IBM i
バージョン 7.2

プログラミング
IBM Rational Development Studio for i
ILE C/C++ コンパイラー参照

SC88-4025-03
(英文原典：SC09-4816-06)

IBM i
バージョン 7.2

プログラミング
IBM Rational Development Studio for i
ILE C/C++ コンパイラー参照

SC88-4025-03
(英文原典：SC09-4816-06)

ご注意!

本書および本書で紹介する製品をご使用になる前に、 A-1 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。

本製品およびオプションに付属の電源コードは、他の電気機器で使用しないでください。

本書は、IBM Rational Development Studio for i (製品番号 5770-WDS)、および新しい版で特に指定のない限り、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。このバージョンは、すべての RISC モデルで稼働するとは限りません。また、CISC モデル上では稼働しません。

本書にはライセンス内部コードについての参照が含まれている場合があります。ライセンス内部コードは機械コードであり、IBM 機械コードのご使用条件に基づいて使用権を許諾するものです。

お客様の環境によっては、資料中の円記号がバックスラッシュと表示されたり、バックスラッシュが円記号と表示されたりする場合があります。

原典： SC09-4816-06
IBM i
Version 7.2
Programming
IBM Rational Development Studio for i
ILE C/C++ Compiler Reference

発行： 日本アイ・ビー・エム株式会社

担当： トランスレーション・サービス・センター

第1刷 2014.4

目次

ILE C/C++ コンパイラー参照

ILE C/C++ コンパイラー参照

本書は、C および C++ プログラミング言語に詳しく、ILE C/C++ コンパイラーを使用して ILE C/C++ アプリケーションを新規作成または既存の ILE C/C++ アプリケーションを保守することを計画しているプログラマーを対象にしています。

IBM i 7.2 の新機能

新規または大幅に変更された情報について説明します。

- 新規 ILE C/C++ 事前定義マクロ。 4-2 ページの『ILE C/C++ 事前定義マクロ』を参照してください。
- 新規 LANGLVL(*EXTENDED0X) 制御言語コマンド・オプションの値。 6-25 ページの『LANGLVL』を参照してください。

「ILE C/C++ コンパイラー参照」の PDF ファイル

この情報の PDF ファイルを表示および印刷することができます。

本書の PDF 版を表示またはダウンロードするには、ILE C/C++ コンパイラー参照を選択してください。

PDF ファイルの保存

PDF を表示または印刷のためにワークステーションに保存するには、次のようにします。

1. ブラウザーで PDF リンクを右クリックします。
2. PDF をローカルに保存するオプションをクリックします。
3. PDF を保存したいディレクトリーを指定します。
4. 「**保存**」をクリックします。

Adobe Reader のダウンロード

これらの PDF を表示または印刷するには、ご使用のシステムに Adobe Reader がインストールされている必要があります。このアプリケーションは、Adobe Web サイト

(www.adobe.com/products/acrobat/readstep.html)  から無料でダウンロードできます。

ILE C/C++ コンパイラー参照について

このセクションでは、コンパイラー参照情報の概要を説明します。

読者には、該当する IBM i メニューや表示、または Control Language (CL) コマンドの使用経験が必要です。また、*ILE 概念* で説明されている ILE の知識も必要です。

前提条件および関連情報

IBM® i の技術情報を検索するには、まず IBM i Information Center をご利用ください。Information Center には以下の Web サイトからアクセスできます。

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

IBM i Information Center には、ソフトウェアのインストール、Linux、WebSphere®、Java™、高可用性、データベース、論理区画、CL コマンド、システム・アプリケーション・プログラミング・インターフェース (API) など、システムに関する最新の更新情報が収められています。さらに、システムのハードウェアおよびソフトウェアの計画、トラブルシューティング、および構成を支援するアドバイザーや検索機能も提供します。

その他の関連情報については、 10-1 ページの『関連情報』を参照してください。

ライセンス・プログラム情報のインストール

ILE C/C++ コンパイラーを使用するシステムには、QSYSINC ライブラリーをインストールする必要があります。

例に関する注

ILE C/C++ コンパイラーの使用方法を示している例は、簡素化されています。これらの例では、C または C++ 言語構成の使用についてのすべては説明しません。例の中には、コードの一部分だけを示し、コードを追加しないとコンパイルできないものもあります。

制御言語コマンド

プロンプトが必要な場合は、CL コマンドを入力して、F4 (プロンプト) を押します。オンライン・ヘルプ情報が必要な場合は、CL コマンド・プロンプトの画面で F1 (ヘルプ) を押します。 CL コマンドはバッチまたは対話モードで使用するか、制御言語プログラムから使用することができます。

CL コマンドの詳細については、以下の IBM i Information Center Web サイトにある「プログラミング」カテゴリーの『*CL および API*』セクションを参照してください。

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

CL コマンドを使用するには、オブジェクト権限が必要です。オブジェクト権限について詳しくは、Information Center Web サイトにある「セキュリティー」カテゴリーの中の『システム・セキュリティーの*計画とセットアップ*』セクションを参照してください。

構文図の見方

- 構文図は、直線で示される経路にしたがって、左から右、上から下の方向に読んでください。

  ▸▸── は、コマンド、ディレクティブ、またはステートメントの先頭を示します。

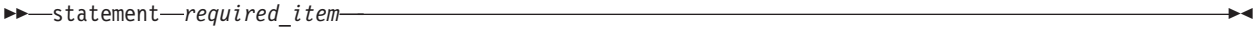
  ──▸ は、コマンド、ディレクティブ、またはステートメント構文が、次の行に続いていることを示します。

  ▸── は、コマンド、ディレクティブ、またはステートメントが、前の行から続いていることを示します。

  ──▸◂ は、コマンド、ディレクティブ、またはステートメントの終わりを示します。

  完全なコマンド、ディレクティブ、またはステートメント以外の構文単位の図は、 ▸── 記号で始まり、──▸ 記号で終わります。

  注: 以下のダイアグラムで、`statement` は、C または C++ コマンド、ディレクティブ、またはステートメントを表しています。
- 必須項目は、次のように水平方向の線 (メインパス) 上に示します。



- 任意指定項目は、次のようにメインパスの下に示します。

- 2 つ以上の項目から選択可能な場合は、スタック内に垂直に記述されます。

  いずれか 1 つの項目の選択が*必須* の場合は、スタック内の項目のいずれか 1 つがメインパス上に記述されます。

  項目を選択しても選択しなくてもよい場合は、縦方向に並んでいる選択項目をすべてメインパスの下に示します。

  デフォルト項目は、メインパスの上に記述されます。

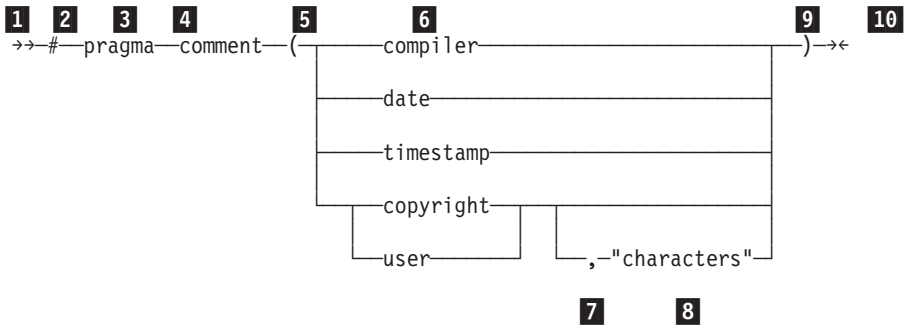
- メインパスの線の上の左に戻る矢印は、繰り返し可能な項目を示します。

スタックの上の繰り返し矢印は、スタック内の項目から複数の項目を選択するか、1 つの項目を繰り返し選択できることを示しています。

- キーワードは、非イタリック体で記述されています。示されているとおりに正確に入力する必要があります (例えば extern)。

  変数は、イタリック体の小文字で記述されます (例えば、*identifier*)。これらはユーザーが指定する名前または値を表しています。
- 構文図に句読記号、括弧、算術演算子、またはその他の記号が示されている場合には、それらを構文の一部として入力しなければなりません。

次の構文図の例では、**#pragma comment** ディレクティブの構文を示しています。 **#pragma** ディレクティブについては、 5-1 ページの『ILE C/C++ プラグマ』を参照してください。



1 構文図の始まりを示します。

2 記号 # を最初に記述します。

3 キーワード pragma は、シンボル # の次に記述されます。

4 キーワード comment は、キーワード pragma の次に記述されます。

5 左括弧が必要です。

6 コメントの型を、表示されている compiler、date、timestamp、copyright、または user のうちいずれか 1 つだけ入力します。

7 コンマが、コメントの型 copyright または user とオプションの文字ストリングの間に必要です。

8 文字ストリングをコンマの次に記述します。文字ストリングは、二重引用符で囲みます。

9 右小括弧は必須です。

10 これが、構文図の終わりを示します。

次の **#pragma comment** ディレクティブの例は、上記のダイアグラムに従っており、構文上正しい例です。

```
#pragma comment(date)
#pragma comment(user)
#pragma comment(copyright,"This text will appear in the module")
```

業界標準

統合言語環境 C/C++ コンパイラーおよびランタイム・ライブラリーは、以下の標準に基づいて設計されています。

- Information Technology - Programming languages - C, ISO/IEC 9899:1990 (C89 とも呼ばれる)
- Information Technology - Programming languages - C, ISO/IEC 9899:1999 (C99 とも呼ばれる)
- Information Technology - Programming languages - C++, ISO/IEC 14882:1998 (C++98 とも呼ばれる)
- Information Technology - Programming languages - C++, ISO/IEC 14882:2003(E) (標準 C++ とも呼ばれる)
- Information Technology - Programming languages - Extension for the programming language C to support decimal floating-point arithmetic, ISO/IEC WDTR 24732. このドラフトのテクニカル・レポートは、C 標準委員会に提出されており、 http://www.open-std.org/JTC1/SC22/WG14/www/docs/n1176.pdf で入手できます。

ILE C は C99 機能のサブセットをサポートします。

ILE C++ は C++0x 機能のサブセットをサポートします。

注: C++0x は、新しいバージョンの C++ プログラミング言語標準です。IBM は、新しい標準の機能の開発および実装を継続します。この言語レベルの実装は、IBM による標準の解釈に基づいています。新しい C++ 標準ライブラリーのサポートを含め、C++0x 標準のすべての機能を IBM が実装し終えるまで、リリースごとに実装が変更される可能性があります。IBM では、IBM による C++0x 標準の新機能の実装に関し、ソース、バイナリー、リストおよびその他のコンパイラー・インターフェースにおいて、以前のリリースとの互換性を維持するための試みは、特に行いません。したがって、これらの新機能は継続的なプログラミング・インターフェースとしては利用しないでください。

C++0x は、ISO/IEC 14882:2011 として批准され公開されました。本書での C++0x への参照はすべて、ISO/IEC 14882:2011 標準と同等です。プログラミング・インターフェースなどの対応情報は、将来のリリースで更新される予定です。

事前定義マクロ

いくつかの事前定義マクロが認識されます。このセクションでは、事前定義マクロについての詳細な情報を提供します。

ILE C/C++ コンパイラーは、次の事前定義マクロを認識します。

- 『ANSI/ISO 標準事前定義マクロ』
- 4-2 ページの『ILE C/C++ 事前定義マクロ』

ANSI/ISO 標準事前定義マクロ

ILE C/C++ コンパイラーは、ANSI/ISO 標準で定義される以下のマクロを認識します。他に規定がない場合、定義された時点のマクロの値は 1 です。

__DATE__
ソース・ファイルがコンパイルされた日付が入っている文字ストリング・リテラル。日付は次の形式になります。

```
"Mmm dd yyyy"
```

各値は、次のとおりです。

- Mmm は月を省略形式 (Jan、Feb、Mar、Apr、May、Jun、Jul、Aug、Sep、Oct、Nov、または Dec) で表します。
- dd は日を表します。日が 10 より小さい場合、最初の d はブランク文字になります。
- yyyy は年を表します。

__FILE__
ソース・ファイルの名前が入った文字ストリング・リテラル。

__LINE__
現行のソース行番号を表す整数。

__STDC__
C C コンパイラーが ANSI 規格に準拠している場合に定義されます。このマクロは、言語レベルが LANGLVL(*ANSI) に設定されている場合、定義されています。

__STDC_VERSION__
C long int 型の整数定数として定義されます。このマクロは、__STDC__ も定義され、その値が 199409L の場合にのみ定義されます。このマクロは、C++ 用には定義されません。

__TIME__
ソース・ファイルがコンパイルされた時刻が入っている文字ストリング・リテラル。時刻は次の形式になります。

```
"hh:mm:ss"
```

各値は、次のとおりです。

- hh は時間を表します。
- mm は分を表します。
- ss は秒を表します。

__cplusplus

C++ C++ プログラムのコンパイル時に定義され、コンパイラーが C++ コンパイラーであることを示します。このマクロには、末尾に下線がありません。このマクロは、C 用には定義されません。

注:

1. 事前定義マクロ名は、#define または #undef プリプロセッサー・ディレクティブのサブジェクトにはなりません。
2. 事前定義 ANSI/ISO 規格マクロ名は、名前の直前の 2 つの下線 (__) 文字、大文字の名前、および名前の直後の 2 つの下線文字から構成されます。
3. コンパイラーがソース・プログラムの後続の行を処理すると、コンパイル中に __LINE__ の値は変更されます。
4. コンパイラーがソース・プログラムの一部である #include ファイルを処理すると、__FILE__ および __TIME__ の値は変更されます。
5. C #line プリプロセッサー・ディレクティブを使用して __LINE__ および __FILE__ を変更することもできます。

例

以下の printf() ステートメントは、事前定義マクロ __LINE__、__FILE__ 、__TIME__、および __DATE__ の値を表示し、プログラムが __STDC__ に基づき ANSI 規格に準拠しているかを示すメッセージを印刷します。

```
#include <stdio.h>
#ifdef __STDC__
#   define CONFORM    "conforms"
#else
#   define CONFORM    "does not conform"
#endif
int main(void)
{
  printf("Line %d of file %s has been executed\n", __LINE__, __FILE__);
  printf("This file was compiled at %s on %s\n", __TIME__, __DATE__);
  printf("This program %s to ANSI standards\n", CONFORM);
}
```

関連情報

事前定義マクロの追加情報については、*ILE C/C++ 解説書* を参照してください。

ILE C/C++ 事前定義マクロ

ILE C/C++ コンパイラーは、本セクションに記述された事前定義マクロを提供します。これらのマクロは、対応するプラグマがプログラム・ソースで呼び出されたか、対応するコンパイラー・オプションが指定された場合に定義されます。他に規定がない場合、定義された時点のマクロの値は 1 です。

__ANSI__

LANGLVL(*ANSI) コンパイラー・オプションが有効な場合に定義されます。このマクロが定義されると、コンパイラーは ANSI/ISO C および C++ 標準に準拠する言語構造体のみを許可します。

__ASYNC_SIG__

C SYSIFCOPT(*ASYNCSIGNAL) コンパイラー・オプションが有効な場合に定義されます。

C++ TERASPACE(*YES *TSIFC) STGMDL(*TERASPACE) DTAMDL(*LLP64) RTBND(*LLP64) コンパイラー・オプションが有効な場合に定義されます。

__BASE_FILE__
1 次ソース・ファイルの完全修飾名を示します。

__BOOL__
C++ bool キーワードが使用できることを示します。

_CHAR_SIGNED __CHAR_SIGNED__
#pragma chars(signed) ディレクティブが有効であるとき、または DFTCHAR コンパイラー・オプションが *SIGNED に設定されるときに、定義されます。このマクロが定義される場合、デフォルト文字型は signed です。

_CHAR_UNSIGNED __CHAR_UNSIGNED__
#pragma chars(unsigned) ディレクティブが有効であるとき、または DFTCHAR コンパイラー・オプションが *UNSIGNED に設定されるときに、定義されます。このマクロが定義される場合、デフォルト文字型が unsigned であることを示します。

__cplusplus98__interface__
C++ LANGLVL(*ANSI) コンパイラー・オプションが指定されている場合に定義されます。

__C99_BOOL
C _Bool データ型がサポートされていることを示します。LANGLVL(*EXTENDED) コンパイラー・オプションが有効な場合に定義されます。

__C99_CPLUSCMT
C C++ 形式コメントがサポートされていることを示します。LANGLVL(*EXTENDED) コンパイラー・オプションが有効な場合に定義されます。

__C99_COMPOUND_LITERAL
複合リテラルがサポートされていることを示します。

C LANGLVL(*EXTENDED) コンパイラー・オプションが有効な場合に定義されます。

C++ LANGLVL(*EXTENDED) または LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__C99_DESIGNATED_INITIALIZER
C 指定された初期設定がサポートされていることを示します。LANGLVL(*EXTENDED) コンパイラー・オプションが有効な場合に定義されます。

__C99_DUP_TYPE_QUALIFIER
C 重複した型修飾子がサポートされていることを示します。LANGLVL(*EXTENDED) コンパイラー・オプションが有効な場合に定義されます。

__C99_EMPTY_MACRO_ARGUMENTS
C 空のマクロ引数がサポートされていることを示します。LANGLVL(*EXTENDED) コンパイラー・オプションが有効な場合に定義されます。

__C99_FLEXIBLE_ARRAY_MEMBER
C 柔軟な配列メンバーがサポートされていることを示します。LANGLVL(*EXTENDED) コンパイラー・オプションが有効な場合に定義されます。

__C99_FUNC__
__func__ 事前定義 ID がサポートされていることを示します。

C LANGLVL(*EXTENDED) コンパイラー・オプションが有効な場合に定義されます。

C++ LANGLVL(*EXTENDED) または LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__C99_HEX_FLOAT_CONST
16 進浮動小数点定数がサポートされていることを示します。

C LANGLVL(*EXTENDED) コンパイラー・オプションが有効な場合に定義されます。

C++ LANGLVL(*EXTENDED) または LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__C99_INLINE
C インライン関数指定子がサポートされていることを示します。LANGLVL(*EXTENDED) コンパイラー・オプションが有効な場合に定義されます。

__C99_LLONG
C C99 スタイルの long long データ型およびリテラルがサポートされていることを示します。LANGLVL(*EXTENDED) コンパイラー・オプションが有効な場合に定義されます。

__C99_MACRO_WITH_VA_ARGS
変数引数を持つ関数のようなマクロがサポートされていることを示します。

C LANGLVL(*EXTENDED) コンパイラー・オプションが有効な場合に定義されます。

C++ LANGLVL(*EXTENDED) または LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__C99_MAX_LINE_NUMBER
最大行番号が 2147483647 であることを示します。

C LANGLVL(*EXTENDED) コンパイラー・オプションが有効な場合に定義されます。

C++ LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__C99_MIXED_DECL_AND_CODE
C 混合宣言およびコードがサポートされていることを示します。LANGLVL(*EXTENDED) コンパイラー・オプションが有効な場合に定義されます。

__C99_MIXED_STRING_CONCAT
C++ ワイド・ストリング・リテラルと非ワイド・ストリング・リテラルの連結がサポートされていることを示します。LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__C99_NON_CONST_AGGR_INITIALIZER
C 非定数集約初期化指定子がサポートされていることを示します。LANGLVL (*EXTENDED) コンパイラー・オプションが有効な場合に定義されます。

__C99_NON_LVALUE_ARRAY_SUB
C 配列のための非左辺値の添え字がサポートされていることを示します。LANGLVL(*EXTENDED) コンパイラー・オプションが有効な場合に定義されます。

__C99_PRAGMA_OPERATOR
_Pragma 演算子がサポートされていることを示します。

C LANGLVL(*EXTENDED) コンパイラー・オプションが有効な場合に定義されます。

C++ LANGLVL(*EXTENDED) または LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__C99_RESTRICT

C++ C99 限定修飾子がサポートされていることを示します。 LANGLVL(*EXTENDED) または LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__C99_STATIC_ARRAY_SIZE

C 関数への配列パラメーターにおける静的キーワードがサポートされていることを示します。LANGLVL(*EXTENDED) コンパイラー・オプションが有効な場合に定義されます。

__C99_VAR_LEN_ARRAY

C 可変長配列がサポートされていることを示します。LANGLVL(*EXTENDED) コンパイラー・オプションが有効な場合に定義されます。

__C99_VARIABLE_LENGTH_ARRAY

C++ 可変長配列がサポートされていることを示します。 LANGLVL(*EXTENDED) または LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__DIGRAPHS__

ダイグラフがサポートされていることを示します。

__EXTENDED__

C LANGLVL(*EXTENDED) コンパイラー・オプションが有効な場合に定義されます。

C++ LANGLVL(*EXTENDED) または LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

このマクロが定義されると、コンパイラーは ILE C/C++ コンパイラー実装によって提供される言語拡張機能を許可します。

__FUNCTION__

現在コンパイルされている関数の名前を示します。C++ プログラムの場合、実際の関数プロトタイプに拡張されます。

__HHW_AS400__

ホスト・ハードウェアが IBM i プロセッサーであることを示します。

__HOS_OS400__

C++ ホスト・オペレーティング・システムが IBM i であることを示します。

__IBMC__

C C コンパイラーのバージョンを示します。これは VRM 形式の整数を返します。ここで、VRM は以下のとおりです。

V はバージョンを表します。

R はリリースを表します。

M はモディフィケーション・レベルを表します。

例えば、IBM i 7.2 コンパイラーを TGTRLS(*CURRENT) コンパイラー・オプションと一緒に使用すると、__IBMC__ は整数値 720 を返します。

__IBMCPP__

C++ ILE C++ コンパイラーが基礎としている AIX® XL C++ コンパイラーのバージョンを示します。これは、コンパイラー・バージョンを表す整数を返します。例えば、IBM i 7.2 コンパイラーを TGTRLS(*CURRENT) コンパイラー・オプションと一緒に使用すると、__IBMCPP__ は整数値 1110 を返します。1110 は、ILE C++ コンパイラーが XL C++ V11.1 コンパイラーに基づいていることを意味します。

__IBM__ALIGN

C++ __align 指定子がサポートされていることを示します。

__IBM_ATTRIBUTES

C++ 型、変数、および関数属性がサポートされていることを示します。
LANGLVL(*EXTENDED) または LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__IBM_COMPUTED_GOTO

C++ 計算された goto ステートメントがサポートされていることを示します。
LANGLVL(*EXTENDED) または LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__IBM_DFP__

10 進浮動小数点型がサポートされていることを示します。

C LANGLVL(*EXTENDED) コンパイラー・オプションが有効な場合に定義されます。

C++ LANGLVL(*EXTENDED) または LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__IBM_EXTENSION_KEYWORD

C++ __extension__ キーワードがサポートされていることを示します。 LANGLVL(*EXTENDED) または LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__IBM_INCLUDE_NEXT

#include_next プリプロセス・ディレクティブがサポートされていることを示します。

__IBM_LABEL_VALUE

C++ ラベルが値としてサポートされていることを示します。 LANGLVL(*EXTENDED) または LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__IBM_LOCAL_LABEL

C++ ローカル・ラベルがサポートされていることを示します。 LANGLVL(*EXTENDED) または LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__IBM_MACRO_WITH_VA_ARGS

C++ Variadic マクロ拡張がサポートされていることを示します。 LANGLVL(*EXTENDED) または LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__IBM_TYPEOF__

__typeof__ または typeof キーワードがサポートされていることを示します。このマクロは、常に C 用に定義されています。

C++ については、LANGLVL(*EXTENDED) または LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__IBMCPP_AUTO_TYPEDEDUCTION

C++ 自動的な型推論機能がサポートされていることを示します。LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__IBMCPP_C99_PREPROCESSOR

C++ C++0x 標準で採用されている C99 プリプロセッサー機能がサポートされていることを示します。LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__IBMCPP_DECLTYPE

C++ decltype 機能がサポートされていることを示します。LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__IBMCPP_DELEGATING_CTORS

C++ 委任コンストラクター機能がサポートされていることを示します。LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__IBMCPP_EXTENDED_FRIEND

C++ 拡張フレンド宣言機能がサポートされていることを示します。 LANGLVL (*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__IBMCPP_EXTERN_TEMPLATE

C++ 明示的インスタンス生成宣言機能がサポートされていることを示します。LANGLVL(*EXTENDED) または LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__IBMCPP_INLINE_NAMESPACE

C++ インライン名前空間定義機能がサポートされていることを示します。LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__IBMCPP_STATIC_ASSERT

C++ 静的アサーション機能がサポートされていることを示します。LANGLVL (*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__IFS_IO__

SYSIFCOPT(*IFSIO) または SYSIFCOPT(*IFS64IO) コンパイラー・オプションが指定される場合に定義されます。

__IFS64_IO__

SYSIFCOPT(*IFS64IO) コンパイラー・オプションが指定される場合に定義されます。このマクロが定義されているとき、_LARGE_FILES および _LARGE_FILE_API も関連する IBM 提供のヘッダー・ファイルで定義されます。

__ILEC400__

C ILE C コンパイラーが使用されていることを示します。

__ILEC400_TGTVRM__

C __OS400_TGTVRM__ マクロと同じです。

_LARGE_FILES

SYSIFCOPT(*IFS64IO) コンパイラー・オプションが有効であり、システム・ヘッダー・ファイル types.h が含まれている場合に定義されます。

_LARGE_FILE_API

SYSIFCOPT(*IFS64IO) コンパイラー・オプションが有効であり、システム・ヘッダー・ファイル types.h が含まれている場合に定義されます。

__LLP64_IFC__

DTAMDL(*LLP64) コンパイラー・オプションが有効な場合に定義されます。

__LLP64_RTBND__

C++ RTBND(*LLP64) コンパイラー・オプションが有効な場合に定義されます。

__LONGDOUBLE64

long double 型のサイズが 64 ビットであることを示します。このマクロは、常に定義されています。

_LONG_LONG
IBM long long データ型がサポートされていることを示します。

C LANGLVL(*EXTENDED) コンパイラー・オプションが有効な場合に定義されます。

C++ LANGLVL(*EXTENDED) または LANGLVL(*EXTENDED0X) コンパイラー・オプションが有効な場合に定義されます。

__NO_RTTI__
C++ OPTION(*NORTTI) コンパイラー・オプションが有効な場合に定義されます。

__OPTIMIZE__
C++ 有効な最適化のレベルを示します。このマクロは OPTIMIZE(10) では未定義です。その他の OPTIMIZE 設定については、マクロは次のように定義されています。

OPTIMIZE(20) では 2

OPTIMIZE(30) では 3

OPTIMIZE(40) では 4

__OS400__
このマクロは、コンパイラーが IBM i オペレーティング・システムで使用されるときは必ず定義されます。

__OS400_TGTVRM__
生成されたオブジェクトの実行が意図されているオペレーティング・システムのバージョン/リリース/モディフィケーション・レベルにマップする整数値を定義します。例えば、ターゲット・リリースが TGTRLS(V7R2M0) コンパイラー・オプションを使用して設定されている場合、このマクロは整数値 720 を返します。

__POSIX_LOCALE__
LOCALETYPE(*LOCALE)、LOCALETYPE(*LOCALEUCS2)、または LOCALETYPE (*LOCALEUTF) コンパイラー・オプションが指定される場合に定義されます。

__RTTI_DYNAMIC_CAST__
C++ OPTION(*RTTIALL) または OPTION(*RTTICAST) コンパイラー・オプションが指定されている場合に定義されます。

__RTTI_TYPE_INFO__
C++ OPTION(*RTTIALL) または OPTION(*RTTITYPE) コンパイラー・オプションが指定されている場合に定義されます。

__SIZE_TYPE__
C 現行のプラットフォームでの基礎となる型 size_t を示します。 IBM i の場合、これは unsigned int です。

__SRCSTMF__
C コンパイル中のソース・ファイルのロケーションが SRCSTMF コンパイラー・オプションによって指定される場合に定義されます。

__TERASPACE__
TERASPACE(*YES *TSIFC) コンパイラー・オプションが指定される場合に定義されます。

__THW_AS400__
ターゲット・ハードウェアが IBM i プロセッサーであることを示します。

__TIMESTAMP__
ソース・ファイルが最後に変更された日時が入っている文字ストリング・リテラル。

日時は次の形式になります。

```
"Day Mmm dd hh:mm:ss yyyy"
```

各値は、次のとおりです。

Day は曜日 (Mon、Tue、Wed、Thu、Fri、Sat、Sun のいずれか) を表します。

Mmm は月を省略形式 (Jan、Feb、Mar、Apr、May、Jun、Jul、Aug、Sep、Oct、Nov、または Dec) で表します。

dd は日を表します。日が 10 より小さい場合、最初の d はブランク文字になります。

hh は時間を表します。

mm は分を表します。

ss は秒を表します。

yyyy は年を表します。

注: その他のコンパイラーはこのマクロをサポートしない可能性があります。このマクロがその他のコンパイラーでサポートされている場合、日時の値はここで示す値と異なる場合があります。

__TOS_OS400__
ターゲット・オペレーティング・システムが IBM i であることを示します。

__UCS2__
LOCALETYPE(*LOCALEUCS2) コンパイラー・オプションが指定される場合に定義されます。

__UTF32__
LOCALETYPE(*LOCALEUTF) コンパイラー・オプションが指定される場合に定義されます。

__wchar_t
typedef wchar_t が定義済みであることを示します。

このマクロは、標準ヘッダー・ファイル stddef.h 内で定義されます。

このマクロは、C++ コンパイラー によって定義されます。

ILE C/C++ プラグマ

このセクションでは、プラグマ・ディレクティブの概要を説明します。

プラグマ・ディレクティブ構文

ILE C/C++ コンパイラーでサポートされるプラグマ・ディレクティブには、以下の 2 つの形式があります。

#pragma name

この形式は、以下の構文を使用します。

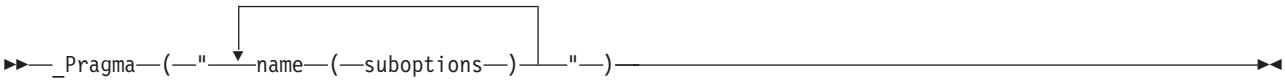
#pragma name 構文

name はプラグマ・ディレクティブ名で、*suboptions* は、適用可能な場合にプラグマに指定できる必須または任意指定のサブオプションです。

_Pragma ("name")

この形式は、以下の構文を使用します。

_Pragma("name") 構文



例えば、次のように記述します。

```
_Pragma ("pack(1)")
```

上記は次のものと同等です。

```
#pragma pack(1)
```

プラグマ・ステートメントのすべての形式において、単一の **#pragma** ステートメントで複数の *name* および *suboptions* を指定することができます。

プラグマの *name* は、特に指定がない限り、マクロ置換されます。コンパイラーは、認識されないプラグマを無視し、そのプラグマを示す通知メッセージ (C++) あるいは警告メッセージ (C) を発行します。

C と C++ の両方のコンパイラーでコンパイルされるコードで、C と C++ の両方に共通ではないプラグマがある場合は、プラグマのまわりに条件付きコンパイル・ディレクティブを追加する必要があります。 (認識されないプラグマは無視されるので、これらのディレクティブは厳密には必要ではありません。) 例え

ば、**#pragma info** は C++ コンパイラーにのみ認識されるため、プラグマのまわりに条件付きコンパイル・ディレクティブを追加することになる場合があります。

```
#ifdef __cplusplus
#pragma info(none)
#endif
```

プラグマ・ディレクティブの範囲

コンパイル単位のソース・コード内であれば、どの時点でも多くのプラグマ・ディレクティブを指定することができます。それ以外の場合、プラグマ・ディレクティブは、その他のディレクティブまたはソース・コード・ステートメントの前に指定してください。それぞれのプラグマの個々の説明には、プラグマの配置の制約がすべて説明されています。

一般的に、プラグマ・ディレクティブは、ソース・プログラムのどのコードの前に指定しても、組み込まれているヘッダー・ファイルなど、コンパイル単位全体に適用されます。ソース・コードの任意の場所に指定できるディレクティブの場合、指定される時点からコンパイル単位の終わりまで適用されます。

コードの選択したセクションのまわりでプラグマ・ディレクティブの相互補完的ペアを使用すると、プラグマの適用範囲をさらに制限することができます。例えば、ソース・コードの選択した部分のみが特定のデータ・モデル設定を使用するように要求するには、**#pragma datamodel** ディレクティブを以下のように使用します。

```
/* Data model may be P128 or LLP64 */
#pragma datamodel(P128)
/* Data model P128 is now in effect */
#pragma datamodel(pop)
/* Prior data model is now in effect */
```

多くのプラグマ・ディレクティブが、スタック・ベースの方法でプラグマ設定を使用可能にしたり使用不可にすることができる「pop」または「reset」サブオプションを提供します。これらのサブオプションの例は、関連するプラグマの説明に記載されています。

プラグマ・ディレクティブの要約

ILE C/C++ コンパイラーは、次のプラグマを認識します。

表 5-1. ILE C/C++ コンパイラーで認識されるプラグマ

プラグマ名	C で有効	C++ で有効
5-3 ページの『argopt』	あり	あり
5-5 ページの『argument』	あり	なし
5-7 ページの『cancel_handler』	あり	あり
5-8 ページの『chars』	あり	あり
5-8 ページの『checkout』	あり	なし
5-9 ページの『コメント』	あり	あり
5-10 ページの『convert』	あり	なし
5-11 ページの『datamodel』	あり	あり
5-12 ページの『define』	なし	あり
5-12 ページの『descriptor』	あり	あり

表 5-1. ILE C/C++ コンパイラーで認識されるプラグマ (続き)

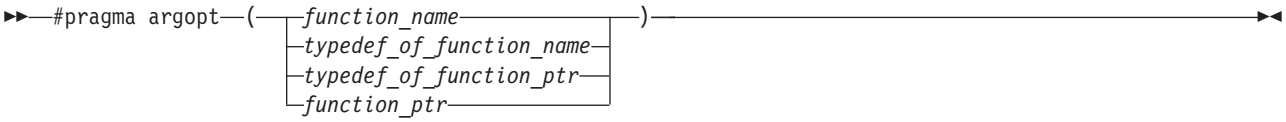
プラグマ名	C で有効	C++ で有効
5-14 ページの『disable_handler』	あり	あり
5-15 ページの『disjoint』	なし	あり
5-15 ページの『do_not_instantiate』	なし	あり
5-16 ページの『enum』	あり	あり
5-20 ページの『exception_handler』	あり	あり
5-23 ページの『hashome』	なし	あり
5-24 ページの『implementation』	なし	あり
5-24 ページの『info』	なし	あり
5-25 ページの『inline』	あり	なし
5-26 ページの『ishome』	なし	あり
5-26 ページの『isolated_call』	なし	あり
5-28 ページの『linkage』	あり	なし
5-29 ページの『map』	あり	あり
5-30 ページの『mapinc』	あり	なし
5-32 ページの『margins』	あり	なし
5-33 ページの『namemangling』	なし	あり
5-34 ページの『namemanglingrule』	なし	あり
5-35 ページの『noargv0』	あり	なし
5-36 ページの『noinline (function)』	あり	なし
5-36 ページの『nomargins』	あり	なし
5-37 ページの『nosequence』	あり	なし
5-37 ページの『nosigtrunc』	あり	なし
5-37 ページの『pack』	あり	あり
5-43 ページの『page』	あり	なし
5-43 ページの『pagesize』	あり	なし
5-44 ページの『pointer』	あり	あり
5-45 ページの『priority』	なし	あり
5-46 ページの『sequence』	あり	なし
5-46 ページの『strings』	あり	あり
5-47 ページの『weak』	なし	あり

個々のプラグマの説明

argopt

C C++

argopt 構文



説明

引数最適化 (argopt) は、実行時パフォーマンスを向上させるプラグマです。結合プロシージャーに適用され、以下によって最適化を達成することができます。

- 汎用レジスター (GPR) へのスペース・ポインター・パラメーターの引き渡し
- 関数から GPR へ戻されるスペース・ポインターの格納

パラメーター

function_name
: 最適化されたプロシージャー・パラメーター引き渡しが指定される関数の名前を指定します。関数は、静的関数、外部的に定義された関数、または現在のコンパイル単位の外側から呼び出される現在のコンパイル単位で定義されている関数のいずれであってもかまいません。

typedef_of_function_name
: 最適化されたプロシージャー・パラメーター引き渡しが指定される関数の型定義の名前を指定します。

typedef_of_function_ptr
: 最適化されたプロシージャー・パラメーター引き渡しが指定される関数ポインターの型定義の名前を指定します。

function_ptr
: 最適化されたプロシージャー・パラメーター引き渡しが指定される関数ポインターの名前を指定します。

使用に関する注意

#pragma argopt ディレクティブを指定した場合でも、プログラムが必ず最適化されるとは限りません。argopt の有効性は変換機構に依存します。

同じ宣言のために、#pragma descriptor と共に #pragma argopt を指定しないでください。コンパイラーは、このプラグマのどちらか一方のみを一度に使用することをサポートします。

関数は、#pragma argopt ディレクティブで指定される前に、宣言 (プロトタイプ化) されるか定義される必要があります。

ボイド・ポインターはスペース・ポインターではないので最適化されません。

#pragma argopt の使用は構造体宣言ではサポートされません。

#pragma argopt は、OS リンケージまたは組み込まれたリンケージを含む関数 (#pragma linkage (function_name, OS) ディレクティブまたは関数に関連付けされた #pragma linkage (function_name, builtin) ディレクティブ、およびその逆を含む関数) に対して指定することはできません。

#pragma argopt は、#pragma exception_handler または #pragma cancel_handler ディレクティブでのハンドラー関数、および **signal()** や **atexit()** などのエラー処理関数として指定される関数に対して無視されます。#pragma argopt ディレクティブは、変数引数リストを含む関数に適用することはできません。

#pragma argopt 有効範囲

#pragma argopt は、関数、関数ポインター、関数ポインターの型定義、またはそれが作動する関数の型定義と同じ有効範囲に配置する必要があります。#pragma argopt が同じ有効範囲にない場合、エラーが出力されます。

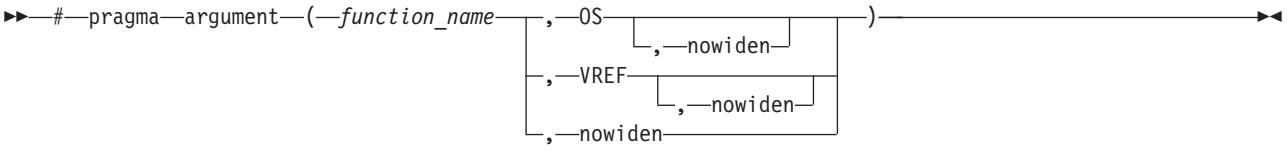
```
#include <stdio.h>

long func3(long y)
{
printf("In func3()\n");
printf("hex=%x,integer=%d\n",y, y);
}
#pragma argopt (func3)           /* file scope of function  */
int main(void)
 {
 int i, a=0;
 typedef long (*func_ptr) (long);
 #pragma argopt (func_ptr)      /* block scope of typedef   */
                                /* of function pointer      */
 struct funcstr
    {
     long (*func_ptr2) (long);
     #pragma argopt (func_ptr2) /* struct scope of function */
                                /* pointer                  */
     };
struct funcstr func_ptr3;
for (i=0; i<99; i++)
 {
  a = i*i;
  if (i == 7)
    {
     func_ptr3.func_ptr2( i );
    }
 }
 return i;
}
```

argument

C

argument 構文



説明

function_name で指定されるプロシージャーまたは型定義に使用する、引数引き渡しおよび受信メカニズムを指定します。

このプラグマはプロシージャーを外部結合プロシージャーとしてのみ識別します。プロシージャーはプラグマ引数ディレクティブと同じソースで定義され、そこから呼び出されます。プラグマ引数ディレクティブが、そのディレクティブで指定されるプロシージャーの定義と同じコンパイル単位で指定される場合、プロシージャーへの引数はプラグマ・ディレクティブで指定される方式を使用して受け取られます。

外部プログラムの呼び出しの詳細については、 5-28 ページの『linkage』プラグマを参照してください。

パラメーター

function_name
外部結合プロシージャーの名前を指定します。

OS OS は、OS リンケージ引数メソッドを使用して、引数が渡されたり、受け取られる (プラグマ・ディレクティブがプロシージャー定義と同じコンパイル単位に存在する場合) ことを示します。非アドレス引数は一時ロケーションにコピーされ、(nowiden が指定されていない場合) 拡大され、コピーのアドレスは呼び出されたプロシージャーに渡されます。アドレスまたはポインターである引数は呼び出されたプロシージャーに直接渡されます。

VREF VREF は、アドレス引数も OS リンケージ・メソッドを使用して引き渡されたり受け取られるという例外がありますが、OS リンケージと同様です。

nowiden
引数が引き渡されたり受け取られる前に拡大されないことを指定します。このパラメーターは引数タイプを指定しないで単独で使用することができます。例えば、`#pragma argument (myfunc, nowiden)` は、プロシージャー myfunc が、典型的な*値による* 方法を使用して、拡大されずに引数を引き渡したり受け取ることを示します。

使用に関する注意

このプラグマは、パラメーターが結合プロシージャーに渡される方法、およびパラメーターが受け取られる方法を制御します。#pragma 引数ディレクティブで指定される関数名は、現在のコンパイル単位で定義することができます。 #pragma 引数ディレクティブは、指定する関数の前に置く必要があります。

影響されるプロシージャーと同じコンパイル単位で #pragma 引数ディレクティブを指定すると、プラグマ引数ディレクティブで指定したとおりに、プロシージャーで引数が受け取られる (送信される) ことがコンパイラーに通知されます。これはプラグマ引数で指定される ILE C で書かれた結合プロシージャーに役立ちます。プロシージャーおよび定義への呼び出しが別のコンパイル単位にある場合、プラグマ引数ディレクティブがその引き渡し方法 (OS、VREF、または nowiden) に関して一致することをユーザーが保証する必要があります。

例えば、下記の 2 つのソース・ファイルにおいて、引数の一時コピーのアドレスは Program 1 の `foo` に引き渡されます。Program 2 の `foo` は一時コピーのアドレスを受け取り、それを逆参照し、その値をパラメーター `a` に割り当てます。2 つのプラグマ・ディレクティブが異なる場合、動作は未定義です。

Program 1	**Program 2**

`#pragma argument(foo, OS, nowiden)`
`void foo(char);`
`void main() {`
`  foo(10);`
`}`
	`#pragma argument(foo, OS, nowiden)`
	`void foo(char a) { a++; }`

以下のいずれかが発生する場合、警告が出され、#pragma 引数ディレクティブは無視されます。

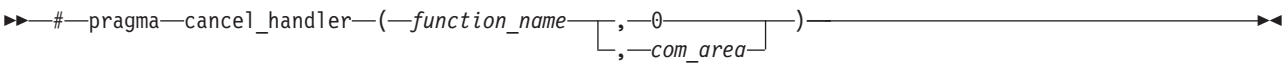
- #pragma 引数ディレクティブがコンパイル単位内の指定された関数の宣言や定義の前にない。

- ディレクティブの *function_name* がプロシージャーの名前またはプロシージャーの型定義ではない。
- ディレクティブで指定された型定義がディレクティブで使用される前にプロシージャーの宣言や定義で使用されている。
- #pragma 引数ディレクティブがこの関数に対して既に指定されている。
- #pragma リンケージ・ディレクティブがこの関数に対して既に指定されている。
- 関数が #pragma 引数ディレクティブの前に既に呼び出されている。

cancel_handler

C C++

cancel_handler 構文



説明

コード内の、#pragma cancel_handler ディレクティブが置かれている点で、指定の関数をユーザー定義の ILE 取り消しハンドラーとして使用可能にすることを指定します。

#pragma cancel_handler ディレクティブによって使用可能に設定された取り消しハンドラーはすべて、ディレクティブを含む関数の呼び出しが終了したときに黙示的に使用不可になります。ハンドラーが #pragma disable_handler ディレクティブによって明示的に使用不可に設定されなかった場合、呼び出しはコール・スタックから削除されます。

パラメーター

function_name
: ユーザー定義の ILE 取り消しハンドラーとして使用される関数の名前を指定します。

com_area
: 例外ハンドラーに情報を渡すために使用されます。*com_area* が必要でない場合、ディレクティブの 2 番目のパラメーターとしてゼロを指定してください。 *com_area* がディレクティブで指定されている場合は、整数、浮動小数点、倍精度、構造体、共用体、配列、列挙型、ポインター、またはパック 10 進のデータ型のいずれかの変数である必要があります。com_area は VOLATILE 修飾子を使用して宣言される必要があります。構造体や共用体のメンバーにすることはできません。

<except.h> および取り消しハンドラーに渡されるポインターの型定義 _CNL_Hndlr_Parms_T の詳細については、「*C/C++ ランタイム・ライブラリー関数*」を参照してください。

使用に関する注意

ハンドラー関数では、パラメーターとして 16 バイトのポインターのみを使用することができます。

この #pragma ディレクティブは、C 言語ステートメント境界および関数定義内部でのみ使用できます。

以下のいずれかが発生した場合、コンパイラーはエラー・メッセージを出力します。
- ディレクティブが C 関数本体の外部または C ステートメントの内部に出現する。
- ハンドラー関数が宣言または定義されていない。
- ハンドラー関数として指定された ID が関数ではない。

- *com_area* 変数が宣言されていない。
- *com_area* 変数が有効なオブジェクト・タイプを持っていない。
- 指定されたハンドラー関数が引数最適化 (#pragma argopt) で指定されている。

#pragma cancel_handler ディレクティブの使用例および使用の詳細については、*ILE C/C++ プログラマーの手引き* を参照してください。

chars

C C++

chars 構文

説明

コンパイラーがすべての char オブジェクトを signed または unsigned として扱うことを指定します。このプラグマは、ソース・ファイル内の C コードまたはディレクティブ (#line ディレクティブの場合を除く) の前になければなりません。

パラメーター

unsigned
すべての char オブジェクトは unsigned 整数として扱われます。

signed すべての char オブジェクトは signed 整数として扱われます。

checkout

C

checkout 構文

説明

*NONE 以外の CHECKOUT コンパイラー・オプションの値が指定されている場合に、コンパイラーがコンパイラー情報を与える必要があるかどうかを指定します。

パラメーター

suspend
コンパイラーが通知メッセージを中断することを指定します。

resume コンパイラーが通知メッセージを再開することを指定します。

使用に関する注意

#pragma チェックアウト・ディレクティブはネストできます。これは、以前に指定した #pragma checkout (suspend) ディレクティブが有効な場合に、#pragma checkout (suspend) ディレクティブが有効でなくなることを意味します。これは #pragma チェックアウト再開ディレクティブにも該当します。

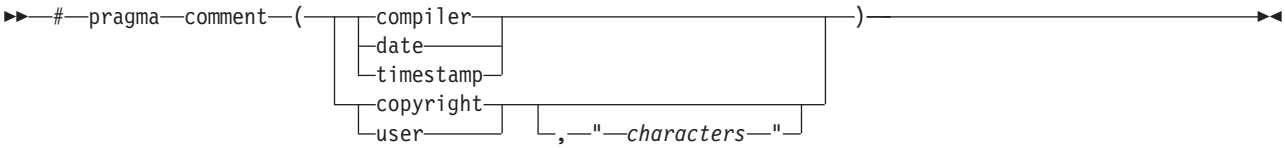
例

```
/*  Assume CHECKOUT(*PPTRACE) had been specified                        */
#pragma checkout(suspend)  /* No CHECKOUT diagnostics are performed    */
   ...
#pragma checkout(suspend)  /* No effect                                */
   ...
#pragma checkout(resume)   /* No effect                                */
   ...
#pragma checkout(resume)   /* CHECKOUT(*PPTRACE) diagnostics continue  */
```

コメント

C C++

comment 構文



説明

プログラムやサービス・プログラム・オブジェクトにコメントを出力します。これは DETAIL (*COPYRIGHT) を含む DSPPGM または DSPSRVPGM によって表示できます。このプラグマは、ソース・ファイル内の C コードまたはディレクティブ (#line ディレクティブの場合を除く) の前になければなりません。

パラメーター

comment プラグマの有効な設定として、以下を使用できます。

compiler
コンパイラーの名前とバージョンが、生成されたプログラム・オブジェクトの最後に出力されます。

date コンパイルの日時が、生成されたプログラム・オブジェクトの最後に出力されます。

timestamp
ソースの最終モディフィケーション日時が、生成されたプログラム・オブジェクトの最後に出力されます。

copyright
characters で指定されたテキストが、生成されたプログラム・オブジェクトにコンパイラーによって置かれ、プログラム実行時にメモリーにロードされます。

user *characters* で指定されたテキストが、生成されたオブジェクトにコンパイラーによって置かれます。ただし、プログラム実行時にメモリーにロードされません。

使用に関する注意

著作権およびユーザー・コメントのタイプは ILE C/C++ コンパイラーに対して実質的に同じです。片方が他方に対して有利なわけではありません。

#pragma comment(copyright) または #pragma comment(user) ディレクティブのテキスト部分における最大文字数は 256 です。

単一のコンパイル単位で使用できる #pragma コメント・ディレクティブの最大数は 1024 です。

convert

C C++

convert 構文

説明

コンパイル時に、ソース・ファイルのある時点以降、ストリング・リテラルを変換するために使用するコード化文字セット ID (CCSID) を指定します。変換は、ソース・ファイルの最後まで、または別の #pragma 変換ディレクティブが指定されるまで継続します。#pragma convert (0) を使用して、以前の #pragma 変換ディレクティブを使用不可にします。変換前のストリング・リテラルの CCSID は、ルート・ソース・メンバーと同じ CCSID です。CCSID 905 および 1026 はサポートされていません。 CCSID は EBCDIC または ASCII のいずれかを使用します。

パラメーター

ccsid ソース・ファイルのストリングとリテラルを変換するために使用するコード化文字セット ID を指定します。値は 0 から 65535 の範囲です。コード・ページの詳細については、*ILE C/C++ ランタイム・ライブラリー関数* を参照してください。

使用に関する注意

デフォルトでは、書式ストリング (`printf()` および `scanf()` など) を構文解析するランタイム・ライブラリー関数は、書式制御ストリングが CCSID 37 にコード化されていることを前提としています。LOCALETYPE(*LOCALEUTF) コンパイラー・オプションが指定されている場合、ランタイム・ライブラリー関数は、プログラムに設定された最後のロケールの CCSID (または、プログラムがロケールを設定していない場合は UTF-8) に書式制御ストリングがコード化されていることを前提とします。

16 進数で指定されるストリングおよび文字定数、例えば、(0xC1) は変換されません。

置換文字は、ソース CCSID と同じシンボル・セットを含まないターゲット CCSID に変換する場合には使用されません。コンパイルは失敗します。

C コンパイラーに対して、値 65535 を含む CCSID が指定される場合、値 0 が指定された場合と同様に動作します。 C++ コンパイラーの場合、ルート・ソース・メンバーの CCSID が仮定されます。ソース・ファイル CCSID 値が 65535 である場合、ソース・ファイルに対してジョブ CCSID が仮定されます。ファイル CCSID が 65535 であり、ジョブ CCSID が 65535 でない場合、ファイル CCSID に対してジョブ CCSID が仮定されます。ファイル CCSID が 65535 で、ジョブ CCSID も 65535 であるが、システム CCSID 値は 65535 でない場合、ファイル CCSID に対してそのシステム CCSID 値が仮定されます。ファイル、ジョブおよびシステム CCSID 値が 65535 である場合、CCSID 037 が仮定されます。

| インクルード処理の場合、ヘッダーの最初の CCSID は、#include の時点で有効な CCSID です。ヘッダー・ファイルの最後で、CCSID は #include の時点で有効であった CCSID に戻されます。推奨されるプログラミング手法は、ヘッダー・ファイル内において、すべての convert(ccsid) プラグマに対し、それに対応する convert(0) プラグマをヘッダー・ファイルの最後までに設定することです。

| C コンパイラーに対して LOCALETYPE(*LOCALE) コンパイラー・オプションが指定される場合、ワイド文字リテラルは変換されません。 C++ コンパイラーの場合、ワイド文字リテラルは、convert プラグマによって要求されたコード・ページに変換されます。 LOCALETYPE(*LOCALEUTF) または LOCALETYPE(*LOCALEUCS2) コンパイラー・オプションが指定される場合、ワイド文字リテラルは変換されません。詳細については、*ILE C/C++ プログラマーの手引き* の『*Using Unicode Support for Wide-Character Literals*』を参照してください。

datamodel

C C++

datamodel 構文

説明

コードのセクションに適用するデータ・モデルを指定します。データ・モデル設定により、明示的な修飾子がない場合にポインター型の解釈が決定されます。

このプラグマは、DTAMDL コンパイラー・コマンド行オプションで指定されたデータ・モデルをオーバーライドします。

パラメーター

P128、p128
__ptr64 キーワードなしで宣言されたポインターのサイズは 16 バイトになります。

LLP64、llp64
__ptr128 キーワードなしで宣言されたポインターのサイズは 8 バイトになります。

pop 以前のデータ・モデル設定をリストアします。以前のデータ・モデル設定が存在しない場合、DTAMDL コンパイラー・コマンド行オプションで指定された設定が使用されます。

使用に関する注意

C++ このプラグマおよび設定は、C++ プログラムで使用される場合、大/小文字を区別します。

#pragma datamodel(LLP64) または #pragma datamodel(llp64) の指定は、TERASPACE(*YES) コンパイラー・オプションも指定されている場合にのみ有効です。

このプラグマで指定されるデータ・モデルは、別のデータ・モデルが指定されるか、#pragma datamodel(pop) が指定されるまで、有効なままです。

例

このプラグマは、ヘッダー・ファイルを折り返す場合に推奨されます。ポインター宣言にポインター修飾子を追加する必要がありません。以下に例を示します。

```
// header file blah.h
#pragma datamodel(P128)     // Pointers are now 16-byte
char* Blah(int, char *);
#pragma datamodel(pop)      // Restore previous setting of datamodel
```

__ptr64 および **__ptr128** ポインター修飾子を使用してデータ・モデルを指定することもできます。この修飾子は、DTAMDL コンパイラー・オプション、および特定のポインター宣言用の #pragma datamodel 設定をオーバーライドします。

__ptr64 修飾子は、TERASPACE(*YES) コンパイラー・オプションも指定されている場合にのみ使用する必要があります。**__ptr128** 修飾子は任意に指定することができます。

以下の例は、プロセス・ローカル・ポインターおよびタグ付けされたスペース・ポインターの宣言を示しています。

```
char * __ptr64   p;  // an 8-byte, process local pointer
char * __ptr128  t;  // a 16-byte, tagged space pointer
```

詳細については、*ILE C/C++ プログラマーの手引き* の『*Using Teraspace*』および *ILE 概念* の『テラスペースおよび単一レベル・ストレージ』を参照してください。

define

C++

define 構文

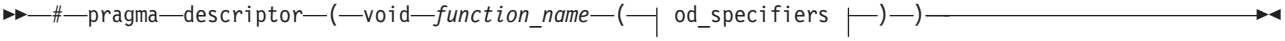
説明

#pragma 定義ディレクティブは、クラスのオブジェクトを実際に定義することなく、強制的にテンプレート・クラスの定義を行います。このプラグマは、宣言が使用できるところならばどこにでも入れることができます。テンプレート関数を効果的または自動的に生成するようにプログラムを編成する場合に使用されます。

descriptor

C C++

descriptor 構文



od_specifiers:

説明

操作記述子は、関数引数に関連したオプションの情報の一部です。この情報は、データ型および長さなどの引数の属性を記述するために使用されます。#pragma 記述子ディレクティブは引数に操作記述子がある関数を識別するために使用されます。

操作記述子は、引数のデータ型の定義が異なる可能性のある他の言語で作成された関数に、引数を渡すときに役立ちます。例えば、C では、ストリングは最初のヌル文字で終了され、ヌル文字を含む文字の連続した配列として定義されます。別の言語では、ストリングは長さ指定子および文字列から構成されるものとして定義される場合があります。C 関数から別の言語で作成された関数にストリングを引き渡す場合、操作記述子が引数と共に渡され、呼び出された関数では、渡されるストリングの長さと型を決定することができます。

ILE C/C++ コンパイラーは #pragma 記述子ディレクティブで指定される関数に引き渡す引数用の操作記述子を生成します。操作記述子が必要だと識別される各引数の記述子タイプ、データ型、および長さが、生成された記述子には含まれます。操作記述子の情報は、ILE API CEEGSI および CEEDOD を使用して、呼び出された関数によってリトリーブすることができます。CL コマンドの詳細については、以下の IBM i Information Center Web サイトにある「プログラミング」カテゴリーの『*CL および API*』セクションを参照してください。

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

関数への引き渡し時に、操作記述子によって正しいストリングの長さを決定するには、ストリングを初期化する必要があります。

ILE C コンパイラーはストリングを記述する操作記述子をサポートします。

注: ILE C/C++ 内の文字ストリングは、以下の方法のいずれかを使用して定義されます。

- char string_name[n]
- char * string_name
- ストリング・リテラル

パラメーター

function_name
: 引数が操作記述子を必要とする関数の名前。

od_specifiers
: ""、ボイド、または * で構成され、コンマによって区切られ、どの関数の引数が操作記述子を持つかを指定するシンボルのリスト。*od_specifier* リストは、関数の *od_specifier* リストが、関数の引数リスト以上の指定子を持つことができない点を除いて、関数の引数リストと同じです。

  - ストリング操作記述子が引数に必要な場合、*od_specifier* パラメーターに対して同等の位置に "" または * を指定する必要があります。
  - 操作記述子が引数に必要ではない場合、*od_specifier* リストの同等の位置のパラメーターに対して *void* を指定します。

使用に関する注意

同じ宣言のために、#pragma argopt と共に #pragma descriptor を指定しないでください。コンパイラーは、このプラグマのどちらか一方のみを一度に使用することをサポートします。

コンパイラーは、以下の条件のいずれかが発生したとき、警告を出し、#pragma 記述子ディレクティブを無視します。

- プラグマ・ディレクティブで指定された ID が関数ではない。
- 関数が別のプラグマ記述子で既に指定されている。
- 関数が静的として宣言されている。
- 関数が #pragma リンケージ・ディレクティブで既に指定されている。
- 指定された関数が `main()` などのユーザー・エントリー・プロシージャーである。
- 関数が #pragma 記述子ディレクティブの前にプロトタイプ化されていない。
- 関数への呼び出しが #pragma 記述子ディレクティブの前に発生する。

操作記述子を使用する場合、以下の制約事項を考慮してください。

- 操作記述子は、機能名で呼び出される関数用にのみ生成されます。関数ポインターで呼び出される関数では、操作記述子は生成されません。
- 操作記述子は C++ 関数宣言には許可されていません。
- 関数引数よりも少ない *od_specifiers* がある場合、残りの *od_specifiers* はデフォルトのボイドになります。
- 関数が可変数の引数を必要とする場合、#pragma 記述子ディレクティブでは、操作記述子が可変引数用ではなく必要な引数用に生成されることを指定できます。
- リテラルまたは配列が明示的に char * にキャストされる場合以外は、それらが操作記述子を必要とする引数としても使用されているときに、リテラルまたは配列でポインター算術計算を行うことは有効ではありません。例えば、F がストリングを引数として解釈する関数であり、F がこの引数用の操作記述子を必要とする場合、F への次のような呼び出しでの引数は有効ではありません。F(a + 1)。ここで、「a」は char a[10] として定義されます。

disable_handler

disable_handler 構文

説明

exception_handler または cancel_handler プラグマのいずれかによって最後に使用可能に設定されたハンドラーを使用不可にします。

このディレクティブは、関数の終了前にハンドラーを明示的に使用不可にする必要がある場合にのみ必要になります。このディレクティブを実行するのは、ハンドラーが使用可能に設定された関数の終了時に、使用可能なすべてのハンドラーが黙示的に使用不可にされるからです。

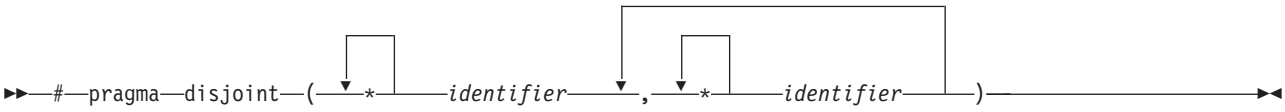
使用に関する注意

このプラグマは、C 言語ステートメント境界および関数定義内部でのみ使用できます。使用可能に設定されたハンドラーがない場合に #pragma disable_handler が指定されると、コンパイラーはエラー・メッセージを出力します。

disjoint

C++

disjoint 構文



説明

このディレクティブは、リストされている ID はどれも、同じ物理ストレージを共用していない (これによって、さらに最適化の機会が提供されます) ことをコンパイラーに通知します。いずれかの ID が実際に物理ストレージを共用している場合、このプラグマはプログラムに誤った結果を引き起こす可能性があります。

ディレクティブの中の ID は、このプラグマが現れるプログラム内のポイントで可視でなければなりません。disjoint 名前リスト内の ID が以下の項目を参照することはできません。

- 構造体または共用体のメンバー
- 構造体、共用体、または列挙型タグ
- 列挙型定数
- 型定義名
- ラベル

例

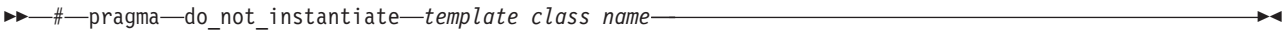
```
int a, b, *ptr_a, *ptr_b;
#pragma disjoint(*ptr_a, b)  // *ptr_a never points to b
#pragma disjoint(*ptr_b, a)  // *ptr_b never points to a
one_function()
{
    b = 6;
    *ptr_a = 7;     // Assignment does not alter the value of b
    another_function(b);    // Argument "b" has the value 6
}
```

外部ポインター **ptr_a** は、外部変数 **b** とストレージを共有することはなく、この外部変数を指すこともないので、**ptr_a** が指すオブジェクトに 7 を代入しても、**b** の値は変わりません。同様に、外部ポインター **ptr_b** は、外部変数 **a** とストレージを共用することはなく、この外部変数を指すこともありません。コンパイラーは、another_function の引数が値 6 を持つことを仮定し、メモリーから変数を再ロードしません。

do_not_instantiate

C++

do_not_instantiate 構文



説明

指定したテンプレート宣言がインスタンス化されないようにします。このプラグマを使用して、定義が提供されているテンプレートの暗黙のインスタンス生成を抑制することができます。

パラメーター

template_class_name
インスタンスを生成されるべきではないテンプレート・クラスの名前。

使用に関する注意

テンプレートのインスタンス生成を手動で処理しており (つまり、コンパイラー・オプション **TEMPLATE(*NONE)** および **TMPLREG(*NONE)** が有効になっている)、指定されたテンプレートのインスタンス生成が別のコンパイル単位に存在している場合は、**#pragma do_not_instantiate** を使用することで、リンク・ステップ中に複数のシンボル定義を取得しないことが保証されます。

クラス・テンプレートの特殊化における **#pragma do_not_instantiate** は、テンプレートの明示的インスタンス生成宣言として扱われます。このプラグマは、C++0x 標準に導入された、明示的インスタンス生成宣言の機能のサブセットを提供します。これは互換性の目的でのみ提供されているもので、推奨されていません。新しいアプリケーションでは、明示的インスタンス生成宣言を代わりに使用してください。 6-25 ページの『LANGLVL』および *ILE C/C++ 解説書* の『明示的インスタンス生成 (C++ のみ)』を参照してください。

例

次の例は、プラグマの使用法を示しています。

```
#pragma do_not_instantiate Stack <int>
```

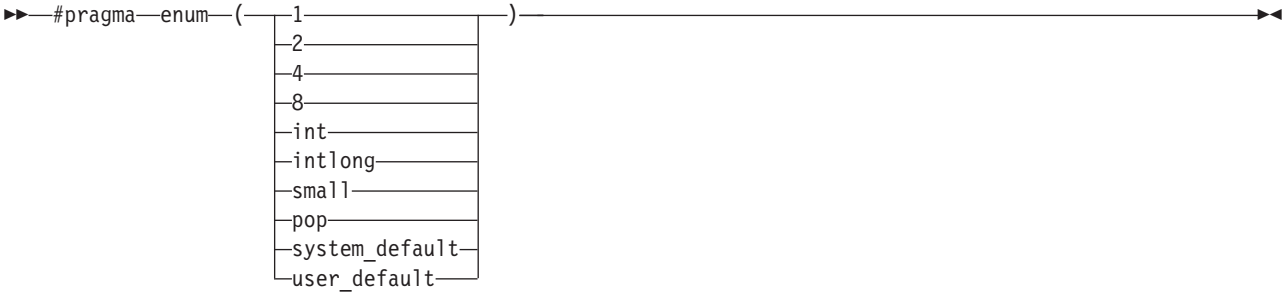
enum

C

enum 構文

C++

enum 構文



説明

列挙型を表す場合にコンパイラーで使用されるバイト数を指定します。後続のすべての enum 定義は、コンパイル単位が終了するか、または別の #pragma enum ディレクティブが検出されるまで、このプラグマに影響を受けます。複数のプラグマが使用されている場合は、直前に検出されたプラグマが有効です。6-40 ページの『ENUM』 ページで説明されているように、このプラグマによって、ENUM コンパイラー・オプションがオーバーライドされます。

パラメーター

1、2、4、8
列挙型が 1、2、4、または 8 バイトのコンテナーで保管されるように指定します。コンテナーの符号は列挙型の値の範囲によって決定されますが、範囲の指定がない場合、符号付きが優先となります。プラグマ **enum(8)** ディレクティブは、C++ でのみ使用可能です。

int 列挙型は ANSI C、または C++ 規格の列挙型表現に保管されます (4 バイト符号付き)。C++ プログラムでは、列挙型の値が ANSI C++ 規格ごとに 2^{31}-1 を超える場合、int コンテナーは 4 バイト符号なしとなります。

intlong 列挙型の値の範囲が int の制限を超えた場合、列挙型が 8 バイトのストレージを専有することを指定します。列挙型の値の範囲が int の制限を超えない場合は、列挙型は 4 バイトのストレージを専有し、enum(int) が指定されていたように表されます。プラグマ **enum(intlong)** ディレクティブは、C++ でのみ使用可能です。

small 後続の列挙型はできる限り最小のコンテナーに置かれ、その列挙型の値が指定されます。コンテナーの符号は列挙型の値の範囲によって決定されますが、範囲の指定がない場合、符号なしが優先となります。

pop 以前に有効だった列挙型のサイズが選択され、現行の設定は破棄されます。

system_default
デフォルトの列挙型サイズが選択されます。デフォルトは small オプションです。

user_default
ENUM コンパイラー・オプションによって指定された列挙型サイズが選択されます。

enum 設定で使用することができる値の範囲は、以下のとおりです。

表 5-2. enum 設定で使用可能な値の範囲

エレメント値の範囲	Enum オプション						
	small (デフォルト)	1	2	4	8 (C++ のみ)	int	intlong (C++ のみ)
0 .. 127	1 バイト符号なし	1 バイト符号付き	2 バイト符号付き	4 バイト符号付き	8 バイト符号付き	4 バイト符号付き	4 バイト符号付き
0 .. 255	1 バイト符号なし	1 バイト符号なし	2 バイト符号付き	4 バイト符号付き	8 バイト符号付き	4 バイト符号付き	4 バイト符号付き
-128 .. 127	1 バイト符号付き	1 バイト符号付き	2 バイト符号付き	4 バイト符号付き	8 バイト符号付き	4 バイト符号付き	4 バイト符号付き
0 .. 32767	2 バイト符号なし	ERROR	2 バイト符号付き	4 バイト符号付き	8 バイト符号付き	4 バイト符号付き	4 バイト符号付き
0 .. 65535	2 バイト符号なし	ERROR	2 バイト符号なし	4 バイト符号付き	8 バイト符号付き	4 バイト符号付き	4 バイト符号付き
-32768 .. 32767	2 バイト符号付き	ERROR	2 バイト符号付き	4 バイト符号付き	8 バイト符号付き	4 バイト符号付き	4 バイト符号付き
0 .. 2147483647	4 バイト符号なし	ERROR	ERROR	4 バイト符号付き	8 バイト符号付き	4 バイト符号付き	4 バイト符号付き
0 .. 4294967295	4 バイト符号なし	ERROR	ERROR	4 バイト符号なし	8 バイト符号付き	C++ 4 バイト符号なし C ERROR	4 バイト符号なし
-2147483648 .. 2147483647	4 バイト符号付き	ERROR	ERROR	4 バイト符号付き	8 バイト符号付き	4 バイト符号付き	4 バイト符号付き
0 .. (2^{63} -1) (C++ のみ)	8 バイト符号なし	ERROR	ERROR	ERROR	8 バイト符号付き	ERROR	8 バイト符号付き
0.. 2^{64} (C++ のみ)	8 バイト符号なし	ERROR	ERROR	ERROR	8 バイト符号なし	ERROR	8 バイト符号なし
-2^{63} .. (2^{63}-1) (C++ のみ)	8 バイト符号付き	ERROR	ERROR	ERROR	8 バイト符号付き	ERROR	8 バイト符号付き

例

以下の例では、#pragma enum およびコンパイラー・オプションのさまざまな使用が示されています。

1. enum の宣言内で #pragma enum を使用し、enum のストレージ割り振りを変更することはできません。以下のコード・セグメントによって警告が生成され、enum オプションの 2 番目の出現は無視されます。

```
#pragma enum ( small )
   enum e_tag { a, b,
#pragma enum ( int ) /* error: cannot be within a declaration */
   c
  } e_var;

#pragma enum ( pop ) /* second pop isn't required */
```

2. C コンパイラーの場合、enum 定数の範囲は、unsigned int または int (signed int) のいずれかの範囲内である必要があります。 C++ コンパイラーの場合、enum 定数の範囲は、unsigned long long または long long (signed long long) のいずれかの範囲内である必要があります。例えば、C コンパイラーが使用されている場合、以下のコード・セグメントにはエラーが含まれていますが、C++ コンパイラーが使用されている場合は、コンパイルは成功します。

```
#pragma enum ( small )
   enum e_tag { a=-1,
                b=2147483648   /* C compiler error: larger than maximum int */
              } e_var;
#pragma enum ( pop )

#pragma enum ( small )
   enum e_tag { a=0,
                b=4294967296 /* C compiler error: larger than maximum int */
              } e_var;
#pragma enum ( pop )
```

3. pop オプションの使用の 1 つとして、メインファイルのデフォルトとは異なる列挙型ストレージを指定しているインクルード・ファイルの最後で、列挙型サイズの設定をポップすることが挙げられます。例えば、以下のインクルード・ファイルでは small_enum.h によってさまざまな最小値の列挙型が宣言された後、インクルード・ファイルの最後で、オプション・スタックの最後の値に指定がリセットされています。

```
#ifndef small_enum_h
#define small_enum_h
/*
 * File small_enum.h
 * This enum must fit within an unsigned char type
*/
#pragma enum ( small )
   enum e_tag {a, b=255};
   enum e_tag u_char_e_var; /* occupies 1 byte of storage */

/* Pop the enumeration size to whatever it was before */
#pragma enum ( pop )
#endif
```

以下のソース・ファイル (int_file.c) には、small_enum.h が含まれています。

```
/*
 * File int_file.c
 * Defines 4 byte enums
*/
#pragma enum ( int )
   enum testing {ONE, TWO, THREE};
   enum testing test_enum;

/* various minimum-sized enums are declared */
#include "small_enum.h"

/* return to int-sized enums. small_enum.h has popped the enum size
*/
   enum sushi {CALIF_ROLL, SALMON_ROLL, TUNA, SQUID, UNI};
   enum sushi first_order = UNI;
```

列挙型 test_enum および first_order は、どちらも 4 バイトのストレージを持つ int 型です。small_enum.h で定義される変数 u_char_e_var のストレージは 1 バイトで、unsigned char データ型で表現されます。

4. 以下のコード・フラグメントが ENUM = *SMALL オプションでコンパイルされる場合:

```
enum e_tag {a, b, c} e_var;
```

enum 定数の範囲は 0 から 2 です。この範囲は上記の表で説明されているすべての範囲内になります。優先順位に基づき、コンパイラーでは事前定義された unsigned char 型が使用されます。

5. 以下のコード・フラグメントが ENUM = *SMALL オプションでコンパイルされる場合:

   ```
   enum e_tag {a=-129, b, c} e_var;
   ```

   enum 定数の範囲は -129 から -127 です。この範囲は short (signed short) および int (signed int) の範囲内のみとなります。 short (signed short) はより小さいため、enum を表す場合に使用されます。

6. ファイル myprogram.c を以下のコマンドを使用してコンパイルする場合:

   ```
   CRTBNDC MODULE(MYPROGRAM) SRCMBR(MYPROGRAM) ENUM(*SMALL)
   ```

   ENUM オプションが #pragma enum ディレクティブによってオーバーライドされない限り、ソース・ファイル内のすべての enum 変数のストレージは最小となります。

7. 以下の行が含まれているファイル yourfile.c をコンパイルする場合:

   ```
   enum testing {ONE, TWO, THREE};
   enum testing test_enum;

   #pragma enum ( small )
   enum sushi {CALIF_ROLL, SALMON_ROLL, TUNA, SQUID, UNI};
   enum sushi first_order = UNI;

   #pragma enum ( int )
   enum music {ROCK, JAZZ, NEW_WAVE, CLASSICAL};
   enum music listening_type;
   ```

   以下のコマンドを使用する場合:

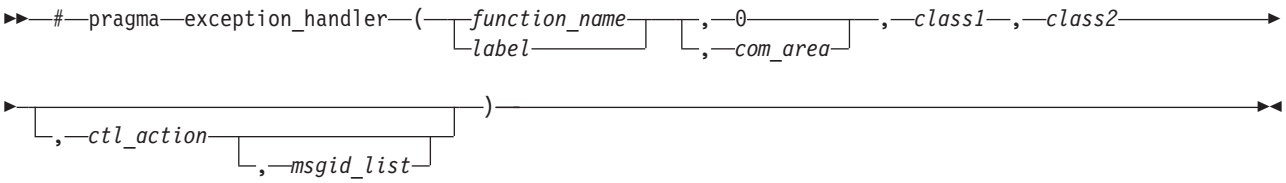
   ```
   CRTBNDC MODULE(YOURFILE) SRCMBR(YOURFILE)
   ```

   enum 変数 test_enum および first_order は最小化されます (すなわち、各ストレージは 1 バイトのみとなります)。その他の enum 変数 (listening_type) は int 型で、ストレージは 4 バイトです。

exception_handler

C C++

exception_handler 構文



説明

#pragma exception_handler が含まれているコード部分で、ユーザー定義の ILE 例外ハンドラーを使用可能にします。

#pragma disable_handler を使用して使用不可にされているのではなく、#pragma exception_handler によって使用可能にされた例外ハンドラーはすべて、それらのハンドラーが使用可能である関数の終了時に、暗黙的に使用不可にされます。

パラメーター

function

ユーザー定義の ILE 例外ハンドラーとして使用される関数の名前を指定します。

label ユーザー定義の ILE 例外ハンドラーとして使用されるラベルの名前を指定します。このラベルは #pragma exception_handler が使用可能である関数内で定義されている必要があります。ハンドラーによって制御が行われているときには、例外は暗黙的にハンドルされ、#pragma exception_handler ディレクティブを含む呼び出しの際にハンドラーによって定義されたラベルでの例外の再開は制御されます。呼び出しスタックが最新の呼び出しから取り消されても、#pragma exception_handler ディレクティブが含まれている呼び出しは取り消されません。#pragma exception_handler の位置にかかわらず、ラベルは関数定義のどのステートメント部分にでも入れることができます。

com_area

通信域で使用されます。*com_area* を指定する必要がない場合、ディレクティブの 2 番目のパラメーターとしてゼロが使用されます。*com_area* がディレクティブで指定されている場合は、整数、浮動小数点、倍精度、構造体、共用体、配列、列挙型、ポインター、またはパック 10 進のデータ型のいずれかの変数である必要があります。com_area は VOLATILE 修飾子を使用して宣言される必要があります。構造体や共用体のメンバーにすることはできません。

class1、*class2*

例外マスクの最初の 4 バイトおよび最後の 4 バイトを指定します。<except.h> ヘッダー・ファイルでは、クラス・マスクに使用することができる値について説明しています。このファイルではこれらの値のマクロ定義についても説明しています。*class1* および *class2* の評価は、必要なマクロ展開がすべて終了した後、整数定数式となる必要があります。以下の有効な *class2* の値をモニターすることができます。

- _C2_MH_ESCAPE
- _C2_MH_STATUS
- _C2_MH_NOTIFY、および
- _C2_FUNCTION_CHECK。

ctl_action

この例外ハンドラーに対してどのアクションが実行される必要があるかを示すように整数定数を指定します。ハンドラーが関数である場合、デフォルト値は _CTLA_INVOKE です。ハンドラーがラベルである場合、デフォルト値は _CTLA_HANDLE です。このパラメーターはオプションです。

<except.h> ヘッダー・ファイルで定義されている有効な例外制御アクションは、以下のとおりです。

#define の名前	**定義済みの値およびアクション**
_CTLA_INVOKE	1 に定義されます。この制御アクションはディレクティブで指定された関数を呼び出しますが、例外はハンドルしません。例外が明確にハンドルされない場合、処理は続行されます。このアクションは関数においてのみ有効です。
_CTLA_HANDLE	2 に定義されます。ハンドラーが呼び出される前に、例外がハンドルされてメッセージがログに記録されます。ハンドラーによって制御されるときには、例外は既にアクティブではありません。例外ハンドラーが戻るとき、例外処理は終了します。このアクションは関数およびラベルにおいて有効です。
_CTLA_HANDLE_NO_MSG	3 に定義されます。例外はハンドルされますが、ハンドラーが呼び出される前にメッセージがログに記録されることはありません。ハンドラーによって制御されるときには、例外は既にアクティブではありません。例外メッセージはログに記録されません。型定義 _INTRPT_Hndlr_Parms_T の Msg_Ref_Key はゼロに設定されています。例外ハンドラーが戻るとき、例外処理は終了します。このアクションは関数およびラベルにおいて有効です。

#define の名前	定義済みの値およびアクション
_CTLA_IGNORE	131 に定義されます。例外はハンドルされ、メッセージはログに記録されます。制御はディレクティブで指定されたハンドラー関数には渡されず、例外がアクティブではなくなります。この例外の原因となった命令の直後の命令から再開されます。このアクションは関数においてのみ有効です。
_CTLA_IGNORE_NO_MSG	132 に定義されます。例外はハンドルされ、メッセージはログに記録されません。制御はディレクティブで指定されたハンドラー関数には渡されず、例外がアクティブではなくなります。この例外の原因となった命令の直後の命令から再開されます。このアクションは関数においてのみ有効です。

msgid_list

メッセージ ID リストが含まれているオプションのストリング・リテラルを指定します。どの ID がメッセージ ID リストにある ID の 1 つと一致しているかについて例外が発生した場合にのみ、例外ハンドラーは効力を持ちます。リストは 7 文字のメッセージ ID で、最初の 3 文字がメッセージ接頭語となり、最後の 4 文字がメッセージ番号となります。各メッセージ ID は複数のスペースやコンマで区切られています。このパラメーターはオプションですが、指定する場合には *ctl_action* を同時に指定する必要があります。

例外ハンドラーで制御するには、*class1* および *class2* の選択基準が満たされている必要があります。また、*msgid_list* を指定する場合、以下の基準に従って、例外がリスト内の少なくとも 1 つのメッセージ ID と一致している必要があります。

- メッセージ ID が例外と正確に一致している。
- メッセージ ID (右端の 2 文字が 00 の場合) の左端の 5 文字が任意の例外 ID と一致している。例えば、CPF5100 のメッセージ ID は、メッセージ ID が CPF51 で始まる任意の例外と一致します。
- メッセージ ID (右端の 4 文字が 0000 の場合) が同じ接頭部を持つ任意の例外 ID と一致している。例えば、CPF0000 のメッセージ ID は、メッセージ ID の接頭部が CPF (CPF0000 から CPF9999) である任意の例外と一致します。
- *msgid_list* を指定したものの、生成される例外がリストで指定された例外ではない。この場合、例外ハンドラーによって制御を行うことはできません。

使用に関する注意

ハンドラー関数では、パラメーターとして 16 バイトのポインターのみを使用することができます。

<except.h> ヘッダー・ファイルで定義されたマクロ _C1_ALL は、すべての有効な *class1* 例外マスクと同等に使用することができます。<except.h> ヘッダー・ファイルで定義されたマクロ _C2_ALL は、4 つの有効な *class2* 例外マスクすべてと同等に使用することができます。

2 進 OR 演算子を使用して、異なる型のメッセージをモニターすることができます。例えば、以下のようになります。

```
#pragma exception_handler(myhandler, my_comarea, 0, _C2_MH_ESCAPE | \
               _C2_MH_STATUS | _C2_MH_NOTIFY, _CTLA_IGNORE, "MCH0000")
```

は、モニター対象である 4 つの *class2* 例外クラスのうちの 3 つに対し、例外モニターをセットアップします。

以下のいずれかが発生した場合、コンパイラーはエラー・メッセージを出力します。

- ディレクティブが C 関数本体の外部または C ステートメントの内部に出現する。
- 指定されたハンドラーが宣言された関数または定義されたラベルではない。

- *com_area* 変数が宣言されていないか、または有効なオブジェクト・タイプが含まれていない。
- 例外クラス・マスクがいずれも有効な整数定数ではない。
- *指定されたハンドラーがラベル (_CTLA_INVOKE, _CTLA_IGNORE, _CTLA_IGNORE_NO_MSG) である場合に許可されない値の 1 つが、ctl_action* である。
- *msgid_list* は指定したが、*ctl_action* は指定していない。
- *msgid_list* のメッセージが有効ではない。大文字でないメッセージ接頭語が有効とみなされない。
- ストリングのメッセージがブランクやコンマで区切られていない。
- ストリングが “ ” で囲まれていないか、または 4 KB より長い。
- 指定されたハンドラー関数が引数最適化 (#pragma argopt) で指定されている。

ラベルがユーザー定義の例外ハンドラーとして使用されると、例外が発生した場合に、exception_handler プラグマと disable_handler プラグマの間のコードの一部がスキップされる可能性があります。このため、例外の範囲内の宣言またはステートメントがスキップされる可能性があり、これにより、変数が初期化されない状況や、(ストレージは宣言時に可変長配列に割り振られるため) 可変長配列にストレージが割り振られない状況が発生します。次に例を示します。

1. i が 0 の場合、z への代入により例外が発生する可能性があります。この場合、制御は LABEL に分岐します。
2. 例外が発生すると、ptr が &z に設定されず、vla にはストレージが割り振られません。
3. ptr が初期化されず、vla のストレージが割り振られない場合、*ptr および vla[0] の使用がイリーガルになる可能性があります。

```
void func(unsigned int i)
{
  unsigned int *ptr = NULL;
  unsigned int z;
#pragma exception_handler(LABEL,0, _C1_ALL, _C2_ALL)
  z = 45/i;    // 1
  ptr = &z;    // 2
  unsigned int vla[z];
#pragma disable_handler
LABEL:
  vla[0] = *ptr;   // 3
  return;
}
```

また、可変長配列は実行時にそのストレージを宣言し、それによりストレージ割り振りの例外が発生する可能性があるため、可変長配列宣言に対して例外ハンドラーを使用可能に設定することをお勧めします。

#pragma exception_handler ディレクティブの使用例および使用の詳細については、*ILE C/C++ プログラマーの手引き* を参照してください。

hashome

C++

hashome 構文

説明

このプラグマは、指定したクラスに #pragma ishome によって指定されるホーム・モジュールがあることをコンパイラーに通知します。このクラスの仮想関数表は、ある特定のインライン関数と同様に、静的には生成されません。その代わり、#pragma ishome が指定されたクラスのコンパイル単位において、外部として参照されます。

パラメーター

className
上記の外部参照を必要とするクラスの名前を指定します。*className* はクラスであり、定義されている必要があります。

AllInlines
className 内のすべてのインライン関数が、外部として参照される必要があることを指定します。この引数では、大/小文字を区別しません。

一致する #pragma hashome がない #pragma ishome が存在する場合、警告が出されます。

5-26 ページの『ishome』も参照してください。

implementation

C++

implementation 構文

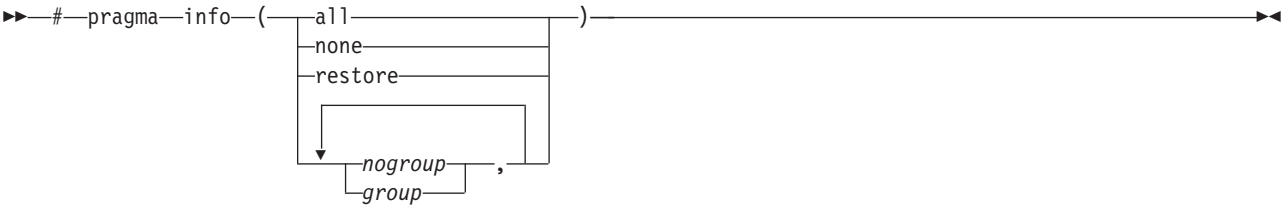
説明

#pragma implementation ディレクティブは、関数テンプレート定義を含むファイルの名前をコンパイラーに通知します。この関数テンプレート定義は、プラグマが含まれているインクルード・ファイル内のテンプレート宣言に対応します。このプラグマは、宣言が使用できるところならばどこにでも入れることができます。テンプレート関数を効果的または自動的に生成するようにプログラムを編成する場合に使用されます。

info

C++

info 構文



説明

このプラグマは、どの診断メッセージがコンパイラーによって作成されるかを制御する場合に使用できます。

パラメーター

all このプラグマが有効である間、すべての診断メッセージが生成されます。

none このプラグマが有効である間、すべての診断メッセージがオフとなります。

restore `pragma info` の以前の設定をリストアします。

nogroup
指定された診断グループに関連付けられるすべての診断メッセージを生成しません。特定のメッセージのグループをオフにするには、そのグループ名の前に「no」を付加します。例えば、**nogen** は CHECKOUT メッセージを抑制します。有効なグループ名を以下にリストします。

group 指定された診断グループに関連付けられるすべての診断メッセージを生成します。有効なグループ名は、以下のとおりです。

lan 言語レベルの効果に関する情報を表示します。

gnr コンパイラーによって一時変数が作成された場合にメッセージを生成します。

cls クラスの使用に関する情報を表示します。

eff 影響を及ぼさないステートメントに関して警告します。

cnd 条件式で起こり得る冗長または問題に関して警告します。

rea 到達不能ステートメントに関して警告します。

par 使用されていない関数仮パラメーターをリストします。

por C/C++ 言語の非ポータブルな使用法をリストします。

trd 起こり得るデータの切り捨てまたは損失に関して警告します。

use 未使用の自動変数または静的変数を検査します。

gen 一般的な CHECKOUT メッセージをリストします。

inline

C

inline 構文

説明

#pragma inline ディレクティブは、*function_name* をインライン化するかどうかを指定します。このプラグマはソース内の任意の場所に入れることができますが、ファイル・スコープ内である必要があります。INLINE(*ON) コンパイラー・オプション・パラメーターが指定されていない場合、このプラグマは効果がありません。 **#pragma inline** が関数で指定されている場合、このインラインによって、すべての呼び出しでインライン化されるように指定された関数が強制されます。関数は選択 (*NOAUTO) モード、および自動 (*AUTO) INLINE モードの両方でインライン化されます。

インライン化によって、関数呼び出しは関数の機械語コードで置換されます。これにより、関数呼び出しのオーバーヘッドが減少し、より多くのコードがオプティマイザーに公開され、より多くの最適化の機会が得られます。

使用に関する注意

- インライン化はコンパイラーの最適化がレベル 30 以上に設定されている場合にのみ、実行されます。
- 直接的には、再帰的関数はインライン化されません。直接の再帰が検出されるまで、再帰的関数は間接的にインライン化されます。
- 変数の引数リストを持つ関数呼び出しは、引数がその引数リストの変数部分で検出された場合には、インライン化されません。
- 関数が関数ポインターによって呼び出される場合、インライン化は行われません。
- *function_name* がプラグマが含まれている同じコンパイル単位で定義されていない場合、pragma inline ディレクティブは無視されます。
- 以下のすべてに該当する場合、関数の定義は破棄されます。
  - 関数が静的である。
  - 関数に独自のアドレスが取得されていない。
  - 関数が呼び出されるいずれの場所においてもインライン化されている。

  このアクションにより、関数が使用されているモジュールおよびプログラム・オブジェクトのサイズを減らすことができます。

関数のインライン化について詳しくは、*ILE C/C++ プログラマーの手引き* の『Function Call Performance』を参照してください。

ishome

C++

ishome 構文

```
►►—#—pragma—ishome—(—className—)————————————————————————————————►◄
```

説明

このプラグマは、指定されたクラスのホーム・モジュールが現行のコンパイル単位であることをコンパイラーに通知します。ホーム・モジュールとは、仮想関数の表などの項目が保管される場所のことです。コンパイル単位の外部から項目が参照される場合、その項目はホームの外部では生成されません。この利点はコードの最小化です。

パラメーター

className
ホームが現行のコンパイル単位となるクラスのリテラル名を指定します。

一致する #pragma hashome がない #pragma ishome が存在する場合、警告が出されます。

5-23 ページの『hashome』も参照してください。

isolated_call

C++

isolated_call 構文

説明

そのパラメーターで指定されている作用以外の副次作用がない、または副次作用に依存しない関数をリストします。

パラメーター

function

ID、演算子関数、型変換関数、または修飾名のいずれかである 1 次式を指定します。ID は型関数のものまたは関数の型定義である必要があります。名前が、多重定義された関数を指す場合、関数のすべての変形が、分離された呼び出しとしてマークされます。

使用に関する注意

リストされた関数にはそのパラメーターで指定されている作用以外の副次作用がない、または副次作用に依存しないことが、プラグマによってコンパイラーに通知されます。次の場合、関数に副次作用があるか、または関数は副次作用に依存していると考えられます。

- volatile 型オブジェクトへのアクセス
- 外部オブジェクトの変更
- 静的オブジェクトの変更
- ファイルの変更
- 別のプロセスまたはスレッドによって変更されたファイルへのアクセス
- 動的オブジェクトの割り振り (戻る前に解放される場合は除く)
- 動的オブジェクトの解放 (同一の呼び出し中に割り振られた場合は除く)
- システム状態の変更 (丸めモードまたは例外処理など)
- 上記のいずれかを行う関数の呼び出し

基本的に、ランタイム環境の状態の変化は副次作用と見なされます。許容される副次作用は、ポインターまたは参照によって渡される関数引数の変更のみです。これ以外の副次作用のある関数は、#pragma isolated_call ディレクティブでリストされたときに誤った結果を導く可能性があります。

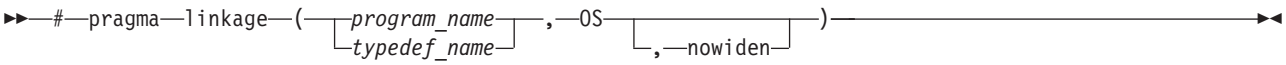
関数を isolated_call としてマークすると、呼び出された関数では外部変数および静的変数を変更できないこと、およびストレージへのペシミスティック参照は、呼び出し関数から (該当する場合) 削除できることが最適化プログラムに指示されます。命令はより柔軟に再配列することができ、その結果、パイプラインの遅延が減少し、プロセッサーでは命令がより高速に実行されます。 1 つの関数に対する同一のパラメーターでの複数の呼び出しは結合することができ、結果が不要な場合、呼び出しを削除でき、呼び出しの順序を変更できます。

指定の関数では、非 volatile 型の外部オブジェクトを検査して、ランタイム環境の非 volatile 状態に依存する結果を返すことが許可されます。また、関数に渡されるポインター引数によって示されるストレージも関数によって変更できます (参照による呼び出し)。関数自体を呼び出す関数、またはローカル静的ストレージに依存する関数は指定しないでください。#pragma isolated_call ディレクティブでそのような関数をリストすると、予測できない結果になる可能性があります。

linkage

C

linkage 構文



説明

指定の関数または関数型定義を IBM i パラメーター引き渡し規則に従った外部プログラムとして識別します。

このプラグマでは、外部プログラムへの呼び出しのみが許可されます。結合プロシージャーの呼び出しについては、5-5 ページの『argument』プラグマを参照してください。

パラメーター

program_name
外部プログラム名を指定します。IBM i プログラム命名規則に従うために #pragma map ディレクティブが指定されている場合以外は、外部名は長さ 10 文字を超えない大文字で指定する必要があります。ただし、#pragma map で指定された名前が長すぎる場合、名前は #pragma linkage 処理中に 255 文字に切り捨てられます。

typedef_name
このプラグマに影響を受ける型定義を指定します。

OS 外部プログラムが IBM i 呼び出し規則に従って呼び出されることを指定します。

nowiden
これが指定されている場合、引数はコピーされて引き渡される前に拡大されません。

使用に関する注意

このプラグマによって、IBM i プログラムから外部プログラムが呼び出されます。外部プログラムは、任意の言語で作成できます。

プラグマは、関数、関数型、および関数ポインター型に適用できます。関数型定義に適用された場合、プラグマの効果はすべての関数、およびオリジナルの型定義によって宣言された新規の型定義にも適用されます。

このディレクティブは、プログラム名 (または型) が宣言される前、または宣言された後に使用できます。ただし、プラグマ・ディレクティブの前に、プログラムを呼び出したり、宣言に型を使用することはできません。

関数または関数ポインターは `int` または `void` のみを返すことができます。

呼び出しの引数は、次の IBM i 引数引き渡し規則に従って引き渡されます。

- 非アドレス引数は一時的な場所にコピーされ、拡大されて (`nowiden` が指定されていない場合のみ)、コピーのアドレスが、呼び出されたプログラムに引き渡されます。
- アドレス引数は呼び出されたプログラムに直接引き渡されます。

次の場合、コンパイラーでは警告メッセージが発行され、#pragma linkage ディレクティブは無視されます。

- プログラムが int または void 以外の戻りの型で宣言されている。
- 関数に 256 を超えるパラメーターが含まれている。
- 別のプラグマ linkage ディレクティブが、関数または関数型に対して既に指定されている。
- 関数が現行のコンパイル単位で定義されている。
- 指定の関数が既に呼び出されているか、または型が宣言の中で既に使用されている。
- #pragma argopt または #pragma argument が、指定された関数または型に対して既に指定されている。
- プラグマ・ディレクティブで指定されたオブジェクトが関数または関数型ではない。
- プラグマ・ディレクティブで指定されたオブジェクトの名前は 10 文字を超過してはならず、超過した場合、名前は切り捨てられる。

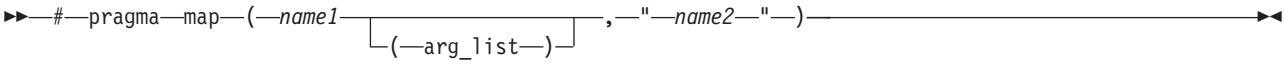
map

C

map 構文

C++

map 構文



説明

ID へのすべての参照を外部的に定義された別の ID へ変換します。

パラメーター

name1 ソース・コードで使用されている名前。C の場合、*name1* は、外部リンケージを持つデータ・オブジェクトまたは関数を表すことができます。 C++ の場合、*name1* は、外部リンケージを持つデータ・オブジェクト、非多重定義関数または多重定義関数、多重定義演算子のいずれかを表すことができます。マップされる名前は、グローバル名前空間にない場合、完全修飾する必要があります。

name1 は、自身が参照されるコンパイル単位内で宣言し、他のコンパイル単位で定義しないでください。 *name1* を、プログラム内の任意の場所にある別の **#pragma map** ディレクティブで使用しないでください。

arg_list

name1 によって指定された多重定義関数または演算子関数の引数のリスト。*name1* によって多重定義関数が指定されると、この関数は括弧で囲み、引数リスト (存在する場合) を組み込む必要があります。 *name1* が非多重定義関数を指定する場合は、*name1* のみが必須で、括弧および引数リストはオプションです。

name2 オブジェクト・コードに現れる名前。*name2* の長さが 65535 バイトを超えた場合、メッセージが発行され、プラグマは無視されます。

name2 は、*name1* が参照されているのと同じコンパイル単位の中で、宣言または定義されます。*name2* は、同じコンパイル単位の中で別の **#pragma map** ディレクティブが使用しているものと同じにしないでください。

使用に関する注意

#pragma map ディレクティブは、プログラムのどこに指定してもかまいません。

関数を実際にマップするには、マップ・ターゲット関数 (*name2*) が、リンク時に使用可能な定義を持つ必要があります。

name2 が C++ リンケージを使用する関数名を指定する場合、*name2* はそのマングル名を使用して指定される必要があります。例えば、以下のようになります。

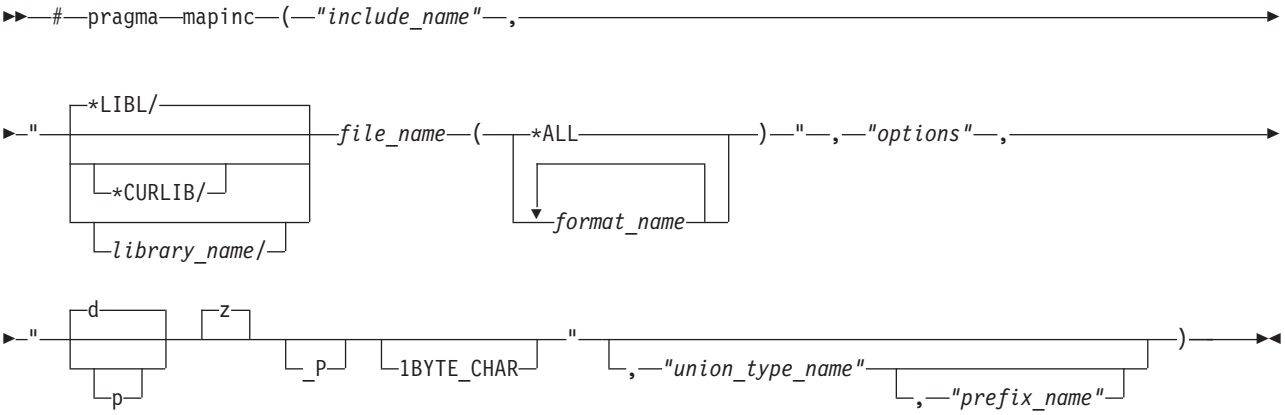
```
int foo(int, int);
#pragma map(foo, "bar__FiT1")

int main( )
{
  return foo(4,5);
}

int bar(int a, int b)
{
  return a+b;
}
```

mapinc

mapinc 構文



説明

データ記述仕様 (DDS) がモジュールに組み込まれることを示します。このディレクティブでは、ファイルおよび DDS レコード・フォーマットが識別され、組み込まれるフィールドについての情報が示されます。関連の include ディレクティブと共にこのプラグマを使用すると、外部ファイル記述で指定されているレコード・フォーマットから型定義が自動的にコンパイラーで生成されるようになります。

パラメーター

include_name
ソース・プログラムの #include ディレクティブで示した名前です。

library_name
外部記述ファイルが含まれているライブラリーの名前です。

file_name
外部記述ファイルの名前です。

format_name
プログラムに組み込まれる DDS レコード・フォーマットを示す必須パラメーターです。複数のレコード・フォーマット (format1 format2)、または 1 ファイル中のすべてのフォーマット (*ALL) を組み込むことができます。

options 指定できる *options* は次のとおりです。

input DDS で INPUT または BOTH として宣言されたフィールドは、型定義構造体に組み込まれます。応答標識は、キーワード INDARA が装置ファイルの外部ファイル記述 (DDS ソース) で指定されていない場合、入力構造体に組み込まれます。

output DDS で OUTPUT または BOTH として宣言されたフィールドは、型定義構造体に組み込まれます。オプション標識は、キーワード INDARA が装置ファイルの外部ファイル記述 (DDS ソース) で指定されていない場合、出力構造体に組み込まれます。

both DDS で INPUT、OUTPUT または BOTH として宣言されたフィールドは、型定義構造体に組み込まれます。オプション標識および応答標識は、キーワード INDARA が装置ファイルの外部ファイル記述 (DDS ソース) で指定されていない場合、両方の構造体に組み込まれます。

key 外部ファイル記述でキーとして宣言されたフィールドが組み込まれます。このオプションはデータベース・ファイルおよび DDM ファイルに対してのみ有効です。

indicators
標識オプションが指定されている場合、標識に対して個別に 99 バイトの構造体が作成されます。このオプションは装置ファイルに対してのみ有効です。

lname このオプションによって、長さ 128 文字までのファイル名を使用できるようになります。ファイル名が 10 文字を超える場合、名前は関連ショート・ネームに変換されます。ショート・ネームは、外部ファイル定義を抽出するために使用されます。ファイル名が 10 文字以下の短い名前の場合、関連ショート・ネームへは変換されません。長さ 30 文字までのレコード・フィールド名がコンパイラーによって型定義で生成されます。

lvlchk 構造の配列型定義が、レベル検査情報用に生成されます (型名 _LVLCHK_T)。型 _LVLCHK_T のオブジェクトへのポインターも生成され、レベル検査情報 (フォーマット名およびレベル ID) で初期化されます。

nullflds
DDS のレコード・フォーマットにヌル可能フィールドが 1 つでもある場合、そのフォーマットの各フィールドに対して文字フィールドを含むヌル・マップ型定義が生成されます。この型定義によって、ユーザーはヌルのフィールドを指定できます (各ヌル・フィールドに 1 を設定し、それ以外はゼロを設定します)。また、nullflds と共に key オプションが使用されている場合に、フォーマットにヌル可能キー・フィールドが 1 つでもある場合、そのフォーマットの各キー・フィールドに対して文字フィールドを含む追加の型定義が生成されます。

物理ファイルおよび論理ファイルに対しては、input、both、key、lvlchk、および nullflds を指定できます。装置ファイルに対しては、input、output、both、indicator、および lvlchk を指定できます。

データ型は、次のいずれか (複数可) に指定でき、スペースで区切る必要があります。

d パック 10 進データ型。

p DDS によるパック 10 進数フィールドが文字フィールドとして宣言されます。

z DDS によるゾーン・フィールドが文字フィールドとして宣言されます。コンパイラーにはゾーン・データ型がないため、これがデフォルトになります。

_P パック構造体が生成されます。

1BYTE_CHAR
DDS で定義された 1 バイト文字用に 1 バイト文字フィールドが生成されます。

" " デフォルト値の d および z が使用されます。

union_type_name
組み込まれた型定義の共用体定義が、名前 union_type_name_t として作成されます。このパラメーターはオプションです。

prefix_name
生成される型定義構造体名の最初の部分を指定します。接頭部が指定されていない場合、ライブラリーおよび file_name が使用されます。

使用に関する注意

#pragma mapinc ディレクティブを外部記述ファイルと共に使用する方法の詳細については、*ILE C/C++ プログラマーの手引き* の『*Using Externally Described Files in a Program*』を参照してください。

margins

C

margins 構文

説明

#pragma ディレクティブが存在するソース・メンバーのレコードのスキャン時に、 left margin が最初の桁として使用され、right margin が最後の桁として使用されるように指定します。

マージンの設定は、その設定が置かれているソース・メンバーにのみ適用され、そのメンバー内のインクルード・ディレクティブで指定されているソース・メンバーには無効です。

パラメーター

left margin
ゼロより大きく、32754 より小さな数値である必要があります。 *left margin* は *right margin* より小さい必要があります。

right margin
ゼロより大きく、32754 より小さな数値か、アスタリスク (*) である必要があります。 *right margin* は *left margin* より大きい必要があります。コンパイラーは、左マージンと右マージンの間をスキャンします。 *right margin* の値としてアスタリスクが指定されている場合、コンパイラーは、入力レコードの最後に指定されている左マージン以降をスキャンします。

使用に関する注意

#pragma margins ディレクティブは、そのディレクティブから別の #pragma margins または nomargins ディレクティブが検出されるまでか、検出されない場合はソース・メンバーの終わりまでの行に有効です。

#pragma margins ディレクティブと #pragma sequence ディレクティブは、併用することができます。この 2 つの #pragma ディレクティブが同じ桁を予約する場合は、#pragma sequence ディレクティブが優先され、その桁はシーケンス番号用に予約されます。

例えば、#pragma margins ディレクティブで 1 および 20 のマージンが指定され、#pragma sequence ディレクティブで 15 から 25 の桁がシーケンス番号用に指定された場合、有効なマージンは 1 および 14 となり、シーケンス番号用に予約される桁は 15 から 25 までとなります。

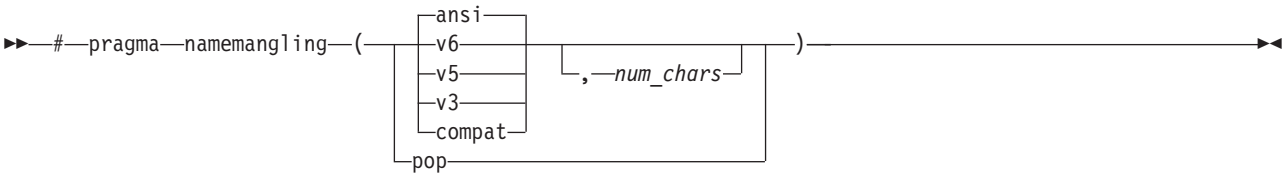
指定されたマージンがサポートされている範囲内にないか、マージンに数値以外の値が含まれている場合は、コンパイル中に警告メッセージが出されて、そのディレクティブは無視されます。

5-36 ページの『nomargins』および 5-46 ページの『sequence』プラグマも参照してください。

namemangling

C++

namemangling 構文



説明

C++ ソース・コードから生成した外部シンボル名のネーム・マングリング方式を選択します。前バージョンのコンパイラーで作成されたリンク・モジュールとのバイナリー互換性を保持するために、このオプションおよびプラグマが提供されます。後方互換性が必要ない場合は、このオプションのデフォルト設定を変更しないことを推奨します。

パラメーター

ansi ネーム・マングリング方式は、関数テンプレートの多重定義を含め、標準 C++ の最新言語機能を完全にサポートしています。 **ansi** がデフォルトです。

v6 ネーム・マングリング方式は、バージョン V5R3M0、V5R4M0、および V6R1M0 のコンパイラーで使用されているのと同じものです。

v5 ネーム・マングリング方式は、バージョン V5R1M0 および V5R2M0 のコンパイラーで使用されているのと同じものです。

v3 ネーム・マングリング方式は、V5R1M0 以前のバージョンのコンパイラーと同じものです。

compat
このオプションは、前述の **v3** と同じです。

num_chars
マングル名に使用可能な最大文字数を指定します。このサブオプションを指定しないと、デフォルト最大値が 255 文字である **v3** および **compat** を除くすべての設定に対してデフォルト最大値 64000 文字が使用されます。

pop 現行のプラグマ設定を破棄し、前のプラグマ・ディレクティブによって指定された設定に戻します。前のプラグマ・ディレクティブが指定されていない場合は、**ansi** のデフォルト設定が使用されます。

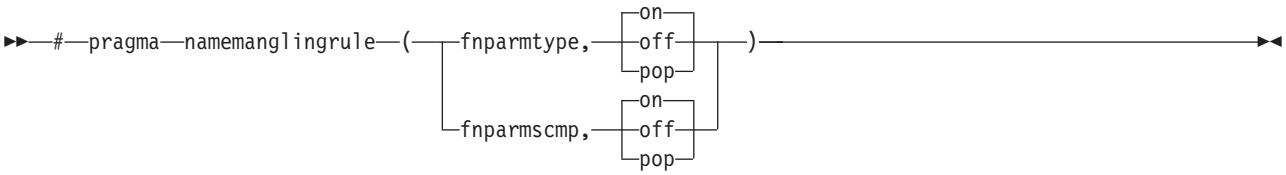
使用に関する注意

RTBND(*LLP64) コンパイラー・オプションが使用されていない場合、**#pragma namemangling** ディレクティブはサポートされません。

namemanglingrule

C++

namemanglingrule 構文



説明

ソース・コードの選択した部分に有効なネーム・マングリング方式、特に関数仮パラメーターの cv 修飾子のマングリングに関して細かい制御を提供します。

#pragma namemanglingrule ディレクティブを使用すると、トップレベルの cv 修飾子が関数仮パラメーターでマングルされるかどうか、またはコンパイラーが繰り返される関数仮パラメーターが同等であるかを比較するときに、中間レベルの cv 修飾子が考慮されるかどうかを制御することができます。

デフォルト

#pragma namemangling(ansi|v6) が有効な場合は **fnparmtype, on** です。それ以外の場合、デフォルトは **fnparmtype, off** です。

#pragma namemangling(ansi) が有効な場合は、**fnparmscmp, on** です。それ以外の場合、デフォルトは **fnparmscmp, off** です。

パラメーター

fnparmtype, on
トップレベルの cv 修飾子は、関数仮パラメーターのマングル名でエンコードされません。また、トップレベルの cv 修飾子は、繰り返される関数仮パラメーターが同等であるかを比較するときは

無視されます。トップレベルの cv 修飾子の使用方法のみが異なる関数仮パラメーターは同等と見なされ、圧縮エンコード方式に従ってマングルされます。この設定は、ILE C++ V5R3M0 以降のリリースと互換性があります。

fnparmtype, off
トップレベルの cv 修飾子は、関数仮パラメーターのマングル名でエンコードされます。また、cv 修飾子の使用方法のみが異なった、繰り返される関数仮パラメーターは同等と見なされず、別個のパラメーターとしてマングルされます。この設定は、ILE C++ V5R2M0 以前のリリースと互換性があります。

fnparmtype, pop
以前に有効だった **fnparmtype** 設定に戻します。以前に有効だった設定がない場合は、デフォルトの **fnparmtype** 設定が使用されます。

fnparmscmp, on
中間レベルの cv 修飾子は、繰り返される関数仮パラメーターが等価に対して比較されるときに考慮されます。中間レベルの cv 修飾子の使用方法が異なる、繰り返される関数仮パラメーターは別個のパラメーターとしてマングルされます。この設定は、ILE C++ V7R1M0 以降のリリースと互換性があります。

fnparmscmp, off
中間レベルの cv 修飾子は繰り返される関数仮パラメーターが同等であるかを比較するときは無視されます。中間レベルの cv 修飾子の使用方法のみが異なる関数仮パラメーターは同等と見なされ、圧縮エンコード方式に従ってマングルされます。この設定は、ILE C++ V6R1M0 以前のリリースと互換性があります。

fnparmscmp, pop
以前に有効だった **fnparmscmp** 設定に戻します。以前に有効だった設定がない場合は、デフォルトの **fnparmscmp** 設定が使用されます。

使用に関する注意

1. **#pragma namemanglingrule** はグローバル、クラス、および関数スコープ内で許可されます。これは、外部リンケージを持つブロック・スコープ関数宣言に影響を与えません。
2. 関数宣言および定義の前に、別のプラグマ設定を指定することができます。後続の宣言および定義内で **#pragma namemanglingrule** 設定が矛盾する場合、コンパイラーはこれらの設定を無視して警告メッセージを出します。
3. RTBND(*LLP64) コンパイラー・オプションが使用されていない場合、**#pragma namemanglingrule** ディレクティブはサポートされません。

noargv0

C

noargv0 構文

説明

ソース・プログラムで argv[0] を使用しないことを指定します。このプラグマによって、小さな C プログラムが多数存在するアプリケーションや、何度も呼び出される小さなプログラムが存在するアプリケーションのパフォーマンスを向上させることができます。

使用に関する注意

#pragma noargv0 は、main() 関数が定義されているコンパイル単位内に置く必要があります。そうでない場合は無視されます。

noargv0 プラグマ・ディレクティブが有効な場合、argv[0] は NULL になります。その引数ベクトル内の他の引数は、このディレクティブの影響を受けません。#pragma noargv0 ディレクティブが指定されていない場合、argv[0] には現在実行中のプログラムの名前が入ります。

noinline (function)

noinline 構文

説明

関数がインライン化されないことを指定します。INLINE コンパイラー・オプションの設定は、この function_name については無視されます。

使用に関する注意

最初に指定されたプラグマが使用されます。#pragma noinline の指定後に同じ関数に対して #pragma inline が指定されると、その関数にはすでに #pragma noinline が指定されていることを示す警告が出されます。

#pragma noinline ディレクティブは、ファイル・スコープにのみ使用できます。ファイル・スコープの外にある場合、プラグマは無視され、警告が出されます。

nomargins

nomargins 構文

説明

入力レコード全体を入力用にスキャンすることを指定します。

使用に関する注意

#pragma nomargins ディレクティブは、そのディレクティブから #pragma margins ディレクティブが検出されるまでか、検出されない場合はソース・メンバーの終わりまでの行に有効です。

5-32 ページの『margins』プラグマも参照してください。

nosequence

C

nosequence 構文

説明

入力レコードにシーケンス番号が含まれないことを指定します。

使用に関する注意

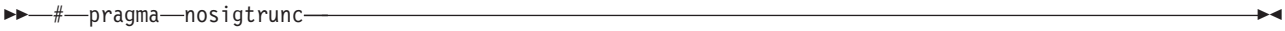
#pragma nosequence ディレクティブは、そのディレクティブから #pragma sequence ディレクティブが検出されるまでか、検出されない場合はソース・メンバーの終わりまでの行に有効です。

5-46 ページの『sequence』プラグマも参照してください。

nosigtrunc

C

nosigtrunc 構文



説明

実行時に、算術演算、代入、キャスト、初期化、または関数呼び出しにおいてパック 10 進数によるオーバーフローが発生した場合に、例外が生成されないことを指定します。このディレクティブは、パック 10 進オーバーフローによって生じるシグナルを抑制します。 #pragma nosigtrunc ディレクティブは、ファイル・スコープにのみ使用できます。関数、ブロック、または関数プロトタイプ・スコープで #pragma nosigtrunc ディレクティブが検出された場合は、警告メッセージが出されて、そのディレクティブは無視されます。

使用に関する注意

この #pragma ディレクティブにはファイル・スコープがあるため、関数定義の外に置く必要があります。そうでない場合は無視されます。その場合でも、一部のパック 10 進演算のコンパイルにおいて、オーバーフローの発生する可能性が高い場合には、警告メッセージが出される可能性があります。パック 10 進 エラーについて詳しくは、*ILE C/C++ プログラマーの手引き* を参照してください。

pack

C C++

pack 構文

説明

#pragma pack ディレクティブは、このディレクティブの後に続く構造体、共用体、またはクラス (C++ のみ) のメンバーに対して使用する位置合わせ規則を指定します。 C++ では、パッキングは*宣言* またはタイプに対して実行されます。この点で、パッキングが*定義* に対しても実行される C とは異なります。

コンパイラー・コマンドと一緒に PACKSTRUCT オプションを使用して、指定された境界に沿ってパッキングを実行させることもできます。詳細については、6-40 ページの『PACKSTRUCT』を参照してください。

パラメーター

1、2、4、8、16
構造体および共用体は、指定されたバイト境界に沿ってパッキングされます。

default コンパイラー・オプション PACKSTRUCT によって指定される位置合わせ規則を選択します。

system デフォルトの IBM i 位置合わせ規則を選択します。

pop、reset
以前に有効だった位置合わせ規則を選択し、現在の規則を破棄します。これは、`#pragma pack ( )` を指定するのと同じです。

以下の例では、単語 *struct* または *union* を *class* の代わりに使用することができます。

#pragma pack は、スタック・ベースで設定されます。 pack の値はすべて、ユーザーのソース・コードの構文解析時に、スタックへとプッシュされます。そのスタックの先頭にある値が、現在のパッキング値です。 #pragma pack (reset)、#pragma pack(pop)、または #pragma pack() ディレクティブが指定されるとそのスタックの先頭がポップされ、そのスタック内の次のエレメントが新しいパッキング値になります。スタックが空の場合、PACKSTRUCT コンパイラー・オプションが指定されていれば、その値が使用されます。指定されていない場合、NATURAL 位置合わせのデフォルトの設定値が使用されます。

PACKSTRUCT コンパイラー・オプションの設定は、#pragma pack ディレクティブによってオーバーライドされますが、スタックの最後尾には必ず残ります。パッキング・オプションではキーワード **_Packed** が最も優先され、#pragma pack ディレクティブや PACKSTRUCT コンパイラー・オプションでこのキーワードをオーバーライドすることはできません。

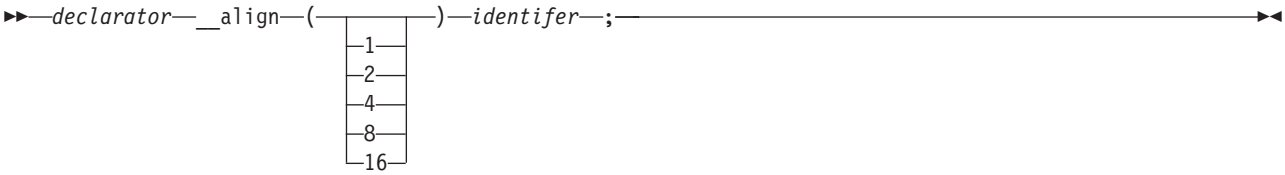
デフォルトでは、どのメンバーもその NATURAL 位置合わせを使用します。メンバーを、そのメンバーの NATURAL 位置合わせより大きな値で位置合わせすることはできません。char 型は、1 バイト境界に沿った位置合わせのみ可能です。 short 型は 1 バイトまたは 2 バイト境界に沿った位置合わせのみ可能で、int 型は 1、2、または 4 バイト境界に沿った位置合わせのみ可能です。

16 バイト・ポインターは、すべて 16 バイト境界に沿って位置合わせされます。 **_Packed**、PACKSTRUCT、および #pragma pack でこれを変更することはできません。 8 バイトのテラスペース・ポインターはどのような位置合わせでも可能ですが、8 バイトの位置合わせをお勧めします。

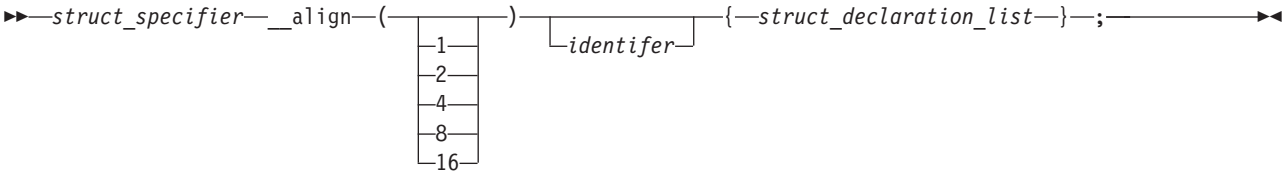
関連演算子および指定子

__align 指定子: **__align** 指定子を使用することで、データ項目または ILE C/C++ 集合体 (ILE C の構造体または共用体、ILE C++ のクラスなど) の位置合わせを指定することができます。ただし、**__align** はその集合体全体の位置合わせにのみ適用されるもので、集合体の中のメンバーの位置合わせには影響しません。また、集合体の特定のメンバー (16 バイト・ポインターなど) に制約事項があるため、メモリー内の集合体が **__align** で指定された境界で位置合わせされるとは限りません。例えば、16 バイト・ポインターはすべて 16 バイト境界で位置合わせされる必要があるため、16 バイト・ポインターが唯一のメンバーである集合体に、16 バイト位置合わせ以外の位置合わせを指定することはできません。

__align 構文



struct_specifier 構文



以下の一部の例に示されているように、データ項目の宣言時または定義時に **__align** 指定子を使用して、明示的に位置合わせを指定することも可能です。

__align 指定子:

- 第 1 レベルの変数および集合体定義の宣言でのみ使用できます。パラメーターおよび自動は無視されます。
- 集合体定義内の個々のエレメントに使用することはできませんが、別の集合体定義内でネストされている集合体定義には使用できます。
- 以下の状態には使用できません。
  - 集合体定義内の個々のエレメント。
  - 不完全な型で宣言されている変数。
  - 定義なしで宣言されている集合体。
  - 配列の個々のエレメント。
  - その他の型 (関数型や enum など) の宣言または定義。
  - 変数の位置合わせのサイズが型の位置合わせのサイズよりも小さい場合。

_Packed 指定子: **_Packed** は、構造体、共用体、および C++ ではクラス定義にも関連付けることができます。C++ では、**_Packed** は typedef で指定する必要があります。これの効果は #pragma pack(1) と同じです。以下のコードは、**_Packed** の有効な使用例および無効な使用例を示しています。これらの例で使用されているキーワード *struct*、*union*、および *class* は、相互に交換して使用することができます。

```
typedef _Packed class SomeClass { /* ... */ } MyClass;  // OK
typedef _Packed union AnotherClass {} PUnion;           // OK
typedef _Packed struct {} PAnonStruct;                  // Invalid, struct must be named
Class Stack { /* ... */ };
_Packed Stack someObject;                               // Invalid, specifier _Packed must be
                                                        // associated with a typedef in C++
_Packed struct SomeStruct { };                          // OK for C, invalid for C++
_Packed union SomeUnion { };                            // OK for C, invalid for C++
```

__alignof 演算子:

```
unary-expression:
    __alignof unary-expression
    __alignof ( type-name )
```

__alignof 演算子は、そのオペランドの位置合わせを戻します。オペランドは、式、または括弧付きの型の名前とすることができます。オペランドの位置合わせは、IBM i 位置合わせ規則に従って決定されます。ただし、関数型や不完全な型を持つ式、型などの括弧付きの名前、またはビット・フィールド・メンバーを指定する式にこれを適用することはできません。この演算子の結果の型は **size_t** です。

例

以下の例では、単語 union または class を、struct の代わりに使用することができます。

1. **#pragma pack スタックのポッピング**

   #pragma pack (pop)、#pragma pack (reset)、または #pragma pack() を指定すると、スタックが 1 つずつポップし、その位置合わせの要求が、その前の #pragma pack より前にアクティブであった状態にリセットされます。例えば、

   ```
   // Default alignment requirements used
   .
   .
   #pragma pack (4)
    struct A {  };
   #pragma pack (2)
    struct B {  };
    struct C {  };
   #pragma pack (reset)
    struct D {  };
   #pragma pack ()
    struct E {  };
   #pragma pack (pop)
    struct F {  };
   ```

   `struct A` がマッピングされると、そのメンバーは #pragma pack(4) に従って位置合わせされます。`struct B` および `struct C` がマッピングされると、そのメンバーは pragma pack(2) に従って位置合わせされます。

   #pragma pack (reset) は、#pragma pack(2) によって指定された位置合わせ要求をポップさせ、その位置合わせ要求を #pragma pack(4) によって指定されたとおりにリセットします。

   `struct D` がマッピングされると、そのメンバーは pragma pack(4) に従って位置合わせされます。#pragma pack () は、#pragma pack(4) によって指定された位置合わせ要求をポップさせ、その位置合わせ要求を、そのファイルの開始時に使用されていたデフォルト値にリセットします。

struct E がマッピングされると、そのメンバーは、ファイルの開始時にアクティブであった (コマンド行で指定された) デフォルトの位置合わせ要求の指定どおりに位置合わせされます。

#pragma pack (pop) の効果は、その前の #pragma pack ディレクティブの効果と同じで、パック・スタックの先頭の値をポップさせます。ただし、PACKSTRUCT コンパイラー・オプションで指定されているように、デフォルトのパック値をパック・スタックから除去することはできません。そのデフォルト値は、struct F の位置合わせに使用されます。

2. **__align & #pragma pack**

```
__align(16) struct S {int i;};          /* sizeof(struct S) == 16   */
struct S1 {struct S s; int a;};         /* sizeof(struct S1) == 32  */
#pragma pack(2)
struct S2 {struct S s; int a;} s2;      /* sizeof(struct S2) == 32  */
                                        /* offsetof(S2, s1) == 0    */
                                        /* offsetof(S2, a) == 16    */
```

3. **#pragma pack**

この例では、データ型はデフォルトでは #pragma pack (8) の指定よりも小さな境界に沿ってパックされるため、より小さな境界 (alignof(S2) = 4) に沿って位置合わせされます。

```
#pragma pack(2)
struct S {                      /* sizeof(struct S) == 48  */
   char a;                      /* offsetof(S, a) == 0     */
   int* b;                      /* offsetof(S, b) == 16    */
   char c;                      /* offsetof(S, c) == 32    */
   short d;                     /* offsetof(S, d) == 34    */
} S;                            /* alignof(S) == 16        */

struct S1 {                     /* sizeof(struct S1) == 10 */
   char a;                      /* offsetof(S1, a) == 0    */
   int b;                       /* offsetof(S1, b) == 2    */
   char c;                      /* offsetof(S1, c) == 6    */
   short d;                     /* offsetof(S1, d) == 8    */
} S1;                           /* alignof(S1) == 2        */

#pragma pack(8)
struct S2 {                     /* sizeof(struct S2) == 12 */
   char a;                      /* offsetof(S2, a) == 0    */
   int b;                       /* offsetof(S2, b) == 4    */
   char c;                      /* offsetof(S2, c) == 8    */
   short d;                     /* offsetof(S2, d) == 10   */
} S2;                           /* alignof(S2) == 4        */
```

4. **PACKSTRUCT コンパイラー・オプション**

PACK STRUCTURE を 2 に設定して以下をコンパイルする場合:

```
struct S1 {                     /* sizeof(struct S1) == 10 */
   char a;                      /* offsetof(S1, a) == 0    */
   int b;                       /* offsetof(S1, b) == 2    */
   char c;                      /* offsetof(S1, c) == 6    */
   short d;                     /* offsetof(S1, d) == 8    */
} S1;                           /* alignof(S1) == 2        */
```

5. **#pragma pack**

PACK STRUCTURE を 4 に設定して以下をコンパイルする場合:

```
#pragma pack(1)
struct A {              // this structure is packed along 1-byte boundaries
   char a1;
   int a2;
};
```

```
#pragma pack(2)
struct B {              // this class is packed along 2-byte boundaries
   int b1;
   float b2;
   float b3;
};

#pragma pack(pop)       // this brings pack back to 1-byte boundaries
struct C {
   int c1;
   char c2;
   short c3;
};

#pragma pack(pop)       // this brings pack back to the compile option,
struct D {              // 4-byte boundaries
   int d1;
   char d2;
};
```

6. **__align**

```
int __align(16) varA;   /* varA is aligned on a 16-byte boundary */
```

7. **_Packed**

```
struct A {                      /* sizeof(A) == 24      */
   int a;                       /* offsetof(A, a) == 0  */
   long long b;                 /* offsetof(A, b) == 8  */
   short c;                     /* offsetof(A, c) == 16 */
   char d;                      /* offsetof(A, d) == 18 */
};

_Packed struct B {              /* sizeof(B) == 15    */
   int a;                       /* offsetof(B, a) == 0  */
   long long b;                 /* offsetof(B, b) == 4  */
   short c;                     /* offsetof(B, c) == 12 */
   char d;                      /* offsetof(B, d) == 14 */
};
```

struct A のレイアウト (* = 埋め込み):

```
|a|a|a|a|*|*|*|*|b|b|b|b|b|b|b|b|c|c|d|*|*|*|*|*|
```

struct B のレイアウト (* = 埋め込み):

```
|a|a|a|a|b|b|b|b|b|b|b|b|c|c|d|
```

8. **__alignof**

```
struct A {
   char a;
   short b;
};

struct B {
   char a;
   long b;
} varb;

int var;
```

上のコード例では、次のようになります。

- __alignof(struct A) = 2
- __alignof(struct B) = 4
- __alignof(var) = 4

- __alignof(varb.a) = 1

```
__align(16) struct A {
   int a;
   int b;
};

#pragma pack(1)
struct B {
   long a;
   long b;
};

struct C {
   struct {
      short a;
      int b;
   } varb;
} var;
```

上のコード例では、次のようになります。

- __alignof(struct A) = 16
- __alignof(struct B) = 4
- __alignof(var) = 4
- __alignof(var.varb.a) = 4

page

C

page 構文

説明

生成されるソース・リストの *n* ページをスキップします。 *n* が指定されていない場合は、次のページが開始されます。

pagesize

C

pagesize 構文

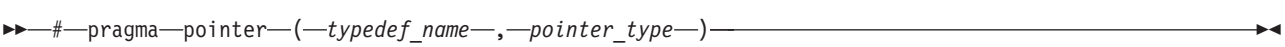
説明

生成されるソース・リストの 1 ページ当たりの行数を *n* にセットします。 pagesize プラグマは、オプション・リスト・ページ (Prolog とも呼ばれます) には影響しない場合があります。

pointer

C C++

pointer 構文



説明

IBM i ポインター型を使用できるようにします。

- スペース・ポインター
- システム・ポインター
- 呼び出しポインター
- ラベル・ポインター
- サスペンド・ポインター
- オープン・ポインター

#pragma pointer ディレクティブで指定された型定義を使用して宣言される変数には、そのディレクティブ内の typedef_name と関連付けられるポインター型があります。<pointer.h> ヘッダー・ファイルには、これらのポインター型の型定義および #pragma ディレクティブが含まれています。このヘッダー・ファイルをソース・コードに組み込むことで、これらの型定義を、これらの型のポインター変数の宣言に直接使用することができます。

パラメーター

pointer_type
以下のいずれかです。

SPCPTR
スペース・ポインター

OPENPTR
オープン・ポインター

SYSPTR
システム・ポインター

INVPTR
呼び出しポインター

LBLPTR
ラベル・コード・ポインター

SUSPENDPTR
サスペンド・ポインター

使用に関する注意

以下のエラーのいずれかが発生すると、コンパイラーは警告を出し、#pragma pointer ディレクティブを無視します。

- ディレクティブで指定されているポインター型が、SPCPTR、 SYSPTR、 INVPTR、 LBLPTR、 SUSPENDPTR、または OPENPTR のいずれでもない。

- 指定された型定義が #pragma pointer ディレクティブより前に宣言されていない。
- ディレクティブの最初のパラメーターとして指定されている ID が、型定義ではない。
- 指定された型定義がボイド・ポインターの型定義ではない。
- 指定された型定義が、#pragma pointer ディレクティブより前の宣言で使用されている。

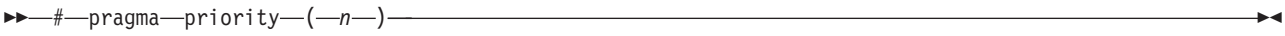
指定される型定義は、ファイル・スコープで定義する必要があります。

IBM i ポインターの使用について詳しくは、*ILE C/C++ プログラマーの手引き* を参照してください。

priority

C++

priority 構文



説明

#pragma priority ディレクティブは、実行時に静的オブジェクトを初期化する際の順序を指定します。

値 *n* は、**INT_MIN** から **INT_MAX** の範囲内の整数リテラルです。デフォルト値は 0 です。負の値は高い優先順位を示し、正の値は低い優先順位を示します。

最初の 1024 個の優先順位 (**INT_MIN** から **INT_MIN** + 1023) は、コンパイラーとそのライブラリーで使用するために予約されます。 #pragma priority は、ソース・ファイルの任意の場所に、何度でも置くことができます。ただし、各プラグマの優先順位は、以前のプラグマの優先順位より大きい値にする必要があります。これは、実行時の静的初期化が宣言どおりの順序で行われるようにするために必要なことです。

例

```
//File one called First.C

#pragma priority (1000)
class A { public: int a; A() {return;} } a;
#pragma priority (3000)
class C { public: int c; C() {return;} } c;
class B { public: int b; B() {return;} };
extern B b;
main()
{
    a.a=0;
    b.b=0;
    c.c=0;
}

//File two called Second.C
#pragma priority (2000)
class B { public: int b; B() {return;} } b;
```

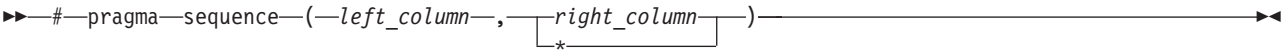
この例では、実行時の静的初期化の実行シーケンスは以下のようになります。

1. ファイル First.C の優先順位 1000 の静的初期化
2. ファイル Second.C の優先順位 2000 の静的初期化
3. ファイル First.C の優先順位 3000 の静的初期化

sequence

C

sequence 構文



説明

シーケンス番号が入るべき入力レコードの桁を指定します。桁の設定は、その設定が指定されているソース設定にのみ適用され、そのメンバー内のインクルード・ディレクティブで指定されているソース・メンバーには無効です。

パラメーター

left column
: ゼロより大きく、32754 より小さい必要があります。 *left column* は *right column* より小さい必要があります。

right column
: ゼロより大きく、32754 より小さい必要があります。 *right column* は、*left column* 以上である必要があります。 *right column* の値として指定されるアスタリスク (*) は、そのシーケンス番号が *left column* と入力レコードの最後までの間に入っていることを示します。

使用に関する注意

#pragma sequence ディレクティブは、そのディレクティブの次の行から有効となります。別の #pragma sequence ディレクティブまたは #pragma nosequence ディレクティブが検出されるか、そのソース・メンバーが終わるまで有効です。

#pragma margins ディレクティブと #pragma sequence ディレクティブは、併用することができます。この 2 つの #pragma ディレクティブが同じ桁を予約する場合は、#pragma sequence ディレクティブが優先され、その桁はシーケンス番号用に予約されます。

例えば、#pragma margins ディレクティブで 1 および 20 のマージンが指定され、#pragma sequence ディレクティブで 15 から 25 の桁がシーケンス番号用に指定された場合、有効なマージンは 1 および 14 となり、シーケンス番号用に予約される桁は 15 から 25 までとなります。

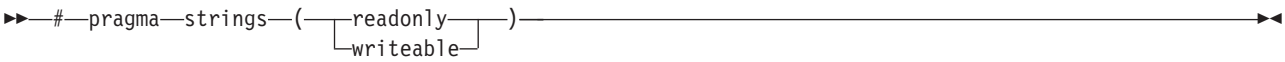
指定されたマージンがサポートされている範囲内にないか、マージンに数値以外の値が含まれている場合は、コンパイル中に警告メッセージが出されて、そのディレクティブは無視されます。

5-37 ページの『nosequence』および 5-32 ページの『margins』プラグマも参照してください。

strings

C C++

strings 構文



説明

コンパイラーが読み取り専用メモリーの中にストリングを配置できること、あるいは、書き込み可能メモリーの中にストリングを配置しなければならないことを指定します。ストリングはデフォルトでは書き込み可能です。このプラグマは、ファイル内の C または C++ コードの前にある必要があります。

注: このプラグマは、*STRDONLY コンパイラー・オプションをオーバーライドします。

weak

C++

weak 構文

説明

弱いグローバル・シンボルであるコンパイラーの ID を識別します。

パラメーター

identifier
: 弱いグローバル・シンボルになると考えられる ID の名前を指定します。

identifier2
: *identifer2* が指定されると、*identifier* が弱いグローバル・シンボルとみなされ、その値は *identifier2* と同じになります。このプラグマを有効にするには、*identifier2* を同じコンパイル単位内で定義する必要があります。

このプラグマは、プログラム内の任意の場所に置かれ、指定された ID を弱いグローバル・シンボルとして識別します。*identifier* は定義することはできませんが、宣言される場合はあります。 *identifier* が宣言され、*identifier2* が指定される場合、*identifier2* の型と互換性がある型にする必要があります。

例

```
#pragma weak func1 = func2
```

制御言語コマンド

このセクションでは、ILE C/C++ コンパイラーで使用される制御言語 (CL) コマンドの概要を説明します。 構文図やパラメーター記述テーブルがあります。

この表は、IBM i コンパイラーで使用される CL コマンドについて説明するものです。

表 6-1. 制御言語コマンド

アクション	コマンド	説明
C モジュールの作成	CRTCMOD	指定されたソースに基づいて、モジュール・オブジェクト (*MODULE) を作成します。
C++ モジュールの作成	CRTCPPMOD	
バインド C プログラムの作成	CRTBNDC	指定されたソースに基づいて、プログラム・オブジェクト (*PGM) を作成します。
バインド C++ プログラムの作成	CRTBNDCPP	

CL コマンドとそのパラメーターは、大文字と小文字のどちらで入力しても構いません。このリファレンスでは、すべて大文字で表記しています。例えば、以下のようになります。

```
CRTCPPMOD MODULE(ABC/HELLO) SRCSTMF('/home/usr/hello.C') OPTIMIZE(40)
```

ILE C/C++ 言語ステートメントは、示されているとおりに正確に入力する必要があります。例えば、fopen、_Ropen のようになります。これは ILE C/C++ コンパイラーが大/小文字を区別するためです。

変数は、*file-name*、*characters*、および *string* などのように、小文字のイタリック体で表されます。これらはユーザーが指定する名前または値を表しています。

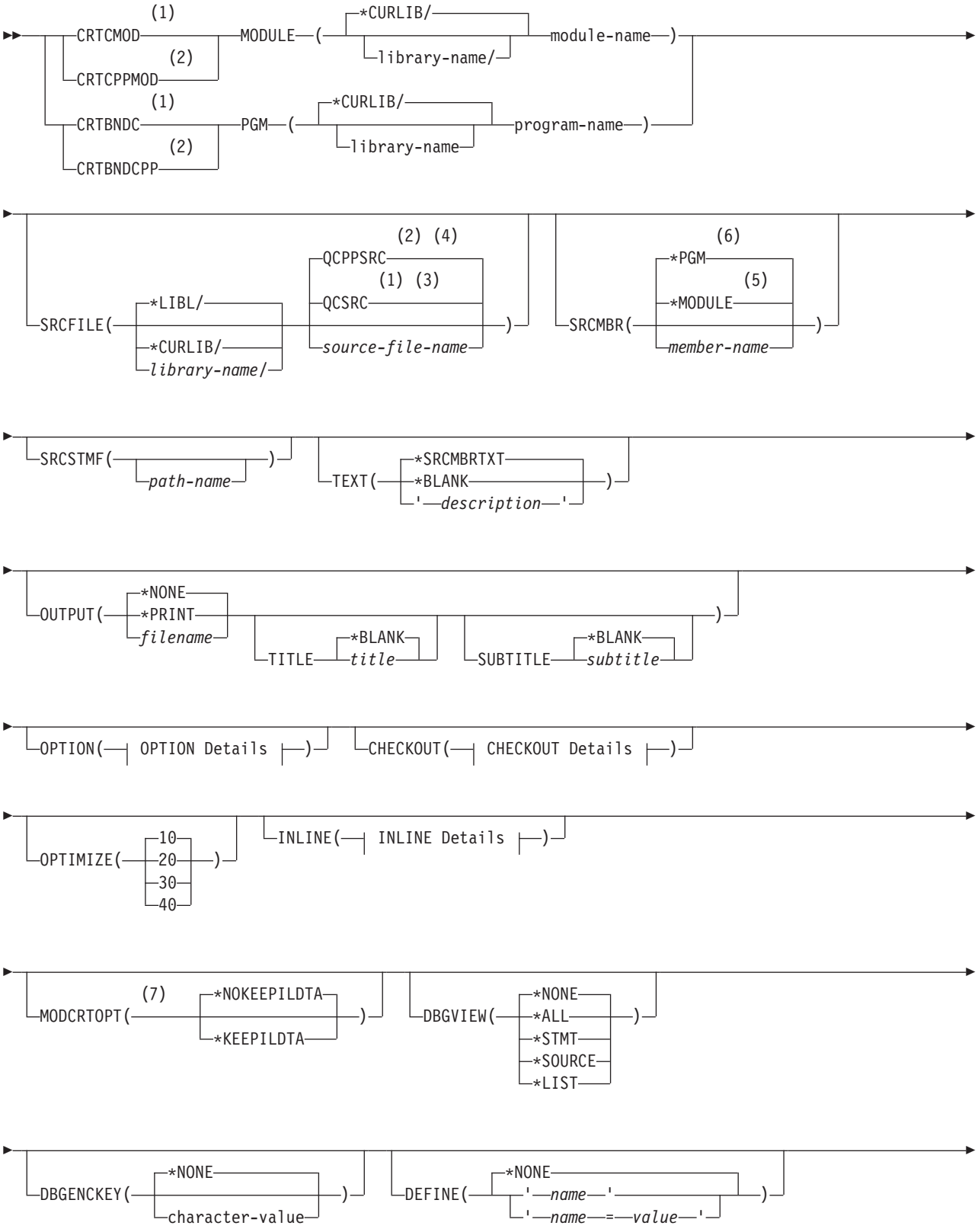
言語ステートメントには、句読記号、括弧、算術演算子、またはその他の記号が含まれている場合があります。これらは、その構文図に示されているとおり、正確に入力する必要があります。

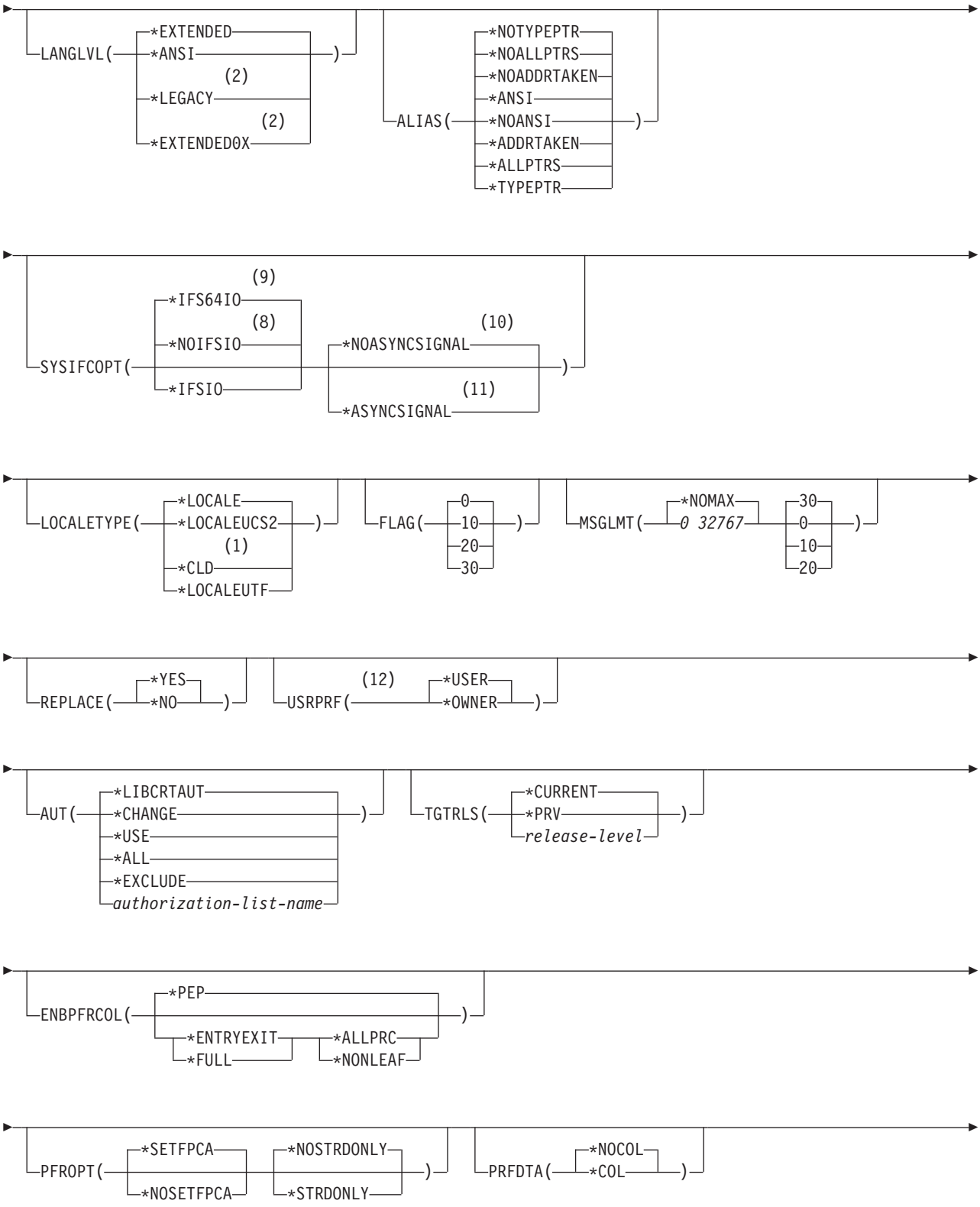
Qshell コマンド行環境からコンパイラーとそのオプションを呼び出すこともできます。Qshell コマンドおよびオプションの書式について詳しくは、7-1 ページの『ixlc コマンドを使用した C/C++ コンパイラーの起動』を参照してください。

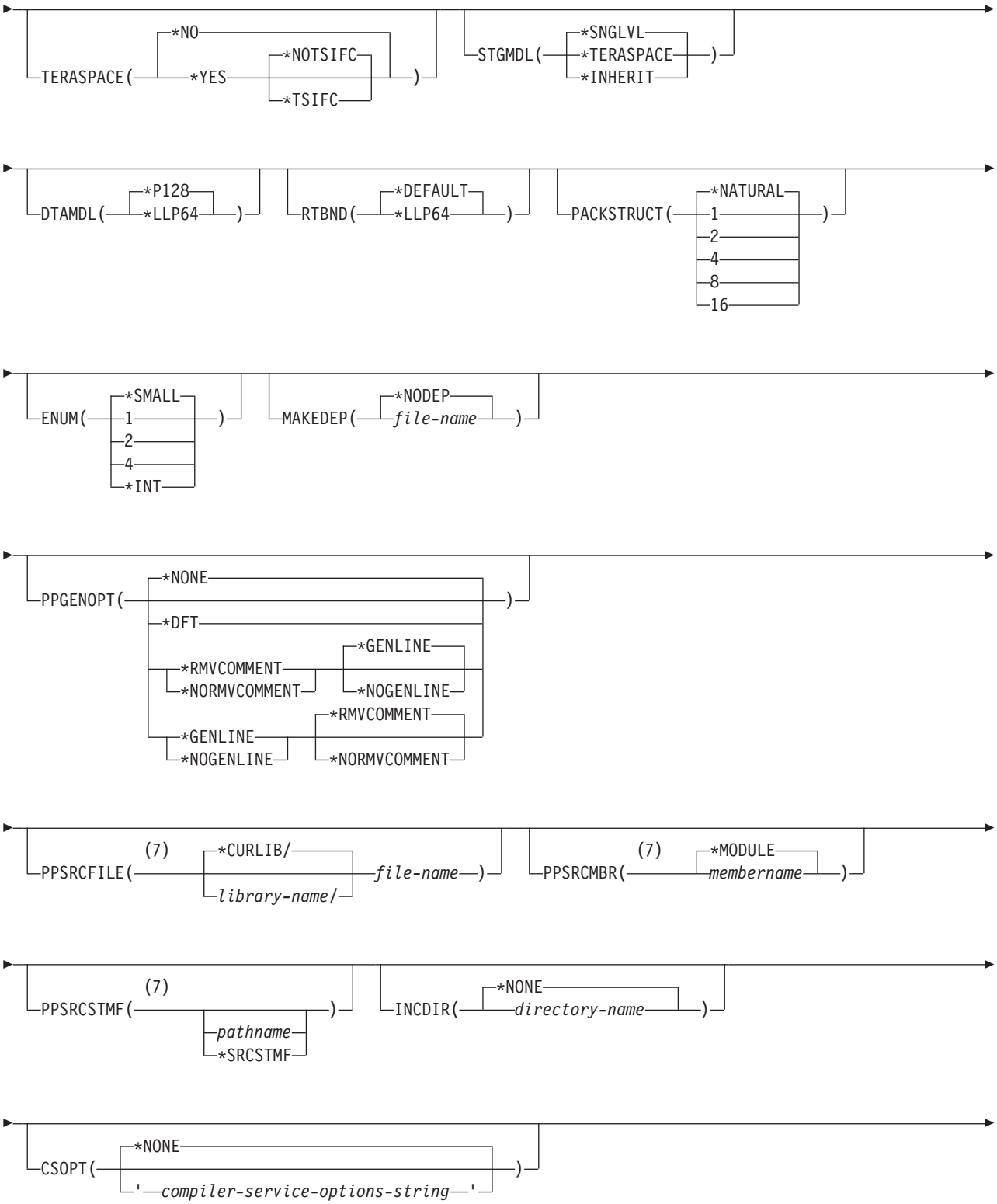
制御言語コマンド構文

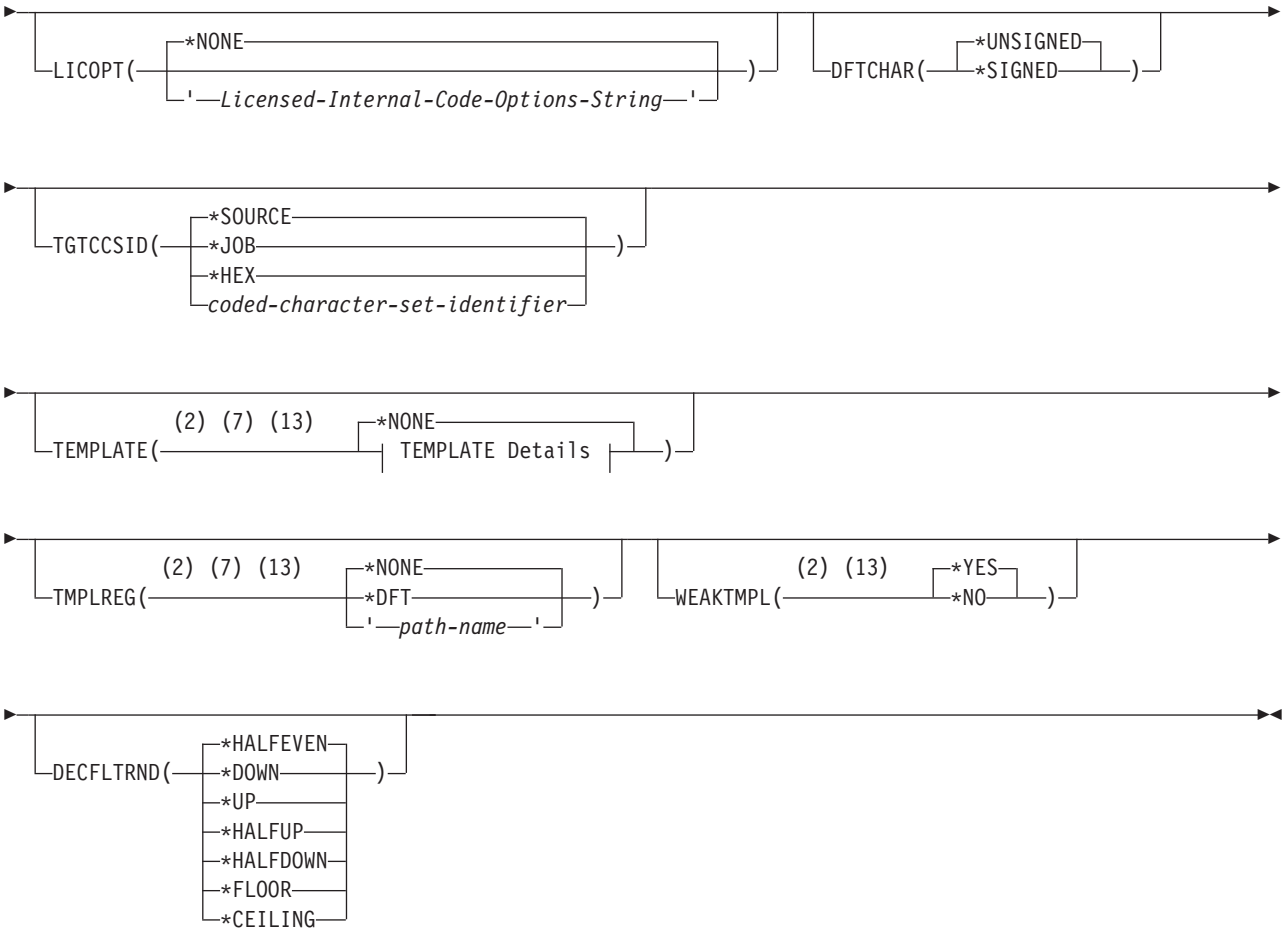
このセクションの構文図には、CRTCMOD、CRTCPPMOD、CRTBNDC、および CRTBNDCPP コマンドのすべてのパラメーターとオプション、および各オプションのデフォルト値が示されています。ほとんどの場合、キーワードはどのコマンドでも同じです。異なる場合は、注意書きが付いています。各オプションについての詳細は、6-7 ページの『制御言語コマンド・オプション』を参照してください。

構文図

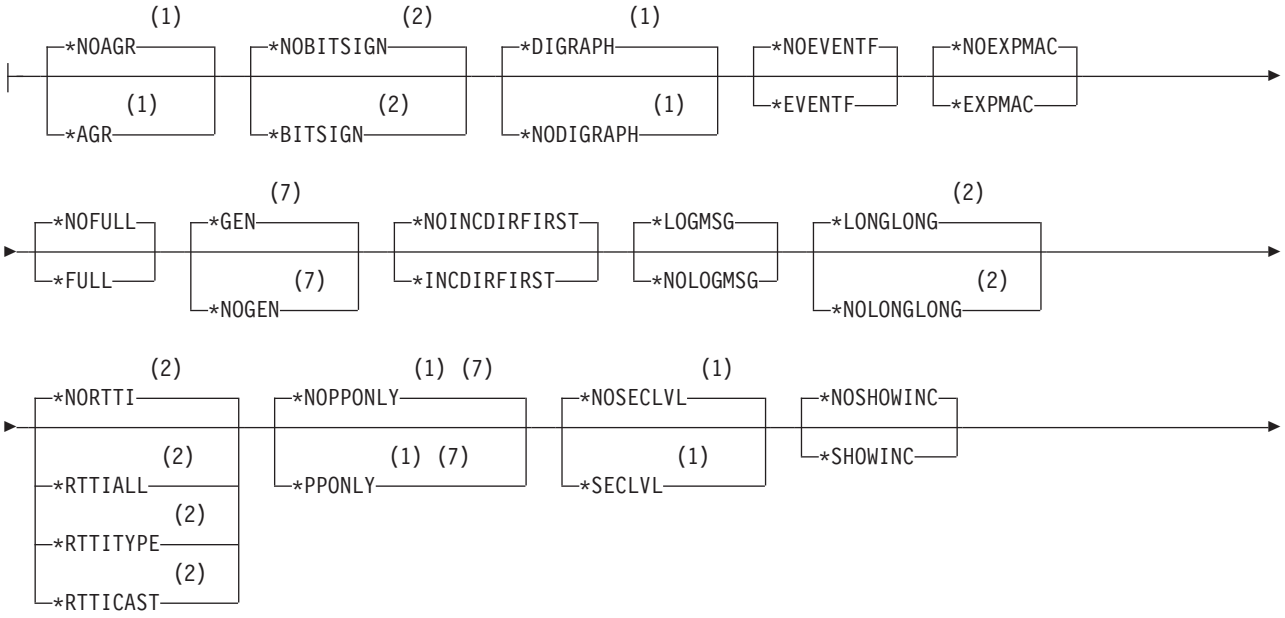

CRTCMOD
(1)
CRTCPPMOD
(2)
MODULE(
*CURLIB/
library-name/
module-name)
CRTBNDC
(1)
CRTBNDCPP
(2)
PGM(
*CURLIB/
library-name
program-name)
SRCFILE(
*LIBL/
*CURLIB/
library-name/
QCPPSRC
(2) (4)
QCSRC
(1) (3)
source-file-name)
SRCMBR(
*PGM
*MODULE
(5)
member-name
(6))
SRCSTMF(
path-name)
TEXT(
*SRCMBRTXT
*BLANK
'description')
OUTPUT(
*NONE
*PRINT
filename
TITLE
*BLANK
title
SUBTITLE
*BLANK
subtitle)
OPTION(
OPTION Details)
CHECKOUT(
CHECKOUT Details)
OPTIMIZE(
10
20
30
40)
INLINE(
INLINE Details)
MODCRTOPT(
(7)
*NOKEEPILDTA
*KEEPILDTA)
DBGVIEW(
*NONE
*ALL
*STMT
*SOURCE
*LIST)
DBGENCKEY(
*NONE
character-value)
DEFINE(
*NONE
'name'
'name=value')

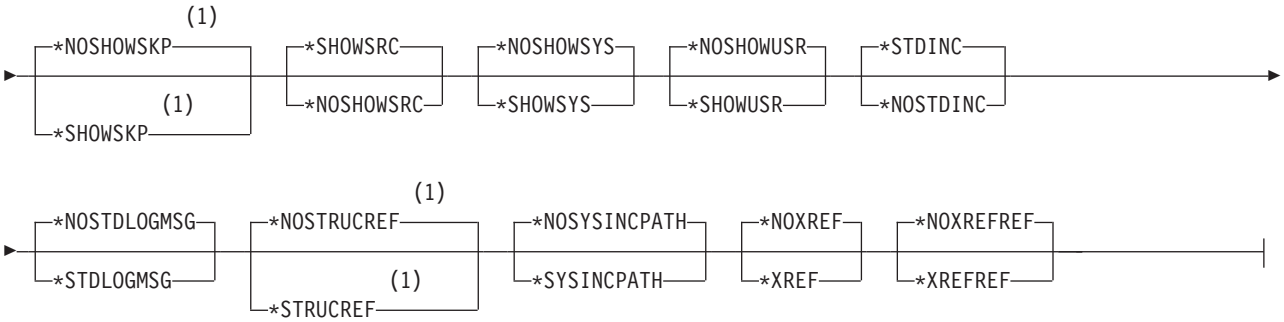

LANGLVL(
*EXTENDED
*ANSI
(2)
*LEGACY
(2)
*EXTENDED0X
)
ALIAS(
*NOTYPEPTR
*NOALLPTRS
*NOADDRTAKEN
*ANSI
*NOANSI
*ADDRTAKEN
*ALLPTRS
*TYPEPTR
)
SYSIFCOPT(
(9)
*IFS64IO
(8)
*NOIFSIO
*IFSIO
(10)
*NOASYNCSIGNAL
(11)
*ASYNCSIGNAL
)
LOCALETYPE(
*LOCALE
*LOCALEUCS2
(1)
*CLD
*LOCALEUTF
)
FLAG(
0
10
20
30
)
MSGLMT(
*NOMAX
0 32767
30
0
10
20
)
REPLACE(
*YES
*NO
)
(12)
USRPRF(
*USER
*OWNER
)
AUT(
*LIBCRTAUT
*CHANGE
*USE
*ALL
*EXCLUDE
authorization-list-name
)
TGTRLS(
*CURRENT
*PRV
release-level
)
ENBPFRCOL(
*PEP
*ENTRYEXIT
*FULL
*ALLPRC
*NONLEAF
)
PFROPT(
*SETFPCA
*NOSETFPCA
*NOSTRDONLY
*STRDONLY
)
PRFDTA(
*NOCOL
*COL
)


TERASPACE(*NO *YES *NOTSIFC *TSIFC)
STGMDL(*SNGLVL *TERASPACE *INHERIT)
DTAMDL(*P128 *LLP64)
RTBND(*DEFAULT *LLP64)
PACKSTRUCT(*NATURAL 1 2 4 8 16)
ENUM(*SMALL 1 2 4 *INT)
MAKEDEP(*NODEP file-name)
PPGENOPT(*NONE *DFT *RMVCOMMENT *NORMVCOMMENT *GENLINE *NOGENLINE *GENLINE *NOGENLINE *RMVCOMMENT *NORMVCOMMENT)
PPSRCFILE((7) *CURLIB/ library-name/ file-name)
PPSRCMBR((7) *MODULE membername)
PPSRCSTMF((7) pathname *SRCSTMF)
INCDIR(*NONE directory-name)
CSOPT(*NONE '-compiler-service-options-string-')

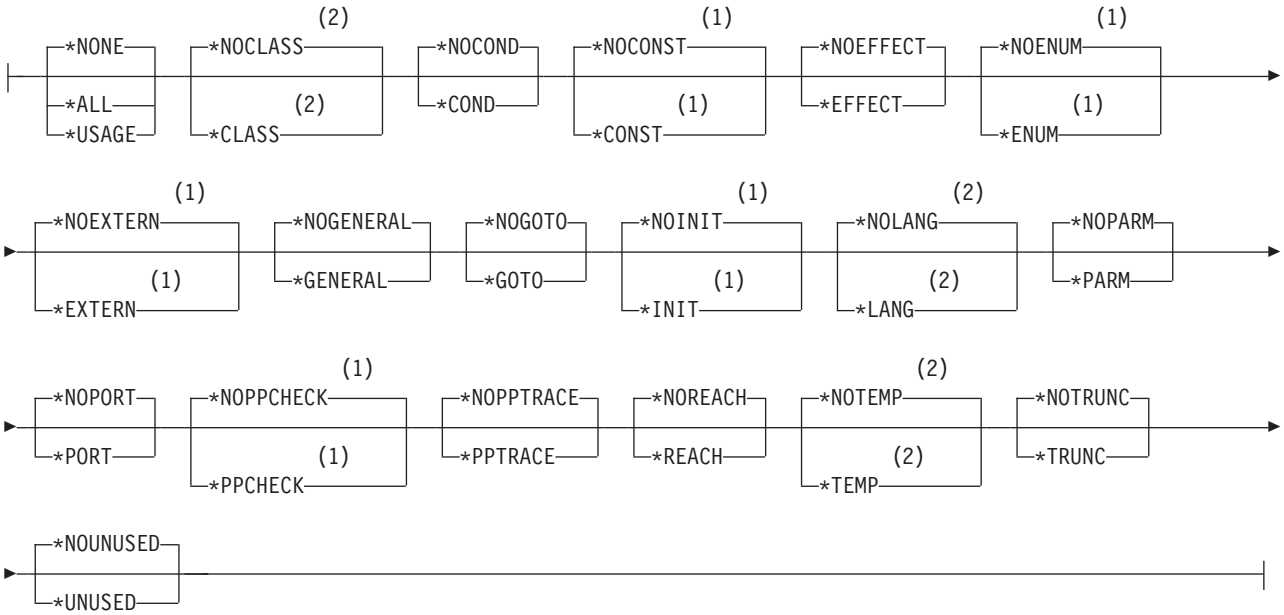

LICOPT(
*NONE
'Licensed-Internal-Code-Options-String'
)
DFTCHAR(
*UNSIGNED
*SIGNED
)
TGTCCSID(
*SOURCE
*JOB
*HEX
coded-character-set-identifier
)
TEMPLATE(
(2) (7) (13)
*NONE
TEMPLATE Details
)
TMPLREG(
(2) (7) (13)
*NONE
*DFT
'path-name'
)
WEAKTMPL(
(2) (13)
*YES
*NO
)
DECFLTRND(
*HALFEVEN
*DOWN
*UP
*HALFUP
*HALFDOWN
*FLOOR
*CEILING
)

OPTION の詳細:

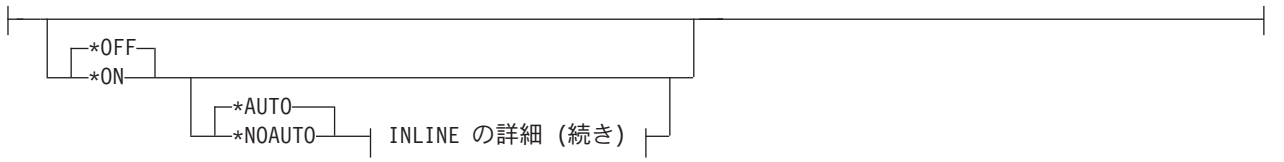

(1)
*NOAGR
(1)
*AGR
(2)
*NOBITSIGN
(2)
*BITSIGN
(1)
*DIGRAPH
(1)
*NODIGRAPH
*NOEVENTF
*EVENTF
*NOEXPMAC
*EXPMAC
*NOFULL
*FULL
(7)
*GEN
(7)
*NOGEN
*NOINCDIRFIRST
*INCDIRFIRST
*LOGMSG
*NOLOGMSG
(2)
*LONGLONG
(2)
*NOLONGLONG
(2)
*NORTTI
(2)
*RTTIALL
(2)
*RTTITYPE
(2)
*RTTICAST
(1) (7)
*NOPPONLY
(1) (7)
*PPONLY
(1)
*NOSECLVL
(1)
*SECLVL
*NOSHOWINC
*SHOWINC



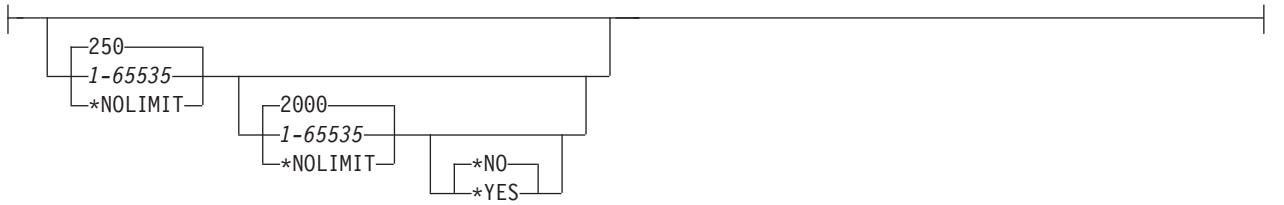
CHECKOUT の詳細:



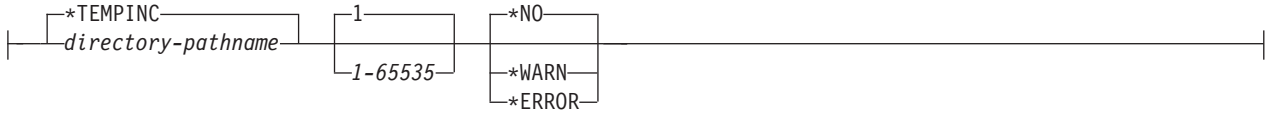
INLINE の詳細:



INLINE の詳細 (続き):



TEMPLATE の詳細:



注:

1 C コンパイラーのみ

2 C++ コンパイラーのみ

3 C コンパイラーのデフォルト設定

4 C++ コンパイラーのデフォルト設定

5 「モジュールの作成」コマンドのみ

6 「バインド済みプログラムの作成」コマンドのみ

7 「モジュールの作成」コマンドのみ

8 C コンパイラーのデフォルト設定

9 C++ コンパイラーのデフォルト設定

10 C コンパイラーのみ

11 C コンパイラーのみ

12 「バインド済みプログラムの作成」コマンドのみ

13 統合ファイル・システム (IFS) を使用する場合にのみ適用可能

制御言語コマンド・オプション

以下のページでは、CRTCMOD、CRTCPPMOD、CRTBNDC、および CRTBNDCPP の各コマンドのキーワードについて説明しています。ほとんどの場合、キーワードはどのコマンドでも同じです。異なる場合は、注意書きが付いています。

説明の中で使用されている「オブジェクト」という用語には、以下の 2 つの意味があります。

- CRTCMOD コマンドまたは CRTCPPMOD コマンドを使用している場合、オブジェクトとはモジュール・オブジェクトを意味します。
- CRTBNDC コマンドまたは CRTBNDCPP コマンドを使用している場合、オブジェクトとはプログラム・オブジェクトを意味します。

MODULE

CRTCMOD および CRTCPPMOD コマンドでのみ有効。コンパイル済み ILE C または C++ モジュール・オブジェクトのモジュール名およびライブラリーを指定します。

MODULE 構文 :

***CURLIB**
これはデフォルトのライブラリー値です。オブジェクトは現行ライブラリーに保存されます。ジョブに現行ライブラリーがない場合は、QGPL が使用されます。

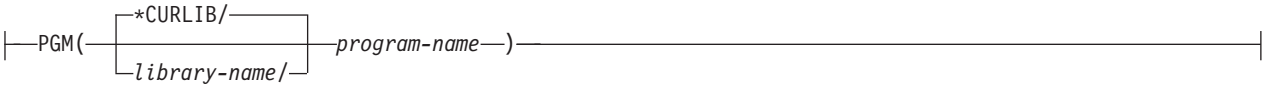
library-name
オブジェクトの保存先のライブラリーの名前を入力します。

module-name
モジュール・オブジェクトの名前を入力します。

PGM

CRTBNDC および CRTBNDCPP コマンドでのみ有効。コンパイル済み ILE C または C++ プログラム・オブジェクトにプログラム名とライブラリーを指定します。

PGM 構文 :



***CURLIB**
これはデフォルトのライブラリー値です。オブジェクトは現行ライブラリーに保存されます。ジョブに現行ライブラリーがない場合は、QGPL が使用されます。

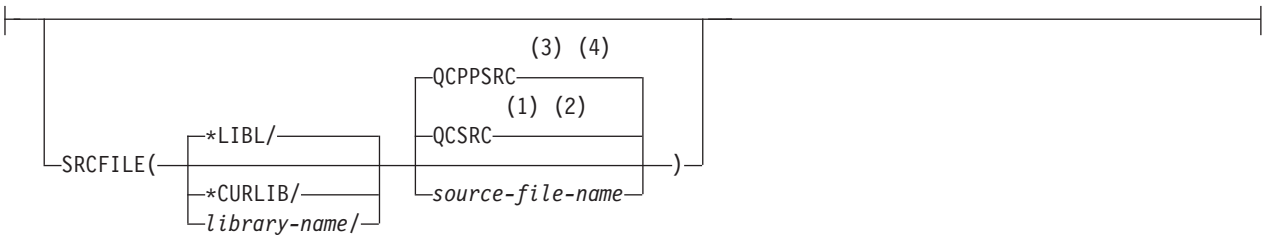
library-name
オブジェクトの保存先のライブラリーの名前を入力します。

program-name
プログラム・オブジェクトの名前を入力します。

SRCFILE

コンパイルする ILE C または C++ ソース・コードを含んでいるファイルのソース物理ファイル名とライブラリーを指定します。

SRCFILE 構文 :



注:

1 C コンパイラーのみ

2 C コンパイラーのデフォルト設定

3 C++ コンパイラーのみ

4 C++ コンパイラーのデフォルト設定

***LIBL**
これはデフォルトのライブラリー値です。ライブラリー・リストを検索して、ソース・ファイルを含んでいるライブラリーを見つけます。

***CURLIB**
現行ライブラリーでソース・ファイルを検索します。ジョブに現行ライブラリーがない場合は、QGPLが使用されます。

library-name
ソース・ファイルを含んでいるライブラリーの名前を入力します。

QCSRC C
コンパイルする ILE C ソース・コードを持つメンバーを含んでいるソース物理ファイルのデフォルト名。

QCPPSRC C++
コンパイルする ILE C++ ソース・コードを持つメンバーを含んでいるソース物理ファイルのデフォルト名。

source-file-name
ILE C または C++ ソース・コードを持つメンバーを含んでいるファイルの名前を入力します。

SRCMBR

ILE C または C++ ソース・コードを含むメンバーの名前を指定します。

SRCMBR 構文 :

注:

1 「モジュールの作成」コマンドのみ

2 「バインド済みプログラムの作成」コマンドのみ

***MODULE**
CRTCMOD または *CRTCPPMOD* コマンドでのみ有効。 MODULE パラメーターで提供されるモジュール名は、ソース・メンバー名として使用されます。これが、メンバー名が指定されていない場合のデフォルトです。

***PGM**
CRTBNDC または *CRTBNDCPP* コマンドでのみ有効。 PGM パラメーターで提供されるプログラム名は、ソース・メンバー名として使用されます。これが、メンバー名が指定されていない場合のデフォルトです。

member-name
ILE C または C++ ソース・コードを含んでいるメンバーの名前を入力します。

SRCSTMF

コンパイルする ILE C または C++ ソース・コードを含んでいるストリーム・ファイルのパス名を指定します。

SRCSTMF 構文 :

パス名は、絶対修飾名と相対修飾名のどちらを指定することもできます。絶対パス名は「/」で始まり、相対パス名は「/」以外の文字で始まります。絶対修飾されている場合、このパス名は完全です。相対修飾の場合は、このパス名の先頭にジョブの現行作業ディレクトリーを付加することによって、パス名が完全なものになります。

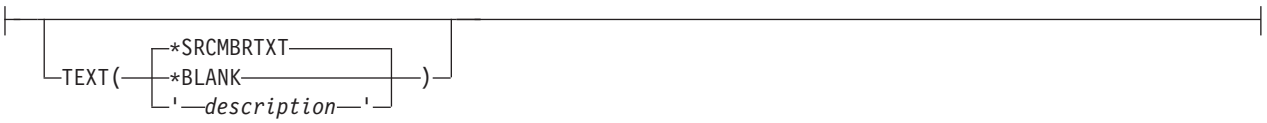
注:

1. SRCMBR パラメーターと SRCFILE パラメーターは、SRCSTMF パラメーターと一緒に指定することはできません。
2. SRCSTMF が指定されている場合、以下のコンパイラー・オプションは無視されます。
   - TEXT(*SRCMBRTXT)
   - OPTION(*STDINC)
   - OPTION(*SYSINCPATH)
3. 混合バイト環境では SRCSTMF パラメーターはサポートされていません。

TEXT

オブジェクトとその機能を説明するテキストを入力することができます。

TEXT 構文 :



***SRCMBRTXT**
デフォルト設定。コンパイル済みオブジェクトには、ソース・ファイル・メンバーに関連付けられているテキスト記述が使用されます。ソース・ファイルがインライン・ファイルまたはデバイス・ファイルの場合、このフィールドはブランクです。

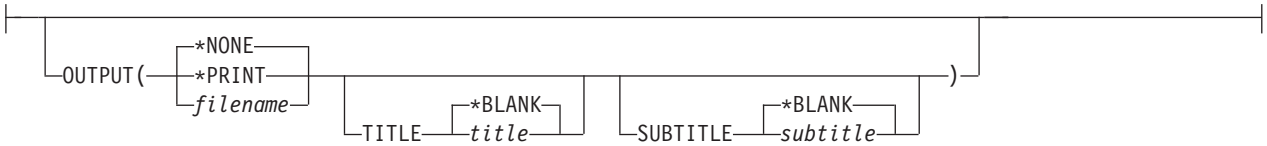
***BLANK**
テキストを表示しないことを指定します。

description
説明のためのテキストを 50 文字以下で入力し、そのテキストを単一引用符で囲みます。引用符は 50 文字のストリングの一部とは見なされません。CRTCMOD または CRTCPPMOD プロンプト画面が使用されている場合は、引用符が付けられます。

OUTPUT

コンパイラー・リストが必要であるかどうかを指定します。

OUTPUT 構文 :



***NONE**
コンパイラー・リストを生成しません。リストが不要の場合は、このデフォルトを使用するとコンパイル時のパフォーマンスが向上します。*NONE が指定されており、リスト関連オプションである *AGR、*EXPMAC、 *FULL、*SECLVL、 *SHOWINC、*SHOWSKP、*SHWSRC、*SHOWSYS、*SHOWUSR、*SHWSRC、*STRUCREF、*XREF、または *XREFREF が OPTION キーワードで指定されている場合、それらのオプションは無視されます。

***PRINT**
コンパイラー・リストをスプール・ファイルとして生成します。

WRKSPLF 内のスプール・ファイル名は、作成されるオブジェクト (プログラムまたはモジュール) と同じ名前です。

filename
コンパイラー・リストは、このストリングで指定されるファイル名で保存されます。

リスト名は統合ファイル・システム (IFS) フォーマット (例えば **/home/mylib/listing/hello.lst**) で指定する必要があります。ライブラリー *mylib* 内のデータ管理ファイル *listing* は、**/QSYS.LIB/***mylib*.lib/*listing*.file/hello.mbr として指定する必要があります。このストリングが「/」で始まっていない場合、このストリングは現行ディレクトリーまたはライブラリーのサブディレクトリーと見なされます。このファイルが存在しない場合は作成されます。

IFS リストを作成するには、データ権限 *WX が必要です。IFS を介してデータ管理ファイル・リストを作成するには、データ権限 *WX、およびオブジェクト権限 *OBJEXIST と *OBJALTER が必要です。

TITLE
コンパイラー・リストのタイトルを指定します。指定できる TITLE の値は以下のとおりです。

***BLANK**
タイトルは生成されません。

title
リストのタイトル・ストリングを指定します (最大 98 文字)。

SUBTITLE
コンパイラー・リストのサブタイトルを指定します。SUBTITLE に指定できる値は以下のとおりです。

***BLANK**
タイトルは生成されません。

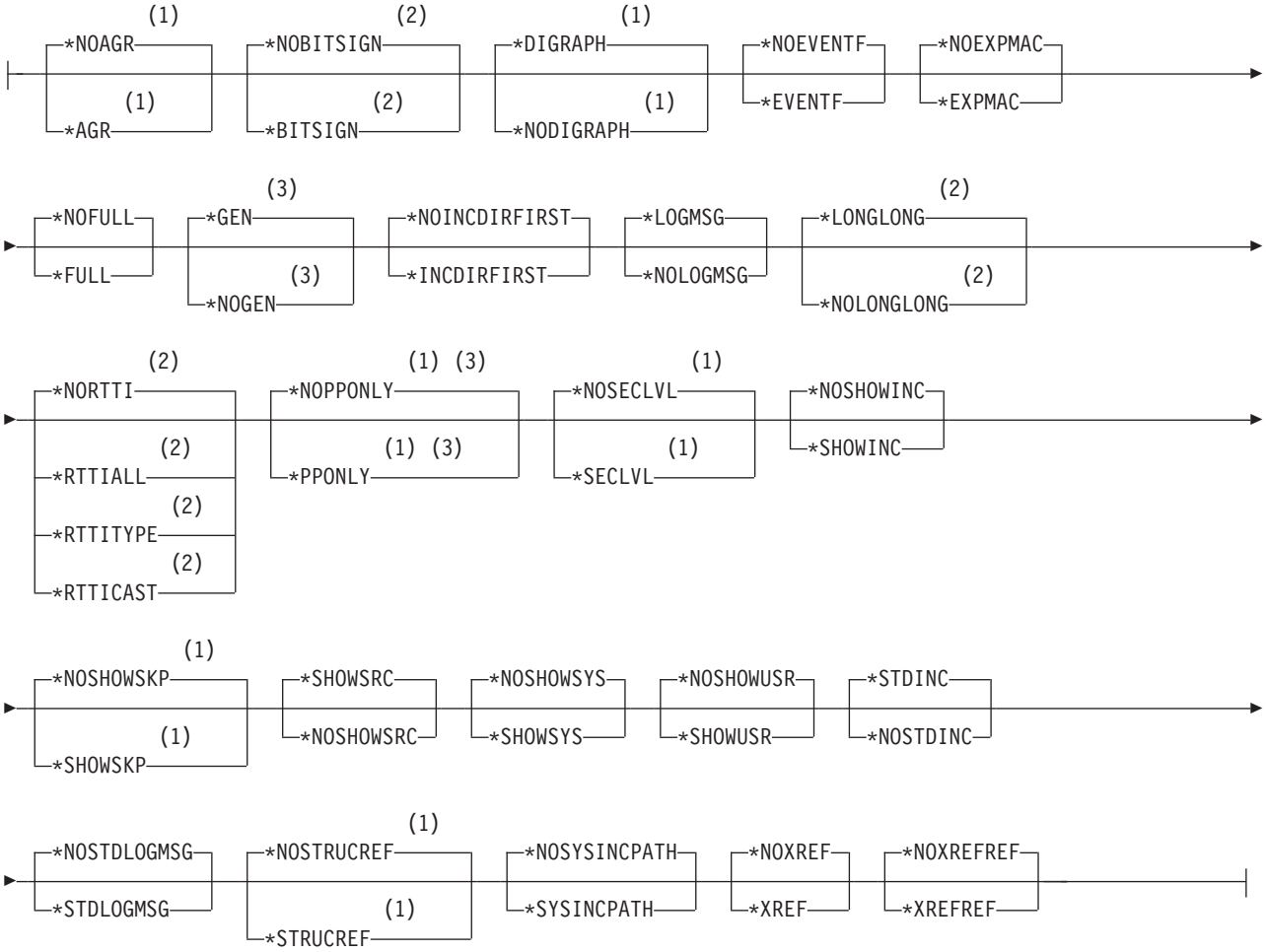
subtitle
リスト・ファイルのサブタイトル・ストリングを指定します (最大 98 文字)。

OPTION

ILE C または C++ ソース・コードのコンパイル時に使用するオプションを指定します。これらのオプションは、空白スペースで区切って、任意の順序で指定できます。オプションの説明で特に注記していない限り、1 つのオプションが複数回指定されている場合、または 2 つのオプションが競合している場合は、最後に指定されたオプションが使用されます。

OPTION 構文 :

OPTION の詳細:



注:

1 C コンパイラーのみ

2 C++ コンパイラーのみ

3 「モジュールの作成」コマンドのみ

指定可能なオプションは次のとおりです。

***NOAGR** C

C++ コンパイラーはこれを受け入れますが、無視します。 デフォルト設定。コンパイラー・リストに集合体構造マップを生成しません。

***AGR** C

C++ コンパイラーはこれを受け入れますが、無視します。 コンパイラー・リストに集合体構造マップを生成します。このマップはソース・プログラム内のすべての構造体のレイアウトを提供し、変数が埋め込まれているかいないかを示します。OUTPUT(*PRINT) を指定する必要があります。

*AGR オプションは *STRUCREF オプションをオーバーライドします。

***NOBITSIGN** C++

デフォルト設定。ビット・フィールドは符号なしです。

***BITSIGN** C++

ビット・フィールドは符号付きです。

***NODIGRAPH** C

デフォルト設定。コンパイラーは 2 文字表記の文字列を認識しません。この設定が有効で 2 文字表記が検出されると、構文エラーが発生する可能性があります。

***DIGRAPH** C

2 文字表記の文字列は、一部のキーボードにはない文字を表す場合に使用できます。文字またはストリング・リテラル内の 2 文字表記の文字列は、プリプロセッシングでは置き換えられません。

***NOEVENTF**

デフォルト設定。連携開発環境/400 (CODE/400) が使用するイベント・ファイルを作成しません。

***EVENTF**

連携開発環境/400 (CODE/400) が使用するイベント・ファイルを作成します。イベント・ファイルは、作成されたモジュールまたはプログラム・オブジェクトが保存されるライブラリー内の EVFEVENT ファイルのメンバーとして作成されます。 EVFEVENT ファイルが存在しない場合は、自動的に作成されます。イベント・ファイル・メンバー名は、作成されるオブジェクトの名前と同じです。通常、イベント・ファイルが作成されるのは、モジュールまたはプログラムを CODE/400 内から作成する場合です。 CODE/400 はこのファイルを使用して、CODE/400 エディターと統合されたエラー・フィードバックを提供します。

***NOEXPMAC**

デフォルト設定。リストのソース・セクション内、またはデバッグ・リスト・ビュー内にマクロを展開しません。

***EXPMAC**

リスト・ビューのソース・セクション内にすべてのマクロを展開します。このサブオプションが、DBGVIEW(*ALL) または DBGVIEW(*LIST) とともに指定されている場合、コンパイラーはエラー・メッセージを発行して、コンパイルを停止します。

***NOFULL**

デフォルト設定。リストまたはデバッグ・リスト・ビューに表示されないコンパイラー出力情報があります。

***FULL**

すべてのコンパイラー出力情報をリストまたはデバッグ・リスト・ビューに表示します。これを設定することにより、リスト関連のオプションがすべてオンになります。*FULL が指定されている場合は、個々のリスト・オプションをオフにできます。それには、*FULL オプションの後にそのオプションに

対して *NO 設定を指定します。このサブオプションが、DBGVIEW(*ALL) または DBGVIEW(*LIST) とともに指定されている場合、コンパイラーはエラー・メッセージを発行して、コンパイルを停止します。

***GEN**
CRTCMOD および *CRTCPPMOD* コマンドでのみ有効。 デフォルト設定。コンパイル・プロセスのすべてのフェーズが実行されます。

C OPTION(*PPONLY) を指定すると、PPGENOPT(*NONE) オプション設定および OPTION(*GEN) オプション設定がオーバーライドされます。代わりに、以下の設定が暗黙指定されます。

- データ管理ソース・ファイルの場合は PPGENOPT(*DFT) PPSRCFILE(QTEMP/QACZEXPAND) PPSRCMBR(*MODULE)
- IFS ソース・ファイルの場合は PPGENOPT(*DFT) PPSRCSTMF(*SRCSTMF)

***NOGEN**
CRTCMOD および *CRTCPPMOD* コマンドでのみ有効。 構文検査の後、コンパイルは停止します。オブジェクトは作成されません。

***NOINCDIRFIRST**
デフォルト設定。コンパイラーは、ユーザー・インクルード・ファイルを最初にルート・ソース・ディレクトリー内で検索し、次に INCDIR オプションで指定されたディレクトリー内で検索します。

***INCDIRFIRST**
コンパイラーはユーザー・インクルード・ファイルを以下のようにして検索します。

1. INCDIR パラメーターでディレクトリーを指定している場合、コンパイラーはそのディレクトリー内で *file_name* を検索します。
2. 複数のディレクトリーが指定されている場合、コンパイラーはコマンド行での出現順にそれらのディレクトリーを検索します。
3. 現在のルート・ソース・ファイルが存在しているディレクトリーを検索します。
4. INCLUDE 環境変数が定義されている場合、コンパイラーは INCLUDE パスでの出現順にディレクトリーを検索します。
5. *NOSTDINC コンパイラー・オプションが選択されていない場合は、デフォルトのインクルード・ディレクトリーである /QIBM/include を検索します。

***LOGMSG**
デフォルト設定。コンパイル・メッセージがジョブ・ログに書き込まれます。

このオプションと FLAG パラメーターを指定している場合は、FLAG パラメーターで指定されている重大度 (およびそれよりも高い重大度) を持つメッセージがジョブ・ログに書き込まれます。

このオプションを指定し、さらに MSGLMT パラメーターでメッセージの最大数を指定している場合は、指定された重大度のメッセージの数がジョブ・ログに書き込まれると、コンパイルは停止します。

***NOLOGMSG**
コンパイル・メッセージをジョブ・ログに書き込みません。

***LONGLONG** C++
デフォルト設定。コンパイラーは longlong データ型を認識して使用します。

***NOLONGLONG** C++
コンパイラーは longlong データ型を認識しません。

***NORTTI** C++

デフォルト設定。コンパイラーは実行時型情報 (RTTI) の typeid 演算子および dynamic_cast 演算子に必要な情報を生成しません。

***RTTIALL** C++

コンパイラーは、RTTI の typeid 演算子および dynamic_cast 演算子に必要な情報を生成します。

***RTTITYPE** C++

コンパイラーは、RTTI の typeid 演算子に必要な情報は生成しますが、dynamic_cast 演算子に必要な情報は生成しません。RTBND(*LLP64) コンパイラー・オプションが指定されていない場合、このオプションはサポートされません。

***RTTICAST** C++

コンパイラーは RTTI の dynamic_cast 演算子に必要な情報は生成しますが、typeid 演算子に必要な情報は生成しません。RTBND(*LLP64) コンパイラー・オプションが指定されていない場合、このオプションはサポートされません。

***NOPPONLY** C

CRTCMOD コマンドでのみ有効。 デフォルト設定。コンパイラーは、*GEN が OPTION のデフォルトとして残っている場合には、コンパイル・シーケンス全体を実行します。

PPGENOPT を *NONE 以外の設定で指定すると、OPTION(*NOPPONLY) および OPTION(*GEN) オプション設定がオーバーライドされます。

注: PPGENOPT コンパイラー・オプションは OPTION(*NOPPONLY) を置き換えます。将来のリリースでは OPTION(*NOPPONLY) のサポートは除去される可能性があります。

***PPONLY** C

CRTCMOD コマンドでのみ有効。 プリプロセッサーが実行され、ライブラリー QTEMP 内のソース・ファイル QACZEXPAND に出力が保存されます。member-name は MODULE パラメーターで指定されている名前と同じです。コンパイル・シーケンスの残りは実行されません。ジョブをバッチ・モードで実行した場合は、ジョブが完了するとその出力は削除されます。

SRCSTMF を指定した場合、コンパイラーは現行ディレクトリー内のストリーム・ファイルに出力を保存します。このファイルの名前は、SRCSTMF 上の ".i" という拡張子の付いたファイルと同じです。

OPTION(*PPONLY) を指定すると、PPGENOPT(*NONE) および OPTION(*GEN) オプション設定がオーバーライドされます。代わりに、以下の設定が暗黙指定されます。

- データ管理ソース・ファイルの場合は PPGENOPT(*DFT) PPSRCFILE(QTEMP/QACZEXPAND) PPSRCMBR(*MODULE)
- IFS ソース・ファイルの場合は PPGENOPT(*DFT) PPSRCSTMF(*SRCSTMF)

注: PPGENOPT コンパイラー・オプションは OPTION(*PPONLY) を置き換えます。将来のリリースでは OPTION(*PPONLY) のサポートは除去される可能性があります。

***NOSECLVL** C

デフォルト設定。第 2 レベル・メッセージ・テキストをリストに生成しません。

***SECLVL** C

第 2 レベル・メッセージ・テキストをリストに生成します。OUTPUT(*PRINT) を指定する必要があります。

***NOSHOWINC**

デフォルト設定。ユーザー・インクルード・ファイルまたはシステム・インクルード・ファイルを、ソース・リスト内またはデバッグ・リスト・ビュー内で展開しません。

***SHOWINC**
ユーザー・インクルード・ファイルおよびソース・インクルード・ファイルの両方を、リストのソース・セクション内またはデバッグ・リスト・ビュー内で展開します。OUTPUT(*PRINT) または DBGVIEW(*ALL、*SOURCE、または *LIST) を指定する必要があります。

この設定により、*SHOWUSR および *SHOWSYS 設定がオンになりますが、*NOSHOWUSR または *NOSHOWSYS、あるいはその両方を *SHOWINC の後に指定することにより、それらの設定をオーバーライドできます。

***NOSHOWSKP** C
デフォルト設定。プリプロセッサーが無視したステートメントは、リストのソース・セクションまたはデバッグ・リスト・ビューには組み込まれません。プリプロセッサー・ディレクティブが false (ゼロ) に評価されると、プリプロセッサーはステートメントを無視します。

***SHOWSKP** C
プリプロセッサーがステートメントをスキップしたか否かには関係なく、すべてのステートメントをソース・リストまたはデバッグ・リスト・ビューに組み込みます。OUTPUT(*PRINT) または DBGVIEW(*ALL または *LIST) を指定する必要があります。

***SHOWSRC**
デフォルト設定。ソース・リストまたはデバッグ・リスト・ビューにソース・ステートメントを表示します。OUTPUT(*PRINT) または DBGVIEW(*ALL、*SOURCE、または *LIST) を指定する必要があります。

***NOSHOWSRC**
ソース・リストまたはデバッグ・リスト・ビューにソース・ステートメントを表示しません。*EXPMAC、*SHOWINC、*SHOWUSR、*SHOWSYS、および *SHOWSKP リスト・オプションを *NOSHOWSRC オプションの後に指定すると、この設定をオーバーライドできます。

***NOSHOWSYS**
デフォルト設定。#include ディレクティブのシステム・インクルード・ファイルを、ソース・リスト内またはデバッグ・リスト・ビュー内で展開しません。

***SHOWSYS**
#include ディレクティブのシステム・インクルード・ファイルを、ソース・リスト内またはデバッグ・リスト・ビュー内で展開します。OUTPUT オプション、*ALL、*SOURCE、または *LIST の DBGVIEW パラメーターを指定する必要があります。#include ディレクティブのシステム・インクルード・ファイルは、不等号括弧 (< >) で囲まれます。

***NOSHOWUSR**
デフォルト設定。#include ディレクティブのユーザー・インクルード・ファイルを、ソース・リスト内またはデバッグ・リスト・ビュー内で展開しません。

***SHOWUSR**
#include ディレクティブのユーザー・インクルード・ファイルを、ソース・リスト内またはデバッグ・リスト・ビュー内で展開します。OUTPUT(*PRINT) または DBGVIEW(*ALL、*SOURCE、または *LIST) を指定する必要があります。 #include ディレクティブのユーザー・インクルード・ファイルは、二重引用符 (" ") で囲まれます。このオプションは、外部記述されているファイルを、ILE C または C++ プログラムで #pragma mapinc を使用して処理するときに生成される型定義を出力する場合に使用します。

***STDINC**
デフォルト設定。コンパイラーは、デフォルトのインクルード・パス (IFS ソース・ストリーム・ファイルの場合は /QIBM/include、データ管理ソース・ファイル・メンバーの場合は QSYSINC) を、検索順序の最後に組み込みます。

***NOSTDINC**
コンパイラーは、デフォルトのインクルード・パス (IFS ソース・ストリーム・ファイルの場合は /QIBM/include、データ管理ソース・ファイル・メンバーの場合は QSYSINC) を検索順序から除去します。

***NOSTDLOGMSG**
デフォルト設定。コンパイラーは標準出力コンパイラー・メッセージを作成しません。

***STDLOGMSG**
コンパイラーは、Qshell 環境では、標準出力コンパイラー・メッセージを作成します。
TGTRLS(*PRV) を指定してコンパイルしている場合、このオプションは何の影響も与えません。

***NOSTRUCREF** 
デフォルト設定。参照されているすべての構造体変数または共用体変数の集合体構造マップをコンパイラー・リストに生成しません。

***STRUCREF** 
参照されているすべての構造体変数または共用体変数の集合体構造マップをコンパイラー・リストに生成します。このマップはソース・プログラム内の参照されているすべての構造体のレイアウトを提供し、変数が埋め込まれているかいないかを示します。

***NOSYSINCPATH**
デフォルト設定。ユーザー・インクルードの検索パスには影響しません。

***SYSINCPATH**
ユーザー・インクルードの検索パスをシステム・インクルードの検索パスに変更します。このオプションは、機能的にはユーザー #include ディレクティブ内の二重引用符 (#include "file_name") を不等号括弧 (#include <file_name>) に変更することと等価です。

***NOXREF**
相互参照テーブルをリストに生成しません。*NOXREF はデフォルトです。

***XREF**
ソース・コード内の ID のリストとその ID が出現する行番号を含む相互参照テーブルを生成します。OUTPUT オプションを指定する必要があります。

*XREF オプションは *XREFREF オプションをオーバーライドします。

***NOXREFREF**
デフォルト設定。相互参照テーブルをリストに生成しません。

***XREFREF**
ソース・コード内の参照されている ID および変数のみとそれらが出現する行番号を含む相互参照テーブルを生成します。OUTPUT オプションを指定する必要があります。

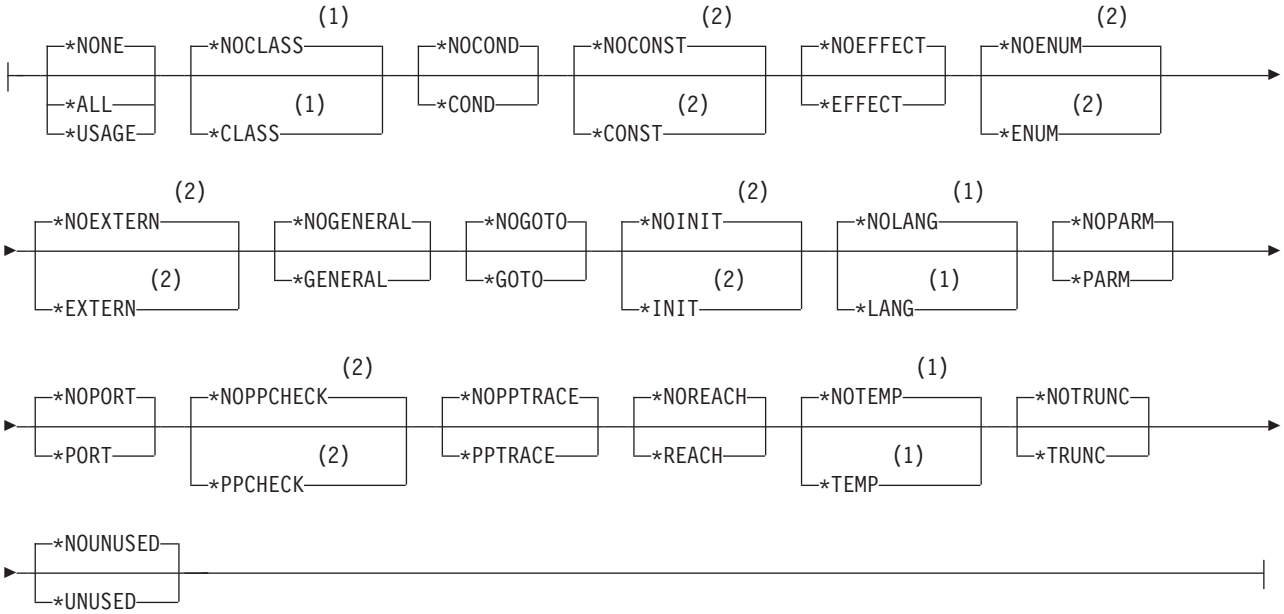
*XREF オプションは *XREFREF オプションをオーバーライドします。

CHECKOUT
起こり得るプログラミング・エラーを示す情報メッセージを生成するために選択するオプションを指定します。1 つのオプションを複数回指定した場合、または、2 つのオプションが競合する場合には、指定した最後のオプションが使用されます。

注: CHECKOUT は、多くのメッセージを作成する場合があります。これらのメッセージがジョブ・ログに送信されないようにするには、OPTION(*NOLOGMSG) およびソース・リスト・オプション OUTPUT(*PRINT) を指定します。

CHECKOUT 構文 :

CHECKOUT の詳細 :



注:

1 C++ コンパイラーのみ

2 C コンパイラーのみ

指定可能なオプションは次のとおりです。

***NONE**
デフォルト設定。CHECKOUT のオプションのすべてを使用不可にします。

***ALL**
CHECKOUT のオプションのすべてを使用可能にします。

***USAGE**

- C *ENUM、*EXTERN、*INIT、*PARM、*PORT、*GENERAL、および *TRUNC の指定と等価。他のすべての CHECKOUT オプションが使用不可です。
- C++ *COND の指定と等価。他のすべての CHECKOUT オプションが使用不可です。

***NOCLASS** C++
デフォルト設定。クラスの使用に関する情報を表示しません。

***CLASS** C++
クラスの使用に関する情報を表示します。

***NOCOND**
デフォルト設定。条件式で起こり得る冗長または問題に関して警告しません。

***COND**
条件式で起こり得る冗長または問題に関して警告します。

***NOCONST** C
デフォルト設定。定数を含む演算に関して警告しません。

***CONST** C
定数を含む演算に関して警告します。

***NOEFFECT**
デフォルト設定。影響を及ぼさないステートメントに関して警告しません。

***EFFECT**
影響を及ぼさないステートメントに関して警告します。

***NOENUM** C
デフォルト設定。列挙型の使用法はリストしません。

***ENUM** C
列挙型の使用法をリストします。

***NOEXTERN** C
デフォルト設定。外部宣言を持つ未使用変数をリストしません。

***EXTERN** C
外部宣言を持つ未使用変数をリストします。

***NOGENERAL**
デフォルト設定。一般の CHECKOUT メッセージをリストしません。

***GENERAL**
一般の CHECKOUT メッセージをリストします。

***NOGOTO**
デフォルト設定。goto 文の出現および使用法をリストしません。

***GOTO**
goto 文の出現および使用法をリストします。

***NOINIT** C
デフォルト設定。明示的に初期化されない自動変数をリストしません。

***INIT** C
明示的に初期化されない自動変数をリストします。

***NOLANG** C++
デフォルト設定。言語レベルの効果に関する情報を表示しません。

***LANG** C++
言語レベルの効果に関する情報を表示します。

***NOPARM**
デフォルト設定。使用しない関数仮パラメーターをリストしません。

***PARM**
使用しない関数仮パラメーターをリストします。

***NOPORT**
デフォルト設定。C または C++ 言語のポータブル以外の使用法をリストしません。

***PORT**
C または C++ 言語のポータブル以外の使用法をリストします。

***NOPPCHECK** C
デフォルト設定。プリプロセッサー・ディレクティブをリストしません。

***PPCHECK** C
すべてのプリプロセッサー・ディレクティブをリストします。

***NOPPTRACE**
デフォルト設定。プリプロセッサーによるインクルード・ファイルのトレースをリストしません。

***PPTRACE**
プリプロセッサーによるインクルード・ファイルのトレースをリストします。

***NOREACH**
デフォルト設定。到達不能ステートメントに関して警告しません。

***REACH**
到達不能ステートメントに関して警告します。

***NOTEMP** C++
デフォルト設定。一時変数に関する情報を表示しません。

***TEMP** C++
一時変数に関する情報を表示します。

***NOTRUNC**
デフォルト設定。起こり得るデータの切り捨てまたは損失に関して警告しません。

***TRUNC**
起こり得るデータの切り捨てまたは損失に関して警告します。

***NOUNUSED**
デフォルト設定。未使用の自動または静的変数を検査しません。

***UNUSED**
未使用の自動変数または静的変数を検査します。

OPTIMIZE

オブジェクトの最適化のレベルを指定します。

OPTIMIZE 構文 :

10 デフォルト設定。生成済みコードは最適化されません。このレベルでは、最短のコンパイル時間になります。

20 コードに一部の最適化が実行されます。

30 生成されたコードに対して全最適化が実行されます。

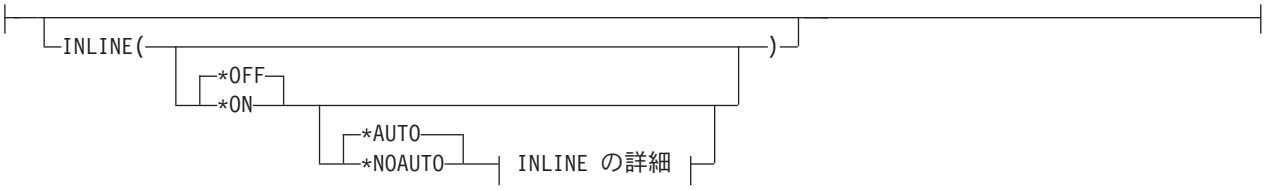
40 レベル 30 で行われたすべての最適化は、生成されたコードで実行されます。さらに、コードは、命令トレースを可能にし、トレース・システム関数を呼び出す、プロシージャーのプロローグおよびエピローグから除去されます。このコードを除去すると、リーフ・プロシージャーの作成が可能になります。リーフ・プロシージャーには、他のプロシージャーへの呼び出しは含まれません。リーフ・プロシージャーへのプロシージャー呼び出しパフォーマンスは、通常のプロシージャーへのプロシージャー呼び出しパフォーマンスよりもかなり高速です。

INLINE

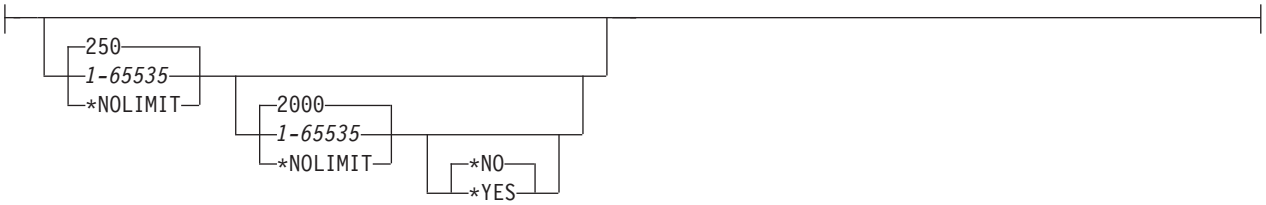
コンパイラーが、関数呼び出しと呼び出し先関数の指示の置換を検討できるようにします。関数をインライン化すると、呼び出しのオーバーヘッドが除去され、最適化が改善されます。何回も呼び出される小関数は、インライン化の適切な候補となります。

注: また、INLINE オプションを指定すると、すべての先行 INLINE オプションを (デフォルト設定を含め) 指定することができます。

INLINE 構文 :



INLINE の詳細 (続き):



指定できる INLINE オプションは次のとおりです。

インライナー
インラインを使用するかどうかを指定します。

***OFF**
デフォルト設定。インラインが、コンパイル単位で実行されないように指定します。

***ON**
インラインが、コンパイル単位で実行されるように指定します。デバッグ・リスト・ビューが指定されると、インライナーはオフになります。

モード
インライナーが、しきい値および制限に基づいて自動的に関数をインライン化しようと試みるかを指定します。

***AUTO**
インライナーが、指定したしきい値および制限に基づいて関数をインライン化できるかを判別するように指定します。#pragma noinline ディレクティブが *AUTO をオーバーライドします。これはデフォルトです。

***NOAUTO**
インライン化にマーク付けられた関数のみが、インライン化の候補とみなされるように指定します。インライン化にマーク付けられた関数は、#pragma inline ディレクティブが指定された C 関数、inline キーワードで宣言された C++ 関数、言語規則によってインライン化にマーク付けられた C++ 関数を含みます。

しきい値
自動インライン化の候補となり得る関数の最大サイズを指定します。サイズは、抽象コード単位で測定されます。抽象コード単位は、関数の実行可能コードのサイズに比例します。C および C++ コードは、コンパイラーによって抽象コード単位に変換されます。

250
しきい値を 250 に指定します。これはデフォルトです。

1-65535
しきい値を 1 から 65535 に指定します。

***NOLIMIT**
しきい値をプログラムの最大サイズとして定義します。

制限
自動インライン化が停止する前に、関数の使用可能な最大相対サイズを指定します。

2000
制限を 2000 に指定します。これはデフォルトです。

1-65535
制限を 1 から 65535 に指定します。

***NOLIMIT**
制限は、プログラムの最大サイズとして定義されます。システム制限が発生する場合があります。

レポート
コンパイラー・リストを使用してインライナー・レポートを作成するかどうかを指定します。

***NO**
インライナー・レポートは作成されません。これはデフォルトです。

***YES**
インライナー・レポートが作成されます。OUTPUT(*PRINT) は、インライナー・レポートを作成するために指定する必要があります。

MODCRTOPT

CRTCMOD および CRTCPPMOD コマンドでのみ有効。 *MODULE オブジェクトが作成されたときに使用するオプションを指定します。これらのオプションを、任意の順序でスペースで区切って指定することができます。1 つのオプションを複数回指定した場合、または、2 つのオプションが競合する場合には、指定した最後のオプションが使用されます。

MODCRTOPT 構文 :

注:

1 「モジュールの作成」コマンドのみ

***NOKEEPILDTA**
デフォルト設定。中間言語データは、*MODULE オブジェクトでは格納されません。

***KEEPILDTA**
中間言語データは、*MODULE オブジェクトで格納されます。

DBGVIEW

作成したプログラム・オブジェクトに使用可能なデバッグのレベルを指定します。また、ソース・レベルのデバッグに使用可能であるソース・ビューを指定します。デバッグ・リスト・ビューを要求すると、インラインがオフになります。

DBGVIEW 構文 :

指定可能なオプションは次のとおりです。

***NONE**
デフォルト設定。コンパイル済みオブジェクトをデバッグするためのデバッグ・オプションのすべてを使用不可にします。

***ALL**
コンパイル済みオブジェクトをデバッグするためのデバッグ・オプションのすべてを使用可能にし、リスト・ビューとソース・ビューを作成します。このサブオプションが、OPTION(*FULL) または OPTION(*EXPMAC) と共に指定されると、コンパイラーはエラー・メッセージを発行し、コンパイルを停止します。

***STMT**
コンパイル済みオブジェクトは、プログラム・ステートメント番号および記号 ID を使用してデバッグできます。

注: *STMT オプションを使用してオブジェクトをデバッグするには、スプール・ファイル・リストが必要です。

***SOURCE**
コンパイル済みオブジェクトをデバッグするためのソース・ビューを生成します。
OPTION(*NOSHOWINC、*SHOWINC、*SHOWSYS、*SHOWUSR) は、作成するソース・ビューの内容を判別します。

注: モジュールが作成された後に、ルート・ソースを変更、名前変更、または移動しないでください。ルート・ソースは、デバッグのためにこのビューを使用するには、同じライブラリー/ファイル/メンバーになければなりません。

***LIST**
コンパイル済みオブジェクトをデバッグするためのリスト・ビューを生成します。 OPTION キーワードで指定されたリスト・オプション (*EXPMAC、 *NOEXPMAC、 *SHOWINC、 *SHOWUSR、 *SHOWSYS、 *NOSHOWINC、 *SHOWSKP、 *NOSHOWSKP) は、スプール・ファイル・リストと、作成されたリスト・ビューの内容を判別します。このサブオプションが、OPTION(*FULL) または OPTION(*EXPMAC) と共に指定されると、コンパイラーはエラー・メッセージを発行し、コンパイルを停止します。

DBGENCKEY

デバッグ・ビューに組み込まれた暗号化プログラム・ソースに使用する暗号鍵を指定します。

DBGENCKEY 構文 :

指定可能なオプションは次のとおりです。

***NONE**
デフォルト設定。暗号鍵は指定されていません。

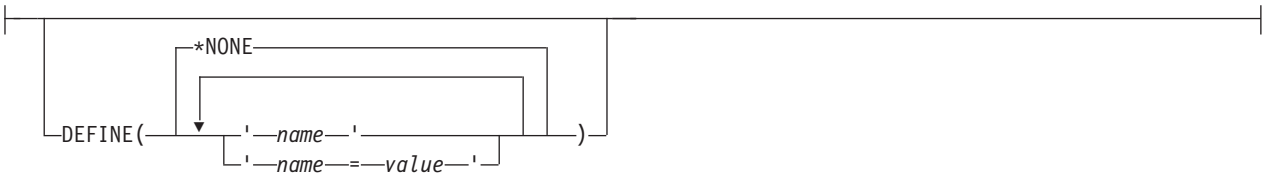
文字値
モジュール・オブジェクトに保管されているデバッグ・ビューに組み込まれた暗号化プログラム・ソースに使用する鍵を指定します。鍵の長さは 1 バイトから 16 バイトの間です。長さが 1 バイトから 15 バイトの鍵は、暗号化のためにブランクで埋めて 16 バイトにされます。長さがゼロの鍵を指定することは、*NONE を指定するのと同等です。

コード・ページ全体で不変ではない文字が鍵に含まれている場合、受動システムが起動システムと同じコード・ページを使用することを保証するかはユーザー次第です。それ以外の場合は、鍵は一致せず、暗号化解除は失敗します。異なるコード・ページを持つシステムで暗号鍵を入力する必要がある場合、すべての EBCDIC コード・ページに不変な文字で鍵を作成することを推奨します。

DEFINE

ファイルがコンパイラーによって処理される前に有効なプリプロセッサー・マクロを指定します。フォーマット DEFINE(マクロ) は、DEFINE('マクロ=1') を指定するのと等価です。

DEFINE 構文 :



***NONE**
デフォルト設定。マクロは定義されません。

name **または** *name=value*
最大 32 のマクロが定義され、マクロの最大長は 80 文字です。単一引用符で各マクロを囲んでください。引用符は 80 文字のストリングの一部とは見なされず、CRTCMOD または CRTCPPMOD プロンプト画面が使用される場合は必要ありません。単一引用符は、大/小文字の区別のあるマクロの場合に必要です。マクロをブランク・スペースで区切ります。*value* を指定しないと、コンパイラーは、値 1 をマクロに割り当てます。

注: コマンドで定義されるマクロは、ソース内で同じ名前のマクロ定義をオーバーライドします。警告メッセージは、コンパイラーによって生成されます。#define max(a,b) ((a)>;(b):(a)?(b)) などの関数に似たマクロは、コマンドでは定義できません。

LANGLVL

ソースがコンパイルされるとき、どのグループの言語機能が組み込まれるかを指定します。LANGLVL が指定されないと、言語レベルは、デフォルトで *EXTENDED に設定されます。

LANGLVL 構文 :

注:

1 C++ コンパイラーのみ

***EXTENDED**
デフォルト設定。プリプロセッサー変数 __EXTENDED__ を定義し、他の言語レベル変数は定義しません。ISO 規格の C および C++、IBM 言語拡張およびシステム特有の機能は使用可能です。 ILE C または C++ のすべての機能が使用可能な場合に、このパラメーターを使用してください。

***ANSI**
C および C++ コンパイルにはプリプロセッサー変数 __ANSI__ および __STDC__、C++ コンパイルのみには __cplusplus98__interface__ を定義し、他の言語レベルの変数は定義しません。ISO 規格 C および C++ のみが使用可能です。

***LEGACY** C++
他の言語レベルの変数は定義しません。構造を、古いレベルの C++ 言語と互換性があるようにしてください。

| ***EXTENDED0X** C++
| *EXTENDED と同じプリプロセッサー変数を定義し、このリリースでサポートされる各 C++11 言語機能に対して個別のプリプロセッサー変数を定義します。このオプションでは、コンパイラーは、ILE C++ のすべての機能と、このバージョンの ILE C++ コンパイラーで実装された現在サポートされている C++11 機能を使用します。 *ILE C/C++ 解説書* の『C++0x 互換性の拡張機能』を参照してください。

| 4-2 ページの『ILE C/C++ 事前定義マクロ』も参照してください。

ALIAS

プログラムに特定のカテゴリーの別名割り当てが含まれているか、プログラムが C/C++ 標準別名割り当て規則に準拠していないかどうかを示します。複数の異なる名前が同じストレージ・ロケーションの別名となっている可能性がある場合、コンパイラーは最適化の一部のスコープを制限します。

ALIAS 構文 :

***ANSI|*NOANSI**

***ANSI** が有効である場合、最適化で型ベースの別名割り当てが使用されます。これは、データ・オブジェクトへのアクセスに安全に使用できる左辺値に制限を加えます。最適化プログラムは、ポインターが指すことができるのは、同じ型のオブジェクトのみであると想定します。

***NOANSI** が有効である場合、最適化プログラムはワーストケースの別名割り当てを想定します。これは、型に関係なく特定の型のポインターが外部オブジェクト、またはアドレスが既に取得されているオブジェクトを指すことができると想定します。

***ADDRTAKEN|*NOADDRTAKEN**

***ADDRTAKEN** が有効である場合、変数はアドレスが取得されない限り、ポインターから切り離されています。アドレスがコンパイル単位に記録されていない変数のクラスは、ポインターを介した間接アクセスから切り離されていると見なされます。

***NOADDRTAKEN** が指定されている場合、コンパイラーは有効な別名割り当て規則に従って別名割り当てを生成します。

***ALLPTRS|*NOALLPTRS**

ALLPTRS** が有効な場合、ポインターに別名が割り当てられることはありません (これは ***TYPEPTR** を暗黙指定します)。 ***ALLPTRS** を指定すると、2 つのポインターが同じ保管場所をポイントしないことをコンパイラーに表明することになります。サブオプション ***ALLPTRS** は、ANSI** が指定されている場合にのみ有効です。

***TYPEPTR|*NOTYPEPTR**

***TYPEPTR** が有効である場合、異なる型へのポインターには別名は割り当てられません。
TYPEPTR** を指定すると、異なるタイプの 2 つのポインターが同じ保管場所をポイントしないことをコンパイラーに表明することになります。サブオプション ***TYPEPTR** は、ANSI** が指定されている場合にのみ有効です。

注:

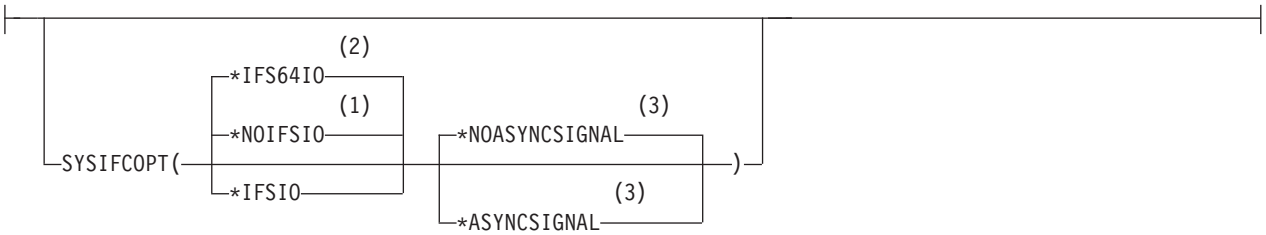
1. 競合する **ALIAS** 設定が指定された場合、最後に指定された設定が使用されます。例えば、**ALIAS(*TYPEPTR *NOTYPEPTR)** が指定された場合は、***NOTYPEPTR** が使用されます。
2. **ALIAS** は、コンパイル中のコードに関する表明をコンパイラーに対して行います。コードに関する表明が FALSE の場合、アプリケーションの実行時に、コンパイラーによって生成されるコードが予測不能の振る舞いをする可能性があります。

3. 以下は、型ベースの別名割り当ての対象となりません。
   - signed 型および unsigned 型。例えば、signed int へのポインターは unsigned int を指すことができます。
   - 文字ポインター型はどの型も指すことができます。
   - volatile または const として限定された型。例えば、const int へのポインターは int を指すことができます。

SYSIFCOPT

作成されるモジュールでの C または C++ ストリーム入出力操作に使用する統合ファイル・システムのオプションを指定します。

SYSIFCOPT 構文 :



注:

1 C コンパイラーのデフォルト設定

2 C++ コンパイラーのデフォルト設定

3 C コンパイラーのみ

***IFS64IO**
C++ コンパイラーのデフォルト設定。作成されるオブジェクトは、サイズが 2 ギガバイトよりも大きいファイルに対する C および C++ ストリーム入出力操作をサポートする 64 ビット統合ファイル・システム API を使用します。このオプションを使用することは、SYSIFCOPT(*IFSIO *IFS64IO) を指定することと等価です。

***NOIFSIO**
C コンパイラーのデフォルト設定。作成されるオブジェクトは、C および C++ ストリーム入出力操作に IBM i データ管理ファイル・システムを使用します。

***IFSIO**
作成されるオブジェクトは、サイズが最大 2 ギガバイトのファイルに対する C および C++ ストリーム入出力操作に、統合ファイル・システム API を使用します。

***NOASYNCSIGNAL** C
デフォルト設定。同期シグナル関数の非同期シグナル関数への実行時マッピングを使用可能にしません。

***ASYNCSIGNAL** C
同期シグナル関数の非同期シグナル関数への実行時マッピングを使用可能にします。このオプションを指定すると、C ランタイム環境で、同期 **signal()** 関数が非同期 **sigaction()** 関数にマップされ、同期 **raise()** 関数が非同期 **kill()** 関数にマップされます。

LOCALETYPE

作成したオブジェクトによって使用されるロケール・サポートのタイプを指定します。

LOCALETYPE 構文 :

注:

1 C コンパイラーのみ

***LOCALE**
デフォルト設定。このオプションでコンパイルされるオブジェクトは、ロケール・オブジェクトのタイプ *LOCALE を使用して、ILE C/C++ コンパイラーおよびランタイムで提供されたロケール・サポートを使用します。このオプションは、V3R7 以降のリリースの IBM i オペレーティング・システムで実行するプログラムでのみ有効です。

***LOCALEUCS2**
このオプションでコンパイルされたオブジェクトは、UNICODE CCSID (13488) の 2 バイト形式で、ワイド文字リテラルを格納します。

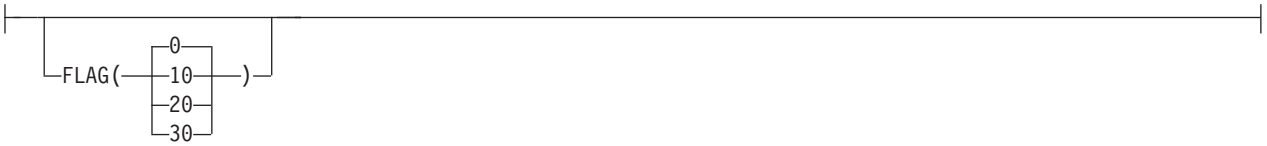
***CLD**
このオプションでコンパイルされるオブジェクトは、ロケール・オブジェクトのタイプ *CLD を使用して前のリリースの ILE C コンパイラーおよびランタイムで提供されたロケール・サポートを使用します。

***LOCALEUTF**
このオプションで作成されたモジュールおよびプログラム・オブジェクトは、*LOCALE オブジェクトによって提供されたロケール・サポートを使用します。ワイド文字タイプは、4 バイトの utf-32 値を含みます。ナロー文字タイプは、utf-8 値を含みます。

FLAG

リストに表示できるメッセージのレベルを指定します。 OPTION(*SECLVL) が指定されない場合は、メッセージの第 1 レベルのテキストのみが含まれます。

FLAG 構文 :



0 デフォルト設定。**通知**レベルで開始されるすべてのメッセージが表示されます。

10 **警告**レベルで開始されるすべてのメッセージが表示されます。

20 **エラー**・レベルで開始されるすべてのメッセージが表示されます。

30 **重大エラー**・レベルで開始されるすべてのメッセージが表示されます。

MSGLMT

コンパイルが停止する前に発生する可能性がある特定の重大度レベルのメッセージの最大数を指定します。

MSGLMT 構文 :

***NOMAX**
デフォルト設定。コンパイルは、指定したメッセージ重大度レベルで発生したメッセージの数に関係なく、続行されます。

0 32767
コンパイルが停止する前に、指定したメッセージ重大度レベルで、またはそれ以上で発生する可能性があるメッセージの最大数を指定します。有効範囲は 0 から 32767 です。

30 デフォルト設定。コンパイルが停止する前に、重大度 30 で *message-limit* メッセージが発生するよう指定します。

0 コンパイルが停止する前に、重大度 0 から 30 で *message-limit* メッセージが発生するよう指定します。

10 コンパイルが停止する前に、重大度 10 から 30 で *message-limit* メッセージが発生するよう指定します。

20 コンパイルが停止する前に、重大度 20 から 30 で *message-limit* メッセージが発生するよう指定します。

REPLACE

オブジェクトの既存バージョンを現行バージョンで置き換えるかどうかを指定します。

REPLACE 構文 :

***YES**
デフォルト設定。既存オブジェクトは新規バージョンで置き換えられます。古いバージョンはライブラリー QRPLOBJ に移され、システム日付および時刻に基づいて名前が変更されます。置き換えられたオブジェクトのテキスト記述は、元のオブジェクトの名前に変更されます。古いオブジェクトが削除されていない場合、そのオブジェクトは次回の IPL で削除されます。

***NO**
既存オブジェクトは置き換えられません。同じ名前を持つオブジェクトが、指定されたライブラリーに存在する場合は、メッセージが表示されてコンパイルは停止します。

USRPRF

CRTBNDC および CRTBNDCPP コマンドでのみ有効。 コンパイル済み ILE C または C++ プログラム・オブジェクトの実行時に使用され、プログラム・オブジェクトがオブジェクトごとに所有している権限を含

んでいるユーザー・プロファイルを指定します。プログラムの所有者またはプログラム・ユーザーのプロファイルは、プログラム・オブジェクトがどのオブジェクトを使用するのかを制御するために使用されます。

USRPRF 構文 :

注:

1 「バインド済みプログラムの作成」コマンドのみ

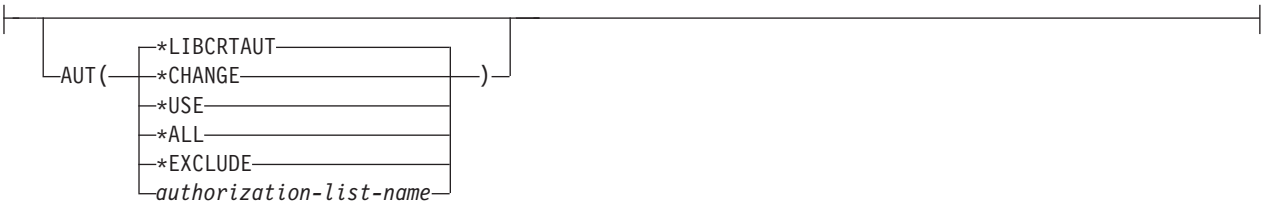
***USER**
デフォルト設定。プログラム・オブジェクトを実行しているユーザーのプロファイルが使用されます。

***OWNER**
プログラム所有者およびプログラム・ユーザーの両方のユーザー・プロファイルに含まれているオブジェクト権限を集めたセットを使用して、プログラム・オブジェクトの処理中にオブジェクトを検出し、それにアクセスします。プログラムによって作成されるオブジェクトは、そのプログラムのユーザーによって所有されます。

AUT

オブジェクトに対して特定権限を持っていないユーザーにオブジェクト権限を指定します。ユーザーが権限リストに載っていない場合、またはそのグループがオブジェクトに対する特定権限を持っていない場合があります。

AUT 構文 :



***LIBCRTAUT**
デフォルト設定。オブジェクトの共通権限は、ターゲット・ライブラリー (作成されたオブジェクトが入っているライブラリー) の CRTAUT キーワードから取られます。この値はオブジェクトの作成時に決められます。オブジェクトの作成後にライブラリーの CRTAUT 値が変更された場合には、新しい値は既存のオブジェクトに影響しません。

***CHANGE**
すべてのデータ権限およびオブジェクトに対するすべての操作を実行する権限を提供します。ただし、所有者に限定されたものまたはオブジェクト権限およびオブジェクト管理権限によって制御されているものを除きます。オブジェクトを変更することも、その基本機能を実行することもできます。

***USE**
オブジェクト操作権、読み取り権限、およびオブジェクトの基本操作の権限を提供します。特定権限を持っていないユーザーはオブジェクトを変更することができません。

***ALL**
所有者に限定されているものまたは権限リスト管理権限によって制御されているものを除き、オブジェ

クトのすべての操作の権限を提供します。オブジェクトの存在を制御したり、そのセキュリティーを指定したり、その基本機能を実行したりすることはできますが、その所有権を転送することはできません。

***EXCLUDE**
特殊権限を持っていないユーザーは、そのオブジェクトにアクセスできません。

authorization-list-name
モジュール・オブジェクトが追加される、ユーザーおよび権限の権限リストの名前を入力します。オブジェクトはこの権限リストによって保護され、オブジェクトの共通権限は *AUTL に設定されます。権限リストは、コマンドが出されるときにシステムに存在していなければなりません。

TGTRLS

作成されるオブジェクトに対してオペレーティング・システムのリリース・レベルを指定します。

TGTRLS 構文 :

***CURRENT**
デフォルト設定。オブジェクトは、システムで稼働しているオペレーティング・システムのリリースで使用されます。例えば、システムで V2R3M5 が稼働している場合、*CURRENT は、バージョン 2 リリース 3 修正 5 がインストールされているシステムでオブジェクトを使用することを示します。またこのオブジェクトは、インストールされているオペレーティング・システムのリリースよりも新しいリリースがインストールされているシステムでも使用できます。

注: V2R3M5 がシステムで稼働中で、作成するオブジェクトを V2R3M0 がインストールされているシステムで使用する場合は、TGTRLS(*CURRENT) ではなく TGTRLS(V2R3M0) を指定してください。

***PRV**
オブジェクトはオペレーティング・システムの前のリリースで使用されます。例えば、システムで V2R3M5 が稼働しており、V2R2M0 がインストールされているシステムで作成するオブジェクトを使用する場合は、*PRV を指定します。またこのオブジェクトは、インストールされているオペレーティング・システムのリリースよりも新しいリリースがインストールされているシステムでも使用できます。

release-level
VxRxMx の形式でリリースを指定します。オブジェクトは、特定のリリースがインストールされているシステム、またはインストールされているオペレーティング・システムのリリースよりも新しいリリースがインストールされているシステムで使用できます。値は、現行バージョン、リリース、および修正レベルによって決まり、新規リリースのたびに変更されます。指定するリリース・レベルが、このコマンドでサポートされている最も古いリリース・レベルよりも古い場合、サポートされているもっとも古いリリースを示すエラー・メッセージが表示されます。

V5R1M0 よりも前のオペレーティング・システム・リリース用にコンパイルを行うと、以下のコンパイラー・オプションのいくつかの設定が無視されることがあります。

- 6-17 ページの『CHECKOUT』
- 6-12 ページの『OPTION』

- 6-10 ページの『OUTPUT』
- 6-34 ページの『PRFDTA』

V5R1M0 よりも前のオペレーティング・システム・リリース用にコンパイルを行うと、以下のオプションが完全に無視されます。

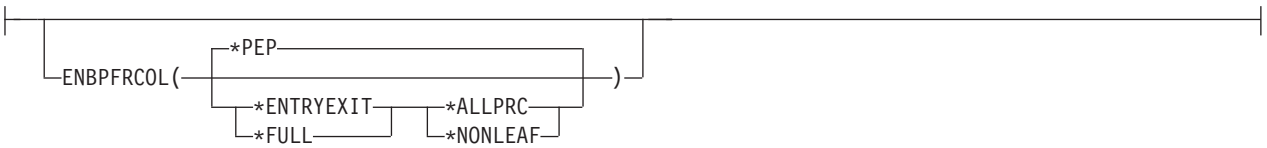
- 6-44 ページの『CSOPT』
- 6-45 ページの『DFTCHAR』
- 6-39 ページの『DTAMDL』
- 6-40 ページの『ENUM』
- 6-44 ページの『INCDIR』
- 6-45 ページの『LICOPT』
- 6-41 ページの『MAKEDEP』
- 6-40 ページの『PACKSTRUCT』
- 6-41 ページの『PPGENOPT』
- 6-38 ページの『STGMDL』
- 6-46 ページの『TGTCCSID』

ENBPFRCOL

パフォーマンス・データの測定コードをオブジェクトに生成するかどうかを指定します。収集されたデータは、システム・パフォーマンス測定ツールで使用することで、アプリケーションのパフォーマンスのプロファイルを作成することができます。作成されたオブジェクトにパフォーマンス測定コードを生成すると、オブジェクトが若干大きくなり、パフォーマンスに影響が出る場合があります。

注: V6R1M0 からは、このパラメーターは作成されたオブジェクトに影響しなくなりました。これは、V6R1M0 より前のリリースとの互換性のためだけに存在します。

ENBPFRCOL 構文 :



***PEP**
デフォルト設定。プログラム入力プロシージャーの入り口および出口でのみ、パフォーマンス統計情報が収集されます。 アプリケーションに関する全体的なパフォーマンス情報を収集したい場合には、この値を選択してください。このサポートは、以前 TPST ツールで提供されていたものと同じです。

***ENTRYEXIT *NONLEAF**
リーフ・プロシージャーでないすべてのプログラムのプロシージャーの入り口および出口で、パフォーマンス統計情報が収集されます。これには、プログラムの PEP が含まれます。

この選択項目が役立つのは、アプリケーションに他の関数を呼び出す関数に関する情報を把握したい場合です。

***ENTRYEXIT *ALLPRC**
すべてのオブジェクトのプロシージャー (リーフ・プロシージャーを含む) の入り口および出口で、パフォーマンス統計情報が収集されます。これには、プログラムの PEP が含まれます。

この選択項目が役立つのは、すべての関数に関する情報を把握したい場合です。アプリケーションで呼び出されるプログラムがすべて、*PEP、*ENTRYEXIT、または *FULL オプションを使用してコンパイルされていることが分かっている場合は、このオプションを使用してください。そうでない場合、アプリケーションが、パフォーマンス測定のできない他のオブジェクトを呼び出しているなら、パフォーマンス測定ツールは、リソースをその呼び出し側のアプリケーションが使用しているものとします。このため、リソースが現在どこで実際に使用されているのかを判別することは困難になります。

***FULL *NONLEAF**
リーフ・プロシージャーでないすべてのプロシージャーの入り口および出口で、パフォーマンス統計情報が収集されます。外部プロシージャー呼び出しの前後でも統計が収集されます。

***FULL *ALLPRC**
リーフ・プロシージャーを含むすべてのプロシージャーの入り口および出口で、パフォーマンス統計情報が収集されます。また、外部プロシージャー呼び出しの前後でも統計が収集されます。

アプリケーションから呼び出される他のオブジェクトが、*PEP、*ENTRYEXIT、または *FULL オプションを使用してコンパイルされていない場合には、このオプションを使用してください。このオプションによって、パフォーマンス測定ツールは、アプリケーションが使用するリソースとアプリケーションで呼び出されるオブジェクト (パフォーマンス測定が可能でない場合であっても) が使用するリソースとを区別します。このオプションは最もコストがかかるものですが、これによって 1 つのアプリケーション内のさまざまなプログラムを選択して分析することができます。

***NONE**
このオブジェクトについて収集されるパフォーマンス・データはありません。パフォーマンス情報が必要でないときに、より小さいオブジェクト・サイズが必要な場合に、このパラメーターを使用してください。

PFROPT

パフォーマンスの向上のために使用できる各種オプションを指定します。これらのオプションは、1 つ以上の空白で区切って、任意の順序で指定できます。1 つのオプションを複数回指定した場合、または 2 つのオプションが競合している場合は、最後に指定したオプションが使用されます。

PFROPT 構文 :

***SETFPCA**
デフォルト設定。コンパイラーは、浮動小数点計算用の ANSI のセマンティクスを実現するために、浮動小数点計算属性を設定します。

***NOSETFPCA**
計算属性は設定されません。このオプションは、作成されるオブジェクトの内部に浮動小数点計算が含まれていない場合に使用されます。

***NOSTRDONLY**
コンパイラーがストリングを書き込み可能メモリーに配置する必要があることを指定します。これはデフォルトです。

***STRDONLY**
コンパイラーがストリングを読み取り専用メモリーに配置できることを指定します。

PRFDTA

モジュールまたはプログラムに対してプログラム・プロファイルをオンにするかどうかを指定します。プロファイルは、ILE アプリケーションでのキャッシュ・ラインおよびメモリー・ページの使用を改善することにより、プログラムまたはサービス・プログラムのパフォーマンスの向上につながることがあります。

PRFDTA 構文 :

注: スタンドアロンの *MODULE オブジェクトのプロファイルは作成できません。

***NOCOL**

デフォルト設定。プロファイル・データの収集は無効です。プロファイル・データがプログラムまたはサービス・プログラム・オブジェクトに含まれている場合、モジュールはそのプロファイル・データを収集しません。

***COL**

プロファイル・データの収集は有効です。プロファイル・データがプログラムまたはサービス・プログラム・オブジェクトに含まれている場合、モジュールはそのプロファイル・データを収集します。

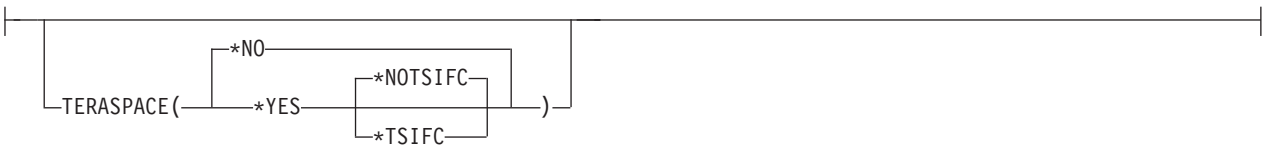
このオプションを使用して、オブジェクトの作成時にデータを収集するコードを生成します。このデータは、プロシージャー内の基本ブロックの実行回数と、プロシージャーの呼び出し回数で構成されます。

注: *COL は、モジュールの最適化レベルが *FULL (30) 以上の場合にのみ有効です。

TERASPACE

作成されたオブジェクトが、テラスペース・ストレージ・ロケーションを参照するアドレスを認識して、それを使用できるかどうかを指定します。

TERASPACE 構文 :



***NO**

デフォルト設定。作成されたオブジェクトは、テラスペース・ストレージのアドレスを認識できません。

注: V6R1M0 からは、すべてのモジュールがテラスペースから割り振られたストレージのアドレッシングを処理するために使用できます。ただし、*NO が指定されている場合は、***YES** 記述にリストされているコンパイラー機能は使用できません。

***YES**

作成されたオブジェクトは、他のテラスペース対応プログラムおよびサービス・プログラムから渡されたパラメーターを含め、テラスペース・ストレージのアドレスを扱うことができます。さらに、以下のコンパイラー機能が使用可能です。

- ポインターを __ptr64 で修飾し、テラスペース・ストレージへのアクセスに使用する 8 バイト・ポインターの作成を許可することができます。
- テラスペース・ストレージ・モデルは、STGMDL(*TERASPACE) コンパイラー・オプションで指定することができます。
- LLP64 データ・モデルは、DTAMDL(*LLP64) コンパイラー・オプションあるいは **#pragma datamodel(llp64)** ディレクティブで指定することができます。
- ポインター間の差の算術計算は、ptrdiff_t の結果の代わりに signed long long の結果を返します。

***NOTSIFC**
コンパイラーは、malloc( ) や shmat( ) などのストレージ関数のテラスペース・バージョンを使用しません。 TERASPACE(*YES) が指定されている場合、***NOTSIFC** がデフォルトです。

***TSIFC**
コンパイラーは、malloc( ) や shmat( ) などのストレージ関数のテラスペース・バージョンを使用しますが、プログラム・ソース・コードを変更する必要はありません。コンパイラーは __TERASPACE__ マクロを定義し、いくつかのストレージ関数名をそれらのテラスペース対応の関数名にマップします。例えば、このコンパイラー・オプションを選択すると、malloc( ) ストレージ関数は _C_TS_malloc( ) にマップされます。

DTAMDL (6-39 ページの『DTAMDL』 ページを参照) および STGMDL (6-38 ページの『STGMDL』 ページを参照) コンパイラー・オプションを TERASPACE コンパイラー・オプションと一緒に使用できます。これらのオプションの有効な組み合わせを、それらの組み合わせを選択したときの効果と共に以下の表に示します。

表 6-2. DTAMDL、STGMDL、および TERASPACE コンパイラー・オプションの有効な組み合わせ

DTAMDL(*P128)	STGMDL		
	(*SNGLVL)	**(*TERASPACE)**	**(*INHERIT)**
	• モジュール/プログラムは、単一レベル・ストアの作業用ストレージを使用するよう設計されています。 • 生成されたコードは、以下を使用する実行をサポートします。 – 単一レベル・ストアの作業用ストレージ – 単一レベル・ストアの動的ストレージ • 作業用ストレージには、16 バイトのスペース・ポインターを使用してのみアクセスできます。 • デフォルトのポインター・サイズは 16 バイトです。	• モジュール/プログラムは、テラスペースの作業用ストレージを使用するよう設計されています。 • 生成されたコードは、以下を使用する実行をサポートします。 – テラスペースの作業用ストレージ – 単一レベル・ストアの動的ストレージ – テラスペースの動的ストレージ • 作業用ストレージには、以下のいずれかを使用してアクセスできます。 – プロセスのローカル・ポインター – 16 バイトのスペース・ポインター • デフォルトのポインター・サイズは 16 バイトです。	• モジュールは、呼び出し側プログラムのストレージ・モデルに応じて、以下のいずれかを使用するよう設計されています。 – 単一レベル・ストアの作業用ストレージ – テラスペースの作業用ストレージ • 生成されたコードは、呼び出し側プログラムのストレージ・モデルに応じて、以下のいずれかを使用する実行をサポートしています。 – 単一レベル・ストアの作業用ストレージ – テラスペースの作業用ストレージ – 単一レベル・ストアの動的ストレージ – テラスペースの動的ストレージ • デフォルトのポインター・サイズは 16 バイトです。
TERASPACE(*NO)	**デフォルト設定**	無効な組み合わせ	無効な組み合わせ
TERASPACE(*YES *NOTSIFC)	• 生成されたコードは、テラスペースを使用する実行もサポートします。 • デフォルトでは、動的ストレージ・インターフェースの単一レベル・ストア・バージョンが使用されます。	• デフォルトでは、動的ストレージ・インターフェースの単一レベル・ストア・バージョンが使用されます。	• デフォルトでは、動的ストレージ・インターフェースの単一レベル・ストア・バージョンが使用されます。
TERASPACE(*YES *TSIFC)	• 生成されたコードは、テラスペースを使用する実行もサポートします。 • デフォルトでは、動的ストレージ・インターフェースのテラスペース・バージョンが使用されます。 • __TERASPACE__ マクロが定義されます。	• デフォルトでは、動的ストレージ・インターフェースのテラスペース・バージョンが使用されます。 • __TERASPACE__ マクロが定義されます。	• デフォルトでは、動的ストレージ・インターフェースのテラスペース・バージョンが使用されます。 • __TERASPACE__ マクロが定義されます。

DTAMDL(*LLP64)	STGMDL		
	(*SNGLVL)	(*TERASPACE)	(*INHERIT)
	• モジュール/プログラムは、単一レベル・ストアの作業用ストレージを使用するよう設計されています。 • 生成されたコードは、以下を使用する実行をサポートします。 – 単一レベル・ストアの作業用ストレージ – 単一レベル・ストアの動的ストレージ – テラスペース • 作業用ストレージには、16 バイトのスペース・ポインターを使用してのみアクセスできます。 • デフォルトのポインター・サイズは 8 バイトです。	• モジュール/プログラムは、テラスペースの作業用ストレージを使用するよう設計されています。 • 生成されたコードは、以下を使用する実行をサポートします。 – テラスペースの作業用ストレージ – 単一レベル・ストアの動的ストレージ – テラスペースの動的ストレージ • 作業用ストレージには、以下のいずれかを使用してアクセスできます。 – プロセスのローカル・ポインター – 16 バイトのスペース・ポインター • デフォルトのポインター・サイズは 8 バイトです。	• モジュールは、呼び出し側プログラムのストレージ・モデルに応じて、以下のいずれかを使用するよう設計されています。 – 単一レベル・ストアの作業用ストレージ – テラスペースの作業用ストレージ • 生成されたコードは、呼び出し側プログラムのストレージ・モデルに応じて、以下のいずれかを使用する実行をサポートしています。 – 単一レベル・ストアの作業用ストレージ – テラスペースの作業用ストレージ – 単一レベル・ストアの動的ストレージ – テラスペースの動的ストレージ • 作業用ストレージには、以下のいずれかを使用してアクセスできます。 – (条件付き) プロセスのローカル・ポインター – 16 バイトのスペース・ポインター • デフォルトのポインター・サイズは 8 バイトです。
TERASPACE(*NO)	無効な組み合わせ	無効な組み合わせ	無効な組み合わせ
TERASPACE(*YES *NOTSIFC)	• デフォルトでは、動的ストレージ・インターフェースの単一レベル・ストレージ・バージョンが使用されます。 • __LLP64_IFC__ マクロが定義されます。	• デフォルトでは、動的ストレージ・インターフェースの単一レベル・ストレージ・バージョンが使用されます。 • __LLP64_IFC__ マクロが定義されます。	• デフォルトでは、動的ストレージ・インターフェースの単一レベル・ストレージ・バージョンが使用されます。 • __LLP64_IFC__ マクロが定義されます。

DTAMDL(*LLP64)	STGMDL		
	(*SNGLVL)	(*TERASPACE)	(*INHERIT)
TERASPACE(*YES *TSIFC)	• デフォルトでは、動的ストレージ・インターフェースのテラスペース・バージョンが使用されます。 • __TERASPACE__ および __LLP64_IFC__ マクロが定義されます。	***テラスペースを最も有効に使用するための推奨設定*** • デフォルトでは、動的ストレージ・インターフェースのテラスペース・バージョンが使用されます。 • __TERASPACE__ および __LLP64_IFC__ マクロが定義されます。	• デフォルトでは、動的ストレージ・インターフェースのテラスペース・バージョンが使用されます。 • __TERASPACE__ および __LLP64_IFC__ マクロが定義されます。

テラスペースを最も有効に使用するために、以下のオプションの組み合わせを指定することができます。

```
TERASPACE(*YES *TSIFC) STGMDL(*TERASPACE) DTAMDL(*LLP64)
```

テラスペース・ストレージについて詳しくは、*ILE C/C++ プログラマーの手引き* の『*Using Teraspace*』および *ILE 概念* の『テラスペースおよび単一レベル・ストレージ』を参照してください。

STGMDL

モジュール・オブジェクトが使用するストレージのタイプ (静的または自動) を指定します。

STGMDL 構文 :

***SNGLVL**
デフォルト設定。モジュールまたはプログラムは、従来の単一レベル・ストレージ・モデルを使用します。オブジェクトの静的および自動ストレージは単一レベル・ストアから割り振られ、16 バイト・ポインターを使用してのみアクセスできます。 TERASPACE(*YES) オプションが指定されている場合は、必要であればモジュールからテラスペース動的ストレージにアクセスできます。

***TERASPACE**
モジュールまたはプログラムは、テラスペース・ストレージ・モデルを使用します。テラスペース・ストレージ・モデルは、単一のジョブに対して最大 1 テラバイトのローカル・アドレス・スペースを提供します。オブジェクトの静的および自動ストレージはテラスペースから割り振られ、8 バイト・ポインターまたは 16 バイト・ポインターのいずれかを使用してアクセスできます。

***INHERIT**
CRTCMOD および CRTCPPMOD コマンドでのみ有効。 作成されたモジュールは、単一レベル・ストレージまたはテラスペース・ストレージのいずれかを使用できます。使用されるストレージのタイプは、呼び出し元が必要としているストレージのタイプによって異なります。

STGMDL(*TERASPACE) または STGMDL(*INHERIT) を TERASPACE(*NO) と一緒に使用すると、コンパイラーはエラーとしてフラグを立て、コンパイルは停止します。

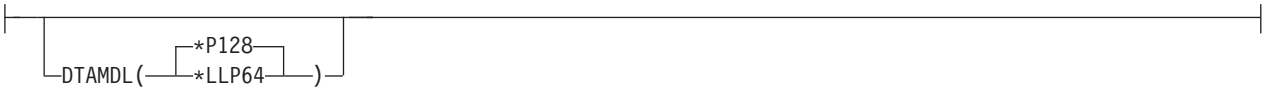
STGMDL、TERASPACE、および DTAMDL コンパイラー・オプションの有効な組み合わせについて詳しくは、6-34 ページの『TERASPACE』を参照してください。

IBM i プラットフォームで使用できるストレージのタイプについて詳しくは、*ILE 概念* の『テラスペースおよび単一レベル・ストレージ』を参照してください。

DTAMDL

明示的な修飾子がない場合に、ポインター型がどのように解釈されるかを指定します。 __ptr64 および __ptr128 タイプの修飾子と datamodel プラグマは、DTAMDL コンパイラー・オプションの設定をオーバーライドします。

DTAMDL 構文 :



***P128**
デフォルト設定。ポインター変数のデフォルト・サイズは 16 バイトです。

***LLP64**
ポインター変数のデフォルト・サイズは 8 バイトであり、コンパイラーは、マクロ __LLP64_IFC__ を定義します。

TERASPACE(*NO) と共に DTAMDL(*LLP64) を使用すると、コンパイラーによってエラーとしてフラグが立てられ、コンパイルが停止します。

詳しくは、 5-11 ページの『datamodel』プラグマを参照してください。

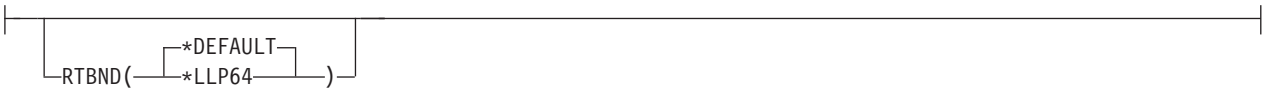
STGMDL、TERASPACE、および DTAMDL コンパイラー・オプションの有効な組み合わせについて詳しくは、 6-34 ページの『TERASPACE』を参照してください。

RTBND

C++

作成されたオブジェクトにランタイム・バインディング・ディレクトリーを指定します。

RTBND 構文 :



***DEFAULT**
デフォルト設定。作成されたオブジェクトは、デフォルトのバインディング・ディレクトリーを使用します。

***LLP64**
作成されたオブジェクトは、64 ビットのランタイム・バインディング・ディレクトリーを使用し、コンパイラーはマクロ __LLP64_RTBND__ を定義します。

PACKSTRUCT

ソース・コード内の構造体、共用体、およびクラスのメンバーに対して使用する位置合わせ規則を指定します。 PACKSTRUCT は、構造体のメンバーおよび構造体自体に対して使用されるパッキング値を設定します。

デフォルトでデータ型が #pragma pack で指定されている境界よりも小さい境界に沿ってパックされている場合、それらのデータ型はそのまま小さい方の境界に沿って位置合わせされます。以下に例を示します。

- char 型は常に 1 バイト境界に沿って位置合わせされます。
- 16 バイト・ポインターは 16 バイト境界に沿って位置合わせされます。PACKSTRUCT、_Packed、および #pragma pack はこの位置合わせを変更することはできません。
- 8 バイト・ポインターは任意に位置合わせできますが、8 バイトの位置合わせをお勧めします。

パッキングおよび位置合わせについて詳しくは、 5-37 ページの『pack』プラグマを参照してください。

PACKSTRUCT 構文 :

***NATURAL**
デフォルト設定。構造体のメンバーに対する自然位置合わせが使用されます。

1 構造体および共用体は 1 バイト境界に沿ってパックされます。

2 構造体および共用体は 2 バイト境界に沿ってパックされます。

4 構造体および共用体は 4 バイト境界に沿ってパックされます。

8 構造体および共用体は 8 バイト境界に沿ってパックされます。

16 構造体および共用体は 16 バイト境界に沿ってパックされます。

ENUM

列挙型を表す場合にコンパイラーで使用されるバイト数を指定します。これは、オブジェクトのデフォルトの列挙型サイズになります。#pragma enum ディレクティブは、このコンパイラー・オプションをオーバーライドします。

ENUM 構文 :

***SMALL**
デフォルト設定。enum のできるだけ小さいサイズを、指定された enum 値に適した値として使用します。

1 すべての enum 変数を 1 バイトのサイズ (できれば符号付き) にします。

2 すべての enum 変数を 2 バイトのサイズ (できれば符号付き) にします。

4 すべての enum 変数を 4 バイトのサイズ (できれば符号付き) にします。

***INT**

- C ANSI C 規格の enum サイズ (4 バイトの符号付き) を使用します。
- C++ ANSI C++ 規格の enum サイズ (列挙型の値 > 2^{31}-1 でない場合、4 バイトの符号付き) を使用します。

MAKEDEP

Qshell make コマンドの記述ファイルに組み込みに適したターゲットを含む、出力ファイルを作成します。

MAKEDEP 構文 :

***NONE**
デフォルト設定。オプションが使用不可であり、ファイルは作成されません。

file-name
作成された出力ファイルのロケーションおよび名前を指示する IFS パスを指定します。

出力ファイルには、入力ファイルの行および各インクルード・ファイルの項目が含まれます。その汎用形式は、次のとおりです。

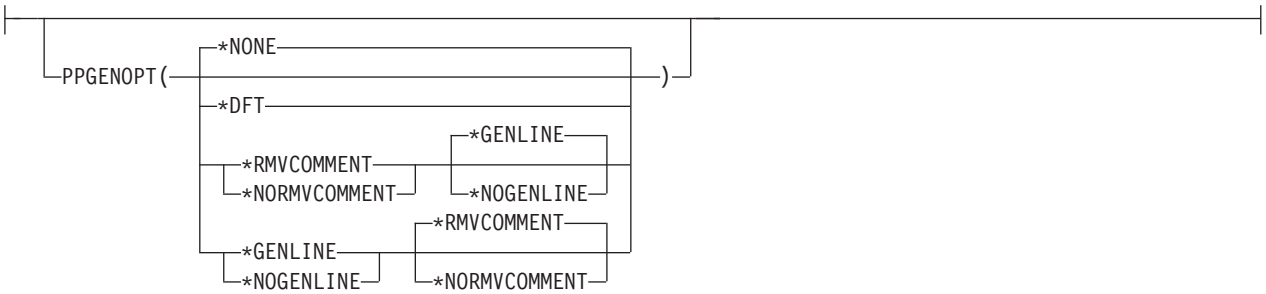
```
file_name.o:file_name.c
file_name.o:include_file_name
```

インクルード・ファイルは、#include プリプロセッサー・ディレクティブの検索順序規則に従ってリストされます。インクルード・ファイルが見つからない場合は、出力ファイルに追加されていません。include 文のないファイルは、入力ファイル名だけを 1 行にリストした出力ファイルを作成します。

PPGENOPT

CRTCMOD または *CRTCPPMOD* コマンドでのみ有効。 プリプロセッサーによって生成された出力を指定します。

PPGENOPT 構文 :



***NONE**
デフォルト設定。プリプロセッサーは出力を生成しません。このオプションを選択すると、PPSRCFILE、PPSRCMBR、および PPSRCSTMF オプションは無効になります。

***DFT**
PPGENOPT(*RMVCOMMENT *GENLINE) を指定することと等価です。

***RMVCOMMENT**
プリプロセス時のコメントを保存します。

***NORMVCOMMENT**
プリプロセス時のコメントを保存しません。

***NOGENLINE**
プリプロセッサー出力に #line ディレクティブを挿入しません。

***GENLINE**
プリプロセッサー出力に #line ディレクティブを作成します。

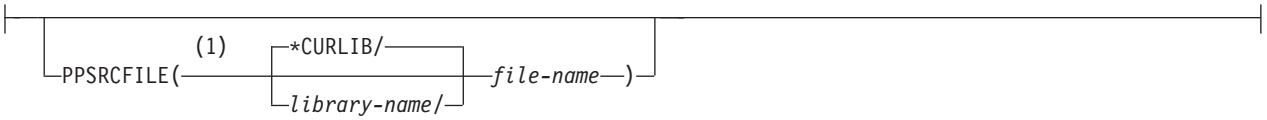
注:

1. PPGENOPT コンパイラー・オプションを *NONE 以外の設定で指定する場合は、以下のいずれかのオプションも入力する必要があります。
   - PPSRCFILE および PPSRCMBR
   - PPSRCSTMF
2. C PPGENOPT を *NONE 以外の設定で指定すると、OPTION(*NOPPONLY) および OPTION(*GEN) オプション設定がオーバーライドされます。
3. C OPTION(*PPONLY) は、PPGENOPT(*NONE) および OPTION(*GEN) オプション設定をオーバーライドします。代わりに、以下の設定が暗黙指定されます。
   - データ管理ソース・ファイルの場合は PPGENOPT(*DFT) PPSRCFILE(QTEMP/QACZEXPAND) PPSRCMBR(*MODULE)
   - IFS ソース・ファイルの場合は PPGENOPT(*DFT) PPSRCSTMF(*SRCSTMF)

PPSRCFILE

CRTCMOD または CRTCPPMOD コマンドでのみ有効。 このオプションは、PPGENOPT オプションと一緒に使用され、プリプロセッサー出力オブジェクトの保存先を定義します。

PPSRCFILE 構文 :



注:

1 「モジュールの作成」コマンドのみ

***CURLIB**
デフォルト設定。オブジェクトは現行ライブラリーに保存されます。ジョブに現行ライブラリーがない場合は、QGPL が使用されます。

library-name
プリプロセッサー出力の保存先のライブラリーの名前。

file-name
プリプロセッサー出力を保存する際の物理ファイル名。このファイルは、まだ存在していない場合に作成されます。

注:

1. PPSRCMBR および PPSRCFILE オプションを PPSRCSTMF オプションと一緒に指定することはできません。
2. C データ管理ファイルに OPTION(*PPONLY) を指定すると、以下の設定が暗黙指定されます。
   - PPGENOPT(*DFT) PPSRCFILE(QTEMP/QACZEXPAND) PPSRCMBR(*MODULE)

PPSRCMBR

CRTCMOD または *CRTCPPMOD* コマンドでのみ有効。 このオプションは、PPGENOPT オプションと一緒に使用され、プリプロセッサー出力の保存先であるメンバーの名前を定義します。

PPSRCMBR 構文 :

注:

1 「モジュールの作成」コマンドのみ

***MODULE**
MODULE パラメーターで提供されるモジュール名は、ソース・メンバー名として使用されます。これが、メンバー名が指定されていない場合のデフォルトです。

member-name
プリプロセッサー出力を含むメンバーの名前を入力します。

注:

1. PPSRCMBR および PPSRCFILE オプションを PPSRCSTMF オプションと一緒に指定することはできません。
2. C データ管理ファイルに OPTION(*PPONLY) を指定すると、以下の設定が暗黙指定されます。
   - PPGENOPT(*DFT) PPSRCFILE(QTEMP/QACZEXPAND) PPSRCMBR(*MODULE)

PPSRCSTMF

CRTCMOD または *CRTCPPMOD* コマンドでのみ有効。 このオプションは、PPGENOPT オプションと一緒に使用され、プリプロセッサー出力の保存先である IFS ストリーム・パス名を定義します。

PPSRCSTMF 構文 :

注:

1 「モジュールの作成」コマンドのみ

path-name
プリプロセッサー出力を含むファイルの IFS パスを入力します。パス名は、絶対修飾名と相対修飾名のどちらを指定することもできます。絶対パス名は「/」で始まり、相対パス名は「/」以外の文字で始まります。絶対修飾されている場合、このパス名は完全です。相対修飾の場合は、このパス名の先頭にジョブの現行作業ディレクトリーを付加することによって、パス名が完全なものになります。

***SRCSTMF**
この設定を選択した場合は、SRCSTMF コマンド・オプションも選択する必要があります。プリプロセッサー出力は、SRCSTMF コマンド・オプションで指定されているのと同じ基本ファイル名で現行ディレクトリーに保存されますが、ファイル名拡張子は **.i** になります。

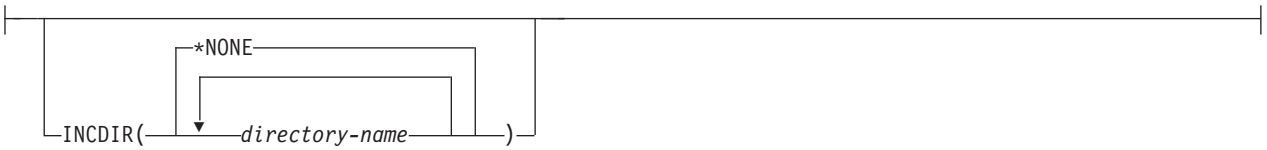
注:

1. PPSRCMBR および PPSRCFILE オプションを PPSRCSTMF オプションと一緒に指定することはできません。
2. 混合バイト環境では SRCSTMF パラメーターはサポートされていません。
3. C IFS ファイルに OPTION(*PPONLY) を指定すると、以下の設定が暗黙指定されます。
   - PPGENOPT(*DFT) PPSRCSTMF(*SRCSTMF)

INCDIR

ソース・ストリーム・ファイルをコンパイルするときにインクルード・ヘッダーを配置するのに使用されるパスを再定義してください。ソース・ファイルのロケーションが SRCSTMF コンパイラー・オプションで IFS パスとして定義されていない場合、または完全絶対パス名が #include ディレクティブに指定されている場合は、このオプションは無視されます。

INCDIR 構文 :



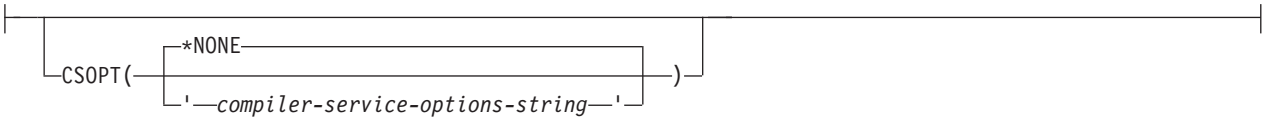
***NONE**
デフォルト設定。デフォルトのユーザー・インクルード・パスの開始時に挿入されるディレクトリーはありません。

directory-name
デフォルトのユーザー・インクルード・パスの開始時に挿入するディレクトリー名を指定します。複数のディレクトリー名を入力することができます。デフォルトのユーザー・インクルード・パスの開始時に入力した順序で、ディレクトリーが挿入されます。

CSOPT

このオプションで、1 つ以上のコンパイラー・サービス・オプションを指定できます。有効なオプション・ストリングが、PTF カバー・レターまたはリリース情報に記述されます。

CSOPT 構文 :



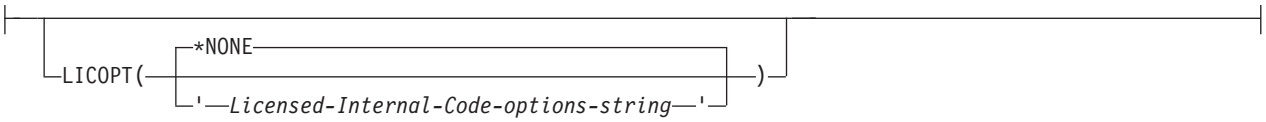
***NONE**
デフォルト設定。コンパイラー・サービス・オプションが選択されていません。

compiler-service-options-string
指定したコンパイラー・サービス・オプションは、モジュール・オブジェクトの作成中に使用されます。

LICOPT

1 つまたは複数のライセンス内部コード・コンパイル時オプションを指定します。このパラメーターによって、各コンパイル時オプションを選択することができます。このパラメーターは、選択した各タイプのコンパイラー・オプションの利点と欠点を理解している上級プログラマーを対象にしています。

LICOPT 構文 :



指定可能なオプションは次のとおりです。

***NONE**
デフォルト設定。コンパイル時の最適化は選択されません。

Licensed-Internal-Code-options-string
モジュール/プログラム・オブジェクトの作成時には、選択されたライセンス内部コードのコンパイル時オプションが使用されます。ある種のオプションでは、作成されたモジュール/プログラムのデバッグ能力が低下する場合があります。 LICOPT オプションの詳細については、*ILE 概念* を参照してください。

DFTCHAR

コンパイラーに、**char** 型のすべての変数を符号付き、または符号なしとして処理するよう指示します。

DFTCHAR 構文 :

***UNSIGNED**
デフォルト設定。**char** 型として宣言されたすべての変数を **符号なし char** 型として処理します。_CHAR_UNSIGNED マクロが定義されます。

***SIGNED**
char 型として宣言されたすべての変数を **符号付き char** 型として処理し、_CHAR_SIGNED マクロを定義します。 TGTRLS オプションが、V5R1M0 より前のターゲット・リリースを指定している場合、この設定は無視されます。

TGTCCSID

作成されたオブジェクトのターゲット・コード化文字セット ID (CCSID) を指定します。このオブジェクトの CCSID は、モジュールの文字データが保存されているコード化文字セット ID を示しています。これには、リテラルの記述に使用される文字データ、ソースによって記述されるコメントと ID 名 (CCSID 5026、930、および 290 の ID 名は除く) が含まれます。

C

ASCII CCSID が入力された場合、コンパイラーはエラー・メッセージを発行して、CCSID を 37 と想定します。

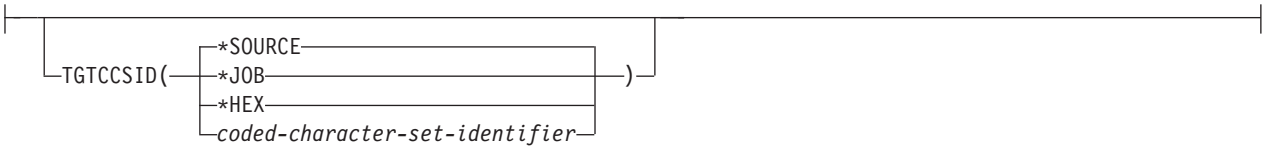
C++

ASCII CCSID が入力された場合、コンパイラーはエラー・メッセージを発行しません。 ASCII CCSID への変換が行われますが、作成されたモジュールの CCSID は 65535 になります。

TGTCCSID オプションは、リストで使用される文字値の CCSID も決定します。ただし、ジョブの CCSID はスプール・ファイルの CCSID であるため、スプール・ファイルに送られるリストはジョブの CCSID にあります。

V5R1 より前のリリース用のコンパイルをターゲットにしている場合、このオプションは無視されます。

TGTCCSID 構文 :



***SOURCE**
デフォルト設定。ルート・ソース・ファイルの CCSID が使用されます。

***JOB**
現行ジョブの CCSID が使用されます。

***HEX**
CCSID 65535 が使用されます。これは、文字データがビット・データとして扱われ、変換されないことを示しています。

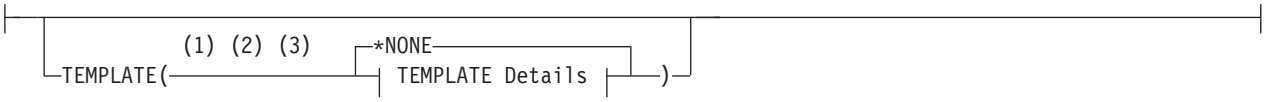
coded-character-set-identifier
使用する特定の CCSID を指定します。

TEMPLATE

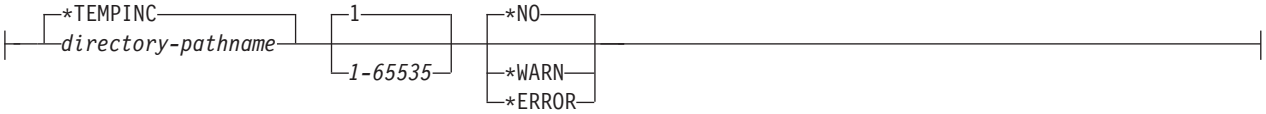
C++

C++ テンプレート生成をカスタマイズするためのオプションを指定します。

TEMPLATE 構文 :



TEMPLATE の詳細:



注:

1 C++ コンパイラーのみ

2 「モジュールの作成」コマンドのみ

3 統合ファイル・システム (IFS) を使用する場合にのみ適用可能

指定可能なオプションは次のとおりです。

***NONE**
自動テンプレート・インスタンス化ファイルは作成されません。テンプレートのフル実装が既知で、そのテンプレートのテンプレート・クラスのオブジェクトが定義されているか、またはモジュール内のそのテンプレートのテンプレート関数が呼び出される場合、コンパイラーはそのようなテンプレートをすべてインスタンス化します。フル実装が不明の場合 (例えば、テンプレート・クラス定義はあるが、そのテンプレート・クラスのメソッド定義がない場合)、モジュール内のそのテンプレートはインスタンス化されません。

注: このことにより、テンプレート仕様が複数のモジュールで使用されている実行可能プログラム内で、コードが重複することがあります。

***TEMPINC**
テンプレートは **tempinc** という名前のディレクトリー内に生成されます。このディレクトリーは、ルート・ソース・ファイルが含まれていたディレクトリー内に作成されています。ソース・ファイルがストリーム・ファイルでない場合は、TEMPINC という名前のファイルが、ソース・ファイルが含まれているライブラリーに作成されます。TEMPLATE(*TEMPINC) および TMPLREG オプションは、相互に排他的です。

directory-pathname
指定されたディレクトリー・ロケーションにテンプレート・インスタンス化ファイルが生成される点を除き、*TEMPLATE(*TEMPINC) と同じです。このディレクトリー・パスは、現行ディレクトリーを起点とした相対パスにすることも、あるいは絶対ディレクトリー・パスにすることもできます。

指定されたディレクトリーが存在していない場合は、ディレクトリーが作成されます。

注:
存在しないディレクトリーが、指定されたディレクトリー・パスに含まれている場合は (例えば、subdir1 が存在しない場合の TEMPLATE(/source/subdir1/tempinc)) エラー状態が発生します。

1-65535
ヘッダー・ファイルごとに *TEMPLATE(*TEMPINC) オプションによって生成されるテンプレート・インクルード・ファイルの最大数を指定します。指定しなかった場合、この設定は **1** にデフォルト設定されます。この設定の最大値は 65535 です。

***NO**
TEMPLATE(*NONE) が指定されていない場合のデフォルト設定。これを指定した場合は、前のバージョンのコンパイラー用に作成されたコードで発生したエラーの数を減少させるために、コンパイラーが構文解析を行うことはありません。

注: このオプションおよび次の 2 つのオプションの設定に関係なく、実装の外部で発生する問題に対してはエラー・メッセージが生成されます。例えば、以下のような構成体の構文解析中またはセマンティック検査中にエラーが検出されると、常にエラー・メッセージが発行されます。

- 関数テンプレートの戻りの型
- 関数テンプレートのパラメーター・リスト
- クラス・テンプレートのメンバー・リスト
- クラス・テンプレートの基本指定子

***WARN**
テンプレート実装を構文解析し、意味エラーに対して警告メッセージを発行します。エラー・メッセージは、構文解析中に検出されたエラーに対しても発行されます。

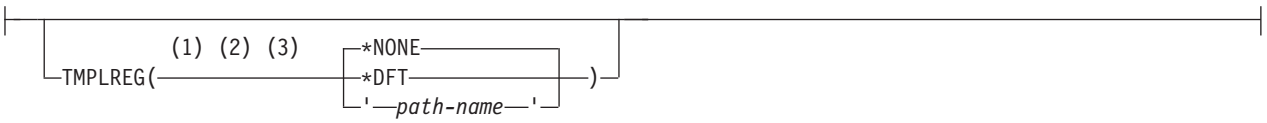
***ERROR**
テンプレートがインスタンス化されない場合でも、テンプレート実装内の問題をエラーとして扱います。

TMPLREG

C++

CRTCPPMOD コマンドでのみ有効。 すべてのテンプレートのレコードを、それらのテンプレートがソースで検出されたときのまま維持し、各テンプレートのインスタンス化が一度だけ行われることを保証します。TMPLREG および TEMPLATE(*TEMPINC) パラメーターは相互に排他的です。

TMPLREG 構文 :



注:

1 C++ コンパイラーのみ

2 「モジュールの作成」コマンドのみ

3 統合ファイル・システム (IFS) を使用する場合にのみ適用可能

指定可能なオプションは次のとおりです。

***NONE**
デフォルト設定。テンプレート情報の追跡にテンプレート・レジストリー・ファイルを使用しません。

***DFT**
ソース・ファイルがストリーム・ファイルの場合は、テンプレート・レジストリー・ファイルが、デフォルトの 'templateregistry' という名前で、ソース・ディレクトリーに作成されます。ソース・ファイルがストリーム・ファイルでない場合は、メンバー QTMPLREG を持つファイル QTMPLREG が、ソースが含まれているライブラリーに作成されます。

path-name
テンプレート・レジストリー情報の保存先のストリーム・ファイルのパス名を指定します。

WEAKTMPL

C++

テンプレート・クラスの静的メンバーに弱い定義を使用するかどうかを指定します。弱く定義された、テンプレート・クラスの静的メンバーは、1 つのプログラムまたはサービス・プログラム内での複数定義の衝突を回避します。

WEAKTMPL 構文 :

指定可能なオプションは次のとおりです。

***YES**
デフォルト設定。テンプレート・クラスの静的メンバーには弱い定義が使用されます。

***NO**
テンプレート・クラスの静的メンバーには弱い定義は使用されません。

一部のプログラムは、他のモジュールにリンクされている場合、強い静的データ・メンバーを要求します。デフォルトのオーバーライドは、コンパイル時にしか行えません。

DECFLTRND

定数 10 進浮動小数点式の評価のコンパイル時の丸めモードを指定します。このオプションは，実行時 10 進浮動小数点丸めモード (setca 組み込み関数を使用して設定される) には影響しません。

DECFLTRND 構文 :

指定可能なオプションは次のとおりです。

***HALFEVEN**
デフォルト設定。最も近い値に丸めます。中間の場合は、偶数にします。例えば、5.22 は 5.2 に、5.67 は 5.7 に、5.55 は 5.6 に、5.65 は 5.6 に丸めます。

***DOWN**
ゼロ方向に丸めるか、あるいは結果を切り捨てにします。例えば、5.22 は 5.2 に、5.67 は 5.6 に、5.55 は 5.5 に、5.65 は 5.6 に丸めます。

***UP**
ゼロから離れるように丸めます。例えば、5.22 は 5.3 に、5.67 は 5.7 に、5.55 は 5.6 に、5.65 は 5.7 に丸めます。

***HALFUP**
最も近い値に丸めます。中間の場合は、ゼロから離れるように丸めます。例えば、5.22 は 5.2 に、5.67 は 5.7 に、5.55 は 5.6 に、5.65 は 5.7 に丸めます。

***HALFDOWN**
最も近い値に丸めます。中間の場合は、ゼロ方向に丸めます。例えば、5.22 は 5.2 に、5.67 は 5.7 に、5.55 は 5.5 に、5.65 は 5.6 に丸めます。

***FLOOR**
負のアフィニティー方向に丸めます。例えば、5.22 は 5.2 に、5.67 は 5.6 に、5.55 は 5.5 に、5.65 は 5.6 に丸めます。

***CEILING**
正の無限大方向に丸めます。例えば、5.22 は 5.3 に、5.67 は 5.7 に、5.55 は 5.6 に、5.65 は 5.7 に丸めます。

ixlc コマンドを使用した C/C++ コンパイラーの起動

このセクションでは、ixlc Qshell コマンドの概要を説明します。

ixlc コマンドを使用することで、コンパイラーを起動し、IBM i Qshell コマンド行からコンパイラー・オプションを指定することができます。モジュール・バインダー・コマンドを指定することができます。ixlc コマンドを AIX の Make ファイルと一緒に使用して、コンパイルを制御することができます。

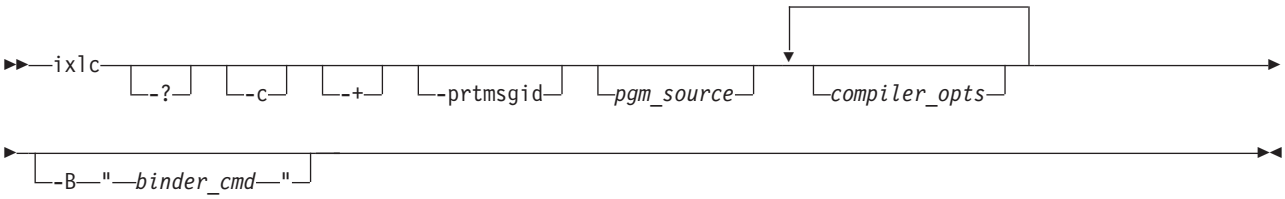
Qshell での ixlc の使用

文字ベース・インターフェース Qshell コマンド行で **ixlc** の IBM i 版を使用すると、以下のことが可能になります。

- IBM i プラットフォームに常駐しているデータ管理ソース・コードをコンパイルする。
- IBM i プラットフォームに常駐している IFS ソース・コードをコンパイルする。
- IBM i プラットフォームに常駐しているヘッダー・ファイルを使用する。

ixlc コマンドとオプションの構文

ixlc コマンドの基本的な構文は以下のとおりです。



各値は、次のとおりです。

ixlc 基本的なコンパイラー・コマンドの呼び出し。デフォルトでは、**ixlc** コマンドはコンパイラーに、バインド済みプログラムの作成を指示します。

-? このフラグを指定すると、ixlc コマンドのヘルプが表示されます。

-c このフラグを指定すると、コンパイラーにモジュールの作成を指示します。

-+ このフラグを指定すると、C++ コンパイラーが起動されます。

-prtmsgid
このフラグを指定すると、コンパイル・エラー・メッセージの追加情報が表示されます。追加情報には、行番号、列番号、メッセージ ID、およびメッセージ重大度が含まれます。

pgm_source
コンパイル中のプログラム・ソース・ファイルの名前を指定します。以下のようにソースの名前を指定することで、IFS ソース・プログラムまたはデータ管理ソース・プログラムをコンパイルすることができます。

```
qsys.lib/.../name.mbr
```

また、-qsrcfile(*library*/*file*) および -qsrcmbr(*member*) の Qshell コンパイラー・オプションを使用してプログラム・ソースの位置を識別することによって、データ管理ソース・プログラムをコンパイルすることも可能です。

compiler_opts
ILE C/C++ コンパイラー・オプションの ixlc 名を指定します。

-B"*binder_cmd***"**
バインダー・コマンドおよびオプションを指定します。例えば、以下のようになります。

```
-B"CRTPGM PGM(library/target) MODULE(...)"
```

使用に関する注意

1. ixlc コマンドおよびオプションは、大/小文字が区別されます。
2. コンパイラーの起動時には、競合するオプションを指定することが可能です。この場合、コマンド行に後で指定されたオプションは、前に指定されたオプションをオーバーライドします。例えば、以下のように指定してコンパイラーを起動するとします。

   ```
   ixlc hello.c -qgen -qnogen
   ```

   これは、以下のように指定するのと同じことです。

   ```
   ixlc hello.c -qnogen
   ```
3. 一部のオプション設定は累積されるため、前に指定された同じオプションを取り消さずに、コマンド行において何度も指定することができます。このようなオプションには、以下のものがあります。
   - OPTION コンパイラー・オプション・グループ内の設定
   - CHECKOUT コンパイラー・オプション・グループ内の設定
   - ALIAS コンパイラー・オプション
   - DEFINE コンパイラー・オプション
   - PPGENOPT コンパイラー・オプション

ixlc コマンド・オプション

以下の表は、「モジュールの作成」および「バインド済みプログラムの作成」コンパイラー・オプションとそれに相当する **ixlc** のマッピングを示しています。コンパイラー・オプションには、この表には示されていない言語および使用に関する制約事項がある場合があります。そのような制約事項については、そのオプションの参照情報を参照してください。

表 7-1. ixlc コマンド・オプション

モジュールの作成/バインド済みプログラムの作成オプション	オプション設定	相当する ixlc とその注
6-7 ページの『MODULE』, 6-8 ページの『PGM』	[*CURLIB/ | *libraryname*/]*name*	-o[*CURLIB/ | *libraryname*/]*name*
	ライブラリーが指定されていない場合、ターゲット・オブジェクトは現在のユーザー・プロファイルによって指定されている現行ライブラリーに送られます。そのユーザーに現行ライブラリーがない場合は、QGPL となります。	
6-8 ページの『SRCFILE』	[*LIBL/ | *CURLIB/ | *libraryname*/] *filename*	-qsrcfile=[*LIBL/ | *CURLIB/ | *libraryname*/] *filename*
6-9 ページの『SRCMBR』	*MODULE | *mbrname*	-qsrcmbr=*mbrname*
6-10 ページの『SRCSTMF』	*pathname*	(*なし、デフォルトのパス名を使用*)
6-10 ページの『TEXT』	*SRCMBRTEXT | *BLANK | *text*	-qtext="*text*"

表 7-1. *ixlc* コマンド・オプション *(続き)*

モジュールの作成/バインド済みプログラムの作成オプション	オプション設定	相当する **ixlc** とその注
6-10 ページの『OUTPUT』	*NONE	-qnoprint
	*PRINT	-qprint
	filename	-qoutput="*filename*"
6-12 ページの『OPTION』	*AGR | *NOAGR	-qagr
	*BITSIGN | *NOBITSIGN	-qbitfields=signed -qbitfields=unsigned
	*DIGRAPH | *NODIGRAPH	-qdigraph -qnodigraph
	*EVENTF | *NOEVENTF	-qeventf -qnoeventf
	*EXPMAC | *NOEXPMAC	-qexpmac -qnoexpmac
	*FULL | *NOFULL	-qfull -qnofull
	*GEN | *NOGEN	-qgen -qnogen
	*INCDIRFIRST | *NOINCDIRFIRST	-qidirfirst
	*LOGMSG | *NOLOGMSG	-qlogmsg -qnologmsg
	*LONGLONG | *NOLONGLONG	-qlonglong -qnolonglong
	*NORTTI | *RTTIALL | *RTTITYPE | *RTTICAST	-qnortti -qrtti=all -qrtti=typeinfo -qrtti=dynamiccast
	*PPONLY | *NOPPONLY	-qpponly
	*SECLVL | *NOSECLVL	-qseclvl -qnoseclvl
	*SHOWINC | *NOSHOWINC	-qshowinc -qnoshowinc
	*SHOWSKP | *NOSHOWSKP	-qshowskp -qnoshowskp
	*SHOWSRC | *NOSHOWSRC	-qsource -qnosource
	*SHOWSYS | *NOSHOWSYS	-qshowsys -qnoshowsys
	*SHOWUSR | *NOSHOWUSR	-qshowusr
	*STDINC | *NOSTDINC	-qstdinc -qnostdinc
	*STDLOGMSG | *NOSTDLOGMSG	-qstdlogmsg -qnostdlogmsg
	*STRUCREF | *NOSTRUCREF	-qrefagr
	*SYSINCPATH | *NOSYSINCPATH	-qsysincpath -qnosysincpath
	*XREF | *NOXREF	-qxref=full -qxref
	*XREFREF | *NOXREFREF	-qattr=full -qattr

表 7-1. ixlc コマンド・オプション (続き)

モジュールの作成/バインド済みプログラムの作成オプション	オプション設定	相当する ixlc とその注
6-17 ページの『CHECKOUT』	*NONE | *USAGE | *ALL	-qinfo=cnd -qinfo=all
	*CLASS | *NOCLASS	-qinfo=cls
	*COND | *NOCOND	-qinfo=cnd
	*CONST | *NOCONST	-qinfo=cns
	*EFFECT | *NOEFFECT	-qinfo=eff
	*ENUM | *NOENUM	-qinfo=enu
	*EXTERN | *NOEXTERN	-qinfo=ext
	*GENERAL | *NOGENERAL	-qinfo=gen
	*GOTO | *NOGOTO	-qinfo=got
	*INIT | *NOINIT	-qinfo=ini
	*LANG | *NOLANG	-qinfo=lan
	*PARM | *NOPARM	-qinfo=par
	*PORT | *NOPORT	-qinfo=por
	*PPCHECK | *NOPPCHECK	-qinfo=ppc
	*PPTRACE | *NOPPTRACE	-qinfo=ppt
	*REACH | *NOREACH	-qinfo=rea
	*TEMP | *NOTEMP	-qinfo=gnr
	*TRUNC | *NOTRUNC	-qinfo=trd
	*UNUSED | *NOUNUSED	-qinfo=use
6-20 ページの『OPTIMIZE』	10 | 20 | 30 | 40	-qoptimize=10 -qoptimize=20 -qoptimize=30 -qoptimize=40 -O
		-O は、-qoptimize=40 の指定に相当します。

表 7-1. ixlc コマンド・オプション (続き)

モジュールの作成/バインド済みプログラムの作成オプション	オプション設定	相当する ixlc とその注
6-21 ページの『INLINE』	*OFF	-qnoinline
	*ON *AUTO | *NOAUTO 250 | 1-65535 | *NOLIMIT 2000 | 1-65535 | *NOLIMIT *NO | *YES	-qinline="*opt1 opt2 opt3 opt4*" 各値は、次のとおりです。 • *opt1* は次のいずれかです。 – auto – noauto • *opt2* は次のいずれかです。 – 250 – *1–65536* – *NOLIMIT • *opt3* は次のいずれかです。 – 2000 – *1–65536* – *NOLIMIT • *opt4* は次のいずれかです。 – norpt – rpt
		それぞれのオプション・グループから 1 つを選択して指定する必要があります。また、選択したそれぞれを、スペースで区切ることも必要です。以下に例を示します。 -qinline="auto 400 3000 rpt"
6-22 ページの『MODCRTOPT』	*KEEPILDATA | *NOKEEPILDATA	-qildta -qnoildta
6-23 ページの『DBGVIEW』	*NONE | *ALL | *STMT | *SOURCE | *LIST	-qdbgview=none -qdbgview=all -qdbgview=stmt -qdbgview=source -qdbgview=list -g
		-g は、-qdbgview=all の指定に相当します。
6-24 ページの『DBGENCKEY』	*NONE | *character value*	-qdbgenckey=string
6-24 ページの『DEFINE』	*NONE | *name* | *name=value*	-D*name*
		name を値 1 で定義します。
6-25 ページの『LANGLVL』	*EXTENDED | *ANSI | *LEGACY | *EXTENDED0X	-qlanglvl=extended -qlanglvl=ansi -qlanglvl=compat366 -qlanglvl=extended0x

表 7-1. *ixlc* コマンド・オプション (続き)

モジュールの作成/バインド済みプログラムの作成オプション	オプション設定	相当する ixlc とその注
6-26 ページの『ALIAS』	*ANSI | *NOANSI | *ADDRTAKEN | *NOADDRTAKEN | *ALLPTRS | *NOALLPTRS | *TYPEPTR | *NOTYPEPTR	-qalias=ansi -qalias=noansi -qalias=addrtaken -qalias=noaddrtaken -qalias=allptrs -qalias=noallptrs -qalias=typeptr -qalias=notypeptr
6-27 ページの『SYSIFCOPT』	*NOIFSIO | **IFSIO | *IFS64IO	-qnoifsio -qifsio -qifsio=64
	*ASYNCSIGNAL | *NOASYNCSIGNAL	-qasyncsignal -qnoasyncsignal
6-28 ページの『LOCALETYPE』	*LOCALE | *LOCALEUCS2 | *LOCALEUTF | *CLD	-qlocale=locale -qlocale=localeucs2 -qlocale=localeutf -qlocale=cld
6-28 ページの『FLAG』	0 | 10 | 20 | 30	-qflag=0 -qflag=10 -qflag=20 -qflag=30
6-29 ページの『MSGLMT』	*NOMAX | *0-32767* 30 | 0 | 10 | 20	-qmsglmt="*limit severity*"。この中で、*limit* は *nomax か、*0* から *32767* までの間の任意の整数で、 *severity* は 0、10、20、または 30 のいずれかになります。デフォルトは -qmsglmt="*nomax 30" です。
6-29 ページの『REPLACE』	*YES | *NO	-qreplace -qnoreplace
6-29 ページの『USRPRF』	*USER | *OWNER	-quser -qowner
6-30 ページの『AUT』	*LIBCRTAUT | *CHANGE | *USE | *ALL | *EXCLUDE	-qaut=libcrtaut -qaut=change -qaut=use -qaut=all -qaut=exclude
6-31 ページの『TGTRLS』	*CURRENT | *PRV | *release_lvl*	-qtgtrls=*current -qtgtrls=*prv -qtgtrls=V*x*R*x*M*x*
6-32 ページの『ENBPFRCOL』	*PEP	-qenbpfrcol=pep
	*ENTRYEXIT *NONLEAF	-qenbpfrcol=entryexitnonleaf
	*ENTRYEXIT *ALLPRC	-qenbpfrcol=entryexitallprc
	*FULL *NONLEAF	-qenbpfrcol=fullnonleaf
	*FULL *ALLPRC	-qenbpfrcol=fullallprc
6-33 ページの『PFROPT』	*SETFPCA | *NOSETFPCA	-qsetfpca -qnosetfpca
	*NOSTRDONLY | *STRDONLY	-qnoro -qro

表 7-1. ixlc コマンド・オプション (続き)

モジュールの作成/バインド済みプログラムの作成オプション	オプション設定	相当する ixlc とその注
6-34 ページの『PRFDTA』	*NOCOL | *COL	-qnoprofile -qprofile
		-qprfdta=*NOCOL -qprfdta=*COL
6-34 ページの『TERASPACE』	*NO	-qteraspace=no
	*YES *NOTSIFC	-qteraspace=notsifc
	*YES *TSIFC	-qteraspace=tsifc
6-38 ページの『STGMDL』	*SNGLVL | *TERASPACE | *INHERIT	-qstoragemodel=snglvl -qstoragemodel=teraspace -qstoragemodel=inherit
6-39 ページの『DTAMDL』	*P128 | *LLP64	-qdatamodel=P128 -qdatamodel=LLP64
6-39 ページの『RTBND』	*DEFAULT | *LLP64	-qrtbnd -qrtbnd=llp64
6-40 ページの『PACKSTRUCT』	1 | 2 | 4 | 8 | 16 | *NATURAL	-qalign=1 -qalign=2 -qalign=4 -qalign=8 -qalign=16 -qalign=natural
6-40 ページの『ENUM』	1 | 2 | 4 | *INT | *SMALL	-qenum=1 -qenum=2 -qenum=4 -qenum=int -qenum=small
6-41 ページの『MAKEDEP』	*NODEP | *filename*	-M*makefile*
6-41 ページの『PPGENOPT』	*NONE | *DFT	-P
	*RMVCOMMENT | *NORMVCOMMENT	-qppcomment -qnoppcomment
	*GENLINE | *NOGENLINE	-qppline -qnoppline
6-42 ページの『PPSRCFILE』	*CURLIB/*filename*	-qppsrcfile=*CURLIB/*filename*
	libraryname/filename	-qppsrcfile=*libraryname/filename*
	filename	-qppsrcfile=*filename*
6-43 ページの『PPSRCMBR』	*MODULE | *mbrname*	-qppsrcmbr=*module -qppsrcmbr=*mbrname*
6-43 ページの『PPSRCSTMF』	*pathname* | *SRCSTMF	-qppfile=*filename* -qppfile=*srcstmf
6-44 ページの『INCDIR』	*NONE | *pathname*	-I*pathname*
	コマンド行で使用する場合は、IBM i プラットフォーム上のディレクトリーを指定します。インクルード環境変数は上書きされます。	
6-44 ページの『CSOPT』	*string*	-qcsopt=*string*
6-45 ページの『LICOPT』	*NONE | *string*	-qlicopt=*string*
6-45 ページの『DFTCHAR』	*SIGNED | *UNSIGNED	-qchar=signed -qchar=unsigned
6-46 ページの『TGTCCSID』	*SOURCE | *JOB | *HEX | *ccsid#*	-qtgtccsid=source -qtgtccsid=job -qtgtccsid=hex -qtgtccsid=*ccsid#*

表 7-1. *ixlc* コマンド・オプション (続き)

モジュールの作成/バインド済みプログラムの作成オプション	オプション設定	相当する ixlc とその注
6-46 ページの『TEMPLATE』	*NONE | *pathname*	-qnotempinc -qtempinc=*pathname*
	1 - 65535	-qtempmax=*1-65535*
	*NO | *WARN | *ERROR	-qtmplparse=no -qtmplparse=warn -qtmplparse=error
6-48 ページの『TMPLREG』	*DFT | *NONE	-qtmplreg -qnotmplreg
6-49 ページの『WEAKTMPL』	*YES | *NO	-qweaktmpl -qnoweaktmpl
6-49 ページの『DECFLTRND』	*HALFEVEN | *DOWN | *UP | *HALFUP | *HALFDOWN | *FLOOR | *CEILING	-ydn -ydz -ydi -ydna -ydnz -ydm -ydp

I/O 考慮事項

このセクションでは、I/O 考慮事項の概要を説明します。

このセクションでは、以下の内容について説明します。
- レコード・ファイルでのデータ管理機能操作
- ストリーム・ファイルでのデータ管理機能操作
- C ストリームおよびファイル・タイプ
- DDS から C/C++ へのデータ・タイプ・マッピング

レコード・ファイルでのデータ管理機能操作

レコード・ファイルで使用可能なデータ管理機能操作および ILE C/C++ 関数の詳細については、次の IBM i Information Center Web サイトの「*ファイルおよびファイル・システム*」カテゴリーの『データベース・ファイル管理』セクションを参照してください。

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

ストリーム・ファイルでのデータ管理機能操作

レコード入出力関数でストリーム・ファイル (型=レコード) を使用するには FILE ポインターを RFILE ポインターにキャストする必要があります。

ストリーム・ファイルで使用可能なデータ管理機能操作および ILE C/C++ 関数の詳細については、次の IBM i Information Center Web サイトの「*ファイルおよびファイル・システム*」カテゴリーの『データベース・ファイル管理』セクションを参照してください。

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

C ストリームおよびファイル・タイプ

以下の表は、どのファイル・タイプがストリームとしてサポートされているかを要約しています。

表 8-1. C ストリームおよびファイル・タイプの処理

ストリーム	データベース	ディスケット	テープ	プリンター	ディスプレイ	ICF	DDM	保管
TEXT	あり	なし	なし	あり	なし	なし	あり	なし
BINARY: 文字を一度に処理	あり	なし	なし	あり	なし	なし	あり	なし
BINARY: レコードを一度に処理	あり	あり	あり	あり	あり	あり	あり	あり

DDS から C/C++ へのデータ・タイプ・マッピング

以下の表は、DDS データ・タイプおよび外部記述ファイルから ILE C/C++ プログラムへフィールドをマップするために使用される、対応する ILE C/C++ 宣言を示しています。 ILE C/C++ コンパイラーは、外部記述ファイルの DDS データ・タイプに基づいて構造体定義にフィールドを作成します。

表 8-2. DDS から C/C++ へのデータ・タイプ・マッピング

DDS データ・タイプ	長さ	小数点以下の桁数	C/C++ 宣言
標識	1	0	char INxx_INyy[n]; 未使用標識 xx から yy 用 char INxx; 使用標識 xx 用
A - 英数字	1-32766	none	char field[n]; (ここで n は 1 から 32766)
A - 英数字可変長 VARLEN キーワード	1-32740	none	_Packed struct { short len; char data[n]; } field; ここで n はフィールドの最大長
B - 2 進	1-4	0	short int field;
B - 2 進	1-4	1-4	char field[2];
B - 2 進	5-9	0	int field;
B - 2 進	5-9	1-9	char field[4];
H - 16 進	1	none	char field;
H - 16 進	2-32766	none	char field[n]; (ここで n は 2 から 32766)
H - 16 進可変長 VARLEN キーワード	1-32740	none	_Packed struct { short len; char data[n]; } field; ここで n はフィールドの最大長
G - グラフィック可変長 VARLEN キーワード	4-1000	none	_Packed struct { short len; wchar_t data[n]; } field; (ここで n は 4 から 1000)
P - パック 10 進	1-31	0-31	decimal (n,p) ここで n は長さで p はオプション d での小数点以下の桁数
S - ゾーン 10 進数	1-31	0-31	char field[n]; (ここで n は 1 から 31)
F - 浮動小数点	[1]	[1]	float field;
F - 浮動小数点	[1]	[1]	double field;
J - DBCS only	4-32766	none	char field[n]; (ここで n は 4 から 32766 の偶数)
E - DBCS either	4-32766	none	char field[n]; (ここで n は 4 から 32766 の偶数)
O - DBCS open	4-32766	none	char field[n]; (ここで n は 4 から 32766)
J - DBCS only 可変長 VARLEN キーワード	4-32740	none	_Packed struct { short len; char data[n]; } field; (ここで n は 4 から 32740 の偶数)
E - DBCS either 可変長 VARLEN キーワード	4-32740	none	_Packed struct { short len; char data[n]; } field; (ここで n は 4 から 32740 の偶数)
O - DBCS open 可変長 VARLEN キーワード	4-32740	none	_Packed struct { short len; char data[n]; } field; (ここで n は 4 から 32740)
T - 時間	8	none	char field[8];
L - 日付	6、8、または 10	none	char field[n]; (ここで n は 6、8、または 10)
Z - タイム・スタンプ	26	none	char field[26];

[1] C 宣言 (浮動小数点または倍精度) は DDS の FLTPCN (浮動小数点精度) キーワードで指定されるものに基づきます。*SINGLE (デフォルト) は浮動小数点、*DOUBLE は倍精度。

詳細情報については、以下の IBM i Information Center Web サイトの『*DDS 解説書*』を参照してください。

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

制御文字

このセクションでは、制御文字の内部 16 進数表記についての詳細な情報を提供します。

以下の表は、ILE C/C++ コンパイラーおよびライブラリーによって使用されるオペレーティング・システム制御シーケンスの、内部 16 進数表記を示しています。

表 9-1. 内部 16 進数表記

印刷表記	内部表記
NUL (ヌル)	0x00
SOH (ヘッディング開始)	0x01
STX (テキスト開始)	0x02
ETX (テキスト終結)	0x03
SEL (選択)	0x04
HT (水平タブ)	0x05
RNL (必須の改行)	0x06
DEL (削除)	0x07
GE (図形エスケープ)	0x08
SPS (スーパースクリプト)	0x09
RPT (繰り返し)	0x0a
VT (垂直タブ)	0x0b
FF (用紙送り)	0x0c
CR (復帰)	0x0d
SO (シフトアウト)	0x0e
SI (シフトイン)	0x0f
DLE (伝送制御拡張)	0x10
DC1 (装置制御 1)	0x11
DC2 (装置制御 2)	0x12
DC3 (装置制御 3)	0x13
RES/ENP (復元または使用可能表示)	0x14
NL (改行)	0x15
BS (バックスペース)	0x16
POC (プログラム・オペレーター通信)	0x17
CAN (取り消し)	0x18
EM (メディア終端)	0x19
UBS (ユニット・バックスペース)	0x1a
CU1 (顧客使用 1)	0x1b
IFS (ファイル・セパレーターの交換)	0x1c
IGS (グループ・セパレーターの交換)	0x1d
IRS (レコード・セパレーターの交換)	0x1e

表 9-1. 内部 16 進数表記 (続き)

印刷表記	内部表記
IUS/ITB (ユニット・セパレーターの交換/中間伝送ブロック終結)	0x1f
DS (数字選択)	0x20
SOS (重要度の開始)	0x21
FS (フィールド・セパレーター)	0x22
WUS (ワード下線)	0x23
BYP/INP (バイパスまたは禁止表示)	0x24
LF (改行)	0x25
ETB (伝送終結ブロック)	0x26
ESC (エスケープ)	0x27
SA (属性設定)	0x28
SM/SW (設定モードまたは切り替え)	0x2a
CSP (制御シーケンス接頭部)	0x2b
MFA (フィールド属性の修正)	0x2c
ENQ (問い合わせ)	0x2d
ACK (認識)	0x2e
BEL (ベル)	0x2f
SYN (同期信号)	0x32
IR (指標戻り)	0x33
PP (表示位置)	0x34
TRN	0x35
NBS (数値バックスペース)	0x36
EOT (伝送終了)	0x37
SBS (下付き文字)	0x38
IT (インデント・タブ)	0x39
RFF (必須の用紙送り)	0x3a
CU3 (顧客使用 3)	0x3b
DC4 (装置制御 4)	0x3c
NAK (否定応答)	0x3d
SUB (置換)	0x3f
(ブランク文字)	0x40

関連情報

このセクションでは、関連するトピックの情報を提供します。

ILE C/C++ プログラミングに関連するトピックの詳細は、以下の IBM 資料を参照してください。

- IBM i Information Web サイトの「プログラミング」カテゴリー内の『CL プログラミング (SD88-5038-06)』セクションでは、IBM i のプログラミングのトピックに関するさまざまな情報を提供しています。トピックには、オブジェクトとライブラリー、CL プログラミング、プログラム間の制御の流れと連絡、CL プログラムでのオブジェクトの処理、および CL プログラムの作成についての概要説明が含まれます。その他のトピックとしては、事前定義メッセージと即時メッセージおよびメッセージの処理、ユーザー定義のコマンドとメニューの定義と作成、デバッグ・モード、ブレークポイント、追跡、および表示機能を含むアプリケーションのテストなどが入っています。
- GDDM® Programming Guide (SC41-0536-00) では、IBM i 図形データ表示管理プログラム (GDDM) を使用してグラフィックス・アプリケーション・プログラムを作成する方法について説明します。多くのプログラム例およびこのプロダクトをデータ処理システムに適合させる方法を理解するのに役立つ情報が含まれています。
- GDDM Reference (SC41-3718-00) では、IBM i 図形データ表示管理プログラム (GDDM) を使用してグラフィックス・アプリケーション・プログラムを作成する方法について説明します。この資料では、GDDM で使用可能なすべてのグラフィック関数が詳細に説明されています。さらに、GDDM に対する高水準言語インターフェースに関する情報を提供します。
- IBM Rational® Development Studio for i: ILE C/C++ プログラマーの手引き(SC09-2712-07) では、ILE C/C++ コンパイラーに関するプログラミング情報を提供します。言語をまたがるプログラムとプロシージャー呼び出し、ロケール、例外処理、データベース、および装置ファイルのプログラミング上の考慮事項が解説されています。例が示されており、プログラミングのパフォーマンス上のヒントも説明されています。
- プログラミング IBM Rational Development Studio for i: ILE C/C++ 解説書 (SC09-7852-03) では、言語のエレメント、ステートメント、およびプリプロセッサー・ディレクティブなど、ILE C/C++ コンパイラーに関する参照情報を提供しています。例が示されており、プログラミングの考慮事項も説明されています。
- ILE C/C++ ランタイム・ライブラリー関数 (SC41-5607-05) では、標準 C ライブラリー関数および ILE C/C++ ライブラリー拡張機能などの ILE C/C++ ライブラリー関数に関する参照情報を提供しています。例が示されており、プログラミングの考慮事項も説明されています。
- ILE 概念 (SD88-5033-08) では、IBM i の統合 Language Environment アーキテクチャーに関する概念および用語について説明しています。扱われるトピックには、モジュールの作成、バインディング、プログラムの実行、プログラムのデバッグ、および例外処理が含まれています。
- IBM i Information Web サイトの「プログラミング」カテゴリー内の『アプリケーション・プログラミング・インターフェース』セクションでは、熟練のアプリケーション・プログラマーおよびシステム・プログラマーに対して、アプリケーション・プログラミング・インターフェース (API) の使用方法について説明しています。プログラマーが API を使用するときに役立つ入門情報および例を提供します。

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。

IBM Corporation
Software Interoperability Coordinator, Department YBWA
3605 Highway 52 N
Rochester, MN 55901
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。

本書はプランニング目的としてのみ記述されています。記述内容は製品が使用可能になる前に変更になる場合があります。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。これらのサンプル・プログラムは特定物として現存するままの状態で提供されるものであり、いかなる保証も提供されません。 IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (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. _年を入れる_.

プログラミング・インターフェース情報

この ILE C/C++ コンパイラー参照資料には、プログラムを作成するユーザーが IBM i のサービスを使用するためのプログラミング・インターフェースが記述されています。

商標

IBM、IBM ロゴおよび ibm.com は、世界の多くの国で登録された International Business Machines Corporation の商標です。他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。現時点での IBM の商標リストについては、『www.ibm.com/legal/copytrade.shtml』 をご覧ください。

Adobe、Adobe ロゴ、PostScript、PostScript ロゴは、Adobe Systems Incorporated の米国およびその他の国における登録商標または商標です。

Microsoft、Windows、Windows NT および Windows ロゴは、Microsoft Corporation の米国およびその他の国における商標です。

UNIX は The Open Group の米国およびその他の国における登録商標です。

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。

他の製品名およびサービス名等は、それぞれ IBM または各社の商標である場合があります。

使用条件

これらの資料は、以下の条件に同意していただける場合に限りご使用いただけます。

個人使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、非商業的な個人による使用目的に限り複製することができます。ただし、IBM の明示的な承諾をえずに、これらの資料またはその一部について、二次的著作物を作成したり、配布 (頒布、送信を含む) または表示 (上映を含む) することはできません。

商業的使用: これらの資料は、すべての著作権表示その他の所有権表示をしていただくことを条件に、お客様の企業内に限り、複製、配布、および表示することができます。 ただし、IBM の明示的な承諾をえずにこれらの資料の二次的著作物を作成したり、お客様の企業外で資料またはその一部を複製、配布、または表示することはできません。

ここで明示的に許可されているもの以外に、資料や資料内に含まれる情報、データ、ソフトウェア、またはその他の知的所有権に対するいかなる許可、ライセンス、または権利を明示的にも黙示的にも付与するものではありません。

資料の使用が IBM の利益を損なうと判断された場合や、上記の条件が適切に守られていないと判断された場合、IBM はいつでも自らの判断により、ここで与えた許可を撤回できるものとさせていただきます。

お客様がこの情報をダウンロード、輸出、または再輸出する際には、米国のすべての輸出入関連法規を含む、すべての関連法規を遵守するものとします。

IBM は、これらの資料の内容についていかなる保証もしません。これらの資料は、特定物として現存するままの状態で提供され、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任なしで提供されます。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[カ行]

[サ行]

[タ行]

[ハ行]

［マ行］

A

C

D

E

H

I

L

M

N

P

Q

S

W

［特殊文字］

プログラム番号: 5770-WDS

Printed in USA

SC88-4025-03

日本アイ・ビー・エム株式会社
〒103-8510 東京都中央区日本橋箱崎町19-21