決まります。 有効な組み合わせを下表に示します。

表 7. *RMODE* および *RENT|NORENT* が常駐モードに与える影響

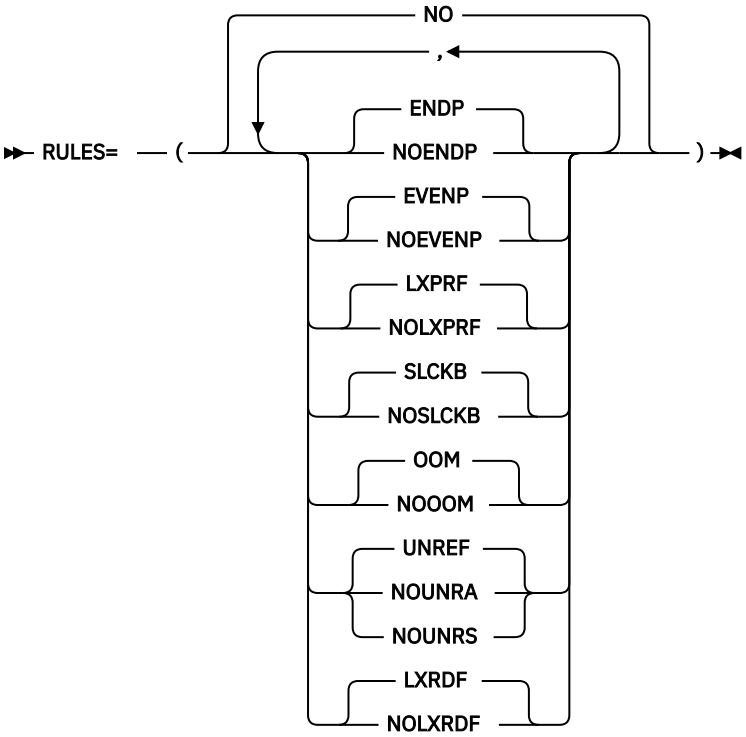
RMODE 設定	**RENT|NORENT 設定**	**割り当てられる常駐モード**
AUTO	RENT	RMODE ANY
AUTO	NORENT	RMODE 24
ANY	RENT	RMODE ANY
ANY	NORENT	適用されない。 コンパイラー・エラーが発行される。
24	RENT	RMODE 24
24	NORENT	RMODE 24

- LP(64) コンパイラー・オプションは RMODE(ANY) を暗黙指定します。 ユーザーが明示的に RMODE(24) を指定した場合は、通知メッセージが出され、設定は無視されます。

RULES

RULES オプションを使用すると、コンパイル時に特定タイプのソース・コードにフラグを立てることにより、プログラムを改善するためにプログラムに関する情報をコンパイラーから要求できます。

構文



デフォルト

RULES = (NO)

RULES=(ENDP,EVENP,LXPRF,SLCKB,OOM,UNREF,LXRDF) と同じ効果を持ちます。

RULES の以下のサブオプションを指定できます (デフォルトを除く)。

(NOENDP)

条件ステートメントの有効範囲が明示範囲終了符号 END-* ではなくピリオドで終了している場合に、コンパイラーから警告メッセージを出します。

(NOEVENP)
コンパイラーは、桁数が偶数であるすべての USAGE PACKED-DECIMAL (COMP-3) データ項目に対して警告メッセージを発行します (そのデータ項目が持つ未使用ビットがゼロ以外の場合に、プログラムの動作が予期しないものになる可能性があるため)。

注 :

- RULES=(NOEVENP) は、未使用の余分なスペースが自らのために予約されている USAGE PACKED-DECIMAL (COMP-3) データ項目を識別できるように支援します。 ただし、このデータ項目が奇数の桁を持つように変更する必要はありません。 そうしたとしても、プログラミングがほんのわずかに良くなるだけです。
- コンパイラーは、偶数桁の PACKED-DECIMAL データ項目に関して、その名前が DFH、DSN、EYU、または SQL で始まる場合はメッセージを発行しません。 つまり、このようなデータ項目は、CICS および Db2 のために生成されたり CICS および Db2 によって生成されたりするデータ項目です。

(NOLXPRF)
効果のない COBOL 機能の使用に対して、コンパイラーから警告メッセージを出します。 このような機能には、算術ステートメントでの USAGE DISPLAY 数値データ項目、MOVE ステートメントでの大量スペース埋め込み、効果のないコンパイラー・オプションなどがあります。

(NOSLCKB)
コンパイラーが遊びバイト (レコード内の遊びバイトまたはレコード間の遊びバイト) を追加することになるすべての SYNCHRONIZED データ項目に対して、コンパイラーから警告メッセージを出します。 遊びバイトの追加を発生させるデータ項目はそれぞれ、コンパイラー診断を受け取ります。

(NOOOM)
コンパイラーは、*integer-1* (最小出現回数) なしで指定されているすべての OCCURS DEPENDING ON 節に関して警告メッセージを発行します。

(NOUNRA)
NOUNRA が指定された場合は、データ項目の定義がユーザー・ソース・プログラムに直接指定されているのかコピー・メンバーのプログラムに含まれているのかに関係なく、FILE SECTION、WORKING-STORAGE SECTION、LOCAL-STORAGE SECTION、および LINKAGE SECTION において参照されない level-01 データ項目および level-77 データ項目 (項目がグループのときに参照される従属項目は含まれない) がすべて報告されます。

(NOUNRS)
NOUNRS が指定されたときは、データ項目の定義がユーザー・ソース・プログラムに直接指定されている場合に限り、FILE SECTION、WORKING-STORAGE SECTION、LOCAL-STORAGE SECTION、および LINKAGE SECTION において参照されない level-01 データ項目および level-77 データ項目 (項目がグループのときに参照される従属項目は含まれない) がすべて報告されます。

注 :

- COBOL では、単一グループ項目の定義が複数のファイルに及ぶ可能性があります。 このようなことが起こる場合、およびグループの level-01 データ項目の定義がメイン・ソース・ファイルに指定されている場合に、NOUNRS が有効になっていると、参照されないデータ項目が報告されます。
- NOUNRA または NOUNRS が有効になっている場合、名前の接頭部が DFH、DSN、EYU、または SQL となっているデータ項目 (すなわち、CICS および Db2 のために生成されたり CICS および Db2 によって生成されたりするデータ項目) は報告されません。

(NOLXRDF)
NOLXRDF が指定されている場合、任意のレベル (level-01 を含む) でデータ項目がさらに小さい項目に再定義される際に、コンパイラーが警告メッセージを出します。

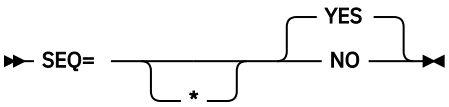
注 :

- RULES のすべてのサブオプションを指定する必要はありません。 サブオプションを指定しない場合は、デフォルトが有効になります。
- RULES は、インストール・デフォルトの 1 つ以上のサブオプションとともに指定する必要があります。

SEQ

SEQ は、ソース・ステートメントがシーケンス番号の昇順になっていることをコンパイラーが検査するかどうかに作用します。

構文



デフォルト
SEQ=YES

YES
ソース・ステートメントが行番号の英数字順 (昇順) になっているかどうかを検査します。

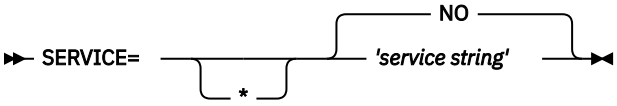
NO
順序検査を行いません。

コンパイル時に SEQ と NUM の両方が有効である場合は、順序は、英数字ではなく数字の照合順序に従って検査されます。

SERVICE

SERVICE は、オブジェクト・モジュールが生成された場合に、オブジェクト・モジュール内にストリングを配置します。 オブジェクト・モジュールがプログラム・オブジェクトにリンクされている場合、ストリングはこのプログラム・オブジェクトとともにメモリーにロードされます。 Language Environment ダンプにトレースバックが含まれている場合は、このストリングがそのトレースバックに組み込まれます。

構文



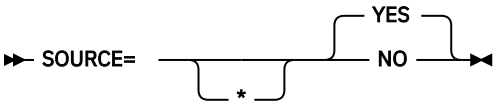
デフォルト
SERVICE=NO

service string の長さは 64 文字に制限されています。

SOURCE

SOURCE は、コンパイラー・リストにソース・ステートメントを含めるかどうかに作用します。

構文



デフォルト
SOURCE=YES

YES
コンパイラー生成の出力に、ソース・ステートメントのリストを 入れると指定します。このリストには、COPY によって組み込まれたステートメントも含まれます。SOURCE=YES を指定すると、SOURCE(DEC) コンパイラー・オプションが有効になります。

注 : SOURCE(HEX) は、コンパイラー呼び出しオプションとしてか、 COBOL ソース・プログラムの PROCESS ステートメントまたは CBL ステートメント内で指定できますが、 SOURCE=HEX をインストールのデフォルトとして指定することはできません。

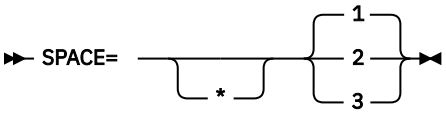
NO
出力にソース・ステートメントを入れません。

メッセージをソース・リストに出力したい場合は、コンパイル時に SOURCE コンパイラー・オプションを有効にしておく必要があります。

SPACE

SPACE は、ソース・リストで 1 行送り、2 行送り、または 3 行送りのいずれを使用するかに作用します。

構文



デフォルト
SPACE=1

1
ソース・ステートメント・リストを 1 行送りにすることを指定します。

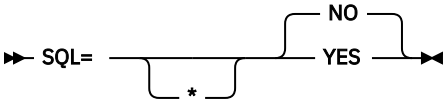
2
ソース・ステートメント・リストを 2 行送りにすることを指定します。

3
ソース・ステートメント・リストを 3 行送りにすることを指定します。

SQL

SQL は、Db2 コプロセッサーを使用可能にするかどうか、および Db2 オプションを指定可能にするかどうかに作用します。

構文



デフォルト
SQL=NO

YES
Db2 コプロセッサーを使用可能にし、Db2 オプションを指定する場合に使用します。 LP(64) コンパイラー・オプションが有効な場合、COBOL ソース・プログラムに SQL ステートメントが含まれていて、Db2 プリコンパイラーが LP(64) でサポートされていなければ、SQL オプションを指定する必要があります。

Db2 コプロセッサーは、データベース要求モジュール (DBRM) を DD 名 DBRMLIB に書き込みます。

NO
ソース・プログラムにある SQL ステートメントを識別して廃棄することを指定します。

SQL=NO は、COBOL ソース・プログラムに SQL ステートメントが含まれていない場合、または COBOL コンパイラーを呼び出す前に別の SQL プリコンパイラーを使用して SQL ステートメントを処理する場合に使用してください。

注 :

- SQL オプションは、どのコンパイラー・オプション・ソースでも指定できます。すなわち、コンパイラー呼び出し、PROCESS ステートメントまたは CBL ステートメント、OPTFILE、またはインストール・デフォルトのどれにでも指定できます。
- Db2 オプションのストリングを区切るには、引用符またはアポストロフィを使用してください。

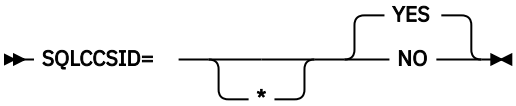
- Db2 オプションは、SQL オプションのカスタマイズの一環として指定することはできません。 (Db2 オプションは、SQL コンパイラー・オプションが呼び出しオプションとして指定される場合、あるいは CBL ステートメントまたは PROCESS ステートメントで指定される場合にのみサポートされます)。 ただし、Db2 製品のインストール・デフォルトをカスタマイズするときには、デフォルトの Db2 オプションを指定することができます。

関連タスク
SQL オプションを使用したコンパイル
(*Enterprise COBOL プログラミング・ガイド*)

SQLCCSID

SQLCCSID は、SQL コンパイラー・オプションが有効な場合に、CODEPAGE コンパイラー・オプションが SQL ステートメントの処理に影響を与えるかどうかを制御します。

構文



デフォルト
SQLCCSID=YES

YES
統合された Db2 コプロセッサー (SQL コンパイラー・オプション) が使用された場合に、CODEPAGE コンパイラー・オプション設定がソース・プログラム内の SQL ステートメントの処理に影響を及ぼすことを示します。

NO
CODEPAGE コンパイラー・オプション設定が、ストリング・リテラルのエンコード方式として、および変換済み SQL ステートメントが含まれる COBOL アプリケーション・ソースのエンコード方式としてのみ使用されることを示します。 Db2 (文字ストリング) ホスト変数は、CODEPAGE コンパイラー・オプションの影響を受けません。 代わりに、Db2 (文字ストリング) ホスト変数のエンコード方式は、DSNHDECP ファイルにある CCSID 値からもたらされます。 つまり、Db2 は、Db2 データ (ホスト変数) のエンコード方式を DSNHDECP ファイルで決定します。

注 :

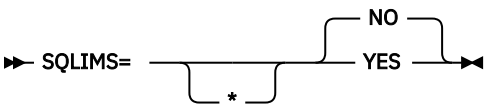
- SQLCCSID オプションは、LP(64) コンパイラー・オプションが有効な場合にサポートされます。 このオプションは LP(32) の場合と同じように動作します。
- SQLCCSID オプションは、統合された Db2 コプロセッサーを使用する場合にのみ効力を持ちます (SQL コンパイラー・オプション)。
- Db2 プリコンパイラーの動作との最高度の互換性を必要とするアプリケーションの場合、NOSQLCCSID オプションを推奨します。

関連参照
SQLCCSID (*Enterprise COBOL プログラミング・ガイド*)

SQLIMS

SQLIMS は、IMS SQL コプロセッサーを使用可能にするかどうか、および情報管理システム (IMS) サブオプションを指定できるかどうかに作用します。

構文



デフォルト
SQLIMS=NO

YES
これは、IMS SQL コプロセッサーを使用可能にして情報管理システム (IMS) サブオプションを指定する場合に使用します。COBOL ソース・プログラムに SQLIMS ステートメントが含まれている場合は、SQLIMS オプションを指定する必要があります。

NO
これは、ソース・プログラムにある SQLIMS ステートメントを診断して廃棄する場合に指定します。

COBOL ソース・プログラムに SQLIMS ステートメントが含まれていない場合は、SQLIMS=NO を使用します。

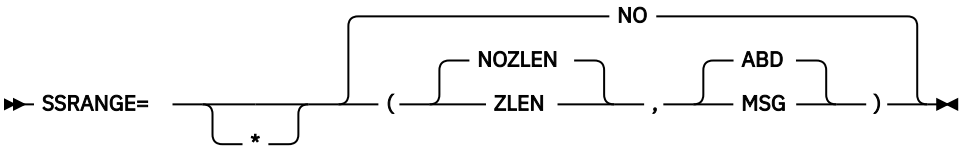
注 :

- SQLIMS オプションは、任意のコンパイラー・オプション・ソース (コンパイラー呼び出し、PROCESS/CBL ステートメント、またはインストール・デフォルト) で 指定できます。
- IMS サブオプションのストリングを区切るには、引用符またはアポストロフィを使用します。
- 長いサブオプション・ストリングを複数のサブオプション・ストリングに分割して、複数の CBL ステートメントにすることができます。
- LP(64) コンパイラー・オプションが有効な場合、SQLIMS オプションはサポートされません。 このオプションがユーザーによって明示的に指定されている場合は、診断メッセージが出されます。

SSRANGE

SSRANGE は、範囲外のストレージ参照がないかを検査するためのコードを生成するかどうかに作用します。

構文



デフォルト
SSRANGE=NO

ZLEN | NOZLEN
割り当てられた区域外のストレージを参照しないようにするために、添え字、参照変更、可変長グループ範囲、および指標を実行時に検査するコードを生成します。 生成されたコードは、関数の引数として指定された、ALL 添え字付けを指定した テーブルに、少なくとも 1 回の出現が含まれているかどうかも検査します。

生成されたコードはまた、OCCURS DEPENDING ON オブジェクトの正しくない設定の結果として、定義された最大長を可変長項目が超えていないかどうかも検査します。 無制限グループまたはその従属項目の場合、検査は参照変更式に対してのみ行われます。 無制限グループに従属するテーブルに対する添字付きまたは索引付きの参照は、検査されません。

ZLEN サブオプションおよび NOZLEN サブオプションは、コンパイラーによる参照変更長の検査方法を制御します。

- ZLEN が有効になっている場合、コンパイラーは、参照変更長がゼロ以上になるようにコードを生成します。 ゼロ長の参照変更が指定されると、実行時に SSRANGE エラーが出力されません。
- NOZLEN が有効になっている場合、コンパイラーは、参照変更長が 1 以上になるようにコードを生成します。 ゼロ長の参照変更が指定されると、実行時に SSRANGE エラーが出力されます。 これは、旧バージョンの COBOL での SSRANGE の動作方法と互換性があります。

MSG | ABD
サブオプション MSG および ABD は、範囲検査が失敗したときの COBOL プログラムの実行時動作を制御します。

- MSG が有効になっているときに範囲検査が失敗すると、ランタイム警告メッセージが発行されます。 また、プログラムの実行は継続され、他の範囲外状態も特定される可能性があります。
- ABD が有効になっているときに範囲検査が失敗すると、最初の範囲外状態でランタイム・エラー・メッセージが発行され、プログラムが異常終了します。 その次の潜在的な範囲外状態を見つけるには、最初の範囲外状態を修正し、プログラムを再コンパイルして再実行します。 他のすべての潜在的な範囲外状態を特定するには、このプロセスを何度も繰り返さなければならない可能性があります。

パフォーマンスの考慮: コンパイル時に SSRANGE=NO 以外のオプションが有効になっていると、オブジェクト・コード・サイズが大きくなり、範囲検査を実行する際のランタイム・オーバーヘッドも大きくなります。

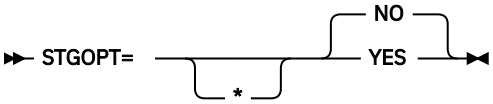
NO
添え字または指標を実行時に検査するためのコードを生成しません。

注 : SSRANGE=NO 以外のオプションを指定すると、コンパイラーによって範囲検査が生成され、実行時に必ず検査が実行されます。 Language Environment のランタイム・オプション CHECK(OFF) を指定しても、コンパイルされた範囲検査を無効にすることはできません。

STGOPT

STGOPT オプションは、ストレージ最適化を制御します。

構文



デフォルト
STGOPT=NO

YES

STGOPT=YES を指定した場合、コンパイラーは以下のデータ項目の一部または全部を破棄する可能性があり、 そのデータ項目に対してストレージを割り振りません。

- 未参照 LOCAL-STORAGE および WORKING-STORAGE のレベル 77 およびレベル 01 の基本データ項目
- レベル 01 グループ項目 (どの従属項目も参照されない場合)
- 未参照特殊レジスター

注 : STGOPT オプションは、VOLATILE 節のあるデータ項目では無視されます。 詳しくは、「*Enterprise COBOL* 言語解説書」で『VOLATILE 節』を参照してください。

コンパイラーは、これらの破棄されたデータ項目をその VALUE 節の値に初期化するコードを生成しません。

また、STGOPT=YES の場合は、パフォーマンスを最適化するためにメモリーにおいて LOCAL-STORAGE SECTION 内のデータ項目を再配列できます。

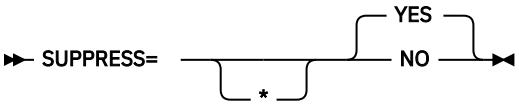
NO
STGOPT=NO を指定すると、すべてのデータ項目 (参照されないデータ項目を含む) のストレージはコンパイラーによって割り振られ、データ項目がパフォーマンス向上のために再配列されることはなく、VALUE 節で定義されたデータ項目はいずれも、たとえ参照されることがなくても必ず初期化されます。

また、RULES=(UNREF | NOUNRA | NOUNRS) オプションを使用すれば、未参照データ項目について警告メッセージを発行するかどうかを制御することもできます。 詳細については、57 ページの『RULES』を参照してください。

SUPPRESS

NOSUPPRESS オプションは、コピーブック情報をリストに表示できるように、プログラム内のすべての COPY ステートメントの SUPPRESS 句を無視する場合に使用します。 コピーブック情報はデバッガーやツールなどで使用でき、その場合それらのソース・コードを変更する必要はありません。

構文



デフォルト
SUPPRESS=YES

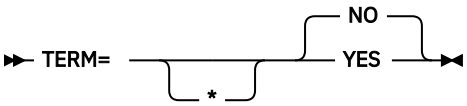
YES
COPY ステートメントの SUPPRESS 句を有効にします。

NO
COPY ステートメントの SUPPRESS 句を無視します。

TERM

TERM は、進行メッセージと診断メッセージを SYSTERM デバイスに送るかどうかに作用します。

構文



デフォルト
TERM=NO

YES
これは、進行メッセージと診断メッセージを SYSTERM ファイルに送信することを指定します。 これは、特に指定がなければデフォルトでユーザーの端末となります。

NO
メッセージを SYSTERM ファイルに送らないことを指定します。

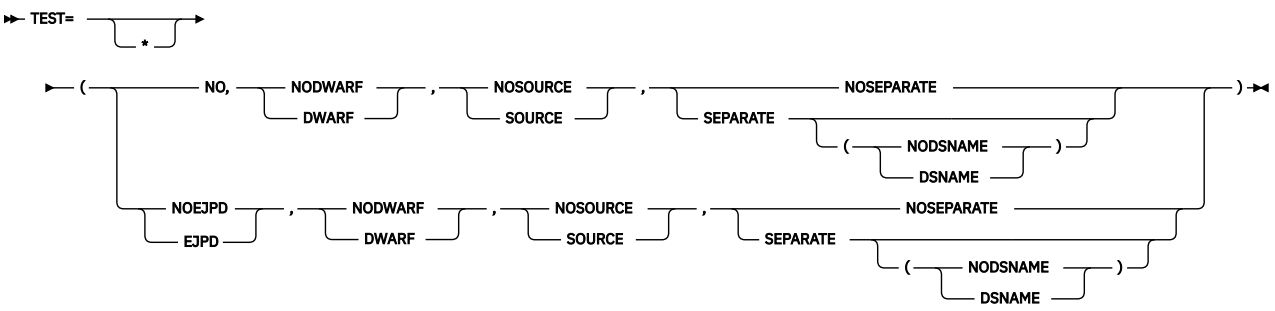
ソース・プログラム内に TERM を指定する場合は、アプリケーション・プログラムごとに SYSTERM DD ステートメントを指定する必要が あります。

TERM は TERMINAL コンパイラー・オプションに対応します。

TEST

TEST は、オブジェクト・コードに作成されるデバッグ情報の量に作用します。 これにより、使用可能なデバッグ・サポートのレベルが決定されます。

構文



注 : このオプションを JCL や CBL/PROCESS で指定する場合とは異なり、サブオプションはすべて必要であり、構文図に示されている順序になっていなければなりません。

デフォルト
TEST=(NO,NODWARF,NOSOURCE,NOSEPARATE)

(NO,DWARF,...)

TEST=(NO,DWARF,...) が有効になっている場合は、基本 DWARF 診断情報がオブジェクト・プログラムに組み込まれ、 SEPARATE も有効になっている場合は別のデバッグ・ファイルに組み込まれます。このオプションにより、アプリケーション障害分析ツール (CEEDUMP や IBM Fault Analyzer など) が使用可能になります。TEST=(NO,DWARF,...) の場合、デバッグ情報は TEST=(DWARF,...) によって入手できる DWARF 情報のサブセットです。TEST=(NO,DWARF,...) が有効になっている場合に生成される DWARF 診断情報は、IBM z/OS Debugger では使用できません。 デバッガーを使用する必要がなく、TEST オプションのパフォーマンスへの影響を回避したいが、一方で CEEDUMP や IBM Fault Analyzer などのアプリケーション障害分析ツールのユーザビリティーが向上している場合は、TEST=(NO,DWARF,...) の使用を検討してください。

コンパイラーによって生成されたデバッグ情報は、業界標準の DWARF 形式です。 DWARF について詳しくは、「*DWARF/ELF エクステンション ライブラリー・リファレンス*」の『*About Common Debug Architecture*』を参照してください。

(NO,NODWARF,...)
TEST=(NO,NODWARF,...) が有効になっている場合、DWARF 診断情報はオブジェクト・プログラムに組み込まれず、別のデバッグ・ファイルに書き込まれることもありません。

注 : SOURCE および SEPARATE は NODWARF と一緒には使用できません。

(NO,...,SOURCE,...)
TEST=(NO,...,SOURCE,...) を指定すると、コンパイラーによって生成される DWARF デバッグ情報に拡張ソース・コードが組み込まれます。

注 : SOURCE は NODWARF と一緒には使用できません。

(NO,...,NOSOURCE,...)
TEST=(NO,...,NOSOURCE,...) を指定すると、生成される DWARF デバッグ情報に拡張ソース・コードは組み込まれません。

(NO,...,SEPARATE(NODSNAME))
TEST=(NO,...,SEPARATE(NODSNAME)) を指定すると、生成される DWARF デバッグ情報は外部ファイルに保管されるため、オブジェクト・プログラムのサイズが大きくなることはありません。コンパイル時に使用される外部ファイル名 (SYSDEBUG データ・セットの名前) はオブジェクト・プログラムに保管されません。 SEPARATE がサブオプションなしで指定されたときのデフォルトは SEPARATE(NODSNAME) です。

注 : SEPARATE は NODWARF と一緒には使用できません。

(NO,...,SEPARATE(DSNAME))
TEST=(NO,...,SEPARATE(DSNAME)) を指定すると、生成される DWARF デバッグ情報は外部ファイルに保管されるため、オブジェクト・プログラムのサイズが大きくなることはありません。コンパイル時に使用される外部ファイル名 (SYSDEBUG データ・セットの名前) はオブジェクト・プログラムに保管されます。 この名前は、実行時に DWARF 情報が要求されるとデフォルトとして使用されます。

注 : SEPARATE は NODWARF と一緒には使用できません。

(NO,...,NOSEPARATE)
TEST=(NO,...,NOSEPARATE) を指定すると、生成される DWARF デバッグ情報はオブジェクト・プログラムに保管されます。

(NO 以外)

(EJPD,...)
TEST=(EJPD,...) および OPT=(1|2) を指定した場合は、以下のようになります。

- IBM z/OS Debugger・コマンド GOTO および JUMPTO が使用可能になります。

- プログラムの最適化が削減されます。最適化はステートメント内で実行されます。最適化がステートメントの境界を越えて実行されることはほとんどありません。

(NOEJPD,...)
TEST=(NOEJPD,...) および OPT=(1|2) を指定した場合は、以下のようになります。

- JUMPTO および GOTO は有効化されません。ただし、SET WARNING OFF IBM z/OS Debugger コマンドを使用する場合、JUMPTO および GOTO は使用可能です。 このシナリオでは、JUMPTO と GOTO の結果は予測不能なものとなります。
- 通常程度のプログラムの最適化が実行されます。

(...,DWARF,...)
TEST=(...,DWARF,...) を指定すると、完全な DWARF 診断情報がオブジェクト・プログラムに組み込まれ、SEPARATE サブオプションが有効になっている場合は別のデバッグ・ファイルに組み込まれます。 このオプションは、CEEDUMP や IBM Fault Analyzer などのアプリケーション障害分析ツールで、最高のユーザビリティーを実現します。

(...,NODWARF,...)
TEST=(...,NODWARF,...) を指定すると、DWARF 診断情報はオブジェクト・プログラムに組み込まれず、別のデバッグ・ファイルに書き込まれることもありません。

注 : SOURCE および SEPARATE は NODWARF と一緒には使用できません。

(...,SOURCE,...)
TEST=(...,SOURCE,...) を指定すると、コンパイラーによって生成される DWARF デバッグ情報に拡張ソース・コードが組み込まれます。

(...,NOSOURCE,...)
TEST=(...,NOSOURCE,...) を指定すると、生成される DWARF デバッグ情報に拡張ソース・コードは組み込まれません。

(...,SEPARATE(NODSNAME))
TEST=(...,SEPARATE(NODSNAME)) を指定すると、生成される DWARF デバッグ情報は外部ファイルに保管されるため、オブジェクト・プログラムのサイズが大きくなることはありません。コンパイル時に使用される外部ファイル名 (SYSDEBUG データ・セットの名前) はオブジェクト・プログラムに保管されません。 SEPARATE がサブオプションなしで指定されたときのデフォルトは SEPARATE(NODSNAME) です。

(...,SEPARATE(DSNAME))
TEST=(...,SEPARATE(DSNAME)) を指定すると、生成される DWARF デバッグ情報は外部ファイルに保管されるため、オブジェクト・プログラムのサイズが大きくなることはありません。コンパイル時に使用される外部ファイル名 (SYSDEBUG データ・セットの名前) はオブジェクト・プログラムに保管されます。 この名前は、実行時に DWARF 情報が要求されるとデフォルトとして使用されます。

注 :

- SEPARATE は NODWARF と一緒には使用できません。
- IBM デバッガーを使用して、SYSDEBUG データ・セットに含まれる DWARF デバッグ情報をデバッグできるようにするには、以下のレベルにある何らかのツールが必要です。
  - IBM Debug for z Systems V14.1 (5655-Q50) (旧 IBM Debug Tool for z/OS) 以降
  - IBM Developer for z Systems V14.1 (5724-T07) 以降
  - IBM Application Delivery Foundation for z Systems V3.1 (5655-AC6) 以降

(...,NOSEPARATE)
TEST=(...,NOSEPARATE) を指定すると、生成される DWARF デバッグ情報はオブジェクト・プログラムに保管されます。

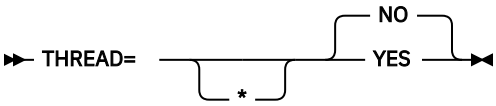
注 : TEST オプションを指定する場合は、CODEPAGE オプションを、COBOL ソース・プログラムで使用する CCSID に設定する必要があります。 特に、DBCS リテラルまたは DBCS ユーザー定義語で日本語

文字を使用するプログラムの場合、CODEPAGE オプションを日本語コード・ページ CCSID に設定してコンパイルしてください。 詳しくは、21 ページの『CODEPAGE』を参照してください。

THREAD

THREAD は、プログラムをマルチスレッド・アプリケーションで使用可能にするかどうかを制御します。

構文



デフォルト
THREAD=NO

YES
複数の POSIX スレッドまたは PL/I タスクを持つ Language Environment・エンクレーブでプログラムを実行できるように指示するには、YES を使用します。

NO
複数の POSIX スレッドまたは PL/I タスクを持つ Language Environment・エンクレーブでプログラムを実行できないように指示するには、NO を使用します。

パフォーマンスの考慮: THREAD コンパイラー・オプションが指定されている場合、自動的に生成される直列化ロジックが原因で実行時のパフォーマンスが低下する可能性があります。

注:

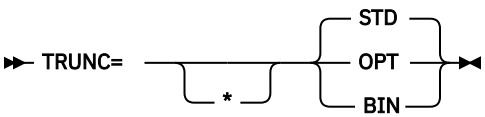
- THREAD コンパイラー・オプションは、LP(64) が有効な場合は無視されます。 ユーザーが明示的に THREAD オプションを指定した場合は、通知メッセージが出されます。
- THREAD コンパイラー・オプションが指定されている場合、プログラムはスレッド化されたアプリケーション内で使用可能に なります。ただし、スレッド化されていないアプリケーションでも THREAD を使用できます。例えば、実行時にアプリケーションに複数の POSIX スレッドまたは PL/I タスクが含まれていない場合、THREAD オプションを指定してコンパイルされたプログラムを CICS 環境で実行できます。
- THREAD コンパイラー・オプションを指定する場合、RENT コンパイラー・オプション も指定する必要があります。同じレベルの優先順位で THREAD と NORENT が指定されている場合、強制的に RENT が使用されます。
- スレッド化されたアプリケーション内で COBOL プログラムを実行するには、実行単位内のすべての COBOL プログラムが THREAD コンパイラー・オプションを指定してコンパイルされている必要があります。
- THREAD コンパイラー・オプションが指定されている場合、以下の言語エレメントはサポートされません。以下の言語エレメントのいずれかを指定すると、エラーとして診断されます。
  - ALTER ステートメント
  - DEBUG-ITEM 特殊レジスター
  - プロシージャー名を含まない GO TO ステートメント
  - PROGRAM-ID パラグラフ内の INITIAL 句
  - ネストされたプログラム
  - RERUN
  - 分割モジュール
  - MERGE ステートメントまたは形式 1 SORT ステートメント
  - STOP *literal* ステートメント
  - USE FOR DEBUGGING ステートメント
- THREAD コンパイラー・オプションを指定してプログラムをコンパイルする場合、呼び出しごとに以下の特殊レジスターが割り振られます。

- ADDRESS OF
- JSON-CODE
- JSON-STATUS
- RETURN-CODE
- SORT-CONTROL
- SORT-CORE-SIZE
- SORT-FILE-SIZE
- SORT-MESSAGE
- SORT-MODE-SIZE
- SORT-RETURN
- TALLY
- XML-CODE
- XML-EVENT
- XML-INFORMATION
- XML-NAMESPACE
- XML-NAMESPACE-PREFIX
- XML-NNAMESPACE
- XML-NNAMESPACE-PREFIX
- XML-NTEXT
- XML-TEXT

TRUNC

TRUNC は、MOVE および算術演算時に 2 進データを切り捨てる方法に作用します。

構文



デフォルト

TRUNC=STD

BIN

インストール・システムのデフォルトとして使用してはなりません。以下を指定します。

- 出力 2 進数フィールドは、PICTURE 節の境界ではなく、 ハーフワード、フルワード、およびダブルワード境界でのみ 切り捨てられます。
- 送信される 2 進数フィールドは、ハーフワード、フルワード、またはダブルワード として扱われ、値は PICTURE 節で暗黙指定される値に限定されるとは想定 されません。
- DISPLAY は、PICTURE 記述に合わせた切り捨てを行わずに、2 進数フィールドの 完全な内容を変換し、出力します。

注: TRUNC(BIN) と NUMCHECK(BIN) が両方とも有効で、エラー・メッセージか異常終了が生成される際に、パフォーマンスを改善するために後で TRUNC(STD|OPT) に切り替えようとしている場合は、データを訂正する必要があります。切り替えない場合は、NUMCHECK(BIN) をオフにして、アプリケーションの実行時間を減らし、エラー・メッセージや異常終了が生成されないようにすることができます。

OPT

コンパイラーは、データが PICTURE および USAGE の指定に従うものと 見なします。コンパイラーは、結果を、ストレージ内のフィールドのサイズ (ハーフワードまたは フルワード) に基づいて処理します。

TRUNC=OPT をお勧めしますが、これを指定するのは、2 進域に移動される データが、2 進項目の PICTURE 節で定義された精度よりも大きい精度の値を 持たない場合に限ってください。それ以外の場合に指定すると、高位桁が切り捨てられることがあります。

STD
85 COBOL 標準に準拠します。

MOVE および算術演算時に算術フィールドを切り捨てる方法を制御します。TRUNC オプションは、MOVE ステートメントおよび演算式に おける 2 進数 (COMP) 受け取りフィールドにのみ適用されます。TRUNC=STD が有効であれば、演算式の (または MOVE ステートメントの 送り出しフィールドの) 最終的な中間結果は、2 進数受け取りフィールド の PICTURE 節内の桁数に切り捨てられます。

パフォーマンスの考慮: TRUNC=OPT を使用すると、余分なコードが生成されず、通常はパフォーマンスが向上します。 一方、TRUNC=BIN または TRUNC=STD を使用した場合は、BINARY データ項目が 変更されると必ず余分なコードが生成されます。TRUNC=BIN を指定すると、通常、これらのオプションの中では比較的処理が 遅くなります。

推奨:

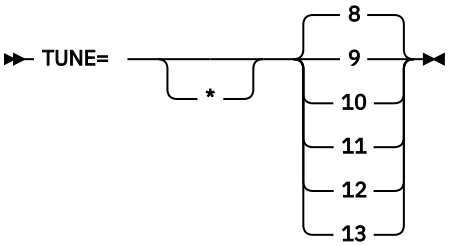
- 他のプロダクトによって設定される 2 進値を使用するプログラムの場合、推奨されるオプションは TRUNC=BIN です。 他のプロダクト (IMS、 Db2、C/C++、FORTRAN、 および PL/I など) が、COBOL の 2 進数データ項目に、それらのデータ項目の PICTURE 節に従わない 値を入れることがあります。
- データが 2 進データ項目用の PICTURE 節に矛盾しない場合は、CICS プログラムで TRUNC=OPT を使用することができます。
- このオプションの設定は、プログラム実行時ロジックに影響を与えます。つまり、同じ COBOL ソース・プログラム でも、このオプションの設定によって結果が異なることがあります。 COBOL ソース・プログラムを正しく実行するために特定の設定を前提としているかどうかを 検証してください。

関連参照
45 ページの『NUMCHECK』

TUNE

TUNE オプションで、実行可能プログラムの最適化の対象となるアーキテクチャーが指定されるようになりました。

構文



デフォルト
ARCH を指定する場合は、デフォルトの TUNE レベルと ARCH レベルが一致している必要があります。ARCH を指定しない場合は、ARCH と TUNE は両方ともデフォルトで 8 に設定されます。

現在サポートされているアーキテクチャー・レベルおよびモデルのグループは以下のとおりです。

8
z/Architecture モードの 2097-xxx (IBM System z10 EC) モデルおよび 2098-xxx (IBM System z10 BC) モデル向けに最適化されたコードを生成します。

9
z/Architecture モードの 2817-xxx (IBM zEnterprise 196) モデルおよび 2818-xxx (IBM zEnterprise 114) モデル向けに最適化されたコードを生成します。

10
z/Architecture モードの 2827-xxx (IBM zEnterprise EC12) モデルおよび 2828-xxx (IBM zEnterprise BC12) モデル向けに最適化されたコードを生成します。

11
z/Architecture モードの 2964-xxx (IBM z13) モデルおよび 2965-xxx (IBM z13s®) モデル向けに最適化されたコードが生成されます。

12
z/Architecture モードの 3906-xxx (IBM z14) モデルおよび 3907-xxx (IBM z14 ZR1) モデル向けに最適化されたコードが生成されます。

13
z/Architecture モードの 8561-xxx (IBM z15) モデル向けに最適化されたコードが生成されます。

TUNE オプションで、実行可能プログラムの最適化の対象となるアーキテクチャーが指定されるようになりました。TUNE のレベルは、有効な ARCH レベルの制約の範囲内で、コンパイラーが使用可能なマシン命令を選択する方法を制御します。 TUNE オプションの目的は、 特定の TUNE アーキテクチャーにおいて、生成されたコードで使用可能なマシン命令からできるだけ高いパフォーマンスを実現することです。

注 : TUNE はパフォーマンスにのみ影響を与え、どのプロセッサー・モデルでアプリケーションを実行できるかには影響しません。

当該アプリケーションを最も頻繁に実行する予定のマシンのアーキテクチャーと一致するように TUNE を選択してください。 アプリケーションを実行できるマシンに合わせてアプリケーションをチューニングするためには、TUNE レベルを常に ARCH レベル以上にする必要があります。 これを強制するために、コンパイラーは ARCH の値を下げるのではなく TUNE の値を上げるという調整を行います。 TUNE は、どこでアプリケーションを実行できるかは指定しません。 これは基本的には最適化オプションです。

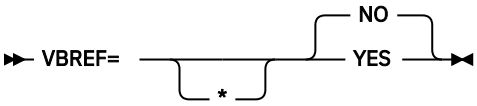
注 : 指定されている ARCH レベルよりも TUNE レベルが低い場合、コンパイラーは、 TUNE レベルを ARCH レベルと一致するように調整します。

関連参照
17 ページの『ARCH』

VBREF

VBREF は、行番号に対するステートメントの相互参照とステートメント使用の要約を作成するかどうかに作用します。

構文



デフォルト
VBREF=NO

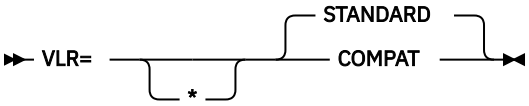
YES
ソース・プログラム内のすべてのステートメント・タイプと、それらが検出された行番号との相互参照を生成します。 VBREF=YES を指定すると、各ステートメントがプログラムで使用された回数の要約も生成されます。

NO
相互参照リストもステートメント要約リストも生成しません。

VLR

VLR オプションは、可変長レコードの READ ステートメントから返されたレコード長がレコード記述と矛盾する場合に、返されるファイル状況に作用します。 これにより、ご使用のプログラムにレコード長の矛盾を引き起こす READ ステートメントがある場合に、旧バージョンから Enterprise COBOL V6 への移行が容易になります。

構文



デフォルト
VLR=STANDARD

READ ステートメントの実行後:

- 読み取られたレコード内の文字位置数がファイルのレコード記述項目で指定された最小サイズより小さい場合、レコード域内の読み取られた最後の有効文字の右側の部分は未定義になります。
- 読み取られたレコード内の文字位置数がファイルのレコード記述項目で指定された最大サイズより大きい場合、最大サイズの右側にあるレコード部分は切り捨てられます。

いずれの場合でも、READ ステートメントは正常に実行され、ファイル状況は VLR コンパイラー・オプション設定に応じて、00 (レコード長の矛盾状態を隠します) または 04 (レコード長の矛盾が発生したことを示します) に設定されます。

COMPAT
VLR=COMPAT は、読み取られたレコードのサイズを、FD 節の "VERYING IN SIZE FROM min TO max" 宣言に基づいて検査します。 VLR=COMPAT を指定した場合、READ ステートメントでレコード長の矛盾が検出されると、状況値 00 を受け取ります。

注 : この設定により、不正な長さの読み取り状態で引き起こされる入出力問題を隠すことができます。 VLR=COMPAT オプションの使用には注意が必要です。 また、READ ステートメントが正しいかどうかを確認してください。

STANDARD
VLR=STANDARD は、読み取られたレコードのサイズを、FD レベル 01 節の宣言に基づいて検査します。 VLR=STANDARD を指定した場合、READ ステートメントでレコード長の矛盾が検出されると、状況値 04 を受け取ります。

FS=04 をテストするためのコードを追加して、レコード内の未定義データをアクセスしないように、また切り捨てられたレコード部分を参照する試行に関する保護例外を受け取らないようにすることができます。

以下の例は、VLR オプションが、読み取られたレコードのサイズを FD 節および FD レベル 01 節の宣言に基づいて検査する方法を示しています。

```
FD MYDD
   block contains 0 records
   record varying in size from 10 to 80
   recording mode V.
01 REC-20
   02 PIC X(20).
01 REC-50.
   02 PIC X(50).
```

表 *8.* 読み取られたレコードの長さとファイル状況

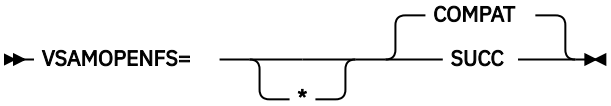
読み取られたレコードの長さ	**=5** **< min varying** **< min level 01**	**= 15** **>= min varying** **< min level 01**	**= 40** **>= min varying <= max varying** **>= min level 01** **<= max level 01**	**= 70** **>= min varying <= max varying** **> max level 01**
COBOL V4	FS=04	FS=00	FS=00	FS=00
COBOL V6 VLR(COMPAT)	FS=04	FS=00	FS=00	FS=00

表 8. 読み取られたレコードの長さとファイル状況 (続き)				
読み取られたレコードの長さ	**=5** **< min varying** **< min level 01**	**= 15** **>= min varying** **< min level 01**	**= 40** **>= min varying <= max varying** **>= min level 01** **<= max level 01**	**= 70** **>= min varying <= max varying** **> max level 01**
COBOL V6 VLR(STANDARD)	FS=04	FS=04	FS=00	FS=04

VSAMOPENFS

VSAMOPENFS オプションは、ファイル整合性検査の確認を必要とする、正常に実行された VSAM OPEN ステートメントから報告されるユーザー・ファイル状況に作用します。

構文



デフォルト
VSAMOPENFS=COMPAT

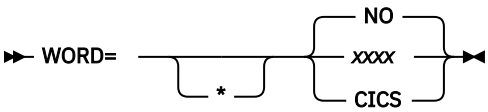
COMPAT
VSAMOPENFS=COMPAT を指定した場合、VSAM OPEN ステートメントが正常に検査されると、ステートメントはファイル状況 97 を返します。 これは、V6 より前の COBOL の実行時動作と互換性があります。

SUCC
VSAMOPENFS=SUCC を指定した場合、VSAM OPEN ステートメントが正常に検査されると、ステートメントはファイル状況 00 を返します。 これにより、ユーザーは、他の正常な操作で通常行うように、返されたファイル状況の最初の桁で 0 の有無を確認するだけで済みます。

WORD

WORD は、コンパイル時に使用する代替予約語テーブルを指示します。

構文



デフォルト
WORD=NO

CICS
CICS 固有の予約語テーブル (IGYCCICS) が、代替予約語テーブルとして提供されます。詳細については、8 ページの『CICS 予約語テーブル (IGYCCICS)』を参照してください。

xxxx
コンパイル時に使用される代替のデフォルト予約語テーブルを指定します。 ***xxxx*** は、使用される予約語テーブルの名前の最後の 文字 (1 から 4 文字の長さ) を表しています。最初の 4 文字は IGYC です。最後の 4 文字は、以下にリストする文字ストリングのいずれかにすることはできず、ドル記号文字 ($) を入れることもできません。

- CBE
- DGEN

- DIAG
- DMAP
- DOPT
- ECWI
- FGEN
- INIT
- LIBO
- LIBR
- LSTR
- LVL0
- LVL1
- LVL2
- LVL3
- LVL8
- OSCN
- PGEN
- RCTL
- RDPR
- RDSC
- RWT
- SAW
- SCAN
- SIMD
- XREF

NO
代替予約語テーブルをデフォルトとして使用しないことを指定します。

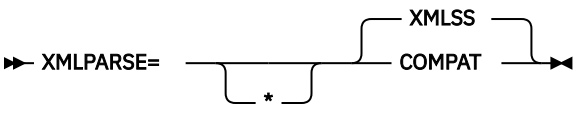
注 :

- WORD の指定は、入力予約語の解釈に影響を与えます。システム名 (UPSI、SYSPUNCH など) および組み込み関数名 を予約語の別名として使用してはなりません。予約語テーブル ABBR 制御ステートメントを使用して関数名を別名として指定する と、その関数名はコンパイラーによって予約語として認識および診断されるので、組み込み関数は実行されません。
- WORD=*XXXX* オプションのデフォルト値を変更すると、FLAGSTD に指定された NO 以外のすべての値と矛盾します。

XMLPARSE

XMLPARSE は、XML 入力の処理に使用するパーサーと、その結果使用可能な XML 構文解析機能を指示します。

構文



デフォルト
XMLPARSE=XMLSS

COMPAT
XML PARSE ステートメントは、COBOL ライブラリーの組み込みコンポーネントである XML パーサーを使用して処理されます。 XML PARSE ステートメントの結果および操作上の動作は、Enterprise COBOL バージョン 3 で取得されるものと互換性があります。 また、XMLPARSE(COMPAT) が使用された場合はバージョン 4 と互換性があります。

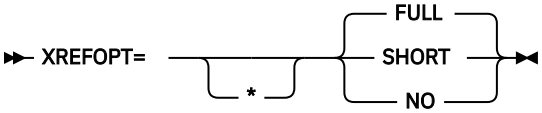
XMLSS
XML PARSE ステートメントは z/OS XML システム・サービス・パーサーを使用して処理されます。 以下の XML 構文解析機能は、このサブオプションが有効である場合にのみ使用できます。

- ネーム・スペースの処理の拡張
- XML PARSE ステートメントの ENCODING、RETURNING NATIONAL、および VALIDATING 句
- UTF-8 でエンコードされた XML 文書の直接的な構文解析のサポート
- 非常に容量が大きい XML 文書の構文解析 (一度に 1 つの XML バッファー) をサポート
- XML 構文解析の System z® Application Assist Processors (zAAPs) へのオフロード

XREFOPT

XREFOPT は、XREF コンパイラー・オプションのデフォルト値を設定します。 これは、リストに作成される相互参照情報の量に作用します。

構文



デフォルト
XREFOPT=FULL

FULL
プログラム内で使用されている名前のソート済み相互参照を生成し、 それらの名前が定義されている行を示します。 COPY/BASIS 相互参照も生成します。 SOURCE=YES も指定されている 場合、組み込み相互参照情報がソースと同じ行に 印刷されます。

SHORT
明示的に参照されている変数のみのソート済みリストを 生成し、COPY/BASIS 相互参照を生成します。

NO
相互参照リストを抑制します。

注 :

- XREFOPT=NO は、DBCSXREF の NO 以外の値と矛盾します。
- デフォルトを XREFOPT=NO に変更しないことをお勧めします。 XREFOPT=NO の場合、COPY/BASIS 相互参照が場合によっては不完全であったり欠落することがあります。

詳しくは、「*Enterprise COBOL* プログラミング・ガイド」で『*XREF*』を参照してください。

ZONECHECK

ZONECHECK は推奨されなくなり、IGYCDOPT に指定できなくなりました。 NUMCHECK=(ZON(ALPHNUM)) を使用すると、ZONECHECK と同じ結果を得られます。

詳細については、45 ページの『NUMCHECK』を参照してください。

関連参照
45 ページの『NUMCHECK』
50 ページの『NUMPROC』
36 ページの『INVDATA』

ZONEDATA

コンパイラーは、ZONEDATA オプションの指示に従って、USAGE DISPLAY データ項目および PACKED-DECIMAL データ項目におけるデータが有効かどうかを判別し、そのデータが有効でない場合はどうすればいいのかを決定します。

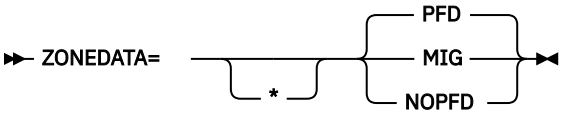
大半のユーザーは USAGE DISPLAY データ項目と USAGE PACKED-DECIMAL データ項目に有効なデータを持っているため、それらのユーザーは ZONEDATA=PFD を使用する必要があります。プログラムが無効データを処理していることを実行時に (NUMCHECK コンパイラー・オプションを使用して) 検出した場合にも、無効データの処理を回避するためにプログラムを変更し、ZONEDATA=PFD を使用する必要があります。

注 : ZONEDATA は非推奨ですが、互換性のために使用できるようにはなっています。 これは INVDATA で置き換えられます。

ZONEDATA オプションが指定されている場合、それは以下のように、同等の INVDATA オプションにマップされます。

```
ZONEDATA=PFD  -> NOINVDATA
ZONEDATA=NOPFD -> INVDATA(NOFORCENUMCMP,CLEANSIGN)
ZONEDATA=MIG -> INVDATA(FORCENUMCMP,NOCLEANSIGN)
```

構文



デフォルト
ZONEDATA=PFD

PFD
ZONEDATA=PFD が有効な場合、コンパイラーは、USAGE DISPLAY データ項目および PACKED-DECIMAL データ項目にあるすべてのデータが有効であることを想定し、数値比較を行う上で可能な限り最も効率的なコードを生成します。 例えば、コンパイラーは数値変換を避けるためにストリング比較を生成する場合があります。

MIG
ZONEDATA=MIG が有効な場合、コンパイラーは、ゾーン 10 進数データ項目内の各桁のゾーン・ビットを無視する数値比較を行う命令を生成します。 例えば、ゾーン 10 進数値は比較前に、パック命令によってパック 10 進数に変換されます。 さらに、無効な数値や無効な符号がゾーン 10 進数データ項目やパック 10 進数データ項目に含まれている場合や、無効なゾーン・ビットがゾーン 10 進数データ項目に含まれている場合、コンパイラーは COBOL V4 (またはそれ以前のバージョン) とは異なる結果を生成する可能性がある既知の最適化を実行しません。

NOPFD

ZONEDATA=NOPFD が有効な場合、コンパイラーは、COBOL V4 (またはそれ以前のバージョン) で NUMPROC=NOPFD|PFD を使用したときに COBOL V4 (またはそれ以前のバージョン) が実行する方法と同じ方法でゾーン 10 進数データの数値比較または英数字比較をするための命令を生成します。

- COBOL V4 (またはそれ以前のバージョン) でゾーン・ビットが処理対象となっていた場合、コンパイラーは英数字比較を生成し、英数字比較でもゾーン 10 進数データ項目内の各桁のゾーン・ビットが数値比較の対象となります。 ゾーン 10 進数値はゾーン 10 進数のままです。
- COBOL V4 でゾーン・ビットが無視されていた場合は、ゾーン 10 進数データ項目内の各桁のゾーン・ビットが無視される数値比較をコンパイラーが生成します。 ゾーン 10 進数値は比較前に、PACK 命令によってパック 10 進数に変換されます。

ゾーン 10 進データの比較の生成時に COBOL V4 以前と同じ方法でコンパイラーがゾーン・ビットを扱うには、COBOL V6 で使用される NUMPROC サブオプションが、COBOL V4 以前で使用された NUMPROC サブオプションと一致しなければなりません。

- COBOL V6 で COBOL V4 (またはそれ以前のバージョン) の NUMPROC=NOPFD 動作を行うには、COBOL V6 で ZONEDATA=NOPFD および NUMPROC=NOPFD を使用します。
- COBOL V6 で COBOL V4 (またはそれ以前のバージョン) の NUMPROC=PFD 動作を行うには、COBOL V6 で ZONEDATA=NOPFD および NUMPROC=PFD を使用します。

さらに、無効な数値や無効な符号がゾーン 10 進数データ項目やパック 10 進数データ項目に含まれている場合や、無効なゾーン・ビットがゾーン 10 進数データ項目に含まれている場合、コンパイラーは COBOL V4 (またはそれ以前のバージョン) とは異なる結果を生成する可能性がある既知の最適化を実行しません。

注 : 符号は、NUMPROC コンパイラー・オプション設定に準じた有効な符号でなければなりません。 さらに、下位バイトには、SIGN IS LEADING または SIGN IS SEPARATE のいずれかによる符号解除および符号付けのために有効なゾーン (x'F') がなければなりません。

どの値を指定した場合でも、COBOL V6 へのマイグレーションを容易にするには、以下のようにします。

- 数値、符号、およびゾーン・ビットが有効な場合に COBOL V6 を使用するときは、COBOL V4 で使用した NUMPROC 設定と同じ設定を ZONEDATA=PFD と一緒に使用します。
- データに無効な数字、無効な符号コード、または無効なゾーン・ビットがある場合は、プログラムが実行時に数値データ項目に無効なデータを持たないように、プログラムまたはシステムを変更してください。

  プログラムまたはシステムを修正したら、お好みの ZONEDATA=PFD オプションを使用できます。 この作業を含めることができず、無効データを使用した実行を続けなければならない場合にのみ、ZONEDATA に対して以下の選択項目を検討してください。

  - NUMPROC=MIG を COBOL V4 で使用した場合、COBOL V6 では ZONEDATA=MIG および NUMPROC=NOPFD を使用します。
  - NUMPROC=NOPFD を COBOL V4 で使用した場合、COBOL V6 では ZONEDATA=NOPFD および NUMPROC=NOPFD を使用します。
  - NUMPROC=PFD を COBOL V4 で使用した場合、COBOL V6 では ZONEDATA=NOPFD および NUMPROC=PFD を使用します。

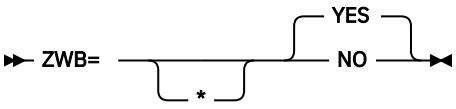
注 : 明らかに無効なデータが検出された場合、これらのオプションを使用したとしても、古いコンパイラーの動作を完全に一致させることは常に可能ではありません。例えば、比較を行う場合でも、ZONEDATA=NOPFD によって、どのようなケースでも COBOL V4 と同じ結果を得られるわけではありません。

パフォーマンスの考慮事項: ZONEDATA=PFD では、ZONEDATA=NOPFD|MIG よりもランタイム・パフォーマンスが向上します。 ZONEDATA=NOPFD|MIG は、NUMPROC=PFD による最適化の一部を無効にします。

ZWB

ZWB は、コンパイラーが、実行時に符号付きゾーン 10 進数フィールドを英数字フィールドと比較する前に、符号付きゾーン 10 進数フィールドから符号を除去するかどうかを決定します。

構文



デフォルト
ZWB=YES

YES
実行中に符号付き外部 10 進数 (DISPLAY) フィールドを英数字フィールドと 比較するときに、符号付き外部 10 進数フィールドから符号を除去します。

NO
実行中に符号付き外部 10 進数 (DISPLAY) フィールドを英数字フィールドと比較するときに、符号付き外部 10 進数フィールドから符号を除去しません。

注:

- このオプションの設定は、プログラム実行時論理に影響を与えます。つまり、同じ COBOL ソース・プログラムでも、このオプションの設定によって結果が異なることがあります。Enterprise COBOL ソース・プログラムを 正しく実行するために、特定の設定を前提としているかどうかを検証してください。
- アプリケーション・プログラマーは、SPACES の入力数値フィールドをテスト するために ZWB=NO を使用します。

第 3 章 Enterprise COBOL のカスタマイズ

以下のセクションでは、Enterprise COBOL をカスタマイズするために変更可能な、提供されるユーザー・ジョブおよびモジュールについて説明します。

Enterprise COBOL に変更を加えることができるのは、この製品のインストールが完了した後に限られます。

それらの変更の 1 つは、SMP/E USERMOD を使用して行います。USERMOD を適用する前に Enterprise COBOL を配布ライブラリーの中に ACCEPT しなかった場合は、 SMP/E RESTORE ステートメントを使用して USERMOD を除去することはできません。USERMOD を配布ライブラリーの中に受け入れてはなりません。USERMOD がインストール先のプログラマーの要件を満たさないことが判明した 場合、USERMOD を除去することがあるからです。

USERMOD によって変更されるモジュールにサービスを適用するときは、その前に USERMOD を除去する必要があります。この場合、サービスが正常にインストールされた後、USERMOD を再適用することができます。

重要 : Enterprise COBOL がインストール先のアプリケーション・プログラマーの要件を満たすようにしてください。Enterprise COBOL のカスタマイズを計画する際には、アプリケーション・プログラマーと相談してください。 そうすることにより、インストール時に行う変更は、インストール先で開発されるアプリケーション・ プログラムを最も適切にサポートすることになります。

Enterprise COBOL をインストールするために必要な情報はすべて、製品に付属のプログラム・ディレクトリー に含まれています。

ユーザー変更の要約

Enterprise COBOL をインストールすると、多数のサンプル変更ジョブが ターゲット・データ・セット IGY.V6R3M0.SIGYSAMP に置かれます。ただし、これらのサンプル変更ジョブは、特定システム用にカスタマイズされていません。 これらのジョブをカスタマイズする必要があります。

79 ページの表 9 は、 これらのサンプル変更ジョブの名前を示しています。これらのジョブについては、示されている参照ページに詳しい説明があります。

IGY.V6R3M0.SIGYSAMP のメンバーを変更して実行依頼する前に、これらのメンバーを個人用データ・セットの 1 つにコピーして、変更を中止する場合のために、未変更のバックアップ・コピーがある状態にしておいてください。

可能な変更についての説明は、JCL 内のコメントに示されています。TSO を使用して、ジョブを変更し、実行依頼することができます。

表 *9. Enterprise COBOL* 用のユーザー変更ジョブの要約

説明	ジョブ	参照ページ
コンパイラー・オプション・デフォルト・モジュールの変更	IGYWDOPT	80 ページの『コンパイラー・オプション・デフォルト・モジュールの変更』
固定として指定されたオプションのオーバーライド	IGYWUOPT	81 ページの『固定として指定されたオプションのオーバーライド』
予約語の変更	IGYWRWD	81 ページの『予約語の変更』

コンパイラー・オプションのデフォルトの変更

コンパイラー・オプションのデフォルトは変更することができ、固定のコンパイラー・オプションをオーバーライドすることができます。

コンパイラー・オプションのデフォルトを変更するには、以下のようにします。

1. オプション・モジュール IGYCDOPT のソースを IGY.V6R3M0.SIGYSAMP から該当するジョブにコピーして、2 行のコメントを置き換えます。

2. IGYCOPT マクロ呼び出しのパラメーターを変更して、 ご使用のシステム用に選択したコンパイラー・オプションに一致するようにします。

変更した IGYCOPT マクロ呼び出しをコーディングする際には、以下の要件に注意してください。

- IGYCOPT 呼び出しの各行 (最終行以外) の 桁 72 に継続文字 (ソース中の X) を置く必要があります。継続行は、桁 16 で開始する必要があります。任意のコンマの後で、コーディングを分割することができます。
- マクロの最初のオプションの前にコンマを置かないでください。
- オプションおよびサブオプションは、大文字で指定する必要があります。ストリングであるサブオプションのみは、大小混合または小文字で指定することが できます。
- ストリング・サブオプションの 1 つに特殊文字 (例えば、組み込みブランク、対応しない右括弧または左括弧) を含める場合は、そのストリングを 二重引用符 (") ではなくアポストロフィ (') で囲む必要があります。連続するアポストロフィまたは引用符のいずれかを使用して、ヌル・ストリングを 指定することができます。

  ストリングの中でアポストロフィ (') または単一アンパーサンド (&) を使用するには、その文字を 2 つ続けて指定する必要があります。最大許容ストリングの長さを超えたかどうかを判別するときや、ストリングの 有効長を設定するときに、このペアは 1 文字として数えられます。
- アポストロフィを使用するときは、どのストリングにおいても、対応しないアポストロフィの使用を避けてください。このエラーは、IGYCOPT 自体の中では検出されません。代わりに、アセンブラーは次のようなメッセージを出します。

  ```
  IEV03  *** ERROR ***  NO ENDING APOSTROPHE
  ```

  このメッセージは、違反しているサブオプションに対する位置関係を示すものではありません。さらに、このエラーが発生すると、どのオプションも正しく解析されません。
- デフォルト値を変更するオプションのみをコーディングしてください。 コーディングしないオプションについては、IGYCOPT マクロがデフォルト値を生成します。

  デフォルトのコンパイラー・オプションを計画する際に役立つワークシートについては、4 ページの『コンパイラー・オプション用の IGYCDOPT ワークシート』を参照してください。 オプションの説明については、11 ページの『第 2 章 Enterprise COBOL コンパイラー・オプション』を参照してください。
- マクロ命令の後に、END ステートメントを置いてください。

注: `igycdopt.asm` での IGYCOPT の呼び出しなど、 いずれかのマクロ呼び出しでアセンブラーが不明なキーワードを検出した場合 そのキーワードは定位置パラメーターとして扱われます。 IGYCOPT マクロの現行バージョンは定位置パラメーターを検査していなかったため、 それらの不明キーワードは完全に無視されていました。 IGYCOPT マクロは変更されて、それらの定位置パラメーターを検出し、適切なエラー・メッセージを出すようになりました。 RC=00 をアセンブルしていた既存のカスタマイズ・マクロを再ビルドすると、 これまで定位置パラメーターとして処理されて無視されていた不明なコンパイラー・オプションのため、 または、正しくない構文 (COBOL カスタマイズ・マクロで `OPTION=` とすべきところが `OPTION()` になっているなど) を使用してコーディングされているコンパイラー・オプションのために、アセンブリー・エラーが発生して再ビルドが失敗する可能性があります。 こうした不明なオプションや正しくコーディングされていないオプションを削除または訂正してから、 IGYCDOPT モジュールを再ビルドする必要があります。

マクロ呼び出しのコーディング方法について詳しくは、「*High Level Assembler for z/OS Language Reference*」を参照してください。

コンパイラー・オプション・デフォルト・モジュールの変更

Enterprise COBOL のコンパイラー・オプションのデフォルトを変更するには、サンプル・ジョブ IGYWDOPT を変更します。

デフォルト値を選択するには、11 ページの『第 2 章 Enterprise COBOL コンパイラー・オプション』に記載されている情報を使用します。

IGYWDOPT の JCL を変更する場合の手順:

1. ご使用のシステムに適した JOB カードを追加します。
2. ご使用のシステムで必要な場合、JES ROUTE カードを追加します。
3. IGYWDOPT 内の 2 行のコメント行を、IGY.V6R3M0.SIGYSAMP にある IGYCDOPT のソースのコピーで置き換えます。
4. IGYCDOPT 内の IGYCOPT マクロ・ステートメントのパラメーターをコーディングして、ご使用のシステムでのデフォルト・コンパイラー・オプションのために 選択した値を反映させます。
5. #GLOBALCSI をグローバル CSI 名に変更します。
6. SET BDY ステートメント上の #TZONE をターゲット・ゾーン名に変更します。

ジョブ IGYWDOPT を変更した後、このジョブを実行依頼します。このジョブが正しく実行されると、条件コード 0 が返されます。 リストの IGY*nnnn* 通知メッセージ を調べて、ご使用のシステムで有効になる デフォルトを確認してください。

固定として指定されたオプションのオーバーライド

固定として指定された 1 つ以上のオプションをオーバーライドすることが必要なアプリケーションがある場合があります。

そのアプリケーションのコンパイル時に STEPLIB または JOBLIB としてアクセスできる 別個のデータ・セット (IGY.V6R3M0.SIGYCOMP データ・セットの前に置く) に、オプション・モジュールの 一時コピーを作成することによって、その他のオプションを提供できます。

サンプル・ジョブ IGYWUOPT は、ユーザー指定のデータ・セットの中に リンク・エディットされるデフォルト・オプション・モジュールを作成します。アセンブリーおよびリンク・エディットは SMP/E の制御範囲外で行われるので、標準のデフォルト・オプション・モジュールは影響を受けません。

IGYWUOPT の JCL を変更する場合の手順:

1. ご使用のシステムに適した JOB カードを追加します。
2. ご使用のシステムで必要な場合、JES ROUTE カードを追加します。
3. IGYWUOPT 内の 2 行のコメント行を、IGY.V6R3M0.SIGYSAMP にある IGYCDOPT のソースのコピーで置き換えます。
4. IGYWUOPT 内の IGYCOPT マクロ・ステートメントのパラメーターを変更して、この固定オプションのオーバーライド・コンパイラー・オプション・モジュールのために選択した値を反映させます。
5. IBM 提供ものとは異なる接頭部を Enterprise COBOL ターゲット・データ・セットに使用することを選択した場合は、SYSLIB DD ステートメント (「<<<<<」でマークされている) を検査して、データ・セット名が正しいことを確認してください。
6. SYSLMOD DD ステートメントにおける DSNAME=YOURLIB を、IGYCDOPT モジュールのバインド先となるデータ・セットの名前に変更します。 選択したデータ・セットに現在入っている IGYCDOPT モジュールは、新しい バージョンで置き換えられることに注意してください。

ジョブ IGYWUOPT を変更した後、このジョブを実行依頼します。このジョブが正常に実行されると、両方のステップが条件コード 0 を返します。 さらに、リストの IGY*nnnn* 通知メッセージ を調べて、標準のデフォルト・オプション・モジュールの代わりにこのモジュールが使用されるときに有効になるデフォルトを確認してください。

予約語の変更

Enterprise COBOL で予約語として扱われている語を変更するには、予約語テーブル・ユーティリティーを使用します。

Enterprise COBOL が使用する予約語は、 この製品で提供されるテーブル (IGYCRWT) で保守されます。CICS 固有の予約語テーブル (IGYCCICS) が、代替予約語テーブルとして提供されています (8 ページの『CICS 予約語テーブル (IGYCCICS)』を参照)。予約語テーブル・ユーティリティー (IGY8RWTU) を 使用して IGYCRWT または IGYCCICS を変更するか、追加の予約語テーブル を作成することによって、予約語を変更することができます。以前に作成したテーブルを変更することもできます。

予約語テーブル・ユーティリティーは、予約語テーブルを作成または変更するための制御ステートメントを受け入れます。 その結果、新しいテーブルには、IBM 提供のテーブルからの予約語と、ユーザーが適用したすべての変更が含まれます。

以下のタイプの変更を予約語テーブルに加えることができます。

- プログラムで使用されたときに通知メッセージで示される語を 追加する。この通知メッセージを作成するには、IGYCRWT 予約語テーブルを変更し、FLAGSTD オプションを使用してコンパイルする必要があります。
- プログラムで使用されたときに重大エラー・メッセージで示される語を 追加する。
- 現在は通知メッセージまたはエラー・メッセージで示される語に、メッセージを出さないことを指示する。

作成する各予約語テーブルには、固有の 1 から 4 文字の ID が必要です。使用できない文字ストリングのリスト については、72 ページの『WORD』を参照してください。

コンパイル時に、コンパイラー・オプション WORD(*xxxx*) の値により、 使用する予約語テーブルが識別されます。*xxxx* は、 メンバー名 IGYC*xxxx* 中に指定した、1 から 4 文字の固有の ID です。複数の予約語テーブルを作成できますが、コンパイル時に指定できる予約語テーブル は 1 つだけです。

注 : 1 つの予約語テーブル内の項目の合計数は、サイズが 1536 バイト (1.5 KB) 以内になるようにしてください。

次に示す例によって、IBM 拡張予約語 ENTRY がプログラムで使用されると、メッセージ 0086 が出されます。

```
INFO  ENTRY
```

次の例では、BOOLEAN、XD、および PARENT の使用が制限されています。これらを使用すると、エラーが発生します。

```
RSTR  BOOLEAN
      XD
      PARENT
```

次の例では、GO TO および ALTER の使用が制限されています。これらを使用すると、エラーが発生します。

```
RSTR  GO
      ALTER
```

次の例では、生成された予約語テーブルにより、すべての 85 COBOL 標準言語の使用が可能になります (ネストされたプログラムを除く)。

```
RSTR IDENTIFICATION(1)  only allow 1 program per compilation unit
RSTR ID(1)              same for the short form
RSTR PROGRAM-ID(1)      only allow 1 program per compilation unit
RSTR GLOBAL             do not allow this phrase at all
```

予約語テーブルの作成または変更

予約語テーブルを作成するか、既存の予約語テーブルのいずれかを変更できます。

以下のように、該当するソース・ファイルのコピーを編集します。

- IGY.V6R3M0.SIGYSAMP 内のメンバー IGY8RWRD (IBM 提供のデフォルト予約語テーブル)
- IGY.V6R3M0.SIGYSAMP 内のメンバー IGY8CICS (IBM 提供の CICS 予約語テーブル)
- ユーザー・ファイル (ユーザー提供の予約語テーブル)

適切な非 SMP/E JCL を変更し、起動する必要もあります。

ファイルには 4 つのパートがあります。つまり、パート I、II、III、および IV です。ファイルおよび非 SMP/E JCL を次のように変更してください。

1. ファイルのプライベート・コピーを作成します。
2. パート I (キーワード MOD を含んでいる行までのすべての行) をスキップします。ファイルのこの部分を変更してはいけません。
3. 予約語を含む行のうち、通知メッセージを発行したくない行があれば、その行の 1 桁目にアスタリスクを付けてパート II を編集します。
4. 重大メッセージが出される必要のない予約語を含む行の桁 1 に アスタリスクを置くことにより、パート III を編集します。
5. 必要な変更を作成する追加の予約語制御ステートメントをコーディングすること により、パート IV を編集します (83 ページの『制御ステートメントのコーディング』を参照)。
6. JCL を変更し、実行します (86 ページの『予約語テーブルを作成する JCL の実行』を参照)。 さらに、新しい予約語テーブルの固有の 1 から 4 文字の ID を作成し、それを JCL 内に指定する必要があります。ユーザー定義の予約語テーブルは、PDSE データ・セット内のプログラム・オブジェクトになります。

制御ステートメントのコーディング

予約語テーブルを作成または変更するには、予約語テーブルに作用する制御ステートメントの構文を理解する必要があります。

下の図は、予約語制御ステートメントのコーディングの形式を示しています。

```
ABBR   reserved-word: user-word [comments]
      [reserved-word: user-word [comments]]
        ?
INFO   COBOL-word [(0 | 1)] [comments]
      [COBOL-word [(0 | 1)] [comments]]
        ?
RSTR   COBOL-word  [(0 | 1)] [comments]
      [COBOL-word [(0 | 1)] [comments]]
        ?
```

図 *2.* 予約語制御ステートメントの構文形式

上の図に示されているように、使用できるキーワードは次のとおりです。

ABBR
既存の予約語の代替形式を指定します。

INFO
プログラムで使用されたときおよび FLAGSTD コンパイラー・オプションが有効なときに通知メッセージで示される語を 指定します。

RSTR
プログラムで使用されたときにエラー・メッセージで示される語を 指定します。

制御ステートメント・キーワード INFO および RSTR で指定した語にはすべて、 その語を使用する Enterprise COBOL プログラム のソース・リストにおいてメッセージでフラグが立てられます。省略語の場合、INFO または RSTR 制御ステートメントでその省略語を指定しない 限り、その省略語はソース・リストにおいてメッセージが出されません。

制御ステートメントのコーディング規則

制御ステートメントをコーディングする際には、規則に従う必要があります。

規則には以下のようなものがあります。

- 制御ステートメントを桁 1 から開始します。
- キーワードと最初のオペランドとの間に、1 つまたは複数のスペースを置きます。

- 2 番目のオペランドを指定するときは、最初のオペランドの 後にコロン (:) と 1 つまたは複数のスペースを置きます。
- 追加の指定を行うには、桁 1 から 5 にブランクを置き、その後に オペランド (複数も可) を指定することにより、制御ステートメントを続けます。
- コメントを指定するには、制御ステートメントの桁 1 にアスタリスク (*) を 置きます。制御ステートメントと同じ行にコメントを置くこともできます。その場合は、コメントを始める前に、オペランドの後に少なくとも 1 つの スペースを置く必要があります。
- 複数の変更を単一の制御ステートメントに指定するには、それぞれの追加の指定を別々の行に置いてください。
- ブランク行を追加してはなりません。

制御ステートメントのオペランドのコーディング

このトピックでは、制御ステートメント内にコーディングするオペランドのタイプを示します。

reserved-word
既存の予約語。

user-word
予約語ではない、ユーザー定義の COBOL 語。

コメント
制御ステートメントと同じ行に置くか、または桁 1 にアスタリスクのある 別個の行に置くコメント。

COBOL-word
システム名、予約語、またはユーザー定義語のいずれかである、最大 30 文字の 語。

制御ステートメントのオペランドのコーディング規則

制御ステートメントのオペランドをコーディングする際には、規則に従います。

規則には以下のようなものがあります。

- user-word は、特定の予約語テーブル内の 1 つの ABBR ステートメントにのみ 使用することができます。
- ABBR ステートメントに指定した reserved-word を、RSTR ステートメントまたは INFO ステートメントのどちらでも指定することができます。
- 特定の reserved-word は、ABBR ステートメントに一度だけ指定する ことができます。
- 特定の COBOL-word は、RSTR ステートメントまたは INFO ステートメントの どちらかに一度だけ指定することができます。

ABBR ステートメント

ABBR ステートメントは、指定された予約語の代替記号を定義します。 この記号は、プログラムのコーディング時に使用できます。

```
ABBR reserved-word: user-word [comments]
```

注:

- user-word は予約語となり、このステートメントに指定した予約語の代わりに 使用することができます。
- reserved-word は、元の定義を持つ予約語のままです。
- ソース・リストには、元のソース (それをコーディングしたときの記号) が示されます。
- コンパイラー出力 (他のリスト、一部のメッセージなど) では、予約語が使用されます。

次の例では、REDEF および SEP は、ソース・プログラムで使用できる省略語となります。ソース・プログラムのコンパイル時に、REDEFINES および SEPARATE の 使用に対する適切な反応が起こります。

```
ABBR   REDEFINES: REDEF
SEPARATE:   SEP
```

INFO ステートメント

INFO ステートメントは、コンパイラーによってフラグを立てる COBOL 語を指定します。 また、このステートメントを使用して、ネストされたプログラムの使用を制御することもできます。

INFO COBOL-word[(0 | 1)] [comments]

1 または 0 を選択して、特定の COBOL-word を一度だけ 使用できるか、またはまったく使用できないかを示すことができます。

0
FLAGSTD コンパイラー・オプションが有効である場合、指定した COBOL-word が使用されると、通知メッセージ 0086 が出されます。

1
指定した COBOL-word は一度だけ使用できます。2 度以上使用されると、通知メッセージ 0195 が出されます。

これらのメッセージは、通知 (I) メッセージとして処理されます。コンパイル条件は変更されません。

RSTR ステートメント

RSTR ステートメントは、プログラムで使用できない COBOL 語を指定します。 また、このステートメントを使用して、ネストされたプログラムの使用を制御することもできます。

RSTR COBOL-word[(0 | 1)] [comments]

1 または 0 を選択して、特定の COBOL-word を一度だけ 使用できるか、またはまったく使用できないかを示すことができます。

0
指定した COBOL-word が使用されると、メッセージ 0084 が出されます。

1
指定した COBOL-word は一度だけ使用できます。2 度以上使用されると、重大メッセージ 0194 が出されます。

以下の予約語は、オプション 1 を使用することでのみ制限 できます。

DATA
ENVIRONMENT
ID
IDENTIFICATION
PROGRAM-ID
WORKING-STORAGE
LOCAL-STORAGE
LINKAGE

非 SMP/E JCL の変更および実行

予約語テーブルを作成または変更するには、IGY.V6R3M0.SIGYSAMP 内のサンプル・ジョブ IGYWRWD を使用します。

このサンプル・ジョブは、IGY.V6R3M0.SIGYSAMP 内の IGY8RWRD または IGY8CICS に基づいたメンバーを、予約語ユーティリティーへの入力として使用します。 これによって、プログラム・オブジェクト IGYC*xxxx* (*xxxx* はユーザー識別) が IGY.V6R3M0.SIGYCOMP 内に作成されます。

IGYWRWD の JCL を変更する場合の手順:

1. インストール・システムに合わせて JOB ステートメントを変更します。
2. 必要に応じて、JES ROUTE レコードを追加します。
3. STEP1 の STEPLIB DD ステートメント上のデータ・セット名を変更して、インストール時に使用したコンパイラー・ターゲット・データ・セット名に一致させます。
4. 変更した予約語テーブルをポイントするため、以下のステップのうちの 1 つのみ を実行します。

- //RSWDREAD DD DSN=... 内のデータ・セット名を、変更済み予約語テーブルのデータ・セット名とメンバー名に変更します。
- RSWDREAD DD を //RSWDREAD DD * で置き換え、その行の直後に 変更済み予約語テーブルを挿入します。

特定の指示については、ジョブ IGYWRWD 内のコメントを参照してください。

5. STEP3 の SYSLMOD DD ステートメント内のデータ・セット名を変更して、 変更済み予約語テーブルを追加する先のデータ・セットの名前に一致させます (SYSLMOD DD ステートメント内の データ・セット名は、コンパイラー・ターゲット・データ・セットの名前でなければなりません)。さらに、SYSLMOD DD ステートメント内のデータ・セット名の後に、変更済み予約語テーブルの名前を括弧で囲んで指定する必要があります。

IGYWRWD の実行後に、予約語テーブル・ユーティリティーからゼロ以外の戻りコードを受け取った場合は、RSWDPRNT DD ステートメントに指定されている出力データ・セットに出力されたエラー・メッセージを調べて間違いを訂正し、このジョブを再実行してください。

予約語テーブルを作成する JCL の実行

予約語テーブルを作成する JCL には、STEP1、STEP2、および STEP3 があります。

この 3 つのステップでは、それぞれ以下のタスクを実行します。

1. 変更済みテーブルを使用して予約語テーブル・ユーティリティーを実行する。
2. 変更済み予約語テーブルをアセンブルする。
3. ランタイム・プログラム・オブジェクトをオブジェクト・モジュールから作成する。

このジョブを実行すると、予約語テーブルが作成されます。この新しいテーブルは、ユーザー指定のライブラリーに入れられ、IGYC の後にユーザー指定の 1 から 4 文字の ID を加えた名前が付けられます。

インストール先でのカタログ式プロシージャーの調整

インストール先の用途に応じて、 カタログ式プロシージャー IGYWC、IGYWCL、IGYWCLG を 調整しなければならない場合があります。

以下の変更を検討してください。

- Enterprise COBOL または Language Environment のターゲット・データ・セットに IBM 提供のものと異なる接頭部を使用した場合、データ・セット名接頭部を変更する。
- LNKLST 連結に IGY.V6R3M0.SIGYCOMP、CEE.SCEERUN、および CEE.SCEERUN2 を入れた場合は、STEPLIB DD ステートメントを除去する。
- 正常に実行されるためには GO ステップのデフォルトの領域サイズよりも大きな領域を必要とするプログラムが大部分を占める環境の場合、デフォルトの領域サイズを変更する。
- UNIT= パラメーターを変更する。

付録 A Enterprise COBOL for z/OS のアクセシビリティー機能

アクセシビリティー機能は、運動や視覚などに障害を持つユーザーが情報技術製品を快適に使用できるように支援します。 z/OS のアクセシビリティー機能は、Enterprise COBOL for z/OS のアクセスを支援します。

アクセシビリティー機能

z/OS は、以下のような主要アクセシビリティー機能を備えています。

- スクリーン・リーダーおよび画面拡大機能ソフトウェアで一般的に使用されるインターフェース
- キーボードのみによるナビゲーション
- 色、コントラスト、フォント・サイズなどの表示属性をカスタマイズする機能

z/OS では、US Section 508 (https://www.access-board.gov/ict/) および Web Content Accessibility Guidelines (WCAG) 2.0 (http://www.w3.org/TR/WCAG20/) に確実に準拠するために、最新の W3C 標準である WAI-ARIA 1.0 (http://www.w3.org/TR/wai-aria/) を使用しています。アクセシビリティー機能を利用するには、最新リリースのスクリーン・リーダーを、この製品でサポートされる最新の Web ブラウザーと併用してください。

IBM Knowledge Center の Enterprise COBOL for z/OS オンライン製品資料は、アクセシビリティーに対応しています。 IBM Knowledge Center のアクセシビリティー機能については、http://www.ibm.com/support/knowledgecenter/en/about/releasenotes.html に説明があります。

キーボード・ナビゲーション

ユーザーは、TSO/E または ISPF を使用して z/OS ユーザー・インターフェースにアクセスできます。

ユーザーは、 IBM Developer for z/OS を使用して、z/OS サービスにアクセスすることもできます。

これらのインターフェースへのアクセスに関する情報については、以下の資料を参照してください。

- *z/OS TSO/E Primer* (http://publib.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/ikj4p120)
- *z/OS TSO/E User's Guide* (http://publib.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/ikj4c240/APPENDIX1.3)
- *z/OS ISPF User's Guide Volume I* (http://publib.boulder.ibm.com/cgi-bin/bookmgr/BOOKS/ispzug70)
- IBM Developer for z/OS Knowledge Center (http://www.ibm.com/support/knowledgecenter/SSQ2R2/rdz_welcome.html?lang=en)

上記の資料には、キーボード・ショートカットまたはファンクション・キー (PF キー) の使用方法を含む TSO/E および ISPF の使用方法が記載されています。 それぞれの資料では、PF キーのデフォルトの設定値とそれらの機能の変更方法についても説明しています。

インターフェース情報

Enterprise COBOL for z/OS のオンライン製品資料は、IBM Knowledge Center で入手でき、標準の Web ブラウザーで表示できます。

PDF ファイルでのアクセシビリティー・サポートは限定的です。 PDF 資料では、オプションのフォント拡大機能およびハイコントラスト表示設定を使用でき、キーボードのみでナビゲートできます。

スクリーン・リーダーで、ピリオドやコンマなどの PICTURE 記号を含む構文図、ソース・コード例、およびテキストを正確に読み上げるには、すべての句読点を読み上げるようにスクリーン・リーダーを設定する必要があります。

支援技術製品は、z/OS のユーザー・インターフェースと連動します。 具体的なガイダンス情報については、z/OS インターフェースへのアクセスに使用する支援技術製品の資料を参照してください。

関連アクセシビリティー情報

標準の IBM ヘルプ・デスクとサポート Web サイトに加え、IBM は、聴覚が不自由なお客様が営業やサポート・サービスにアクセスするために使用できる TTY 電話サービスを立ち上げました。

TTY サービス
800-IBM-3383 (800-426-3383)
(北米内)

IBM およびアクセシビリティー

IBM のアクセシビリティーへの取り組みについて詳しくは、IBM Accessibility (www.ibm.com/able) を参照してください。

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あります。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 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

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。

〒 103-8510
東京都中央区日本橋箱崎町 19 番 21 号
日本アイ・ビー・エム株式会社
法務・知的財産
知的財産権ライセンス渉外

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。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。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。

Intellectual Property Dept. for Rational Software
IBM Corporation
5 Technology Park Drive
Westford, MA 01886
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、その製品の供給者、出版物、もしく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、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

IBM の将来の方向または意向に関する記述については、予告なしに変更または撤回される場合があり、単に目標を示しているものです。

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した創作物には、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) (西暦年). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。© Copyright IBM Corp. 1996, 2020.

プライバシー・ポリシーに関する考慮事項:

サービス・ソリューションとしてのソフトウェアも含めた IBM ソフトウェア製品 (「ソフトウェア・オファリング」) では、製品の使用に関する情報の収集、エンド・ユーザーの使用感の向上、エンド・ユーザーとの対話またはその他の目的のために、Cookie はじめさまざまなテクノロジーを使用することがあります。多くの場合、ソフトウェア・オファリングにより個人情報が収集されることはありません。 IBM の「ソフトウェア・オファリング」の一部には、個人情報を収集できる機能を持つものがあります。 ご使用の「ソフトウェア・オファリング」が、これらの Cookie およびそれに類するテクノロジーを通じてお客様による個人情報の収集を可能にする場合、以下の具体的事項をご確認ください。

この「ソフトウェア・オファリング」は、Cookie もしくはその他のテクノロジーを使用して個人情報を収集することはありません。

この「ソフトウェア・オファリング」が Cookie およびさまざまなテクノロジーを使用してエンド・ユーザーから個人を特定できる情報を収集する機能を提供する場合、 お客様は、このような情報を収集するにあたって適用される法律、ガイドライン等を遵守する必要があります。これには、エンドユーザーへの通知や同意の要求も含まれますがそれらには限られません。

このような目的での Cookie を含む様々なテクノロジーの使用の詳細については、『IBM オンラインでのプライバシー・ステートメント』(http://www.ibm.com/privacy/details/jp/ja/) の『クッキー、ウェブ・ビーコン、その他のテクノロジー』および「IBM Software Products and Software-as-a-Service Privacy Statement」(http://www.ibm.com/software/info/product-privacy) を参照してください。

プログラミング・インターフェース情報

IBM Enterprise COBOL for z/OS では、お客様のインストール済み環境で IBM Enterprise COBOL for z/OS のサービスを使用するプログラムを作成するためのマクロは提供されていません。

重要 : IBM Enterprise COBOL for z/OS マクロをプログラミング・インターフェースとして使用しないでください。

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。

Java およびすべての Java 関連の商標およびロゴは Oracle やその関連会社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。

用語集

この用語集に記載されている用語は、COBOL における意味に従って定義されています。 これらの用語は、他の言語では同じ意味を持つことも、持たないこともあります。

この用語集には、以下の資料からの用語および定義が記載されています。

- 「*ANSI INCITS 23-1985, Programming languages - COBOL*」(「*ANSI INCITS 23a-1989, Programming Languages - COBOL - Intrinsic Function Module for COBOL*」および「*ANSI INCITS 23b-1993, Programming Languages - Correction Amendment for COBOL*」で改訂)
- 「*ISO 1989:1985, Programming languages - COBOL*」は「*ISO/IEC 1989/AMD1:1992, Programming languages - COBOL: Intrinsic function module*」および「*ISO/IEC 1989/AMD2:1994, Programming languages - Correction and clarification amendment for COBOL*」に改訂されました。
- 「*ANSI X3.172-2002, American National Standard Dictionary for Information Systems*」
- *INCITS/ISO/IEC 1989-2002, Information technology - Programming languages - COBOL*
- *INCITS/ISO/IEC 1989:2014, Information technology - Programming languages, their environments and system software interfaces - Programming language COBOL*

米国標準規格 (ANS) の定義の前にはアスタリスク (*) を付けています。

A

簡略複合比較条件 (* abbreviated combined relation condition)
連続する一連の比較条件の中で、 共通のサブジェクトまたは共通のサブジェクトと共通の比較演算子を明示的に省略したことにより生ずる複合条件。

異常終了 (abend)
プログラムの異常終了。

16 MB 境界より上 (above the 16 MB line)
いわゆる 16 MB 境界より上で、2 GB バーより下のストレージ。 このストレージは、31 ビット・モードでのみアドレス可能である。 1980 年代に IBM が MVS™/XA アーキテクチャーを導入するまで、プログラムの仮想ストレージは 16MB に制限されていました。24 ビット・モードでコンパイルされたプログラムは、架空のストレージ境界線の下に保持されているかのように、16MB のスペースに対してのみアドレッシングが可能です。 VS COBOL II 以降、31 ビット・モードでコンパイルされたプログラムは、16 MB 境界より上に配置することができます。

アクセス・モード (* access mode)
ファイル内のレコードを操作するに当たって使用する方式。

実際の小数点 (* actual decimal point)
10 進小数点文字のピリオド (.) またはコンマ (,) によるデータ項目における小数点位置の物理表現。

実際の文書エンコード (actual document encoding)
XML 文書のエンコード・カテゴリーで、以下のいずれかとなる。XML パーサーは文書の最初の数バイトを調べて判別する。

- ASCII
- EBCDIC
- UTF-8
- UTF-16 (ビッグ・エンディアンまたはリトル・エンディアンのいずれか)
- これ以外のサポートされないエンコード
- 認識不能なエンコード

英字名 (* alphabet-name)
`ENVIRONMENT DIVISION` の `SPECIAL-NAMES` 段落のユーザー定義語であり、特定の文字セットまたは照合シーケンス (あるいはその両方) に名前を割り当てるもの。

英字 (* alphabetic character)
英字またはスペース文字。

英数字位置 (alphanumeric character position)
「文字位置 *(character position)*」を参照。

英字データ項目 (alphabetic data item)
記号 A のみを含む PICTURE 文字ストリングが記述されたデータ項目。英字データ項目は USAGE DISPLAY を持ちます。

英数字 (* alphanumeric character)
コンピューターの 1 バイト文字セットの任意の文字。

英数字データ項目 (alphanumeric data item)
暗黙的または明示的に USAGE DISPLAY として記述された、カテゴリー英数字、英数字編集、または数字編集を持つデータ項目を 指す一般的な呼び方。

英数字編集データ項目 (alphanumeric-edited data item)
少なくとも 1 つの記号 A または X のインスタンスおよび少なくとも 1 つの単純挿入記号 B、0、または / を含んでいる、PICTURE 文字ストリングで記述されたデータ項目。 英数字編集データ項目は USAGE DISPLAY を持ちます。

英数字関数 (* alphanumeric function)
コンピューターの英数字セットからの 1 つ以上の文字のストリングで値が構成されている関数。

英数字グループ項目 (alphanumeric group item)
GROUP-USAGE NATIONAL 節なしで定義されたグループ項目。 INSPECT、STRING、および UNSTRING などの操作の場合、英数字グループ項目は、実際のグループの内容にかかわらず、その内容すべてが USAGE DISPLAY として記述されているかのように処理されます。 グループ内の基本項目を処理する必要のある操作 (MOVE CORRESPONDING、ADD CORRESPONDING、または INITIALIZE など) の場合、英数字グループ項目はグループ・セマンティクスを使用して処理されます。

英数字リテラル (alphanumeric literal)
次のセットからの開始区切り文字を有するリテラル。'、"、X'、X"、Z'、または Z"。 この文字ストリングには、コンピューターの有する文字セットの任意の文字を含めることができる。

代替レコード・キー (* alternate record key)
基本レコード・キー以外のキーで、 その内容が索引付きファイルの中のレコードを識別するもの。

米国規格協会 (American National Standards Institute: ANSI)
米国で認定された組織が自発的工業規格を作成して維持する手順を設定する組織であり、製造業者、消費者、および一般の利害関係者で構成される。

引数 (argument)
(1) ID、リテラル、算術式、または関数 ID で、これにより関数の評価に使用する値を指定する。 (2) CALL または INVOKE ステートメント の USING 句のオペランドであり、呼び出されたプログラムまたは起動されたメソッドに値を渡すのに使用されます。

算術式 (* arithmetic expression)
数値リテラル、数値基本項目 (算術演算子で区切られた ID とリテラルなど) を表す ID、1 つの算術演算子で区切られた 2 つの算術式、または括弧で囲まれた算術式。

算術演算 (* arithmetic operation)
ある算術ステートメントが実行されることにより、 またはある算術式が計算されることにより生じるプロセスで、 そこで指定された引数に対して数学的に正しい解が求められる。

算術演算子 (* arithmetic operator)
次に示す集合に属する 1 文字、または 2 文字で構成された固定した組み合わせ。

文字	意味
+	加算
-	減算
*	乗算
/	除算

文字	意味
**	指数

算術ステートメント (* arithmetic statement)
算術演算を実行させるステートメント。 算術ステートメントには、ADD、COMPUTE、DIVIDE、MULTIPLY、および SUBTRACT の各ステートメントがある。

配列 (array)
データ・オブジェクトで構成される集合体。それぞれのオブジェクトは添え字付けによって一意的に参照できる。 配列は、COBOL ではテーブルに類似する。

昇順キー (* ascending key)
データ項目を比較する際の規則に一致するように、最低のキー値から始めて最高のキー値へとデータを順序付けている値に即したキー。

ASCII
情報交換用米国標準コード。 7 ビットのコード化文字をベースとする 1 つのコード化文字セット (パリティー・チェックを含む 8 ビット) を 使用する標準コードであり、データ処理システム、データ通信システム、および関連装置の間での情報交換に使用される。 ASCII セットは、制御文字と図形文字から構成されている。

IBM は、ASCII に対する拡張 (文字 128 から 255) を定義している。

ASCII DBCS
「*2 バイト ASCII (double-byte ASCII)*」を参照。

割り当て名 (assignment-name)
COBOL ファイルの編成を識別する名前で、 システムがこれを認識する際に使用する。

想定小数点 (* assumed decimal point)
データ項目の中に実際には小数点のための文字が入っていない小数点位置。 想定小数点には、論理的な意味があり、物理的には表現されない。

AT END 条件 (AT END condition)
次のような特定の条件のもとで、READ、RETURN、または SEARCH ステートメントを実行した場合に引き起こされる条件。

- 順次アクセス・ファイルに対して READ ステートメントを実行中に、そのファイル内に次の論理レコードが存在しない場合、または相対レコード番号中の有効数字の桁数が相対キー・データ項目のサイズより大きい場合、またはオプションの入力ファイルが使用可能でない場合。
- RETURN ステートメントの実行中に、関連するソート・ファイルまたはマージ・ファイルについての次の論理レコードが存在しない場合。
- SEARCH ステートメントの実行中に、関連する WHEN 句のいずれかで指定された条件を満足することなく、検索操作が終了した場合。

B

基本文字セット (basic character set)
言語のワード、文字ストリング、および区切り文字の作成時に使用される基本的な文字セット。 基本文字セットは 1 バイトの EBCDIC でインプリメントされる。 拡張文字セットには DBCS 文字が含まれる。これは、コメント、リテラル、およびユーザー定義語で使用できる。

85 COBOL 標準 における「*COBOL* 文字セット *(COBOL character set)*」と同義。

ビッグ・エンディアン (big-endian)
メインフレームおよび AIX® ワークステーションがバイナリー・データおよび UTF-16 文字を保存するときに使用するデフォルト形式。 この形式では、バイナリー・データ項目の最下位バイトが最高位アドレスにあり UTF-16 文字の最下位バイトが最高位アドレスにあります。リトル・エンディアン *(little-endian)* と比較。

バイナリー項目 (binary item)
2 進表記 (基数 2 の数体系) で表される数値データ項目。 等価の 10 進数は、10 進数字 0 から 9 に演算符号を加えたもので構成される。 項目の左端のビットは、演算符号。

二分探索 (binary search)
二分探索の各段階では、 データ・エレメント集合が 2 つに分割される。 数が奇数の場合は適切なアクションが取られる。

ブロック (* block)
通常は 1 つ以上の論理レコードで構成される物理的データ単位。 大容量記憶ファイルの場合、ある論理レコードの一部がブロックに入ることがある。 ブロックのサイズは、そのブロックが含まれているファイルのサイズと直接関係はなく、 そのブロックに含まれているか、そのブロックにオーバーラップしている論理レコードのサイズとも直接関係はない。「物理レコード *(physical record)*」と同義。

ブール条件 (boolean condition)
ブール条件は、ブール・リテラルが true であるか false であるかを決定する。 ブール条件は定数条件式でのみ使用できる。

ブール・リテラル (boolean literal)
true 値を示す B'1'、または false 値を示す B'0' のどちらか。 ブール・リテラルは定数条件式でのみ使用できる。

停止点 (breakpoint)
通常は命令によって指定されるコンピューター・プログラムの場所であり、プログラムの実行は外部からの介入またはモニター・プログラムによって割り込まれる場合がある。

バッファー (buffer)
入力データまたは出力データを一時的に保持するために使用されるストレージの一部分。

組み込み関数 (built-in function)
組み込み関数 *(intrinsic function)* を参照。

ビジネス・メソッド (business method)
ビジネス・ロジックまたはアプリケーションのルールをインプリメントする Enterprise Bean のメソッド。 (Oracle)

バイト (byte)
特定の数のビット (通常 8 ビット) から成るストリングであり、1 つの単位として処理され、1 つの文字または制御機能を表す。

バイト・オーダー・マーク (BOM) (byte order mark (BOM))
UTF-16 または UTF-32 テキストの先頭に使用して、後続テキストのバイト・オーダーを示す Unicode 文字。バイト・オーダーには、「ビッグ・エンディアン (big-endian)」または「リトル・エンディアン (little-endian)」がある。

バイトコード (bytecode)
Java コンパイラーによって生成され、Java インタープリターによって実行される、マシンから独立したコード。(Oracle)

C

呼び出し可能サービス (callable services)
言語環境プログラムでは、従来の言語環境プログラム定義の呼び出しインターフェースを使用して COBOL プログラムが呼び出すことができる一連のサービス。 言語環境プログラムの規約を共有するすべてのプログラムがこれらのサービスを使用できる。

呼び出し先プログラム (called program)
CALL ステートメントの対象となるプログラム。 呼び出し先プログラムと呼び出し側プログラムが実行時に結合されて、1 つの実行単位が作成される。

呼び出し側プログラム (* calling program)
別のプログラムへの CALL を実行するプログラム。

標準分解 (canonical decomposition)
複数の Unicode 文字を使用して単一の合成済み Unicode 文字を表す方法。 通常、標準分解は、ラテン語文字と発音区別符号が個別に表されるように発音区別符号付きのラテン語文字を分離するために使用される。 合成済み Unicode 文字とその標準分解を表す例については、合成済み文字 *(precomposed character)* を参照。

ケース構造 (case structure)
結果として生じた多数のアクションの中から選択を行うために、一連の条件をテストするプログラム処理ロジック。

カタログ式プロシージャー (cataloged procedure)
プロシージャー・ライブラリー (SYS1.PROCLIB) と呼ばれる区分データ・セットに置かれた一連のジョブ制御ステートメント。 カタログ式プロシージャーを使用すると、JCL をコーディングする時間を節約して、エラーを減らすことができる。

CCSID
コード化文字セット *ID (coded character set identifier)* を参照。

世紀ウィンドウ (century window)
2 桁年号が固有に決まる 100 年間のこと。 COBOL プログラマーが使用できる世紀ウィンドウには、いくつかのタイプがある。

- ウィンドウ操作組み込み関数 DATE-TO-YYYYMMDD、DAY-TO-YYYYDDD、および YEAR-TO-YYYY については、引数*-2 (argument-2)* によって世紀ウィンドウを指定する。
- 言語環境プログラム呼び出し可能サービスについては、CEESCEN で世紀ウィンドウを指定する。

文字 (* character)
言語のそれ以上分割できない基本単位。

文字エンコード・ユニット (character encoding unit)
コード化文字セット内の 1 つのコード・ポイントに相当するデータの単位。 1 つ以上の文字エンコード・ユニットを使用して、コード化文字セットの文字が表現される。 エンコード・ユニット とも呼ばれる。

USAGE NATIONAL の場合、文字エンコード・ユニットは、 UTF-16 の 2 バイト・コード・ポイントに対応している。

USAGE DISPLAY の場合、文字エンコード・ユニットは、1 つのバイトに対応している。

USAGE DISPLAY-1 の場合、文字エンコード・ユニットは、 DBCS 文字セットの 2 バイト・コード・ポイントに対応している。

文字位置 (character position)
1 文字を保持または表示するために必要な物理ストレージまたは表示スペースの量。 この用語はどのような文字のクラスにも適用される。 文字の特定のクラスについては、以下の用語が適用される。

- 英数字文字位置。USAGE DISPLAY を使用して表される DBCS 文字。
- *DBCS* 文字位置。USAGE DISPLAY-1 を使用して表される DBCS 文字。
- 国別文字位置。USAGE NATIONAL を使用して表される文字。UTF-16 の 文字エンコード・ユニットと同義。

文字セット (character set)
テキスト情報を表すために使用されるエレメントの集合。 ただし、コード化表現は想定されていません。 コード化文字セット *(coded character set)* も参照。

文字ストリング (character string)
COBOL ワード、リテラル、PICTURE 文字ストリング、またはコメント記入項目を形成する一連の連続した文字。 文字ストリングは区切り文字で区切らなければならない。

チェックポイント (checkpoint)
ジョブ・ステップを後で再始動することができるように、 ジョブとシステムの状況に関する情報を記録しておくことができるところ。

クラス (* class)
ゼロ、1 つ、または複数のオブジェクトの共通の動作およびインプリメンテーション を定義するエンティティー。 同じ具体化を共用するオブジェクトは、同じクラスのオブジェクトとみなされる。 クラスは階層として定義でき、あるクラスを別のクラスから継承することができる。

クラス (オブジェクト指向) (class (object-oriented))
ゼロ、1 つ、または複数のオブジェクトの共通の動作およびインプリメンテーション を定義するエンティティー。 同じ具体化を共用するオブジェクトは、同じクラスのオブジェクトとみなされる。

クラス条件 (* class condition)
項目の内容がすべて英字であるか、すべて数字であるか、 すべて DBCS であるか、すべて漢字であるか、あるいはクラス名の定義においてリストされた文字だけで構成されるかという命題で、それに関して真の値を判別することができる。

クラス定義 (* class definition)
クラスを定義する COBOL ソース単位。

クラス階層 (class hierarchy)
オブジェクト・クラス間の関係を示すツリーのような構造。 最上部に 1 つのクラスが置かれ、その下に 1 つ以上のクラスの層が置かれる。 「継承階層 *(inheritance hierarchy)*」と同義。

クラス識別記入項目 (* class identification entry)
IDENTIFICATION DIVISION の CLASS-ID 段落内の 記入項目であり、クラス名を指定する節と、選択した属性をクラス定義に割り当てる節を含む。

クラス名 (オブジェクト指向) (class-name (object-oriented))
オブジェクト指向 COBOL クラス定義の名前。

クラス名 (データの) (* class-name (of data))
ENVIRONMENT DIVISION の SPECIAL-NAMES 段落で定義されるユーザー定義語であり、真理値を定義できる命題に名前を割り当てる。データ項目の内容は、クラス名の定義にリストされている文字だけで構成される。

クラス・オブジェクト (class object)
クラスを表す実行時オブジェクト。

節 (* clause)
記入項目の属性を指定するという目的で順番に並べられた連続する COBOL 文字ストリング。

クライアント (client)
オブジェクト指向プログラミングにおいて、クラス内の 1 つ以上のメソッドからサービスを要求するプログラムまたはメソッド。

COBOL 文字セット (COBOL character set)
COBOL 構文を作成する際に使用される文字セット。 完全な COBOL 文字セットは、以下の文字で構成される。

文字	意味
0,1, . . . ,9	数字
A,B, . . . ,Z	英大文字
a,b, . . . ,z	英小文字
	スペース
+	正符号
-	負符号 (-) (ハイフン)
*	アスタリスク
/	斜線 (スラッシュ)
=	等号
$	通貨符号
,	コンマ
;	セミコロン
.	ピリオド (小数点、終止符)
"	引用符
'	アポストロフィ
(	左括弧

文字	意味
)	右括弧
>	より大きい
<	より小さい
:	コロン
_	下線

COBOL ワード (* COBOL word)
ワード *(word)* を参照。

コード・ページ (code page)
すべてのコード・ポイントに図形文字および制御機能の意味を割り当てるもの。 例えば、あるコード・ページでは、8 ビット・コードに対して 256 コード・ポイントに文字と意味を割り当て、別のコード・ページでは、7 ビット・コードに対して 128 コード・ポイントに文字と意味を割り当てることができる。 ワークステーション上の英語の IBM コード・ページは IBM-1252 で、ホストは IBM-1047 である。コード化文字セット *(coded character set)*。

コード・ポイント (code point)
コード化文字セット (コード・ページ) に定義する固有のビット・パターン。 コード・ポイントには、グラフィック・シンボルおよび制御文字が割り当てられる。

コード化文字セット (coded character set)
文字セットを設定し、その文字セットの文字とコード化表現との間の関係を設定する明確な規則の集まり。 コード化文字セットの例として、ASCII もしくは EBCDIC コード・ページで、または Unicode 対応の UTF-16 エンコード・スキームで表す文字セットがある。

コード化文字セット ID (CCSID) (coded character set identifier (CCSID))
特定のコード・ページを識別する 1 から 65,535 までの IBM 定義番号。

照合シーケンス (* collating sequence)
コンピューターに受け入れ可能な文字のシーケンスで、 ソート、マージ、比較、および索引付きファイルの順次処理を目的として順序付けしたもの。

列 (* column)
印刷行または参照形式行におけるバイト位置。 列は、行の左端の位置から始めて行の右端の位置まで、1 から 1 ずつ増やして番号が付けられる。 列は 1 つの 1 バイト文字を保持する。

複合条件 (* combined condition)
2 つ以上の条件を AND または OR 論理演算子で結合した結果生じる条件。 条件 *(condition)* および 複合否定条件 *(negated combined condition)* も参照。

結合文字 (combining characters)
他の前後の Unicode 文字を変更するために使用される Unicode 文字。 通常、結合文字は、ラテン語文字を変更するために使用される Unicode 発音区別符号。 合成済み文字 *(precomposed character)* を参照して、ラテン語文字 U+0061 (a) とともに使用される結合文字 U+0308 (¨) の例を確認すること。

コメント記入項目 (* comment-entry)
ドキュメンテーションに使用される IDENTIFICATION DIVISION 内の項目で、実行に影響しない。

コメント行 (comment line)
行の標識区域のアスタリスク (*) と、または、プログラム・テキスト区域 (区域 A および区域 B) の最初の文字ストリングとしてのアスタリスクと大なり記号 (*>) と、 その行の区域 A および区域 B に続くコンピューターの文字セットの任意の文字によって表されるソース・プログラム行。 コメント行は、文書化にのみ役立つ。 行の標識区域では斜線 (/)、そしてその行の区域 A および B ではコンピューター文字セットの任意の文字で表される特殊形式のコメント行があると、コメントの印刷前に改ページが行われる。

共通プログラム (* common program)
別のプログラムに直接的に含まれているにもかかわらず、その別のプログラムに直接的または間接的に含まれている任意のプログラムから呼び出すことができるプログラム。

コンパイル (* compile)
(1) 高水準言語で表現されたプログラムを、中間言語、アセンブリー言語、または コンピューター言語で表現されたプログラムに変換すること。 (2) あるプログラミング言語で書かれたコンピューター・プログラムから、プログラムの全体的なロジック構造を利用することによって、または 1 つの記号ステートメントから複数のコンピューター命令を作り出すことによって、またはアセンブラーの機能のようにこれら両方を使用することによって、マシン言語プログラムを生成すること。

コンパイル変数 (compilation variable)
ある特定のリテラル値のシンボル名、あるいは DEFINE ディレクティブまたは DEFINE コンパイラー・オプションによって指定されたコンパイル時演算式のシンボル名。

コンパイル時 (* compile time)
COBOL コンパイラーによって、COBOL ソース・コードが COBOL オブジェクト・プログラムに変換される時間。

コンパイル時演算式 (compile-time arithmetic expression)
DEFINE ディレクティブおよび EVALUATE ディレクティブに、または定数条件式に指定されている算術式のサブセット。 コンパイル時演算式における、正規演算式との違い:

- 指数演算子を指定することはできません。
- オペランドはすべて、整数リテラルか、すべてのオペランドが整数リテラルである演算式でなければなりません。
- 式はゼロによる除算が発生しないように指定する必要があります。

コンパイラー (compiler)
高水準言語で記述されたソース・コードをマシン言語のオブジェクト・コードに変換するプログラム。

コンパイラー指示ステートメント (compiler-directing statement)
コンパイル時にコンパイラーに特定の処置を行わせるステートメント。 標準コンパイラー指示ステートメントには、 COPY、REPLACE、および USE がある。

複合条件 (* complex condition)
1 つ以上の論理演算子が 1 つ以上の条件に基づいて作動する条件。 条件 *(condition)*、単純否定条件 *(negated simple condition)*、および 複合否定条件 *(negated combined condition)* も参照。

複合 ODO (complex ODO)
次のような OCCURS DEPENDING ON 節の特定の形式。

- 可変位置項目またはグループ: DEPENDING ON オプションを指定した OCCURS 節によって記述されたデータ項目の後に、非従属データ項目またはグループが続く。 グループは英数字グループでも国別グループでも構いません。
- 可変位置テーブル: DEPENDING ON オプションを指定した OCCURS 節によって記述されたデータ項目の後に、OCCURS 節に よって記述された非従属データ項目が続く。
- 可変長エレメントを持つテーブル: OCCURS 節によって記述されたデータ項目に、DEPENDING ON オプションを指定した OCCURS 節によって記述された従属データ項目が含まれている。
- 可変長エレメントを持つテーブルの指標名。
- 可変長エレメントを持つテーブルのエレメント。

コンポーネント (component)
(1) 関連ファイルからなる機能グループ化。 (2) オブジェクト指向プログラミングでは、特定の機能を実行し、他のコンポーネントやアプリケーションと連携するように設計されている、再使用可能なオブジェクトまたはプログラム。 JavaBeans は、コンポーネントを作成するための、Oracle が提供するアーキテクチャーである。

合成形式 (composed form)
標準分解による合成済み Unicode 文字の表記。 詳しくは、合成済み文字 *(precomposed character)* を参照。

コンピューター名 (* computer-name)
プログラムがコンパイルまたは実行されるコンピューターを識別するシステム名。

条件 (例外) (condition (exception))
言語環境プログラムによって使用可能にされる、あるいは認識される例外。したがって、ユーザー条件処理ルーチンと言語条件処理ルーチンの活動化に適している。 アプリケーションの通常のプログラミングされたフローを変えるもの。 条件は、ハードウェアまたはオペレーティング・システムによって検出され、その結果、割り込みが起こる。 このほかにも、条件は言語特定の生成コードまたは言語ライブラリー・コードによっても検出できる。

条件 (式) (condition (expression))
ある真の値が決定される実行時のデータの状況。 条件という用語が本書で 一般形式の「条件」(条件-*1*、条件-*2*、...) の中、またはこれに関連して使用された場合は、. .) 次のいずれかである。オプションとして括弧で囲まれた単純条件からなる条件式、あるいは、単純条件、論理演算子、および括弧の構文的に正しい組み合わせ (真理値を判別できる) からなる複合条件。 単純条件 *(simple condition)*、複合条件 *(complex condition)*、単純否定条件 *(negated simple condition)*、複合条件 *(combined condition)*、および 複合否定条件 *(negated combined condition)* も参照。

条件式 (* conditional expression)
EVALUATE、IF、PERFORM、または SEARCH ステートメントの中で指定される単純条件または複合条件。 単純条件 *(simple condition)* および 複合条件 *(complex condition)* も参照。

条件句 (* conditional phrase)
ある条件ステートメントが実行された結果得られる条件の真理値の判別に基づいてとられるべき処置を指定する句。

条件ステートメント (* conditional statement)
条件の真理値を判別することと、オブジェクト・プログラムの次の処理がこの真理値によって決まることを指定するステートメント。

条件変数 (* conditional variable)
1 つまたは複数の値を持つデータ項目であり、これらの値が、そのデータ項目に割り当てられた条件名を持つ。

条件名 (* condition-name)
条件変数が想定できる値のサブセットに名前を割り当てるユーザー定義語。または、インプリメントする人が定義したスイッチまたは装置の状況に割り当てられるユーザー定義語。

条件名条件 (* condition-name condition)
真理値を判別できる命題で、かつ、条件変数の値が、その条件変数と関連する条件名に属する一連の値のメンバーである命題。

*** CONFIGURATION SECTION**
ENVIRONMENT DIVISION のセクションであり、ソース・プログラムとオブジェクト・プログラムの全体的な仕様およびクラス定義を記述する。

CONSOLE
オペレーター・コンソールに関連する COBOL 環境名。

定数条件式 (constant conditional expression)
IF ディレクティブで、または EVALUATE ディレクティブの WHEN 句で使用される可能性がある条件式のサブセット。

定数条件式は、以下の項目のいずれかでなければなりません。

- 両方のオペランドがリテラルであるか、リテラル項のみを含む算術式である比較条件。条件は比較条件の規則に従う必要があり、以下の追加事項があります。
  - オペランドは同じカテゴリーでなければなりません。算術式は数値カテゴリーです。
  - リテラルが指定され、それらが数値リテラルではない場合、関係演算子は “IS EQUAL TO”、“IS NOT EQUAL TO”、“IS =”、“IS NOT =”、または “IS <>” でなければなりません。

  比較条件 *(relation condition)* も参照。
- 定義済み条件。定義済み条件 *(defined condition)* も参照。
- ブール条件。 ブール条件 *(boolean condition)* も参照。

- 前述の単純条件の形式を、AND、OR、および NOT を使用して複合条件に結合することによって形成した複合条件。 簡略複合比較条件を指定することはできません。複合条件 *(complex condition)* も参照。

含まれているプログラム (contained program)
別の COBOL プログラムにネストされている COBOL プログラム。

連続項目 (* contiguous items)
DATA DIVISION 内の連続する記入項目によって記述され、相互に一定の階層関係を持っている項目。

コピーブック (copybook)
一連のコードが含まれたファイルまたはライブラリー・メンバーであり、コンパイル時に COPY ステートメントを使用してソース・プログラムに組み込まれる。 ファイルはユーザーが作成する場合、COBOL によって提供される場合、 または他の製品によって供給される場合とがある。 「コピー・ファイル *(copy file)*」と同義。

カウンター (* counter)
他の数字を使ってその数字分だけ増減したり、 あるいは 0 または任意の正もしくは負の値に変更またはリセットしたりできるようにした、 数または数表現を収めるために使用されるデータ項目。

相互参照リスト (cross-reference listing)
コンパイラー・リストの一部であり、プログラム内においてファイル、フィールド、および標識が定義、参照、および変更される場所に関する情報が入る。

通貨記号値 (currency-sign value)
数字編集項目に保管される通貨単位を識別する文字ストリング。 典型的な例としては、$、USD、EUR などがある。 通貨記号値は、CURRENCY コンパイラー・オプションで定義するか、ENVIRONMENT DIVISION の SPECIAL-NAMES 段落内の CURRENCY SIGN 節によって定義することができる。 CURRENCY SIGN 節が指定されない場合、NOCURRENCY コンパイラー・オプションが有効であれば、ドル記号 ($) がデフォルトの通貨記号値として使用される。 通貨記号 *(currency symbol)* も参照。

通貨記号 (currency symbol)
数字編集項目内の通貨記号値の部分を示すために、PICTURE 節で使用される文字。 通貨記号は、CURRENCY コンパイラー・オプションで定義するか、ENVIRONMENT DIVISION の SPECIAL-NAMES 段落内の CURRENCY SIGN 節によって定義することができる。 CURRENCY SIGN 節が指定されない場合、NOCURRENCY コンパイラー・オプションが有効であれば、ドル記号 ($) がデフォルトの通貨記号値および通貨記号として使用される。 通貨記号と通貨符号値は複数定義可能。 通貨記号値 *(currency sign value)* も参照。

現行レコード (* current record)
ファイル処理において、 ファイルに関連するレコード域の中で使用可能なレコード。

現行ボリューム・ポインター (* current volume pointer)
順次ファイルの現行のボリュームを指している概念上のエンティティー。

D

データ節 (* data clause)
COBOL プログラムの DATA DIVISION のデータ記述記入項目に現れる節で、データ項目の特定の属性を記述する情報を提供する。

データ記述項目 (* data description entry)
COBOL プログラムの DATA DIVISION 内の記入項目であり、レベル番号の後に必要に応じてデータ名が続き、その後に必要に応じて一連のデータ節で構成されるもの。

DATA DIVISION
COBOL プログラムまたはメソッドの 1 つの部 (division)。 使用するファイルとファイルに含まれるレコード、必要となる内部 WORKING-STORAGE レコード、COBOL 実行単位内の複数のプログラムで使用可能なデータなど、プログラムまたはメソッドで処理するデータを記述する。

データ項目 (* data item)
COBOL プログラムにより、または関数評価の規則により、定義されたデータの単位 (リテラルを除く)。

データ・セット (data set)
「ファイル *(file)*」の同義語。

データ名 (* data-name)
データ記述項目で記述されたデータ項目に名前を割り当てるユーザー定義語。 一般形式で使用された場合、データ名は、その形式の規則で特に許可されていない限り、参照変更、添え字付け、または修飾してはならないワードを表す。

DBCS
2 バイト文字セット *(double-byte character set (DBCS))* を参照

DBCS 文字 (DBCS character)
IBM 2 バイト文字セット (DBCS) に定義された任意の文字。

DBCS 文字位置 (DBCS character position)
文字位置 *(character position)* を参照。

DBCS データ項目 (DBCS data item)
少なくとも 1 つの記号 G または少なくとも 1 つの記号 N (NSYMBOL(DBCS) コンパイラー・オプションが有効なとき) を含んでいる PICTURE 文字ストリングで記述されたデータ項目。DBCS データ項目は USAGE DISPLAY-1 を持っています。

デバッグ行 (* debugging line)
行の標識区域に文字 D がある行のこと。

デバッグ・セクション (* debugging section)
USE FOR DEBUGGING ステートメントが含まれているセクション。

宣言文 (* declarative sentence)
区切り記号のピリオドによって終了する 1 つの USE ステートメントから構成されるコンパイラー指示文。

宣言部分 (* declaratives)
PROCEDURE DIVISION の先頭に書き込まれた 1 つ以上の特殊目的セクションの集合であり、その先頭にはキーワード DECLARATIVE が付き、その最後にはキーワード END DECLARATIVES が続いている。宣言部分は、セクション・ヘッダー、USE コンパイラー指示文、および 0 個、1 個、または複数個の関連する段落で構成される。

編集解除 (* de-edit)
項目の編集解除された数値を判別するために、数字編集データ項目からすべての編集文字を論理的に除去すること。

定義済み条件 (defined condition)
コンパイル変数が定義されているかどうかをテストするコンパイル時条件。 定義済み条件は、IF ディレクティブで、または EVALUATE ディレクティブの WHEN 句で指定される。

範囲区切りステートメント (* delimited scope statement)
明示的範囲終了符号を含んでいるステートメント。

区切り文字 (* delimiter)
1 つの文字、または一連の連続する文字であり、文字ストリングの終わりを識別し、 その文字ストリングを後続の文字ストリングから区切る。 区切り文字は、これを使用して区切られる文字ストリングの一部ではない。

従属領域 (dependent region)
IMS において、メッセージ・ドリブン・プログラム、バッチ・プログラム、またはオンライン・ユーティリティーを含む MVS 仮想記憶領域。

降順キー (* descending key)
データ項目を比較する際の規則に一致するように、 最高のキー値から始めて最低のキー値へとデータを順序付けている値に付けられるキー。

数字 (digit)
0 から 9 までの任意の数字。COBOL では、この用語を用いて他の記号を参照することはない。

桁位置 (* digit position)
1 つの桁を保管するために必要な物理ストレージの大きさ。 この大きさは、データ項目を定義するデータ記述項目に 指定された用途によって異なる。

直接アクセス (* direct access)
前回アクセスしたデータへの参照には依存せず、 プロセスがデータの位置にのみ依存する方法で、 データを記憶装置から取得したり、データを記憶装置に入れる機能。

表示浮動小数点データ項目 (display floating-point data item)
暗黙的または明示的に USAGE DISPLAY として記述されており、外部浮動小数点データ項目を記述する PICTURE 文字ストリングを持っている、データ項目。

部 (* division)
部の本体と呼ばれる、0 個、1 個、または複数個のセクションまたは段落の集合であり、特定の規則に従って形成および結合されたもの。 それぞれの部は、部のヘッダーおよび関連した部の本体で構成される。 COBOL プログラムには、見出し部、環境部、データ部、および手続き部の 4 つの部がある。

部の見出し (* division header)
ワードとその後に続く、部の先頭を示す分離文字ピリオドの組み合わせ。 部のヘッダーは次のとおり。

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.
```

DLL
ダイナミック・リンク・ライブラリー *(DLL) (dynamic link library (DLL))* を参照。

DLL アプリケーション (DLL application)
インポートしたプログラム、関数、または変数を参照するアプリケーション。

DLL リンケージ (DLL linkage)
DLL および NODYNAM オプションを 使用してコンパイルされたプログラム内の CALL。CALL は、別個のモジュール内のエクスポートされた名前に解決されるか、または、別個のモジュールに定義されたメソッドの INVOKE に解決される。

Do 構造 (do construct)
構造化プログラミングでは、DO ステートメントを使えば、プロシージャー内の複数のステートメントをグループ化できる。 COBOL では、インライン PERFORM ステートメントが同様に機能する。

do-until
構造化プログラミングにおいて、do-until ループは、少なくとも 1 回は実行され、所定の条件が真になるまで実行される。 COBOL では、TEST AFTER 句を PERFORM ステートメントで使用すれば、同様に機能する。

do-while
構造化プログラミングにおいて、do-while ループは、所定の条件が真である場合、およ び真である間に実行される。 COBOL では、TEST BEFORE 句を PERFORM ステートメントで使用すれば、同様に機能する。

文書タイプ宣言 (document type declaration)
あるクラスの文書に対する文法を規定するマークアップ宣言を含む、または 指示する XML エレメント。この文法は、文書タイプ定義または DTD とも呼ばれる。

文書タイプ定義 (document type definition (DTD))
XML 文書のクラスの文法。 文書タイプ宣言を参照。

2 バイト ASCII (double-byte ASCII)
DBCS 文字および 1 バイト ASCII 文字を含む IBM の文字セット (「ASCII DBCS」とも呼ばれる)。

2 バイト EBCDIC (double-byte EBCDIC)
DBCS 文字および 1 バイト EBCDIC 文字を含む IBM の文字セット (「EBCDIC DBCS」とも呼ばれる)。

2 バイト文字セット (double-byte character set (DBCS))
それぞれの文字が 2 バイトで表現される 1 組の文字。 256 個のコード・ポイントで表現される記号より多くの記号を含んでいる 言語 (日本語、中国語、および韓国語など) は、2 バイト文字セットを必要とする。 各文字に 2 バイトが必要なため、DBCS 文字の入力、表示、および印刷には、DBCS を受け入れ可能なハードウェアおよびサポートされるソフトウェアが必要。

DWARF
DWARF は UNIX International Programming Languages Special Interest Group (SIG) で開発された。言語に依存しないデバッグ情報を提供することにより、さまざまな言語の、統一された方法でのシンボリックなソース・レベル・デバッグのニーズを満たすように設計されている。 DWARF ファイルには、さまざまなエレメントに編成されたデバッグ・データが含まれている。 詳しくは、「*DWARF/ELF エクステンション ライブラリー・リファレンス*」の『*DWARF program information*』を参照。

動的アクセス (* dynamic access)
1 つの OPEN ステートメントの実行範囲内において、特定の論理レコードを、大容量記憶ファイルからは順次アクセス以外の方法で取り出したりそのファイルに入れたりでき、またファイルからは順次アクセスの方法で取り出せるアクセス・モード。

動的 CALL (dynamic CALL)
DYNAM オプションおよび NODLL オプションを使用してコンパイルされたプログラム内の CALL *literal* ステートメント、または NODLL オプションを使用してコンパイルされたプログラム内の CALL *identifier* ステートメント。

動的長 (dynamic-length)
実行時に変化する可能性がある論理長を持つ項目を記述する形容詞。

動的長基本項目 (dynamic-length elementary item)
データ宣言項目に DYNAMIC LENGTH 節が含まれる基本データ項目。

動的長グループ (dynamic-length group)
従属動的長基本項目を含むグループ項目。

ダイナミック・リンク・ライブラリー (DLL) (dynamic link library (DLL))
リンク時ではなく、ロード時または実行時にプログラムにバインドされる実行可能コードおよびデータが入ったファイル。 複数のアプリケーションが DLL 内のコードおよびデータを同時に共用することができる。 DLL はプログラムの実行可能ファイルの一部ではないが、実行可能ファイルを正しく実行するためには必要となる可能性がある。

動的ストレージ域 (dynamic storage area (DSA))
動的に獲得されるストレージであり、レジスター保管域、および動的ストレージ割り振 りに使用可能な区域 (プログラム変数など) から構成される。 DSA は、プログラムまたは関数が呼び出されるときに割り振られ、 呼び出しインスタンスの継続時間の間持続します。 DSA は通常、言語環境プログラムによって管理されるスタック・セグメント内に割り振られる。

E

EBCDIC (拡張 2 進化 10 進コード) (* EBCDIC (Extended Binary-Coded Decimal Interchange Code))
8 ビット・コード化文字をベースとするコード化文字セット。

EBCDIC 文字 (EBCDIC character)
EBCDIC (拡張 2 進化 10 進コード) セットに含まれているいずれかの記号。

EBCDIC DBCS
「*2 バイト EBCDIC (double-byte EBCDIC)*」を参照。

編集データ項目 (edited data item)
0 の抑止または編集文字の挿入、あるいはその両方を行うことによって変更されたデータ項目。

編集用文字 (* editing character)
次に示す集合に属する 1 文字、または 2 文字で構成される固定した組み合わせ。

文字	意味
	スペース
0	ゼロ
+	正符号
-	負符号
CR	貸方
DB	借方

文字	意味
Z	ゼロの抑止
*	小切手変造防止
$	通貨符号
,	コンマ (小数点)
.	ピリオド (小数点)
/	斜線 (スラッシュ)

EGCS
拡張図形文字セット *(extended graphic character set) (EGCS)* を参照。

EJB
Enterprise JavaBeans を参照。

EJB コンテナー (EJB container)
J2EE アーキテクチャーの EJB コンポーネント契約を実装するコンテナー。 この契約は、セキュリティー、並行性、ライフ・サイクル管理、トランザクション、デプロイメント、およびその他のサービスを含んでいるエンタープライズ Bean 用のランタイム環境を指定します。 EJB コンテナーは、EJB サーバーまたは J2EE サーバーによって提供される。 (Oracle)

EJB サーバー (EJB server)
EJB コンテナーにサービスを提供するソフトウェア。 EJB サーバーは、1 つ以上の EJB コンテナーをホストできる。 (Oracle)

エレメント (テキスト・エレメント) (element (text element))
1 つのデータ項目または動詞の記述などのようなテキスト・ストリングの 1 つの論理単位で、その前にエレメント・タイプを識別する固有のコードが付けられたもの。

基本項目 (* elementary item)
論理的にそれ以上細分できないものとして記述されているデータ項目。

カプセル化 (encapsulation)
オブジェクト指向プログラミングでは、オブジェクトの固有の詳細を隠すのに使用される技法。 オブジェクトは、基礎構造を露出しなくても、データの照会と操作を行うインターフェースを提供する。「情報隠蔽 *(information hiding)*」と同義。

エンクレーブ (enclave)
言語環境プログラムのもとで実行される場合、エンクレーブは実行単位に類似している。 エンクレーブは、LINK および C の system() 関数の使用によって、他のエンクレーブを作成できる。

エンコード・ユニット (encoding unit)
文字エンコード・ユニット *(character encoding unit)* を参照。

END CLASS マーカー (end class marker)
語の組み合わせに分離文字ピリオドが続いたもので、COBOL クラス定義の終わりを示す。 クラス終了マーカーは次のとおり。

```
END CLASS クラス名.
```

メソッド終了マーカー (end method marker)
語の組み合わせに分離文字ピリオドが続いたもので、COBOL メソッド定義の終わりを示す。 メソッド終了マーカーは次のとおり。

```
END METHOD method-name.
```

PROCEDURE DIVISION の終わり (* end of PROCEDURE DIVISION) (* end of PROCEDURE DIVISION)
COBOL ソース・プログラムにおいて、それ以後にはプロシージャーが存在しない物理的な位置。

プログラム終了マーカー (* end program marker)
語の組み合わせに分離文字ピリオドが続いたもので、COBOL ソース・プログラムの終わりを示す。 プログラム終了マーカーは、次のように記述する。

```
END PROGRAM program-name.
```

Enterprise Bean
ビジネス・タスクを実装し、EJB コンテナーに存在するコンポーネント。 (Oracle)

Enterprise JavaBeans
オブジェクト指向の分散エンタープライズ・レベル・アプリケーションの開発とデプロイメントのために、Oracle によって定義されたコンポーネント・アーキテクチャー。

項目 (* entry)
分離文字ピリオドで終了させられる連続する節の記述セットであり、COBOL プログラムの IDENTIFICATION DIVISION、ENVIRONMENT DIVISION、または DATA DIVISION に書き込まれる。

環境節 (* environment clause)
ENVIRONMENT DIVISION 記入項目の一部として現れる節。

ENVIRONMENT DIVISION
COBOL プログラム、クラス定義、またはメソッド定義の 4 つの主コンポーネントの 1 つ。 ENVIRONMENT DIVISION では、ソース・プログラムがコンパイルされるコンピューターと、オブジェクト・プログラムが実行されるコンピューターを記述する。 この部では、ファイルの論理概念とそのレコードの間のリンケージ、およびファイルが保管される装置の物理的局面を提供する。

環境名 (environment-name)
IBM が指定する名前であり、システム論理装置、プリンターおよびカード穿孔装置の制御文字、報告書コード、またはプログラム・スイッチ、あるいはそれらの組み合わせを識別する。環境名が ENVIRONMENT DIVISION の簡略名と関連付けられている場合は、その簡略名を、置換が有効な任意の形式で置き換えることができる。

環境変数 (environment variable)
コンピューター環境の一部の局面を定義する多数の変数のいずれかであり、その環境で動作するプログラムからアクセス可能。 環境変数は、動作環境に依存するプログラムの動作に影響を与える。

実行時 (execution time)
実行時 *(run time)* を参照。

実行時環境 (execution-time environment)
ランタイム環境 *(runtime environment)* を参照。

明示的範囲終了符号 (* explicit scope terminator)
特定の PROCEDURE DIVISION ステートメントの有効範囲を終わらせる予約語。

指数 (exponent)
別の数 (底) をべき乗する指数を示す数。 正の指数は乗算を示し、負の指数は除算を示し、小数の指数は数量の根を示す。 COBOL では、指数式は記号 ** の後に指数を付けて表す。

式 (* expression)
算術式または条件式。

拡張モード (* extend mode)
ファイルに対する EXTEND 句の指定のある OPEN ステートメントが実行されてから、そのファイルに対する REEL または UNIT 句の指定のない CLOSE ステートメントが実行される前までの、ファイルの状態。

拡張図形文字セット (extended graphic character set) (EGCS)
それぞれの図形文字を表すために 2 バイトを必要とする図形文字セット (漢字文字セットなど)。 現在は、*2* バイト文字セット *(DBCS)* と呼ばれるようになった。

Extensible Markup Language
XML を参照。

拡張 (extensions)
85 COBOL 標準 に記述されているものの他に、IBM コンパイラーでサポートされる COBOL 構文およびセマンティクス。

外部コード・ページ (external code page)
XML 文書では、CODEPAGE コンパイラー・オプションによって 指定された値。

外部データ (* external data)
プログラムの中で外部データ項目および外部ファイル結合子として記述されるデータ。

外部データ項目 (* external data item)
実行単位の 1 つ以上のプログラムにおいて外部レコードの一部として記述されるデータ項目であり、その項目が記述されている任意のプログラムから参照することができる。

外部データ・レコード (* external data record)
実行単位の 1 つ以上のプログラムにおいて記述される論理レコードであり、そのデータ項目は、それらが記述されている任意のプログラムから参照できる。

外部 10 進数データ項目 (external decimal data item)
ゾーン *10* 進数データ項目 *(zoned decimal data item)* および 国別 *10* 進数データ項目 *(national decimal data item)* を参照。

外部ファイル結合子 (* external file connector)
実行単位の 1 つ以上のオブジェクト・プログラムにアクセス可能なファイル結合子。

外部浮動小数点データ項目 (external floating-point data item)
表示浮動小数点データ項目 *(display floating-point data item)* および 国別浮動小数点データ項目 *(national floating-point data item)* を参照。

外部プログラム (external program)
最外部プログラム。 ネストされていないプログラム。

外部スイッチ (* external switch)
インプリメントする人によって定義され、名前が付けられたハードウェア装置またはソフトウェアで、2 つの選択的な状態のうち一方が存在することを示すために使用される。

F

ファクトリー・データ (factory data)
いったんクラスに割り振られ、クラスのすべてのインスタンスに共用されるデータ。 ファクトリー・データは、クラス定義の FACTORY 段落の DATA DIVISION の WORKING-STORAGE SECTION 内に宣言される。Java private 静的データと同義。

ファクトリー・メソッド (factory method)
オブジェクト・インスタンスとは無関係に、クラスによってサポートされる メソッド。 ファクトリー・メソッドは、クラス定義の FACTORY 段落に宣言される。Java public 静的メソッドと同義。これらは通常、オブジェクトの作成をカスタマイズするために使用されます。

形象定数 (* figurative constant)
ある予約語を使用することによって参照されるコンパイラー生成の値。

ファイル (* file)
論理レコードの集合。

ファイル属性対立条件 (* file attribute conflict condition)
あるファイルで入出力操作を実行する試みが失敗し、 プログラムの中でそのファイルに対して指定したファイル属性が、 そのファイルの固定属性と一致していないこと。

ファイル節 (* file clause)
DATA DIVISION の記入項目であるファイル記述項目 (FD 記入項目) およびソート・マージ・ファイル記述項目 (SD 記入項目) のいずれかの一部として現れる節。

ファイル結合子 (* file connector)
ファイルに関する情報が入っており、ファイル名と物理ファイルの間のリンケージとして、さらにファイル名とその関連レコード域の間のリンケージとして使用されるストレージ域。

ファイル制御 (File-Control)
ソース・プログラムで用いられるデータ・ファイルが宣言されて いる環境部の段落の名前。

ファイル制御ブロック (FCB) (file control block)
I/O ルーチンのアドレス、それらがどのようにオープンおよびクローズされたかに関する情報、およびファイル情報ブロック (FIB) へのポインターを含むブロック。

ファイル制御項目 (* file control entry)
SELECT 節と、ファイルの関連物理属性を宣言するすべての従属節。

FILE-CONTROL 段落 (FILE-CONTROL paragraph)
ENVIRONMENT DIVISION 内の段落であり、この中では、特定のソース単位で使用されるデータ・ファイルが宣言される。

ファイル記述項目 (* file description entry)
DATA DIVISION の FILE SECTION の中にある記入項目。レベル標識 FD と、それに続くファイル名、および、必要に応じて、次に続く一連のファイル節から構成される。

ファイル名 (* file-name)
DATA DIVISION の FILE SECTION の中のファイル記述項目またはソート・マージ・ファイル記述項目で記述されるファイル結合子に名前を付けるユーザー定義語。

ファイル編成 (* file organization)
ファイルの作成時に確立される永続的な論理ファイル構造。

ファイル位置標識 (file position indicator)
概念的エンティティーであり、索引付きファイルの場合は参照キー内の現行キーの値、順次ファイルの場合は現行レコードのレコード番号、相対ファイルの場合は現行レコードの相対レコード番号が入っている。あるいは、次の論理レコードが存在しないことを示すか、オプションの入力ファイルが使用可能でないことを示すか、AT END 条件が既に存在していることを示すか、もしくは有効な次のレコードが設定されていないことを示す。

*** FILE SECTION**
DATA DIVISION のセクションであり、ファイル記述項目、ソート・マージ・ファイル記述項目、および関連するレコード記述が入っている。

ファイル・システム (file system)
データ・レコードおよびファイル記述プロトコルの特定のセットに準拠するファイルの集合、およびこれらのファイルを管理する一連のプログラム。

固定ファイル属性 (* fixed file attributes)
ファイルに関する情報であり、ファイルの作成時に設定され、それ以降はファイルが存在する限り変更できない。 これらの属性には、ファイルの編成 (順次、相対、指標付き)、 基本レコード・キー、代替レコード・キー、コード・セット、 最小および最大のレコード・サイズ、レコード・タイプ (固定長、可変長)、 索引付きファイルのキーの照合シーケンス、ブロック化因数、埋め込み文字、 レコード区切り文字がある。

固定長レコード (* fixed-length record)
ファイル記述項目またはソート・マージ記述項目が、 すべてのレコードのバイトの個数が同じであるように要求しているファイルに関連付けられたレコード。

固定小数点項目 (fixed-point item)
PICTURE 節で定義される数値データ項目であり、オプションの符号の位置、その中に含まれる桁数、およびオプションの小数点の位置を指定するもの。 2 進数、パック 10 進数、または外部 10 進数のいずれかのフォーマットをとることができる。

浮動コメント標識 (*>) (floating comment indicators (*>))
浮動コメント標識がプログラムのテキスト域 (領域 A プラス領域 B) 内の最初の文字ストリングである場合は、この行がコメント行であることを示します。 また、浮動コメント標識がプログラムのテキスト領域内の 1 つ以上の文字ストリングの後にある場合は、インライン・コメントを示します。

浮動小数点 (floating point)
実数を 1 対の数表示で表す、数を表記するための形式。 浮動小数点表記では、固定小数点部分 (最初の数表示) と、暗黙浮動小数点の底を指数で表される数だけ累乗して得られる値 (2 番目の数表示) との積が、実数になります。 例えば、数値 0.0001234 の浮動小数点表記は 0.1234 -3 です (ここで、0.1234 は 小数部であり、-3 は指数です)。

浮動小数点データ項目 (floating-point data item)
小数部と指数が入っている数値データ項目。 その値は、小数部に、指数で指定されただけ累乗された数字データ項目の底を乗算することによって得られる。

フォーマット (* format)
データの集合の特定の配列。

機能 (* function)
ステートメントの実行中に参照された時点で決定される値を持つ、 一時的なデータ項目。

関数 ID (* function-identifier)
関数を参照する文字ストリングと分離文字の構文的に正しい組み合わせ。 関数で表現されるデータ項目は、関数名と引数 (ある場合) によって一意的に識 別される。 関数 ID は、参照修飾子を含むことができる。 英数字関数を参照する関数 ID は、 一定の制限に従いつつ ID が指定できる一般フォーマットの中ならばどこにでも指定できる。 整数関数または数字関数を参照する関数 ID は、算術式が指定できる一般フォーマットの中ならばどこにおいても指定できる。

機能名 (function-name)
必要な引数を伴って関数の値を決定する呼び出しを行うメカニズムに付けられる名前を表すワード。

関数ポインター・データ項目 (function-pointer data item)
入り口点を指すポインターを保管できるデータ項目。 USAGE IS FUNCTION-POINTER 節で定義されるデータ項目に、関数入り口点のアドレスが含められる。 一般的に、C および Java プログラムと通信するために使用される。

G

ガーベッジ・コレクション (garbage collection)
参照されなくなったオブジェクト用のメモリーが Java ランタイム・システムによって自動的に解放されること。

グローバル名 (* global name)
1 つのプログラムにおいてのみ宣言されるが、そのプログラム、またはそのプログラム内に含まれている任意のプログラムから参照できる名前。 条件名、データ名、ファイル名、レコード名、報告書名、およびいくつかの特殊レジスターが、グローバル名となり得る。

グローバル参照 (global reference)
メソッドの有効範囲外にあるオブジェクトの参照。

グループ項目 (group item)
(1) 複数の従属データ項目で構成されるデータ項目。 英数字グループ項目 *(alphanumeric group item)* および 国別グループ項目 *(national group item)* を参照。 (2) 国別グループまたは英数字グループとして明示的に (またはコンテキスト で) 限定されていない場合、この用語は一般のグループを指します。

グループ区切り文字 (grouping separator)
読みやすさのために数値を何桁かまとめて区切るのに使用される文字。 デフォルトの文字はコンマです。

H

ヘッダー・ラベル (header label)
(1) 記録メディア・ユニットのデータ・レコードの前にある、データ・セットのラベル。 (2)「ファイル開始ラベル *(beginning-of-file label)*」の同義語。

隠蔽 (メソッド) (hide (a method))
親クラスで同じメソッド名により定義されたファクトリーまたは静的メソッドを (サブクラスで) 再定義すること。 したがって、サブクラスのメソッドは親クラスのメソッドを隠蔽する。

高位終了 (* high-order end)
文字ストリングの左端の文字。

ハイパースペース (hiperspace)
z/OS 環境で、プログラムがバッファーとして使用できる最大 2 GB までの連続する仮想記憶アドレス範囲。

I

IBM COBOL 拡張部分 (IBM COBOL extension)
85 COBOL 標準 に記述されているものの他に、IBM コンパイラーでサポートされる COBOL 構文およびセマンティクス。

IDENTIFICATION DIVISION
COBOL プログラム、クラス定義、またはメソッド定義の 4 つの主コンポーネントの 1 つ。 IDENTIFICATION DIVISION では、プログラム、クラス、またはメソッドを識別する。 IDENTIFICATION DIVISION には、作成者名、インストール、または日付を含めることができる。

ID (* identifier)
データ項目に名前を付けるための文字ストリングと分離文字の構文的に正しい組み合わせ。 関数ではないデータ項目を参照するときは、ID は、データ名と、修飾子、添え字、または 参照修飾子 (一意的に参照するために必要な場合) から構成される。 関数であるデータ項目を参照する際には、関数 ID が使われる。

IGZCBSN
COBOL/370 リリース 1 のブートストラップ・ルーチン。 これは、COBOL/370 リリース 1 プログラムを含んでいるモジュールとリンク・エディットしなければならない。

IGZCBSO
COBOL (MVS および VM 版) リリース 2、COBOL (OS/390® および VM 版)、および Enterprise COBOL のブートストラップ・ルーチン。これは、COBOL (MVS および VM 版) リリース 2、COBOL (OS/390 および VM 版) または Enterprise COBOL プログラムを含んでいるモジュールとリンク・エディットしなければならない。

IGZEBST
VS COBOL II のブートストラップ・ルーチン。 これは、VS COBOL II リリース 1 プログラムを含んでいるモジュールとリンク・エディットしなければならない。

ILC
言語間通信。言語間通信は、他の高水準言語を呼び出すかまたは他の高水準言語によって呼び出されるプログラムとして定義されています。アセンブラーは高水準言語と見なされません。このため、アセンブラー言語プログラム への呼び出しおよびアセンブラー言語プログラムからの呼び出しは ILC と見なされま せん。

命令ステートメント (* imperative statement)
命令の動詞で開始して、行うべき無条件の処置を指定するステートメント。または明示範囲終了符号によって区切られた条件ステートメント (範囲区切りステートメント)。 1 つの命令ステートメントは、一連の命令ステートメントから構成することができる。

暗黙の範囲終了符号 (* implicit scope terminator)
終了していないステートメントが前にある場合、その範囲を区切る分離文字ピリオド。 または、前にある句の中に含まれるステートメントがある場合、 そのステートメントの範囲の終わりをそれが現れることによって示すステートメントの句。

IMS
情報管理システム (Information Management System)。IBM のライセンス製品。IMS は、階層データベース、データ通信 (DC)、変換処理、およびデータベースのバックアウトとリカバリーをサポートする。

指標 (* index)
コンピューターのストレージ域またはレジスター。 ここにはテーブル内の個々のエレメントを識別するものを表現する内容が入る。

指標データ項目 (* index data item)
指標名に関連する値を、インプリメントする人が指定した形式で収めることができるデータ項目。

索引付きデータ名 (indexed data-name)
データ名とそれに続く括弧で囲まれた 1 つまたは複数の指標名から構成される ID。

索引付きファイル (* indexed file)
指標付き編成のファイル。

指標付き編成 (* indexed organization)
各レコードがそのレコードの中にある 1 つまたは複数のキー値によって識別される永続的な論理ファイル構造。

指標付け (indexing)
指標名を使用した添え字付け と同義。

索引名 (* index-name)
特定のテーブルに関連付けられた指標に付ける名前を表すユーザー定義語。

継承 (inheritance)
クラスのインプリメンテーションを、別のクラスを基にして使用するメカニズム。 定義により、継承するクラスは継承されるクラスに準拠する。 Enterprise COBOL は 多重継承 をサポートしない。サブクラスは、必ず 1 つの即時スーパークラスを有する。

継承階層 (inheritance hierarchy)
クラス階層 *(class hierarchy)* を参照。

初期設定プログラム (* initial program)
実行単位内にプログラムが呼び出されるたびに初期状態に置かれるプログラム。

初期状態 (* initial state)
プログラムが実行単位の中に呼び出された最初期のそのプログラムの状態。

インライン (inline)
ルーチン、サブルーチン、または他のプログラムに分岐せずに、 連続的に実行されるプログラム内の命令。

インライン・コメント (inline comments)
インライン・コメントは、プログラムのテキスト域にある、前に 1 つ以上の文字ストリングが付いた浮動コメント標識 (*>) で示され、コンパイル・グループの任意の行に書き込むことができます。 浮動コメント標識に続く、領域 B の終わりまでの文字はすべて、コメント・テキストです。

入力ファイル (* input file)
入力モードでオープンされるファイル。

入力モード (* input mode)
ファイルに対する INPUT 句の指定のある OPEN ステートメントが実行されてから、そのファイルに対する REEL または UNIT 句の指定のない CLOSE ステートメントが実行される前までの、ファイルの状態。

入出力ファイル (* input-output file)
I-O モードでオープンされるファイル。

*** INPUT-OUTPUT SECTION**
ENVIRONMENT DIVISION のセクションであり、オブジェクト・プログラムまたはメソッドに必要なファイルおよび外部メディアに名前を付け、実行時にデータの伝送および処理に必要な情報を提供する。

入出力ステートメント (* input-output statement)
個々のレコードに対して操作を行うことにより、またはファイルを 1 つの単位として操作することにより、ファイルの処理を行うステートメント。 入出力ステートメントには、ACCEPT (ID 句付き)、CLOSE、DELETE、DISPLAY、OPEN、READ、REWRITE、SET (TO ON または TO OFF 句付き)、START、および WRITE がある。

入力プロシージャー (* input procedure)
ソートすべき特定のレコードの解放を制御する目的で、形式 1 SORT ステートメントの実行時に制御が渡されるステートメントの集合。

インスタンス・データ (instance data)
オブジェクトの状態を定義するデータ。 クラスによって導入されるインスタンス・データは、クラス定義の OBJECT 段落の DATA DIVISION の WORKING-STORAGE SECTION に定義される。 オブジェクトの状態には、クラスが導入した、現行クラスによって継承されているインスタンス変数の状態も含まれる。 インスタンス・データの個々のコピーは、各オブジェクト・インスタンスごとに作成される。

整数 (* integer)
(1) 小数点の右側に桁位置がない数値リテラル。 (2) DATA DIVISION に定義される数値データ項目であり、小数点の右側に桁位置を含まないもの。 (3) 関数の起こりうるすべての評価の戻り値で、小数点の右側の桁がすべてゼロであることが定義されて いる数字関数。

整数関数 (integer function)
カテゴリーが数字で、その定義が小数点の右側に桁位置を持たない関数。

対話式システム生産性向上機能 (ISPF) (Interactive System Productivity Facility (ISPF))
TSO または VM ユーザーに対してメニュー方式のインターフェースを提供する IBM ソフトウェア・プロダクト。ISPF には、ライブラリー・ユーティリティー、強力なエディター、およびダイアログ管理が組み込まれています。

言語間通信 (ILC) (interlanguage communication (ILC))
異なるプログラム言語で書かれた複数のルーチンが通信できること。 ILC サポートにより、各種言語で書かれたコンポーネント・ルーチンからアプリケーションを簡単に構築することができる。

中間結果 (intermediate result)
連続して行われる算術演算の結果を収める中間フィールド。

内部データ (* internal data)
プログラムの中で記述されるデータで、すべての外部データ項目および外部ファイル結合子を除いたもの。 プログラムの LINKAGE SECTION で記述された項目は、内部データとして扱われる。

内部データ項目 (* internal data item)
実行単位内の 1 つのプログラムの中で記述されるデータ項目。 内部データ項目は、グローバル名を持つことができる。

内部 10 進数データ項目 (internal decimal data item)
USAGE PACKED-DECIMAL または USAGE COMP-3 として記述されており、項目を数値として定義する PICTURE 文字ストリング (記号 9、S、P、または V の有効な組み合わせ) を持っている、データ項目。 「パック *10* 進数データ項目 *(packed-decimal data item)*」と同義。

内部ファイル結合子 (* internal file connector)
実行単位内にあるただ 1 つのオブジェクト・プログラムのみがアクセスできるファイル結合子。

内部浮動小数点データ項目 (internal floating-point data item)
USAGE COMP-1 または USAGE COMP-2 として記述されているデータ項目。 COMP-1 は、単精度浮動小数点データ項目を定義します。 COMP-2 は、倍精度浮動小数点データ項目を定義します。 内部浮動小数点データ項目に関連した PICTURE 節はありません。

レコード内データ構造 (* intrarecord data structure)
連続したデータ記述記入項目のサブセットによって定義される、1 つの論理レコードから得られるグループ・データ項目および基本データ項目の集合全体。 これらのデータ記述記入項目には、レコード内データ構造を記述している最初のデータ記述記入項目のレベル番号より大きいレベル番号を持つすべての記入項目が含まれる。

組み込み関数 (intrinsic function)
よく使用される算術関数のような事前定義関数で、組み込み関数参照によって呼び出される。

無効キー条件 (* invalid key condition)
索引付きファイルまたは相対ファイルに関連するキーの特定値が無効であると判別された場合に生じる、実行時の条件。

*** I-O-CONTROL**
ENVIRONMENT DIVISION 段落の名前。この段落では、再実行開始点についてのオブジェクト・プログラム要件、複数データ・ファイルによる同じ区域の共用、および単一入出力装置上の複数のファイル・ストレージが指定される。

I-O-CONTROL 記入項目 (* I-O-CONTROL entry)
ENVIRONMENT DIVISION の I-O-CONTROL 段落内の記入項目であり、プログラム実行中に指定のファイルへのデータの伝送と処理を行うために必要な情報を提供する節が入っている。

入出力モード (* I-O mode)
ファイルに対する I-O 句の指定のある OPEN ステートメントが実行されてから、そのファイルに対する REEL または UNIT 句の指定のない CLOSE ステートメントが実行される前までの、ファイルの状態。

入出力状況 (* I-O status)
入出力操作の結果としての状況を示す 2 文字の値を収める概念上のエンティティー。 この値は、そのファイルについてのファイル制御記入項目で FILE STATUS 節を使用することによって、プログラムに使用可能にされる。

is-a
継承階層におけるクラスおよびサブクラスの特徴を表す関係。 あるクラスに対して is-a 関係を持つサブクラスは、そのクラスから継承する。

ISPF
対話式システム生産性向上機能 *(ISPF) (Interactive System Productivity Facility (ISPF))* を参照。

反復構造 (iteration structure)
ある条件が真である間、あるいはある条件が真になるまで、一連のステートメントが繰り返して実行されるプログラムの処理ロジック。

J

J2EE
Java 2 Platform, Enterprise Edition (J2EE) を参照。

Java 2 Platform, Enterprise Edition (J2EE)
エンタープライズ・アプリケーションの開発とデプロイメントのために、Oracle によって定義された環境。 J2EE プラットフォームは、多層の Web ベース・アプリケーションを開発するための機能を提供するサービス、アプリケーション・プログラミング・インターフェース (API)、およびプロトコルで構成されている。 (Oracle)

Java Batch Launcher and Toolkit for z/OS (JZOS)
従来のバッチ環境で実行されて z/OS システム・サービスにアクセスする z/OS Java アプリケーションの開発を支援するツールのセット。

Java バッチ処理プログラム (JBP) (Java batch-processing program (JBP))
オンライン・データベースおよび出力メッセージ・キューにアクセスできる IMS バッチ処理プログラム。JBP はオンラインで実行されますが、バッチ環境のプログラムと同様に、JCL を使用して、または TSO セッション内で開始されます。

Java バッチ処理領域 (Java batch-processing region)
Java バッチ処理プログラムだけがスケジュールされる IMS 従属領域。

Java Database Connectivity (JDBC)
Java プログラムのデータベースへのアクセスを可能にする API を定義する、Oracle の仕様。

Java メッセージ処理プログラム (JMP) (Java message-processing program (JMP))
トランザクションによって駆動され、オンライン IMS データベースとメッセージ・キューにアクセスできる、Java アプリケーション・プログラム。

Java メッセージ処理領域 (Java message-processing region)
Java メッセージ処理プログラムだけがスケジュールされる IMS 従属領域。

Java Native Interface (JNI)
Java 仮想マシン (JVM) 内で実行される Java コードが、他のプログラム言語で記述されたアプリケーションおよびライブラリーと連携できるようにするプログラミング・インターフェース。

Java 仮想マシン (JVM) (Java virtual machine (JVM))
コンパイル済みの Java プログラムを実行する中央演算処理装置のソフトウェア・インプリメンテーション。

JavaBeans
移植可能で、プラットフォームに依存しない、再使用可能なコンポーネント・モデル。 (Oracle)

JBP
Java バッチ処理プログラム (JBP) (Java batch-processing program (JBP)) を参照。

JDBC
Java Database Connectivity (JDBC) を参照。

JMP
Java メッセージ処理プログラム (JMP) (Java message-processing program (JMP)) を参照。

ジョブ制御言語 (JCL) (job control language (JCL))
ジョブをオペレーティング・システムに識別させ、ジョブの要件を記述するために使われる制御言語。

JSON
JSON (JavaScript Object Notation) とは、単純なデータ交換フォーマットである。

JVM
Java 仮想マシン *(JVM) (Java virtual machine (JVM))* を参照。

JZOS
Java Batch Launcher と *Toolkit for z/OS* を参照。

K

K
記憶容量に関連して使用されるときは、2 の 10 乗。 10 進表記では 1024。

キー (* key)
レコードの位置を識別するデータ項目、またはデータの順序付けを識別するための一連のデータ項目。

参照キー (* key of reference)
索引付きファイルの中のレコードをアクセスするために現在使用されている基本キーまたは代替キー。

キーワード (* keyword)
コンテキスト・センシティブ語または予約語。その語の表示フォーマットがソース単位で使用されるときは、その語は必須である。

キロバイト (KB) (kilobyte (KB))
1 キロバイトは 1024 バイトに相当する。

L

言語名 (* language-name)
特定のプログラミング言語を指定するシステム名。

Language Environment
z/OS 言語環境プログラム の省略名。C、C++、COBOL、FORTRAN、および PL/I アプリケーションに共通のランタイム環境およびランタイム・サービスを提供する一連のアーキテクチャー構造および インターフェース。 言語環境プログラム に準拠するコンパイラーでコンパイルされたプログラムおよび、Java アプリケーションに必要です。

言語環境プログラム 準拠 (Language Environment-conforming)
言語環境プログラム の規約に準拠したオブジェクト・コードを生成するコンパイラー製品 (Enterprise COBOL、COBOL (OS/390 および VM 版)、COBOL (MVS および VM 版)、C/C++ for MVS & VM、PL/I for MVS & VM など) の特性。

最後に使われた状態 (last-used state)
内部値がプログラム終了時と同じままで、初期値にリセットされない、プログラムの状態を言う。

文字 (* letter)
以下の 2 つのセットのいずれかに属する文字。

1. 英大文字: 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
2. 英小文字: 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

レベル標識 (* level indicator)
特定のタイプのファイルを識別するか、または階層での位置を識別する 2 つの英字。 DATA DIVISION 内のレベル標識には、CD、FD、および SD がある。

レベル番号 (* level-number)
階層構造におけるデータ項目の位置を示すか、またはデータ記述記入項目の特性を示す、2 桁の数字で表されたユーザー定義語。 1 から 49 までの範囲のレベル番号は、論理レコードの階層構造におけるデータ項目の位置 を示す。 1 から 9 のレベル番号は、1 桁の数字として書き込むことも、 0 の後に有効数字を書き込むこともできる。 レベル番号 66、77、および 88 は、データ記述項目の特性を識別する。

ライブラリー名 (* library-name)
COBOL ライブラリーの名前を表すユーザー定義語。与えられたソース・プログラムをコンパイルするためにコンパイラーが使用するライブラリーを識別する。

ライブラリー・テキスト (* library text)
COBOL ライブラリーの中にある一連のテキスト・ワード、コメント行、 インライン・コメント、区切り文字のスペース、または区切り文字の疑似テキスト区切り文字。

リリアン日 (Lilian date)
グレゴリオ暦の開始以降の日数。 第 1 日は 1582 年 10 月 15 日、金曜日。 リリアン日フォーマットは、 グレゴリオ暦の考案者であるルイジ・リリオにちなんだ名称。

*** LINAGE-COUNTER**
ページ本体内の現在位置を指す値を収めた特殊レジスター。

リンク (link)
(1) リンク接続 (伝送メディア) と、それぞれがリンク接続の終端にある 2 つのリンク・ステーションの組み合わせ。1 つのリンクは、マルチポイントまたはトークンリング構成において、複数のリンク間で共用できる。 (2) データ項目あるいは 1 つ以上のコンピューター・プログラムの部分を相互接続すること。例えば、リンケージ・エディターによってオブジェクト・プログラムをリンクして実行可能ファイルを作成すること。

LINKAGE SECTION
呼び出し先のプログラムまたはメソッドの DATA DIVISION 内のセクションであり、呼び出し側プログラムまたはメソッドから使用可能なデータ項目が記述される。 これらのデータ項目は、呼び出し側プログラムまたはメソッドおよび呼び出し先プログラムまたはメソッドの両方から参照できる。

リンカー (linker)
z/OS バインダー (リンケージ・エディター) を指す用語。

リテラル (literal)
ストリングを構成するために配列された文字によって、または形象定数を使用することによって、その値が決められる文字ストリング。

リトル・エンディアン (little-endian)
Intel プロセッサーが 2 進データおよび UTF-16 文字を保管するために使用するデフォルト形式。 この形式では、2 進数データ項目の最上位バイトが最上位のアドレスになり、 UTF-16 文字の最上位バイトが最上位のアドレスになる。 ビッグ・エンディアン *(big-endian)* と比較。

ローカル参照 (local reference)
メソッドの有効範囲内にあるオブジェクトの参照。

ロケール (locale)
プログラム実行環境の一連の属性であり、文化的に重要な考慮事項を示す。例えば、文字コード・ページ、照合シーケンス、日時形式、通貨表記、数値表記、または言語など。

*** LOCAL-STORAGE SECTION**
DATA DIVISION のセクションであり、VALUE 節で割り当てられた値に応じて、呼び出し単位で割り振りまたは解放が行われるストレージを定義する。

論理演算子 (* logical operator)
予約語 AND、OR、または NOT のいずれか。 条件の形成において、AND または OR、あるいはその両方を論理連結語として使用できる。 NOT は論理否定に使用できる。

論理レコード (* logical record)
最も包括的なデータ項目。 レコードのレベル番号は 01。 レコードは、基本項目またはグループ項目のどちらでもよい。 「レコード *(record)*」と同義。

下位終了 (* low-order end)
文字ストリングの右端の文字。

M

メインプログラム (main program)
プログラムとサブルーチンからなる階層において、プロセス内でプログラムが実行されたときに最初に制御を受け取るプログラム。

Make ファイル (makefile)
アプリケーションに必要なファイルのリストが収められたテキスト・ファイル。 make ユーティリティーはこのファイルを使用して、ターゲット・ファイルを最新の変更で更新する。

大容量記憶 (* mass storage)
データを順次と非順次の 2 つの方法で編成して保管しておくことができるストレージ・メディア。

大容量記憶装置 (* mass storage device)
磁気ディスクなど、大きな記憶容量を持つ装置。

大容量記憶ファイル (* mass storage file)
大容量記憶メディアに格納されたレコードの集合。

メガバイト、MB (* megabyte (MB))
1 メガバイトは 1,048,576 バイトに相当する。

マージ・ファイル (* merge file)
MERGE ステートメントによってマージされるレコードの集まり。 マージ・ファイルは、マージ機能により作成され、マージ機能によってのみ使用できる。

メッセージ処理プログラム (MPP) (message-processing program (MPP))
トランザクションによって駆動され、オンライン IMS データベースとメッセージ・キューにアクセスできる、IMS アプリケーション・プログラム。

メッセージ・キュー (message queue)
メッセージがアプリケーション・プログラムによって処理されたり端末に送信されたりする前に、 キューに入れられるデータ・セット。

メソッド (method)
オブジェクトによってサポートされる操作の 1 つを定義し、そのオブジェクトに対する INVOKE ステートメントによって実行されるプロシージャー・コード。

メソッド定義 (* method definition)
メソッドを定義する COBOL ソース・コード。

メソッド見出し記入項目 (* method identification entry)
IDENTIFICATION DIVISION の METHOD-ID 段落内の記入項目。この記入項目には、メソッド名を指定する節が入っている。

メソッドの起動 (method invocation)
あるオブジェクトから別のオブジェクトへの通信で、受信オブジェクトにメソッドを実行するように要求するもの。

メソッド名 (method-name)
オブジェクト指向操作の名前。 メソッドを起動するのに使用する場合、名前は、英数字リテラル、国別リテラル、カテゴリー英数字データ項目、またはカテゴリー国別データ項目にすることができます。メソッドを定義する METHOD-ID 段落で使用する場合、名前は英数字リテラルまたは国別リテラルにする必要があります。

メソッド隠蔽 (method hiding)
「隠蔽 *(hide)*」を参照。

メソッド多重定義 (method overloading)
「多重定義 *(overload)*」を参照。

メソッドのオーバーライド (method overriding)
「オーバーライド *(override)*」を参照。

簡略名 (* mnemonic-name)
ENVIRONMENT DIVISION において、指定されたインプリメントする人の名前に関連したユーザー定義語。

モジュール定義ファイル (module definition file)
プログラム・オブジェクト内のコード・セグメントを記述するファイル。

MPP
メッセージ処理プログラム *(MPP) (message-processing program (MPP))* を参照。

マルチタスキング (multitasking)
2 つ以上のタスクの並行実行またはインターリーブド実行を可能にする操作モード。

マルチスレッド化 (multithreading)
コンピューター内で複数のパスを使用して実行を行う並行操作。 「マルチプロセッシング *(multiprocessing)*」と同義。

N

名前 (name)
COBOL オペランドを定義する 30 文字を超えないで構成されたワード。

ネーム・スペース (namespace)
XML 名前空間 *(XML namespace)* を参照。

国別文字 (national character)
(1) 国別リテラルまたは USAGE NATIONAL の UTF-16 文字。 (2) UTF-16 で表される任意の文字。

国別文字データ (national character data)
UTF-16 で表されるデータの一般参照。

国別文字位置 (national character position)
文字位置 *(character position)* を参照。

国別データ (national data)
「国別文字データ *(national character data)*」を参照。

国別データ項目 (national data item)
カテゴリー国別、国別編集、または USAGE NATIONAL の数字編集のデータ項目。

国別 10 進数データ項目 (national decimal data item)
暗黙的または明示的に USAGE NATIONAL として記述されており、PICTURE の記号 9、S、P、および V の有効な組み合わせを含んでいる、外部 10 進数データ項目。

国別編集データ項目 (national-edited data item)
少なくとも 1 つの N のインスタンスおよび単純挿入記号 B、0、または / の少なくとも 1 つを含んでいる PICTURE 文字ストリングで記述されている、データ項目。 国別編集データ項目は USAGE NATIONAL を持ちます。

国別浮動小数点データ項目 (national floating-point data item)
暗黙的または明示的に USAGE NATIONAL として記述されており、浮動小数点データ項目を記述する PICTURE 文字ストリングを持っている、外部浮動小数点データ項目。

国別グループ項目 (national group item)
明示的または暗黙的に GROUP-USAGE NATIONAL 節で記述されたグループ項目。 国別グループ項目は、INSPECT、STRING、および UNSTRING などの操作で、カテゴリー国別の基本データ項目として定義されているかのように処理されます。 英数字グループ項目内で USAGE NATIONAL データ項目を定義するのとは対照的に、この処理により、国別文字の埋め込みおよび切り捨てが確実に正しく行われます。 グループ内の基本項目を処理する必要のある操作 (MOVE CORRESPONDING、ADD CORRESPONDING、および INITIALIZE など) の場合、国別グループは グループ・セマンティクスを使用して処理されます。

固有文字セット (* native character set)
OBJECT-COMPUTER 段落で指定されたコンピューターに関連した、インプリメントする人が定義した文字セット。

固有照合シーケンス (* native collating sequence)
OBJECT-COMPUTER 段落で指定されたコンピューターに関連した、インプリメントする人が定義した照合シーケンス。

ネイティブ・メソッド (native method)
COBOL などの別のプログラム言語で記述されたインプリメンテーションを備える Java メソッド。

複合否定条件 (* negated combined condition)
論理演算子 NOT とその直後に括弧で囲んだ複合条件を続けたもの。 条件 *(condition)* および 複合条件 *(combined condition)* も参照。

単純否定条件 (* negated simple condition)
論理演算子 NOT とその直後に単純条件を続けたもの。 条件 *(condition)* および 単純条件 *(simple condition)* も参照。

ネストされたプログラム (nested program)
他のプログラムの中に直接的に含まれているプログラム。

次の実行可能文 (* next executable sentence)
現在のステートメントの実行完了後に制御が移される次の文。

次の実行可能なステートメント (* next executable statement)
現在のステートメントの実行完了後に制御が移される次のステートメント。

次のレコード (* next record)
ファイルの現行レコードに論理的に続くレコード。

独立項目 (* noncontiguous items)
WORKING-STORAGE SECTION および LINKAGE SECTION 内の基本データ項目で、他のデータ項目と階層上の関係を持たないもの。

独立項目 (* noncontiguous items)
別のデータ項目への階層関係がない、WORKING-STORAGE および LINKAGE SECTION の基本デ ータ項目。

非数字項目 (* nonnumeric item)
その内容を、コンピューターの文字セットからの文字の任意の組み合わせで構成して記述することができるデータ項目。特定のカテゴリーの非数字項目は、さらに制限された文字セットから形成することができる 。

ヌル (null)
無効なアドレスの値をポインター・データ項目に割り当てるために使用される形象定数。 NULL を使えるところならばどこでも、NULLS を使用できる。

数字 (* numeric character)
次のような数字に属する文字。 0、1、2、3、4、5、6、7、8、9。

数値データ項目 (numeric data item)
(1) 記述により内容が数字 0 から 9 より選ばれた文字で表される値に制限されるデータ項目。符号付きである場合、この項目は +、-、または他の表記の演算符号も含むことができます。 (2) カテゴリー数値、内部浮動小数点、または外部浮動小数点のデータ項目。 数値データ項目は、USAGE DISPLAY、NATIONAL、PACKED-DECIMAL、BINARY、COMP、COMP-1、COMP-2、COMP-3、COMP-4、または COMP-5 を持つことができます。

数字編集データ項目 (numeric-edited data item)
印刷出力の際に使用するのに適したフォーマットの数値データを含むデータ項目。 データ項目は、外部 10 進数字の 0 から 9 の数字、小数点、コンマ、通貨符号、符号制御文字、その他の編集記号から構成される。 数字編集項目は、USAGE DISPLAY または USAGE NATIONAL のいずれかで表すことができる。

数字関数 (* numeric function)
クラスとカテゴリーは数字だが、考えられる評価のいくつかにおいて整数関数の要件を満たさないような関数。

数値項目 (* numeric item)
その内容の記述が、「0」から「9」までの数字から選択された文字で表される値に制限されるデータ項目。符号付きの場合は、その項目には +、-、または他の演算符号の表記を入れることもできる。

数値リテラル (* numeric literal)
1 つ以上の数字から構成されるリテラルで、小数点または代数符号あるいはその両方を含むことができる。 小数点は右端の文字であってはならない。 代数符号がある場合には、それが左端の文字でなければならない。

O

オブジェクト (object)
状態 (そのデータ値) および演算 (そのメソッド) を持つエンティティー。 オブジェクトは状態と動作をカプセル化する手段である。 クラス内の各オブジェクトは、そのクラスの 1 つのインスタンスであると言われる。

オブジェクト・コード (object code)
コンパイラーまたはアセンブラーからの出力。 それ自体が実行可能なマシン・コードか、 またはその種のコードの作成を目的としての処理に適する。

*** OBJECT-COMPUTER**
ENVIRONMENT DIVISION にある段落の名前であり、ここではオブジェクト・プログラムが実行されるコンピューター環境が記述される。

オブジェクト・コンピューター記入項目 (* object computer entry)
ENVIRONMENT DIVISION の OBJECT-COMPUTER 段落内の記入項目。この記入項目には、オブジェクト・プログラムが実行されるコンピューター環境を記述する節が入っている。

オブジェクト・デック (object deck)
リンケージ・エディターへの入力として適切なオブジェクト・プログラムの部分。「オブジェクト・モジュール (*object module*)」および「テキスト・デック (*text deck*)」と同義。

オブジェクト・インスタンス (object instance)
単一の、場合によっては多数のオブジェクト。COBOL クラス定義の Object 段落における指定に基づいて インスタンス生成される。 オブジェクト・インスタンスは、そのクラス定義に記述されたデータおよびすべての継承データのコピーを保持する。 オブジェクト・インスタンスに関連付けられたメソッドには、そのクラス定義で定義されたメソッドおよびすべての 継承メソッドが含まれる。

オブジェクト・インスタンスは Java クラスのインスタンスにすることができる。

オブジェクト・モジュール (object module)
オブジェクト・デック *(object deck)* または テキスト・デック *(text deck)* と同義。

項目のオブジェクト (* object of entry)
COBOL プログラムの DATA DIVISION 記入項目内の一連のオペランドと予約語であり、その記入項目のサブジェクトの直後に続く。

オブジェクト指向プログラミング (object-oriented programming)
カプセル化および継承の概念に基づいたプログラミング・アプローチ。 プロシージャー型プログラミング技法とは異なり、オブジェクト指向プログラミングでは、何かが達成される方法ではなく、問題を含むデータ・オブジェクトとその操作方法に重点を置く。

オブジェクト・プログラム (object program)
問題を解決するためにデータと相互に作用することを目的とする実行可能なマシン言語命令とその他の要素の集合または グループ。 このコンテキストでは、オブジェクト・プログラムとは一般に、COBOL コンパイラーがソース・プログラムまたはクラス定義を操作した 結果得られるマシン言語である。 あいまいになる危険がない場合には、オブジェクト・プログラム という用語の代わりに プログラム というワードだけが使用される。

オブジェクト・リファレンス (object reference)
クラスのインスタンスを識別する値。 クラスが指定されなかった場合、オブジェクト参照は一般的なものとなり、任意のクラスのインスタンスに適用できる。

オブジェクト時 (*object time)
オブジェクト・プログラムが実行されるとき。 実行時 *(run time)* と同義。

廃止される言語エレメント (* obsolete element)
2002 COBOL 標準 から削除された 85 COBOL 標準 の COBOL 言語エレメント。

ODO オブジェクト (ODO object)
次の例では、X が OCCURS DEPENDING ON 節のオブジェクト (ODO オブジェクト) である。

```
WORKING-STORAGE SECTION.
01  TABLE-1.
    05  X                   PIC S9.
    05  Y OCCURS 3 TIMES
          DEPENDING ON X    PIC X.
```

ODO オブジェクトの値によって、テーブル内の ODO サブジェクトの数が決まる。

ODO 対象 (ODO subject)
上記の例では、Y が OCCURS DEPENDING ON 節の サブジェクト (ODO サブジェクト) である。 テーブル内の ODO サブジェクトの数である Y の値は、X の値によって決まる。

オープン・モード (* open mode)
OPEN ステートメントが実行されてから、REEL および UNIT 句の指定のない CLOSE ステートメントが実行される前までのファイルの状態。 個々のオープン・モードは、OPEN ステートメントの中で、INPUT、OUTPUT、I-O、または EXTEND のいずれかとして指定する。

オペランド (* operand)
(1) オペランドの一般的な定義は、「操作の対象となるコンポーネント」である。 (2) 本書の目的に沿った言い方をすれば、ステートメントや記入項目の形式中に現れる小文字または日本語で書かれた語 (または語群) はオペランドと見なされ、そのオペランドによって指示されたデータに対して暗黙の参照を行う。

演算、操作 (operation)
オブジェクトに関して要求できるサービス。

演算符号 (* operational sign)
値が正であるか負であるかを示すために数字データ項目または数値リテラルに付けられる代数符号。

オプション・ファイル (optional file)
オブジェクト・プログラムが実行されるたびに必ずしも使用可能でなくてもよいものとして宣言されているファイル。

オプショナル・ワード (* optional word)
言語を読みやすくする目的でのみ特定の形式で含められる予約語。 このようなワードが表示されている形式をソース単位内で使用する場合、そのワードの有無はユーザーが選択できる。

出力ファイル (* output file)
出力モードまたは拡張モードのいずれかでオープンされるファイル。

出力モード (* output mode)
OUTPUT または EXTEND 句の指定のある OPEN ステートメントが実行されてから、 REEL および UNIT 句の指定のない CLOSE ステートメントが実行される前までのファイルの状態。

出力プロシージャー (* output procedure)
形式 1 SORT ステートメントの実行中にソート機能が完了した後で制御が渡されるステートメントの集合、または MERGE ステートメントの実行中に、要求があればマージ機能がマージ済みの順序になっているレコードのうち次のレコードを選択できるようになった後で制御が渡されるステートメントの集合。

オーバーフロー条件 (overflow condition)
ある演算結果の一部が意図した記憶単位の容量を超えたときに起こる条件。

多重定義 (overload)
同じクラスで使用可能な別のメソッドと同一の名前を使い (ただし、異なるシグニチャーを使用して)、メソッドを定義すること。 シグニチャー *(signature)* も参照。

指定変更 (override)
サブクラスのインスタンス・メソッド (親クラスから継承された) を再定義すること。

P

パッケージ (package)
関連する Java クラスの集まり。個々に、または全体としてインポートすることができる。

パック 10 進数データ項目 (packed-decimal data item)
内部 *10* 進数データ項目 *(internal decimal data item)* を参照。

埋め込み文字 (padding character)
物理レコード内の未使用文字位置を埋めるのに使用される英数字または国別文字。

ページ (page)
データの物理的分離を表す、出力データの垂直分割。 分離は、内部論理要件または出力メディアの外部特性、あるいはその両方に基づいて行われる。

ページ本体 (* page body)
行を記述できる、または行送りすることができる (またはその両方ができる) 論理ページの部分。

段落 (* paragraph)
PROCEDURE DIVISION では、段落名の後に分離文字ピリオドが続き、その後に 0 個以上の文が続く。 IDENTIFICATION DIVISION および ENVIRONMENT DIVISION では、段落ヘッダーの後に 0 個以上の記入項目が続く。

段落ヘッダー (* paragraph header)
予約語の後に分離文字ピリオドが付いたもので、IDENTIFICATION DIVISION および ENVIRONMENT DIVISION において段落の始まりを示すもの。 IDENTIFICATION DIVISION で許可されている段落ヘッダーは次のとおり。

```
PROGRAM-ID. (Program IDENTIFICATION
   DIVISION)
CLASS-ID. (Class IDENTIFICATION DIVISION)
METHOD-ID. (Method IDENTIFICATION
   DIVISION)
AUTHOR.
INSTALLATION.
DATE-WRITTEN.
DATE-COMPILED.
SECURITY.
```

ENVIRONMENT DIVISION で許可されている段落ヘッダーは次のとおり。

```
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
REPOSITORY. (Program or Class
   CONFIGURATION SECTION)
FILE-CONTROL.
I-O-CONTROL.
```

段落名 (* paragraph-name)
PROCEDURE DIVISION の中の段落を識別し開始するユーザー定義語。

パラメーター (parameter)
(1) 呼び出し側プログラムと呼び出し先プログラムの間で受け渡されるデータ。 (2) メソッド呼び出しの USING 句内のデータ・エレメント。引数によって、呼び出されたメソッドが要求された操作を実行するために使用できる追加情報を与える。

Persistent Reusable JVM
トランザクション間で JVM をリセットすることによりトランザクション処理用にシリアルに再利用できる JVM。 リセット・フェーズでは、JVM が既知の初期状態に復元される。

句 (* phrase)
連続する 1 つ以上の COBOL 文字ストリングを配列したセットで、COBOL プロシージャー・ステートメントまたは COBOL 節の一部を構成する。

物理レコード (* physical record)
ブロック *(block)* を参照。

ポインター・データ項目 (pointer data item)
アドレス値を保管できるデータ項目。 これらのデータ項目は、USAGE IS POINTER 節を使用してポインターとして明示的に定義される。 ADDRESS OF 特殊レジスターは、ポインター・データ項目として暗黙的に定義されている。 ポインター・データ項目は、 他のポインター・データ項目と等しいかどうかを比較したり、 他のポインター・データ項目に内容を移動することができる。

移植する、ポート (port)
(1) 異なるプラットフォームで実行できるようにコンピューター・プログラムを変更すること。 (2) インターネット・プロトコルでは、Transmission Control Protocol (TCP) プロトコルまたは User Datagram Protocol (UDP) プロトコルと高水準のプロトコルまたはアプリケーションの間の特定の論理結合子。 ポートはポート番号によって識別される。

可搬性 (portability)
あるアプリケーション・プラットフォームから別のアプリケーション・プラットフォームに、ソース・プログラムに比較的わずかな変更を加えるだけでアプリケーション・プログラムを移行できる能力。

合成済み文字 (precomposed character)
標準分解により複数の Unicode 文字を使用して表すことができる単一の Unicode 文字。 合成済み文字は合成文字形式と同じ物理表記を持たない。 例えば、Unicode 文字 U+00E4 (ä) は、Unicode 文字 U+0061 + U+0308 (ä) (ラテン語小文字 a + 結合発音区別符号) の組み合わせとして表すことができる

合成済み文字。 通常、合成済み文字は、発音区別符号を持つラテン語文字や他の結合文字を表すために使用される。

事前初期設定 (preinitialization)
プログラム (特に非 COBOL プログラム) からの複数の呼び出しの準備として の COBOL ランタイム環境の初期設定。 この環境は、明示的に終了されるまで終了されない。

基本レコード・キー (* prime record key)
索引付きファイルのレコードを固有なものとして識別する内容を持つキー。

優先順位番号 (* priority-number)
セグメンテーションの目的で、PROCEDURE DIVISION 内のセクションを分類するユーザー定義語。セグメント番号には 0 から 9 までの文字だけしか使用できない。セグメント番号は 1 桁または 2 桁として表すことができる。

private
ファクトリー・データまたはインスタンス・データに適用されるため、そのデータを定義するクラスのメソッドだけがアクセス可能である。

プロシージャー (* procedure)
PROCEDURE DIVISION 内にある 1 つの段落または論理的に連続する段落のグループ、あるいは 1 つのセクションまたは論理的に連続するセクションのグループ。

プロシージャー・ブランチ・ステートメント (* procedure branching statement)
ソース・コードの中にステートメントが書かれている順番どおりに次の実行可能ステートメントに制御の移動をせず、別のステートメントに明示的に制御の移動を引き起こすステートメント。 プロシージャー分岐ステートメントは次のとおり。ALTER、CALL、EXIT、EXIT PROGRAM、GO TO、MERGE (OUTPUT PROCEDURE 句付き)、PERFORM および SORT (INPUT PROCEDURE または OUTPUT PROCEDURE 句付き)、XML PARSE。

PROCEDURE DIVISION
COBOL の部の 1 つで、問題を解決するための命令を記述する。

プロシージャー統合 (procedure integration)
COBOL 最適化プログラムの機能の 1 つであり、実行されるプロシージャーまたは含まれ ているプログラムへの呼び出しを単純化する。

PERFORM プロシージャー統合とは、PERFORM ステートメントが、実行されるプロシージャーによって置き換えられるプロセスのこと。 含まれているプログラムのプロシージャー統合とは、含まれているプログラムへの呼び出しがプログラム・コードによって置き換えられるプロセスのこと。

プロシージャー名 (* procedure-name)
PROCEDURE DIVISION の中にある段落またはセクションに名前を付けるために使用される ユーザー定義語。 プロシージャー名は、段落名 (これは修飾することができる) またはセクション名から構成される。

プロシージャー・ポインター (procedure pointer)
入り口点を指すポインターを保管できるデータ項目。 USAGE IS PROCEDURE-POINTER 節を付けて定義したデータ項目が、 プロシージャーへの入り口点のアドレスを収める。

プロシージャー・ポインター・データ項目 (procedure-pointer data item)
入り口点を指すポインターを保管できるデータ項目。 USAGE IS PROCEDURE-POINTER 節で定義されるデータ項目には、プロシージャー入り口点のアドレスが入っている。一般的に、COBOL および言語環境プログラムのプログラムと通信するために使用される。

プロセス (process)
プログラムの全部または一部の実行中に発生する一連のイベント。 複数のプロセスを並行して実行することができ、1 つのプロセス内で実行されるプログラムはリソースを共用することができる。

プログラム (program)
(1) コンピューターによる処理に適した一連の命令。 処理には、コンパイラーを使用してプログラムの実行準備をすることやランタイム環境を使用してプログラムを実行することが含まれます。 (2) 1 つ以上の相互に関係のあるモジュールの論理アセンブリー。 同じプログラムの複数のコピーを異なるプロセスで実行することができます。

プログラム名 (program-name)
IDENTIFICATION DIVISION とプログラム終了マーカーにおいて、COBOL ソース・プログラムを識別するユーザー定義語または英数字リテラル。

プログラム識別記入項目 (* program identification entry)
IDENTIFICATION DIVISION の PROGRAM-ID 段落内の記入項目であり、プログラム名を指定し、選択されたプログラム属性をプログラムに割り当てる節が入っている。

プログラム名 (program-name)
IDENTIFICATION DIVISION およびプログラム終了マーカーにおいて、COBOL ソース・プログラムを識別するユーザー定義語または英数字リテラル。

プロジェクト (project)
ダイナミック・リンク・ライブラリー (DLL) や他の実行可能ファイル (EXE) などのターゲットを作成するのに必要な、データおよびアクションの完全セット。

疑似テキスト (* pseudo-text)
ソース・プログラムまたは COBOL ライブラリーにおいて、疑似テキスト区切り文字によって区切られた一連のテキスト・ワード、コメント行、インライン・コメント、または区切り文字スペース (疑似テキスト区切り文字を含まない)。

疑似テキスト区切り文字 (* pseudo-text delimiter)
疑似テキストを区切るために使用される隣接した 2 つの等号文字 (==)。

句読文字 (* punctuation character)
以下のセットに属する文字。

文字	意味
,	コンマ
;	セミコロン
:	コロン
.	ピリオド (終止符)
"	引用符
(	左括弧
)	右括弧
	スペース
=	等号

Q

QSAM (待機順次アクセス方式) (QSAM (Queued Sequential Access Method))
基本順次アクセス方式 (BSAM) の拡張版。 この方式を使用する場合、キューは、処理を待機する入力データ・ブロック、 または処理が終了して補助ストレージまたは出力装置への転送を待機する出力データ・ブロックで形成される。

修飾されたデータ名 (* qualified data-name)
データ名と、その後に連結語の OF または IN とデータ名修飾子を続けたものが 1 つ以上のセットで続いて構成される ID。

修飾子 (* qualifier)
(1) レベル標識と関連付けられるデータ名または名前であり、参照の際に、別のデータ名 (修飾子に従属する項目の名前) と 一緒に、または条件名と一緒に使用される。 (2) セクション名。 そのセクションの中で指定されている段落名と共に参照する際に使用される。 (3) ライブラリー名。 そのライブラリーと関連付けられたテキスト名と共に参照する際に使用される。

R

ランダム・アクセス (* random access)
キー・データ項目のプログラム指定値を使って、 相対ファイルまたは索引付きファイルから取り出したり、削除したり、 またはそこに入れたりする論理レコードを識別するアクセス・モード。

レコード (* record)
論理レコード *(logical record)* を参照。

レコード域 (* record area)
DATA DIVISION の FILE SECTION 内のレコード記述項目で記述されるレコードを処理する目的で割り振られるストレージ域。 FILE SECTION では、レコード域の現行の文字位置の数は、明示または暗黙の RECORD 節によって決められる。

レコード記述 (* record description)
レコード記述項目 *(record description entry)* を参照。

レコード記述項目 (* record description entry)
特定のレコードに関連したデータ記述項目全体。 「レコード記述 *(record description)*」と同義。

記録モード (recording mode)
ファイル内の論理レコードの形式。 レコード・モードは、F (固定長)、V (可変長)、S (スパン)、 または U (不定フォーマット) とすることができる。

レコード・キー (record key)
索引付きファイル内のレコードを識別する内容を持つキー。

レコード名 (* record-name)
COBOL プログラムの DATA DIVISION 内のレコード記述項目で記述されるレコードに名前を付けるユーザー定義語。

レコード番号 (* record number)
編成が順次であるファイル内のレコードの順序数。

レコード・モード (recording mode)
ファイル内の論理レコードの形式。 記録モードは、F (固定長)、V (可変長)、S (スパン)、または U (不定形式) とすることができる。

再帰 (recursion)
それ自体を呼び出すプログラム、または、それ自体で呼び出したプログラムのいずれかによって直接あるいは間接に呼び出されるプログラム。

再起可能 (recursively capable)
PROGRAM-ID ステートメントで RECURSIVE 属性が指定されていれば、プログラムは再帰可能である (再帰的に呼び出すことができる)。

リール (reel)
ストレージ・メディアの個別部分。その大きさはインプリメントする人によって決定され、1 つのファイルの一部、1 つのファイルの全部、または任意の個数のファイルが収容される。 「ユニット *(unit)*」および「ボリューム *(volume)*」と同義。

再入可能 (reentrant)
プログラムまたはルーチンの属性。この属性によって、プログラム・オブジェクトの 1 つのコピーを複数のユーザーが共用できる。

参照形式 (* reference format)
COBOL ソース・プログラムを記述するに際して標準的な方式を提供する形式。

参照変更 (reference modification)
新規のカテゴリー英数字、カテゴリー DBCS、またはカテゴリー国別のデータ 項目を定義する方法であり、USAGE DISPLAY、DISPLAY-1、または NATIONAL データ項目の左端文字および左端文字位置を基準にした長さを指定して定義する方法です。

参照修飾子 (* reference-modifier)
固有のデータ項目を定義する文字ストリングと分離文字の構文的に正しい組み合わせ。 区切り用の左括弧区切り文字、左端の文字位置、区切り文字のコロン、 任意指定の長さ、および区切り用の右括弧区切り文字を含む。

関係 (* relation)
関係演算子 *(relational operator)* または 比較条件 *(relation condition)* を参照。

比較文字 (* relation character)
以下のセットに属する文字。

文字	意味
>	より大きい
<	より小さい
=	に等しい

比較条件 (* relation condition)
ある算術式、データ項目、英数字リテラル、または索引名の値が、他の算術式、データ項目、英数字リテラル、または索引名の 値と特定の関係があるという命題 (それに対して真理値を判別する)。 関係演算子 *(relational operator)* も参照。

比較演算子 (* relational operator)
比較条件の構造で使用される、予約語、比較文字、連続する予約語のグループ、または 連続する予約語と比較文字のグループ。 使用できる演算子とそれらの意味は次のとおり。

文字	意味
IS GREATER THAN	より大きい
IS >	より大きい
IS NOT GREATER THAN	より大きくない
IS NOT >	より大きくない
IS LESS THAN	より小さい
IS <	より小さい
IS NOT LESS THAN	より小さくない
IS NOT <	より小さくない
IS EQUAL TO	に等しい
IS =	に等しい
IS NOT EQUAL TO	に等しくない
IS NOT =	に等しくない
IS GREATER THAN OR EQUAL TO	より大きいか等しい
IS >=	より大きいか等しい
IS LESS THAN OR EQUAL TO	より小さいか等しい
IS <=	より小さいか等しい

相対ファイル (* relative file)
相対編成のファイル。

相対キー (* relative key)
相対ファイルの中の論理レコードを識別するための内容を持つキー。

相対編成 (* relative organization)
各レコードが、レコードのファイル内における論理的順序位置を指定する 0 より大きい整数値によって、固有なものとして識別される永続的な論理ファイル構造。

相対レコード番号 (* relative record number)
相対編成ファイル内でのレコードの序数。 この数値は、整数の数値リテラルとして扱われる。

予約語 (* reserved word)
COBOL ソース・プログラムの中で使用することができるが、ユーザー定義語またはシステム名としてプログラムの中で使用されてはならないワードのリスト中に挙げられている COBOL ワード。

リソース (* resource)
オペレーティング・システムの制御下に置かれており、実行中のプログラムによって使用できる機能またはサービス。

結果の ID (* resultant identifier)
算術演算の結果が収められるユーザー定義のデータ項目。

再使用可能環境 (reusable environment)
事前初期設定用の古い COBOL インターフェース (RTEREUS ランタイム・オプション)、または言語環境プログラム・インターフェース CEEPIPI のいずれかを使用して、アセンブラー・プログラムをメインプログラムとして設定するときに、再使用可能環境が作成されます。

ルーチン (routine)
コンピューターに操作または一連の関連操作を実行させる、COBOL プログラム内の一連のステートメント。 言語環境プログラムでは、プロシージャー、関数、またはサブルーチンのいずれかを指す。

ルーチン名 (* routine-name)
COBOL 以外の言語で記述されたプロシージャーを識別するユーザー定義語。

実行時 (* run time)
オブジェクト・プログラムが実行されるとき。 「オブジェクト時 *(object time)*」と同義。

ランタイム環境 (runtime environment)
COBOL プログラムが実行される環境。

実行単位 (* run unit)
1 つの独立型オブジェクト・プログラム、あるいは COBOL の CALL または INVOKE ステートメントによって相互作用し、実行時に 1 つのエンティティーとして機能する複数のオブジェクト・プログラム。

S

SBCS
1 バイト文字セット *(SBCS) (single-byte character set (SBCS))* を参照。

範囲終了符号 (scope terminator)
PROCEDURE DIVISION の特定のステートメントの終わりを示す COBOL 予約語。これは明示的なもの (例えば、END-ADD など) であることもあれば、暗黙のもの (分離文字ピリオド) であることもある。

セクション (* section)
ゼロ、1 つ、または複数の段落またはエンティティー (セクション本体と呼ばれる) と、その最初のものの前にセクション・ヘッダーが付いているもの。 各セクションは、 セクション・ヘッダーとそれに関連付けられたセクション本体から構成される。

セクション・ヘッダー (* section header)
後ろに分離文字ピリオドが付いたワードの組み合わせで あり、ENVIRONMENT、DATA、または PROCEDURE の各部において、セクションの始まりを示すもの。 ENVIRONMENT DIVISION および DATA DIVISION では、 セクション・ヘッダーは、予約語の後に分離文字ピリオドを続けたものから構成される。 ENVIRONMENT DIVISION で許可されているセクション・ヘッダーは次のとおり。

```
CONFIGURATION SECTION.
INPUT-OUTPUT SECTION.
```

DATA DIVISION で許可されているセクション・ヘッダーは次のとおり。

```
FILE SECTION.
WORKING-STORAGE SECTION.
LOCAL-STORAGE SECTION.
LINKAGE SECTION.
```

PROCEDURE DIVISION では、セクション・ヘッダーは、セクション名、その後に続く予約語 SECTION、およびその後の分離文字ピリオドから構成される。

セクション名 (* section-name)
PROCEDURE DIVISION の中にあるセクションに名前を付けるユーザー定義語。

セグメント化 (segmentation)
85 COBOL 標準 分割モジュールに基づく Enterprise COBOL の機能。 セグメンテーション機能は、セクション・ヘッダーの優先順位番号を使用して、セクションを固定セグメントまたは独立セグメントに割り当てる。 セグメント種別は、セグメントに含まれるプロシージャーが初期状態の制御を受け取るか、最後に使われた状態の制御を受け取るかに影響を及ぼす。

選択構造 (selection structure)
条件が真であるか偽であるかに応じて、ある一連のステートメントか、または別の一連のステートメントが実行されるというプログラムの処理ロジック。

文 (* sentence)
1 つ以上のステートメントの並びで、その最後のものは、分離文字ピリオドで終了する。

別々にコンパイルされたプログラム (* separately compiled program)
あるプログラムをそこに含まれたプログラムと共に、 他のすべてのプログラムとは別個にコンパイルしたときのそのプログラム。

区切り文字 (* separator)
文字ストリングを区切るために使用される、1 文字または連続する 2 文字以上。

区切り文字のコンマ (* separator comma)
文字ストリングを区切るために使われる、後ろに 1 つのスペースが続く 1 つのコンマ (,)。

分離文字ピリオド (* separator period)
文字ストリングを区切るために使われる、後ろに 1 つのスペースが続く 1 つのピリオド (.)。

区切り文字のセミコロン (* separator semicolon)
文字ストリングを区切るために使われる、後ろに 1 つのスペースが続く 1 つのセミコロン (;)。

順序構造 (sequence structure)
一連のステートメントが、順序どおりに実行されるプログラムの処理ロジック。

順次アクセス (* sequential access)
ファイル内のレコードの並び方によって規定されている、 論理レコードの連続した前後関係順に、論理レコードをファイルから取り出したり、 ファイルに書き込んだりするアクセス・モード。

順次ファイル (* sequential file)
順次編成のファイル。

順次編成 (* sequential organization)
レコードがファイルに書き込まれるときに確定されたレコードの前後関係によって識別されるような永続的な論理ファイル構造。

逐次探索 (serial search)
最初のメンバーから始めて最後のメンバーで終わるように、 ある集合のメンバーが連続的に検査される探査方法。

Session Bean
EJB において、クライアントによって作成され、通常は 1 つのクライアント/サーバー・セッションの期間だけ存在する Enterprise Bean。 (Oracle)

77 レベル記述記入項目 (77-level-description-entry)
レベル番号 77 を持つ不連続データ項目を記述するデータ記述記入項目。

符号条件 (* sign condition)
データ項目や算術式の代数値が、0 より小さいか、大きいか、または等しいかという命題で、それに関して真理値が判別できる。

シグニチャー (signature)
(1) ある操作とそのパラメーターの名前。 (2) あるメソッドの名前と その仮パラメーターの数と型。

単純条件 (* simple condition)
以下のセットから選択される任意の単一条件。

- 比較条件
- クラス条件

- 条件名条件
- スイッチ状況条件
- 符号条件

条件 *(condition)* および 単純否定条件 *(negated simple condition)* も参照。

1 バイト文字セット (single-byte character set (SBCS))
各文字が 1 バイトで表現される文字のセット。 *ASCII* および *EBCDIC (*拡張 *2* 進化 *10* 進コード*) (EBCDIC (Extended Binary-Coded Decimal Interchange Code))* も参照。

遊びバイト (レコード内) (slack bytes (within records))
複数の基本データ項目を正しく位置合わせするために、コンパイラーによってデータ項目間に挿入されるバイト。 遊びバイトには意味のあるデータは含まれない。 正しい位置合わせを行うために遊びバイトが必要なときは、SYNCHRONIZED 節によって、コン パイラーに遊びバイトを挿入させる。

遊びバイト (レコード間) (slack bytes (between records))
複数の基本データ項目を正しく位置合わせするために、プログラマーによってファイルのブロック化論理レコードの間に挿入されるバイト。 場合によっては、レコード間に遊びバイトを挿入することによってバッファー内で処理されるレコードのパフォーマンスが改善される。

ソート・ファイル (* sort file)
形式 1 SORT ステートメントによってソートされるレコードの集まり。ソート・ファイルは、ソート機能によってのみ作成され使用される。

ソート・マージ・ファイル記述項目 (* sort-merge file description entry)
DATA DIVISION の FILE SECTION の中にある記入項目。レベル標識 SD と、それに続くファイル名、および、必要に応じて、次に続く一連のファイル節から構成される。

*** SOURCE-COMPUTER**
ENVIRONMENT DIVISION にある段落の名前であり、ここではソース・プログラムがコンパイルされるコンピューター環境が記述される。

コンパイル用コンピューター記入項目 (* source computer entry)
ENVIRONMENT DIVISION の SOURCE-COMPUTER 段落内の記入項目であり、ソース・プログラムがコンパイルされるコンピューター環境を記述する節が入っている。

ソース項目 (* source item)
SOURCE 節によって指定される ID で、印刷可能な項目の値を提供する。

ソース・プログラム (source program)
ソース・プログラムは、他の形式や記号を使用して表現することができるが、本書では、構文的に正しい COBOL ステートメントの集合を常に指している。 COBOL ソース・プログラムは、IDENTIFICATION DIVISION または COPY ステートメントで開始され、指定された場合はプログラム終了マーカーで終了するか、または追加のソース・プログラム行なしで終了する。

ソース単位 (source unit)
COBOL ソース・コードの 1 単位で、個別にコンパイルできる。プログラムまたはクラス定義。 コンパイル単位 とも呼ばれる。

特殊文字 (special character)
以下のセットに属する文字。

文字	意味
+	正符号
-	負符号 (-) (ハイフン)
*	アスタリスク
/	斜線 (スラッシュ)
=	等号
$	通貨符号
,	コンマ

文字	意味
;	セミコロン
.	ピリオド (小数点、終止符)
"	引用符
'	アポストロフィ
(	左括弧
)	右括弧
>	より大きい
<	より小さい
:	コロン
_	下線

SPECIAL-NAMES
ENVIRONMENT DIVISION にある段落の名前。この段落では、環境名がユーザー指定の簡略名と関連付けられる。

特殊名記入項目 (* special names entry)
ENVIRONMENT DIVISION の SPECIAL-NAMES 段落内の記入項目。この記入項目は、通貨記号を指定したり、小数点を選択したり、シンボリック文字を指定したり、インプリメントする人の名前をユーザー指定の簡略名と関連付けたり、英字名を文字セットまたは照合シーケンスと関連付けたり、クラス名を一連の文字と関連付けたりするための手段を提供する。

特殊レジスター (* special registers)
コンパイラーの生成する特定のストレージ域のことで、 その基本的な使用法は、具体的な COBOL 機能を使用したときに作り出される情報を記憶することである。

標準データ・フォーマット (* standard data format)
COBOL データ部でデータの特性を記述するために使用される概念。この概念の もとでは、データの特性は、データが内部的にコンピューターに、または特定の外部メディアに保管される方法に適した形式ではなく、印刷ページ上での無限の長さと幅を持つデータ 外観に適した形式で表現される。

ステートメント (* statement)
COBOL ソース・プログラムに書かれる、動詞を冒頭に置いた、ワード、リテラル、および区切り記号の構文的に正しい組み合わせ。

構造化プログラミング (structured programming)
コンピューター・プログラムを編成してコーディングするための技法であり、この技法 では、プログラムはセグメントの階層で構成され、それぞれのセグメントには 1 つの 入り口点と 1 つの出口点がある。 制御は、構造の下方へと渡され、 階層内のより上位レベルへの無条件ブランチは行われない。

サブクラス (* subclass)
別のクラスから継承するクラス。 継承関係にある 2 つのクラスをまとめて考える場合、継承する側、つまり継承先のクラスをサブクラスといい、継承される側、つまり継承元のクラスをスーパークラスという。

項目のサブジェクト (* subject of entry)
DATA DIVISION の記入項目内において、レベル標識またはレベル番号の直後に現れるオペランドまたは予約語。

サブプログラム (* subprogram)
呼び出し先プログラム *(called program)* を参照。

添え字 (* subscript)
整数、(オプションで演算子 + または - 付きの整数が後ろにある) データ名、あるいは (オプションで演算子 + または - 付きの整数が後ろにある) 索引名のいずれかによって表されるオカレンス番号。これによりテーブル内の特定のエレメントを識別する。 可変数の引数を認める関数では、添え字付き ID を関数引数として使用する場合は、添え字に ALL を使用することができる。

添え字付きデータ名 (* subscripted data-name)
データ名とその後の括弧で囲まれた 1 つ以上の添え字から構成される ID。

置換文字 (substitution character)
ソース・コード・ページからターゲット・コード・ページへの変換の際に、ターゲット・コード・ページで定義されていない文字を表すのに使用される文字。

スーパークラス (* superclass)
別のクラスによって継承されるクラス。 サブクラス *(subclass)* も参照。

サロゲート・ペア (surrogate pair)
UTF-16 形式のユニコードで、共に 1 つのユニコード図形文字を表すエンコード方式ペアの単位。 ペアの最初の単位は上位サロゲート と呼ばれ、第 2 の単位は下位サロゲート と 呼ばれる。 上位サロゲートのコード値の範囲は、X'D800' から X'DBFF' である。 下位サロゲートのコード値の範囲は、X'DC00' から X'DFFF' である。 サロゲート・ペアは、Unicode 16 ビット・コード化 文字セットに適合する文字を 65,536 文字を超えて提供する。

スイッチ状況条件 (switch-status condition)
オンまたはオフに設定可能な UPSI スイッチが、特定の状況に設定されているという命題で、これに関して真理値を判別することができる。

シンボリック文字 (* symbolic-character)
ユーザー定義の形象定数を指定するユーザー定義語。

構文 (syntax)
(1) 意味や解釈および使用の方法に依存しない、文字同士または文字のグループ同士の間の関係。 (2) 言語における表現の構造。 (3) 言語構造を支配する規則。 (4) 記号相互の関係。 (5) ステートメントの構築にかかわる規則。

システム名 (* system-name)
オペレーティング環境と連絡し合うために使用される COBOL ワード。

T

テーブル (* table)
DATA DIVISION の中で OCCURS 節によって定義される、論理的に連続するデータ項目の集合。

テーブル・エレメント (* table element)
テーブルを構成する繰り返し項目の集合に属するデータ項目。

テキスト・デック (text deck)
オブジェクト・デック *(object deck)* または オブジェクト・モジュール *(object module)* と同義。

テキスト名 (* text-name)
ライブラリー・テキストを識別するユーザー定義語。

テキスト・ワード (* text word)
以下のいずれかの文字から成る COBOL ライブラリー、ソース・プログラム、または疑似テキスト内のマージン A およびマージン R の間の、1 文字または連続した文字のシーケンス。

- スペース以外の区切り記号、疑似テキスト区切り文字、英数字リテラルの開始と終了の区切り文字。ライブラリー、ソース・プログラム、または疑似テキスト内のコンテキストに関係なく、右括弧文字と左括弧文字は常に テキスト・ワードと見なされる。
- リテラル。英数字リテラルの場合には、そのリテラルの境界となる開始の引用符と終了の引用符を含むリテラル。
- コメント行および区切り記号によって囲まれたワード COPY を除く、その他の連続する一連の COBOL 文字で、区切り記号でもリテラルでもないもの。

スレッド (thread)
プロセスの制御下にあるコンピューター命令のストリーム (プロセス内のアプリケーションによって開始される)。

トークン (token)
COBOL エディターでは、プログラムにおける意味の単位。 トークンには、データ、言語キーワード、ID、またはその他の言語構文の一部を含めることができる。

トップダウン設計 (top-down design)
関連付けられた諸機能が、構造の各レベルで実行されるようにする階層構造を使ったコンピューター・プログラムの設計。

トップダウン開発 (top-down development)
構造化プログラミング *(structured programming)* を参照。

トレーラー・ラベル (trailer-label)
(1) 記録メディア・ユニットのデータ・レコードの後にある、データ・セットのラベル。 (2)「ファイル終わりラベル *(end-of-file label)*」の同義語。

トラブルシューティング (troubleshoot)
コンピューター・ソフトウェアの使用中に問題を検出し、突き止め、除去すること。

真の値 (* truth value)
2 つの値 (真または偽) のどちらか一方によって、 条件評価の結果を表したもの。

型式化オブジェクト・リファレンス (typed object reference)
指定されたクラスまたはそのサブクラスのオブジェクトだけを参照できるデータ名。

U

単項演算子 (* unary operator)
正符号 (+) または負符号 (-)。 算術式の変数や算術式の左括弧の前に置き、それぞれ +1 または -1 を式に乗算する。

無制限テーブル (unbounded table)
上限として 整数-*2* を指定するのではなく、`OCCURS 整数-1 to UNBOUNDED` によるテーブル。

Unicode
現代世界の各国の言語で記述されるテキストの交換、処理、表示をサポートする汎用文字エンコード標準。UTF-8、UTF-16、UTF-32 など、Unicode を表現する複数のエンコード・スキームが ある。Enterprise COBOL では、国別データ・タイプの表記としてビッグ・エンディアン・フォーマットの UTF-16 を使用して Unicode をサポートしている。

URI (Uniform Resource Identifier (URI))
リソースを一意に指す文字のシーケンスのことで、Enterprise COBOL では、名前空間の ID。URI 構文は、文書「*Uniform Resource Identifier (URI): Generic Syntax*」で定義されています。.

ユニット (unit)
直接アクセスのモジュールであり、その大きさは IBM によって決められている。

汎用オブジェクト参照 (universal object reference)
どのクラスのオブジェクトでも参照できるデータ名。

非制限ストレージ (unrestricted storage)
2 GB バーより下のストレージ。 16 MB 境界より上または下がある。 16 MB 境界より上では、31 ビット・モードでのみ、アドレス可能。

不成功の実行 (* unsuccessful execution)
ステートメントの実行が試みられたが、そのステートメントに指定された操作すべてを実行できなかったこと。 あるステートメントの実行不成功は、そのステートメントによって参照されるデータには影響を及ぼさないが、状況表示には影響を与える可能性がある。

UPSI スイッチ (UPSI switch)
ハードウェア・スイッチの機能を実行するプログラム・スイッチ。 UPSI-0 から UPSI-7 の 8 つのスイッチがある。

URI
URI を参照。

ユーザー定義語 (* user-defined word)
節やステートメントの形式を満たすためにユーザーが提供する必要のある COBOL ワード。

V

変数 (* variable)
オブジェクト・プログラムの実行によって変更を受ける可能性のある値を持つデータ項目。 算術式で使われる変数は、数字基本項目でなければならない。

可変長項目 (variable-length item)
OCCURS 節の DEPENDING 句で記述された表を含んだグループ項目。

可変長レコード (* variable-length record)
ファイル記述項目またはソート・マージ・ファイル記述項目が、 文字位置の数が可変であるレコードを許容しているファイルに関連付けられているレコード。

可変オカレンス・データ項目 (* variable-occurrence data item)
可変オカレンス・データ項目とは、反復される回数が可変であるテーブル・エレメントを言う。 そのような項目は、そのデータ記述記入項目内に OCCURS DEPENDING ON 節を持っているか、またはそのような項目に従属していなければならない。

可変位置グループ (* variably located group)
同じレコード内の可変長テーブルに続くグループ項目 (可変長テーブルに従属するわけではない)。 グループ項目は、英数字グループでも国別グループでも構いません。

可変位置項目 (* variably located item)
同じレコード内の可変長テーブルに続くデータ項目 (可変長テーブルに従属するわけではない)。

動詞 (* verb)
COBOL コンパイラーまたはオブジェクト・プログラムによってとられる処置を表すワード。

ボリューム (volume)
外部ストレージのモジュール。 テープ装置の場合はリール、直接アクセス装置の場合はユニット。

ボリューム切り替え処理手順 (volume switch procedures)
ファイルの終わりに達する前にユニットまたはリールの終わりに達したとき、自動的に実行されるシステム固有の処理手順。

VSAM ファイル・システム (VSAM file system)
COBOL の順次編成、相対編成、および索引編成をサポートするファイル・システム。

W

Web サービス (web service)
特定のタスクを実行し、HTTP や SOAP といったオープン・プロトコルを介してアクセス可能なモジュラー・アプリケーション。

空白文字 (white space)
文書にスペースを挿入する文字。 空白文字には以下のものがある。

- スペース
- 水平タブ
- 復帰
- 改行
- 次の行

Unicode 標準では上記のように呼ばれる。

ワード (* word)
ユーザー定義語、システム名、予約語、または関数名を形成する、30 文字を超えない文字ストリング。

*** WORKING-STORAGE SECTION**
独立項目または WORKING-STORAGE レコード、あるいはその両方から構成される、WORKING-STORAGE データ項目を記述する DATA DIVISION のセクション。

ワークステーション (workstation)
コンピューターの総称 (パーソナル・コンピューター、3270 端末、インテリジェント・ワークステーション、および UNIX 端末を含む)。ワークステーションはメインフレームまたはネットワークに接続されることがよくある。

ラッパー (wrapper)
オブジェクト指向コードとプロシージャー指向コード間のインターフェースを提供するオブジェクト。 ラッパーを使用すると、他のシステムがプログラムを再利用したり、プログラムにアクセスしたりできるようになる。

X

x
PICTURE 節内の記号であり、コンピューターの有する文字セットの任意の文字を含めることができる。

XML
Extensible Markup Language。 マークアップ言語を定義するための標準メタ言語。SGML から派生した、SGML のサブセットである。 XML では、SGML の複雑で使用頻度の低い部分が省略され、文書タイプを扱うアプリケーションの作成、構造化情報の作成および管理、異種コンピューター・システム間での 構造化情報の伝送および共用がはるかに容易になっている。 XML を使用するときに、SGML で必要とされるような堅固なアプリケーションや 処理は不要である。 XML は、World Wide Web Consortium (W3C) の主導で開発された。

XML データ (XML data)
XML エレメントを持つ階層構造に編成されたデータ。 データ定義は XML エレメント・タイプ宣言で定義される。

XML 宣言 (XML declaration)
使用している XML のバージョンや文書のエンコードなど、XML 文書の特性を指定する XML テキスト。

XML 文書 (XML document)
W3C XML 規格で定義されているとおり正しい形式のデータ・オブジェクト。

XML ネーム・スペース (XML namespace)
W3C XML ネーム・スペース仕様によって定義されたメカニズムで、エレメント名および属性名の集まりの有効範囲を制限する。 一意的に選択された XML 名前空間によって、複数の XML 文書または XML 文書内の複数のコンテキストでエレメント名または属性名が一意的に識別されます。

XML スキーマ (XML schema)
W3C によって定義されたメカニズムで、XML 文書の構造と内容を記述し、制約する。 XML スキーマは、それ自体が XML で表され、特定タイプ (購入注文など) の XML 文書のクラスを効率的に定義する。

Z

z/OS UNIX ファイル・システム (z/OS UNIX file system)
階層構造で編成されたファイルとディレクトリーの集合であり、z/OS UNIX を使用してアクセスできる。

ゾーン 10 進数データ項目 (zoned decimal data item)
暗黙的または明示的に USAGE DISPLAY として記述されており、PICTURE の記号 9、S 、P、および V の有効な組み合わせを含んでいる、外部 10 進数データ項目。 ゾーン 10 進数データ項目の内容は、文字 0 から 9 で表され、必要に応じて符号が付きます。 PICTURE ストリングが符号を指定しており、SIGN IS SEPARATE 節が指定されている場合、符号は 文字 + または - として表されます。 SIGN IS SEPARATE が指定されていない場合、符号は、符号位置の最初の 4 ビットをオーバーレイする 1 つの 16 進数字です (先行または末尾)。

#

85 COBOL 標準 (85 COBOL Standard)
以下の標準によって定義された COBOL 言語。

- 「*ANSI INCITS 23-1985, Programming languages - COBOL*」は「*ANSI INCITS 23a-1989, Programming Languages - COBOL - Intrinsic Function Module for COBOL*」および「*ANSI INCITS 23b-1993, Programming Languages - Correction Amendment for COBOL*」に改訂されました。
- 「*ISO 1989:1985, Programming languages - COBOL*」は 「*ISO/IEC 1989/AMD1:1992, Programming languages - COBOL: Intrinsic function module*」および「*ISO/IEC 1989/AMD2:1994, Programming languages - Correction and clarification amendment for COBOL*」に改訂されました。

2002 COBOL 標準 (2002 COBOL Standard)
以下の標準によって定義された COBOL 言語。

- *INCITS/ISO/IEC 1989-2002, Information technology - Programming languages - COBOL*

2014 COBOL 標準 (2014 COBOL Standard)
以下の標準によって定義された COBOL 言語。

- *INCITS/ISO/IEC 1989:2014, Information technology - Programming languages, their environments and system software interfaces - Programming language COBOL*

リソース・リスト

Enterprise COBOL for z/OS

COBOL for z/OS の資料

以下の資料が「Enterprise COBOL for z/OS ライブラリー」にあります。

- *What's new*
- カスタマイズ・ガイド (SC43-3366-02)
- 言語解説書 (SC43-3367-02)
- プログラミング・ガイド (SC43-3368-02)
- 移行ガイド (GC43-3369-02)
- パフォーマンス・チューニング・ガイド (SC43-4104-01)
- メッセージおよびコード (SC43-4107-01)
- *Program Directory* (GI13-4526-02)
- *Licensed Program Specifications* (GI13-4532-02)

ソフトコピー資料

次のコレクション・キットには、Enterprise COBOL およびその他の製品資料が含まれます。 これらは http://www.ibm.com/e-business/linkweb/publications/servlet/pbi.wss にあります。

- *z/OS Software Products Collection*
- *z/OS and Software Products DVD Collection*

サポート

Enterprise COBOL for z/OS のご使用の際に問題がある場合は、サイト: https://www.ibm.com/support/home/product/B984385H82239E03/Enterprise_COBOL_for_z/OS を参照してください。そこでは最新のサポート情報が提供されています。

関連資料

z/OS ライブラリー資料

以下の資料が「z/OS ライブラリー」にあります。

ランタイム・ライブラリー拡張機能

- 共通デバッグ・アーキテクチャー ライブラリー・リファレンス
- 共通デバッグ・アーキテクチャー ユーザーズ・ガイド
- *DWARF/ELF* エクステンション ライブラリー・リファレンス

z/Architecture

- *z/Architecture* 解説書

z/OSDFSMS

- カタログのためのアクセス方式サービス・プログラム
- *Checkpoint/Restart*

- *Macro Instructions for Data Sets*
- データ・セットの使用法
- *Utilities*

z/OS DFSORT

- アプリケーション・プログラミング・ガイド
- インストールおよびカスタマイズ

z/OS ISPF

- ダイアログ開発者 ガイドとリファレンス
- ユーザーズ・ガイド 第 *1* 巻
- ユーザーズ・ガイド 第 *2* 巻

z/OS 言語環境プログラム

- 概念
- カスタマイズ
- デバッグのガイド
- *Language Environment Vendor Interfaces*
- プログラミング・ガイド
- プログラミング・リファレンス
- ランタイム・メッセージ
- ランタイム マイグレーション・ガイド
- *ILC (*言語間通信*)* アプリケーションの作成

z/OS MVS

- *JCL* 解説書
- *JCL* ユーザーズ・ガイド
- プログラミング*:* 高水準言語向け呼び出し可能サービス
- プログラム管理*:* ユーザーズ・ガイドおよび解説書
- システム・コマンド
- *z/OS Unicode Services* ユーザーズ・ガイドおよび解説書
- *z/OS XML System Services* ユーザーズ・ガイドおよび解説書

z/OS TSO/E

- コマンド解説書
- 入門
- ユーザーズ・ガイド

z/OS UNIX システム・サービス

- コマンド解説書
- プログラミング*:* アセンブラー呼び出し可能サービス 解説書
- ユーザーズ・ガイド

z/OS XL C/C++

- プログラミング・ガイド
- ランタイム・ライブラリー・リファレンス

CICS Transaction Server for z/OS

以下の資料が「CICS ライブラリー」にあります。

- *CICS* アプリケーションの開発
- *API (EXEC CICS)* リファレンス
- *CICS* システム・プログラムの開発
- グローバル・ユーザー出口リファレンス
- *XPI Reference*
- *CICS* での *EXCI* の使用

COBOL 報告書作成プログラム・プリコンパイラー

- *Programmer's Manual*、SC26-4301
- *Installation and Operation*、SC26-4302

Db2 for z/OS

以下の資料が「Db2 ライブラリー」にあります。

- アプリケーション・プログラミングおよび *SQL* ガイド
- コマンド解説書
- *SQL* 解説書

IBM Debug for z/OS (以前の IBM Debug for z Systems および IBM Debug Tool for z/OS)

IBM Debug for z/OS については、IBM Debug for z/OS ライブラリーを参照してください。

IBM Debug for z Systems および IBM Debug Tool for z/OS は、IBM Debug for z/OS に置き換えられています。COBOL 文書ライブラリーで、IBM Debug for z Systems および IBM Debug Tool for z/OS の参照がすべて変更されているわけではありません。 デバッガーを最新レベルにアップグレードして、デバッグ機能の全範囲を使用できるようにすることをお勧めします。場合によっては、COBOL アプリケーションの作成に使用している Enterprise COBOL のレベルに応じて、デバッガーを特定のバージョンにアップグレードする必要があります。

- IBM Debug Tool V13.1 は、Enterprise COBOL V5.1 以前のバージョンをサポートしています。
- IBM Debug for z Systems V14.0 は、Enterprise COBOL V6.1 以前のバージョンをサポートしています。
- IBM Debug for z Systems V14.1 は、Enterprise COBOL V6.2 以前のバージョンをサポートしています。
- IBM Debug for z/OS V14.2 は、Enterprise COBOL V6.3 以前のバージョンをサポートしています。

お客様のニーズに最も適している IBM デバッグ製品を見つけるには、https://www.ibm.com/support/knowledgecenter/SSQ2R2_14.2.0/com.ibm.debug.cg.doc/common/dcompo.html?sc=SSQ2R2_latest を参照してください。

IBM Developer for z/OS (以前の IBM Developer for z Systems)

IBM Developer for z Systems に関する情報は、IBM Developer for z/OS ライブラリーにあります。

注 : IBM Developer for z Systems および Rational® Developer for z Systems は IBM Developer for z/OS に置き換えられています。

以下の資料が「IBM Publications Center」にあり、資料番号で検索できます。

IMS

- *Application Programming API Reference*、SC18-9699
- *Application Programming Guide*、SC18-9698

WebSphere® Application Server for z/OS

- *Applications* , SA22-7959

Softcopy publications for z/OS

以下のコレクション・キットには、z/OS および関連製品資料が含まれます。

- *z/OS CD Collection Kit*、SK3T-4269

Java

- *IBM SDK for Java - Tools Documentation*、publib.boulder.ibm.com/infocenter/javasdk/tools/index.jsp
- *The Java 2 Enterprise Edition Developer's Guide*、download.oracle.com/javaee/1.2.1/devguide/html/DevGuideTOC.html
- *Java 2 on z/OS*、www.ibm.com/servers/eserver/zseries/software/java/
- *The Java EE 5 Tutorial*、download.oracle.com/javaee/5/tutorial/doc/
- *The Java Language Specification, Third Edition* (Gosling 他著)、java.sun.com/docs/books/jls/
- *The Java Native Interface*、download.oracle.com/javase/1.5.0/docs/guide/jni/
- *JDK 5.0 Documentation*、download.oracle.com/javase/1.5.0/docs/

JSON

- JavaScript Object Notation (JSON)、www.json.org

Unicode および文字表現

- *Unicode*、www.unicode.org/
- *Character Data Representation Architecture Reference and Registry*, SC09-2190

XML

- *Extensible Markup Language (XML)*、www.w3.org/XML/
- *Namespaces in XML 1.0*、www.w3.org/TR/xml-names/
- *Namespaces in XML 1.1*、www.w3.org/TR/xml-names11/
- *XML specification*、www.w3.org/TR/xml/

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]

A

B

C

D

E

F

H

I

L

M

N

O

P

Q

R

S

T

V

W

X

Z

プログラム番号: 5655-EC6

SC43-3366-02