IBM XL Fortran for Linux, V16.1

コンパイラー・リファレンス
(リトル・エンディアン・ ディストリビューション用)

バージョン ***16.1***

SC43-4663-00

IBM XL Fortran for Linux, V16.1

コンパイラー・リファレンス
(リトル・エンディアン・ ディストリビューション用)

バージョン ***16.1***

SC43-4663-00

お願い

本書および本書で紹介する製品をご使用になる前に、 389 ページの『特記事項』に記載されている情報をお読みください。

本書は、IBM XL Fortran for Linux, V16.1 (プログラム 5765-J10; 5725-C75)、および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。正しいレベルの製品をご使用になるようお確かめください。

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： SC27-8050-00
IBM XL Fortran for Linux, V16.1
Compiler Reference
for Little Endian Distributions
Version 16.1
First edition

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

目次

本書について

本書は、IBM® XL Fortran for Linux, V16.1 コンパイラーについて記述し、コンパイル環境の設定方法、ならびに Fortran 言語で作成されたプログラムのコンパイル、リンク、および実行の方法について説明します。本書には、一連の XL Fortran 資料の他の参照資料の関連トピックへの相互参照も含まれています。

本書の対象読者

本書は、IBM XL Fortran for Linux, V16.1 コンパイラーを操作するすべてのユーザー、Linux オペレーティング・システムに精通しているすべてのユーザー、およびこれまでに Fortran のプログラミングを多少なりとも経験したことがあるすべてのユーザーを対象としています。 XL Fortran を初めて使用するユーザーでも、XL Fortran に固有の機能に関する情報を見つけることができます。本書は、コンパイラー機能 (特にオプション) を理解したり、その機能を効率的なソフトウェア開発に使用する方法を理解したりするときに役立つ可能性があります。

本書の使用法

本書では、コンパイラーの構成、XL Fortran プログラムのコンパイル、リンク、実行などのトピックが扱われていますが、以下の情報については他の資料で扱われているため本書では説明されていません。

- インストールおよびシステム要件。「*XL Fortran* インストール・ガイド」および製品 README ファイルを参照してください。
- XL Fortran 機能の概要。「*XL Fortran* はじめに」を参照してください。
- XL Fortran プログラム言語の構文、意味構造、および実装。「*XL Fortran* ランゲージ・リファレンス」を参照してください。
- 最適化、ポーティング、OpenMP/SMP プログラミング。これらについては、「*XL Fortran* 最適化およびプログラミング・ガイド」を参照してください。
- コンパイラーの使用に関連するオペレーティング・システム・コマンド。Linux 固有ディストリビューションのマニュアル・ページ・ヘルプと資料を参照してください。

本書の構成

本書では、最初にコンパイラーについて概説してから、コンパイラーを呼び出す前に実行する必要のある作業について簡単に説明します。次に、コンパイラー・オプションと問題のデバッグに関する参照情報を説明します。

本書には次のトピックが説明されています。

- 1 ページの『第 1 章 概要』 から 23 ページの『第 4 章 XL Fortran プログラムの編集、コンパイル、リンク、実行』 までで説明するのは、コンパイル環境のセットアップについて、各種コンパイル・モードに必要となる環境変数のセットアップについて、構成ファイルのカスタマイズについて、入力ファイル、出

力ファイル、コンパイラー・リスト、およびメッセージのタイプについて、ならびにプリプロセッサーとリンケージ・エディターの呼び出しに固有となる情報についてです。

- 61 ページの『第 5 章 機能カテゴリー別コンパイラー・オプションの要約』では、機能カテゴリーごとにコンパイラー・オプションを編成します。名前でオプションを検索することや、機能カテゴリー・テーブルを使用してオプションを検索したりオプションにリンクしたりすることが可能です。
- 81 ページの『第 6 章 XL Fortran コンパイラー・オプションの詳細記述』 では、コンパイラー・オプションをアルファベット順にソートして、個別に説明しています。説明では、実例とトピックに関連したリストが提供されます。
- 327 ページの『第 7 章 コンパイラー・コマンド参照』には、XL Fortran に組み込まれているコマンドの詳細な説明があります。
- 335 ページの『第 8 章 64 ビット環境での XL Fortran の使用』 では、64 ビット環境の場合のアプリケーション開発を説明します。
- 337 ページの『第 9 章 コンパイラー・ライセンス使用状況の追跡』では、コンパイラーの使用状況トラッキングについて説明します。この章には、使用されているコンパイラーの数がフローティング・ユーザー・ライセンス資格の数を超えているかどうかを検出するのに役立つ情報が記載されています。
- 341 ページの『第 10 章 問題判別とデバッグ』および 351 ページの『第 11 章 XL Fortran コンパイラー・リストについて』 では、デバッグを行ったり、コンパイラー・リストを理解したりします。
- 359 ページの『第 12 章 XL Fortran 技術情報』および 365 ページの『第 13 章 XL Fortran 内部制限』では、上級のプログラマーが、異常な問題を診断したり、特殊な環境でコンパイラーを実行したりするために必要となる情報について説明します。

規則

活字の規則

以下の表では、IBM XL Fortran for Linux, V16.1 の資料で使用されている活字の規則について説明します。

表 *1.* 活字の規則

書体	意味	例
小文字の太字	呼び出しコマンド、実行可能ファイル名、およびコンパイラー・オプション。	コンパイラーには、さまざまな Fortran 言語レベルおよびコンパイル環境をサポートするために、**xlf** という基本呼び出しコマンドとその他のいくつかのコンパイラー呼び出しコマンドが備わっています。 実行可能プログラムのデフォルトのファイル名は、**a.out** です。
イタリック	パラメーターまたは変数。実際の名前と値はユーザーによって提供されます。イタリックは新規用語の導入にも使用されます。	要求された *size* よりも大きいものを戻す場合には、 *size* パラメーターの更新を確認してください。

表 1. 活字の規則 (続き)

書体	意味	例
下線	コンパイラー・オプションまたはディレクティブのパラメーターのデフォルト設定。	nomaf | maf
モノスペース	プログラム・コードの例、プログラム・コードに対する参照、ファイル名、パス名、コマンド・ストリング、またはユーザー定義名。	myprogram.f をコンパイルおよび最適化するには、`xlf myprogram.f -O3` と入力します。
大文字の太字	Fortran プログラミング・キーワード、ステートメント、ディレクティブ、および組み込みプロシージャー。コンパイラー・オプション/サブオプションを呼び出すために必要な最小文字数を示すためにも大文字が使用される可能性があります。	**ASSERT** ディレクティブは、ディレクティブの直後に続く **DO** ループにのみ適用され、ネストされた **DO** ループには適用されません。

限定を示すエレメント (アイコンおよび大括弧分離文字)

言語エレメントまたはプログラミング・モデルの説明では、この情報はアイコンおよびマーク付き大括弧分離文字を使用して、以下のようにテキストのセグメントを示しています。

表 2. 限定を示すエレメント

アイコン	大括弧分離文字のテキスト	意味
F2008	Fortran 2008 の始まり /	このテキストは、Fortran 2008 標準の IBM XL Fortran 実装環境を記述しています。[1]
F2008	Fortran 2008 の終わり	
F2003	Fortran 2003 の始まり /	このテキストは、Fortran 2003 標準の IBM XL Fortran 実装環境を記述しており、以降のすべての標準に適用されます。[1]
F2003	Fortran 2003 の終わり	
TS 29113	TS 29113 の始まり /	このテキストは、技術仕様 29113 (TS 29113 と呼ばれる) を実装した IBM XL Fortran について記述しています。[1]
TS 29113	TS 29113 の終わり	
IBM	IBM 拡張の始まり /	テキストは、標準の言語仕様に対する IBM XL Fortran 拡張機能である機能を説明します。
IBM	IBM 拡張の終わり	

表 2. 限定を示すエレメント (続き)

アイコン	大括弧分離文字のテキスト	意味
CUDA Fortran CUDA Fortran	CUDA Fortran の始まり / CUDA Fortran の終わり	このテキストは、CUDA Fortran、IBM XL Fortran によって提供される CUDA Fortran サポート、またはその両方を記述しています。
GPU GPU	GPU の始まり / GPU の終わり	このテキストがある個所には、NVIDIA GPU に計算をオフロードする処理に関係する情報が記述されています。

注:

1. 情報に Fortran 言語標準のアイコンまたは大括弧分離文字が付いている場合は、この特定の Fortran 言語標準および以降のすべての標準に適用されます。そうでない場合は、すべての Fortran 言語標準に適用されます。

構文図

本書中では、ダイアグラムは XL Fortran 構文を図示します。このセクションは、そのダイアグラムを解釈したり使用したりするときに役立ちます。

- 構文図は線のパスに沿って、左から右、上から下へと読んでいきます。

  ▸▸── 記号は、コマンド、ディレクティブ、またはステートメントの開始を示します。

  ──▸ 記号は、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。

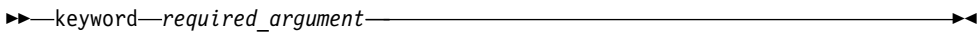
  ▸── 記号は、コマンド、ディレクティブ、またはステートメントが前の行から続いていることを示します。

  ──▸◂ 記号は、コマンド、ディレクティブ、またはステートメントの終了を示します。

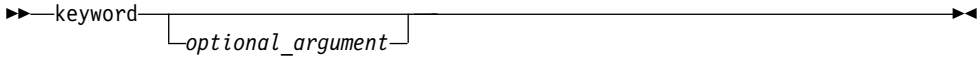
  完結したコマンド、ディレクティブ、またはステートメント以外の構文単位の図であるフラグメントは、|── 記号で始まり ──| 記号で終わります。

  IBM XL Fortran 拡張機能は、構文図内で番号が付けられ、図のすぐ下に注記されています。

  プログラム単位、プロシージャー、構文、インターフェース・ブロック、および派生型定義は複数の個別ステートメントで構成されます。これらの項目については、構文表記をボックスで囲み、個々の構文図は、対応する Fortran ステートメントに必要な順序で表示されます。
- 必須項目は、次のように横線 (メインパス) 上に表示されます。



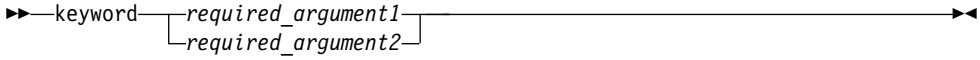
- オプショナル項目は、次のようにメインパスの下側に表示されます。



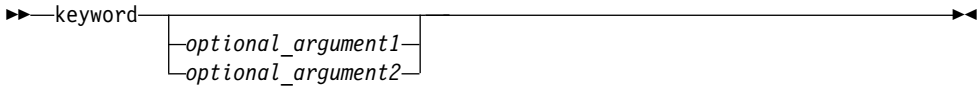
注: オプショナル項目 (構文図にはない) は角括弧 ([および]) で囲まれます。例えば、[UNIT=]u

- 2 つ以上の項目から選択できる場合は、縦に重ねて表示されます。

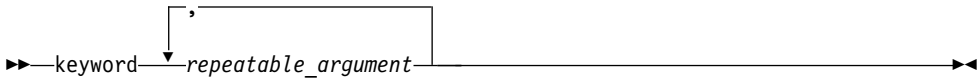
項目の中から 1 つを選択しなければならない 場合は、スタックの 1 つの項目がメインパスに表示されます。



項目の 1 つを選択することがオプショナルの場合は、スタック全体がメインパスの下に表示されます。



- 主線の上にある左に戻る矢印 (反復矢印) は、スタックされた項目から複数個選択できること、あるいは単一の項目を繰り返すことができることを示します。区切り文字も示されます (それがブランク以外の場合)。

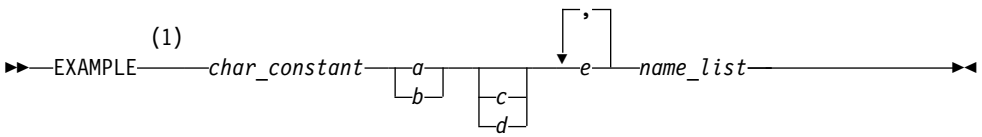


- デフォルトの項目はメインパスの上に表示されます。

- キーワードは、イタリックでない文字で示され、示されているとおりに入力する必要があります。
- 変数は、イタリック体の小文字で示されます。変数は、ユーザー指定の名前や値を表します。変数またはユーザー指定名が *_list* で終わっている場合、コンマで区切られたこれらの項目のリストを指定できます。
- 句読記号、括弧、算術演算子、または他のそのような記号が表示されている場合は、構文の一部として入力する必要があります。

サンプル構文図

次に示すのは解釈付きの構文図の例です。



注:

1 IBM 拡張

この図表は次のように解釈します。

- キーワード EXAMPLE を入力します。
- EXAMPLE は IBM 拡張です。
- *char_constant* に値を入力します。
- *a* または *b* のいずれかの値を入力します。両方は入力しないでください。
- オプションとして、*c* または *d* のいずれかの値を入力します。
- *e* に少なくとも 1 つの値を入力します。複数の値を入力する場合は、それぞれの値の間にコンマが必要です。
- *name_list* に *name* の値を少なくとも 1 つ入力します。複数の値を入力する場合は、それぞれの間にコンマが必要です。 (*_list* 構文は、*e* に対する以前の構文と同等です。

構文ステートメントの読み方

構文ステートメントは左から右に読みます。

- 個々の必須引数は、特殊表記を付けずに記述されます。
- { } 記号で囲まれた選択項目からは、1 つを選択する必要があります。
- オプションの引数は、[ ] 記号で囲まれています。
- 選択項目のグループから選択できる場合は、それらの選択項目は | 文字で区切られます。
- 繰り返せる引数の後には、省略符号 (...) が続きます。

構文記述の例

```
EXAMPLE char_constant {a|b}[c|d]e[,e]... name_list{name_list}...
```

次のリストは、構文ステートメントを説明しています。

- キーワード EXAMPLE を入力します。
- *char_constant* に値を入力します。
- *a* または *b* のいずれかの値を入力します。両方は入力しないでください。
- オプションとして、*c* または *d* のいずれかの値を入力します。
- *e* に少なくとも 1 つの値を入力します。複数の値を入力する場合は、それぞれの値の間にコンマが必要です。
- オプションで、*name_list* に *name* の値を少なくとも 1 つ入力します。複数の値を入力する場合は、それぞれの *name* の間にコンマが必要です。

注: 同じ例が構文ステートメントと構文図の両方で使用されています。

本書の例

本書の例は、特に断りのない限り、単純な形式でコーディングされており、ストレージの節約、エラーのチェック、高速パフォーマンスの実現、特定の成果を達成するために使用可能なすべての方法の提示などの試みはなされていません。

インストール情報の例は、例 または基本例 としてラベル付けられています。基本例 は、デフォルト・インストール中に実行する手順を説明するためのもので、ほとんど変更せずに、またはまったく変更せずに使用できます。

使用されている用語についての注記

本書の用語の中には、次のように、短縮されているものがあります。

- 用語 フリー・ソース・フォーム形式 は多くの場合、フリー・ソース・フォーム と表記しています。
- 用語 固定ソース・フォーム形式 は多くの場合、固定ソース・フォーム と表記しています。
- 用語 *XL Fortran* は多くの場合、*XLF* と表記しています。

関連情報

以下のセクションでは、XL Fortran に関連した情報を説明します。

使用可能なヘルプ情報

IBM XL Fortran 情報

XL Fortran は、以下の形式で製品資料を提供しています。

- クイック・スタート・ガイド

  クイック・スタート・ガイド (`quickstart.pdf`) は、IBM XL Fortran for Linux, V16.1 を即時に使用し始めるときに読むものです。デフォルトでは、XL Fortran ディレクトリー、およびインストール DVD の `¥quickstart` ディレクトリーにあります。

- README ファイル

  README ファイルには、製品情報に対する変更と訂正も含め、最新の情報が含まれています。 README ファイルは、デフォルトではインストール DVD の XL Fortran ディレクトリー、ルート・ディレクトリー、およびサブディレクトリーにあります。

- インストール可能な man ページ

  man ページは製品に準備されているコンパイラー呼び出しとすべてのコマンド行ユーティリティーに対して提供されています。man ページのインストールおよびアクセスについての指示は、「*IBM XL Fortran for Linux, V16.1* インストール・ガイド」に記載されています。

- オンライン製品資料

完全に検索可能な HTML ベースの資料が IBM Knowledge Center (http://www.ibm.com/support/knowledgecenter/SSAT4T_16.1.0/com.ibm.compilers.linux.doc/welcome.html) で参照可能です。

- PDF 文書

  PDF 文書は、Web サイト https://www.ibm.com/support/knowledgecenter/SSAT4T_16.1.0/com.ibm.compilers.linux.doc/download_pdf.html でも入手できます。

  以下のファイルは、XL Fortran 製品情報のフル・セットを構成しています。

  注**:** 他の XL Fortran PDF 文書への相互参照リンクにアクセスできるようにするには、すべての製品資料ファイルが入っている .zip ファイルを unzip してください。また、それぞれの文書をローカル・マシン上の同じディレクトリーにダウンロードできます。

表 *3. XL Fortran PDF* ファイル

文書タイトル	**PDF** ファイル名	説明
IBM XL Fortran for Linux, V16.1 の新機能, *GC43-4669-00*	`whats_new.pdf`	IBM XL Fortran for Linux, V16.1 コンパイラーにおける新機能の要旨のほか、ユーザーの利点に従ってカテゴリー化された新機能が記載されています。
IBM XL Fortran for Linux, V16.1 はじめに, *GC43-4656-00*	`getstart.pdf`	XL Fortran の概要のほか、ご使用の環境のセットアップと構成、プログラムのコンパイルとリンク、およびコンパイル・エラーのトラブルシューティングに関する情報が記載されています。
IBM XL Fortran for Linux, V16.1 インストール・ガイド, *GC43-4662-00*	`install.pdf`	XL Fortran のインストール方法と基本的なコンパイルおよびプログラム実行のための環境の構成方法に関する情報が含まれています。
IBM XL Fortran for Linux, V16.1 移行ガイド, *GC43-4666-00*	`migrate.pdf`	以前のリリースの XL Fortran によって、または他のコンパイラーによって、さまざまなプラットフォームで以前にコンパイルされたプログラムをコンパイルするために XL Fortran を使用する際のマイグレーションの考慮事項が記載されています。
IBM XL Fortran for Linux, V16.1 コンパイラー・リファレンス, *SC43-4663-00*	`compiler.pdf`	さまざまなコンパイラー・オプションおよび環境変数についての情報が含まれます。
IBM XL Fortran for Linux, V16.1 ランゲージ・リファレンス, *SC43-4667-00*	`langref.pdf`	移植性および非機密標準への準拠に対応した言語拡張機能、コンパイラー・ディレクティブ、および組み込みプロシージャーなどの、IBM がサポートする Fortran プログラミング言語についての情報が含まれます。

表 3. XL Fortran PDF ファイル (続き)

文書タイトル	**PDF** ファイル名	説明
IBM XL Fortran for Linux, V16.1 最適化およびプログラミング・ガイド, *SC43-4657-00*	`proguide.pdf`	アプリケーションの移植、言語間呼び出し、浮動小数点演算、入出力、アプリケーションの最適化と並列処理、XL Fortran 高性能ライブラリーなど、高度なプログラミングのトピックに関する情報が記載されています。
Getting Started with CUDA Fortran programming using IBM XL Fortran for Linux, V16.1, GC43-4658-00	`getstart_cudaf.pdf`	XL Fortran で提供される CUDA Fortran サポートに関する詳細情報が記載されています。これには、CUDA Fortran プログラムのコンパイラー・フロー、コンパイル・コマンド、有用なコンパイラー・オプションとマクロ、サポートされる CUDA Fortran 機能、および制限事項が含まれています。

PDF ファイルを読むには、Adobe Reader を使用します。Adobe Reader を持っていない場合は、Adobe の Web サイト (http://www.adobe.com) からダウンロードできます (ライセンス条項に従う必要があります)。

IBM Redbooks® 資料、ホワイト・ペーパー、他の記事など、XL Fortran に関する詳細は、Web (http://www.ibm.com/support/docview.wss?uid=swg27036672) から入手できます。

コンパイラーについて詳しくは、Power® コミュニティー (http://ibm.biz/xl-power-compilers) の XL コンパイラーを参照してください。

その他の IBM 情報

- *ESSL* 製品資料 は http://www.ibm.com/support/knowledgecenter/SSFHY8/essl_welcome.html?lang=en で入手できます。

標準および仕様

XL Fortran は、以下の標準および仕様をサポートするように設計されています。本書に記載されているいくつかのフィーチャーの正確な定義については、以下の標準および仕様を参照してください。

- *American National Standard Programming Language FORTRAN, ANSI X3.9-1978.*
- *American National Standard Programming Language Fortran 90, ANSI X3.198-1992.*
- *ANSI/IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1985.*
- *Federal (USA) Information Processing Standards Publication Fortran, FIPS PUB 69-1.*
- *Information technology - Programming languages - Fortran, ISO/IEC 1539-1:1991.* (この情報では、非公式な名前である Fortran 90 を使用しています。)
- *Information technology - Programming languages - Fortran - Part 1: Base language, ISO/IEC 1539-1:1997.* (この情報では、非公式な名前である Fortran 95 を使用しています。)

- *Information technology - Programming languages - Fortran - Part 1: Base language, ISO/IEC 1539-1:2004.* (この情報では、非公式な名前である Fortran 2003 を使用しています。)
- *Information technology - Programming languages - Fortran - Part 1: Base language, ISO/IEC 1539-1:2010.* (この情報では、非公式な名前である Fortran 2008 を使用しています。現在、この標準に対して部分サポートを提供しています。)
- *Information technology - Further interoperability of Fortran with C, ISO/IEC TS 29113:2012.* (本書では、この資料の非公式な名前である「技術仕様 29113」(別名: TS 29113) が使用されています。現時点では、この仕様には部分サポートが提供されています。)
- *Military Standard Fortran DOD Supplement to ANSI X3.9-1978, MIL-STD-1753* (米国、国防総省標準)。 XL Fortran は、Fortran 90 標準に順次取り込まれてもいるこの標準において文書化されている拡張のみをサポートしていることに注意してください。
- *OpenMP Application Program Interface Version 3.1* (フルサポート)、*OpenMP Application Program Interface Version 4.0* (部分サポート)、および *OpenMP Application Program Interface Version 4.5* (部分サポート)。 http://www.openmp.org で入手可能。

その他の IBM 情報

- *ESSL* 製品資料 は、http://www.ibm.com/support/knowledgecenter/SSFHY8/essl_welcome.html?lang=en で入手できます。

テクニカル・サポート

追加のテクニカル・サポートは、XL Fortran のサポート・ページ (https://www.ibm.com/support/home/product/U128148Q26691I65/XL_Fortran_for_Linux) から利用できます。このページには、Technote や他のサポート情報の豊富なセレクションへの検索機能を備えたポータルがあります。

必要なものが見つからない場合には、compinfo@cn.ibm.com に E メールを出して問い合わせることができます (英文でのみ対応)。

XL Fortran に関する最新の情報に関しては、https://www.ibm.com/us-en/marketplace/xl-fortran-linux-compiler-powerにある製品情報サイトをご覧ください。

第 1 章 概要

IBM XL Fortran for Linux, V16.1 は、最適化を行う標準ベースの Linux オペレーティング・システム用のコマンド行コンパイラーであり、Power Architectureの Power ハードウェアで稼働します。XL Fortran コンパイラーを使用することで、アプリケーション開発者は、Linux オペレーティング・システム用の最適化された 64 ビットのアプリケーションの作成および保守を行うことができます。 また、このコンパイラーは、最適化手法の変化に富んだポートフォリオを提供し、それによりアプリケーション開発者は Power プロセッサーの重層的なアーキテクチャーを活用することができます。

Fortran プログラム言語の実装は、言語標準への準拠の徹底により、異なる環境間での移植性のプロモートを意図されています。言語仕様に厳密に準拠するプログラムは、異なる環境間で最大の移植性を持ちます。理論的には、標準に準拠するコンパイラーで正しくコンパイルするプログラムは、ハードウェアの差異が許容する範囲で、すべての他の準拠コンパイラーのもとで、正しくコンパイルおよび実行します。書かれているプログラム言語の拡張機能を正しく活用するプログラムは、オブジェクト・コードの効率を改善します。

XL Fortran は大規模で、複雑な、計算主体のプログラムに使用できます。また、C を使用して言語間呼び出しをサポートします。 SIMD (single-instruction, multiple data) 並列処理を必要とするアプリケーションに対しては、最適化手法によりパフォーマンスの向上が達成でき、ベクトル・プログラミングより労働集約的ではありません。IBM によって開発された、多くの最適化はコンパイラー・オプションおよびディレクティブによって制御されます。

XL Fortran に用意されている CUDA Fortran サポートを使用すると、ご使用のシステムですべての NVIDIA GPU を活用することができます。

第 2 章 XL Fortran の機能の概要

このセクションでは、XL Fortran コンパイラー、言語、開発環境の機能について概要を説明します。本章は、XL Fortran を評価するユーザー、または製品についてより詳しく知りたい新規ユーザーを対象としています。

ハードウェアおよびオペレーティング・システムのサポート

IBM XL Fortran for Linux, V16.1 は、 いくつかの Linux 配布版でサポートされています。

サポートされる配布版および要件のリストについては、「*XL Fortran for Linux* インストール・ガイド」および README ファイルを参照してください。

POWER8® 以上のプロセッサー は、PowerPC® プロセッサーのタイプです。本書において、PowerPC プロセッサーに関する記述または参照は、POWER8 以上のプロセッサー にも当てはまります。

異なるハードウェア構成を最大限に利用するために、コンパイラーはアプリケーションの実行に使用するマシン構成に基づいて、パフォーマンス調整のための多数のオプションを提供しています。

言語サポート

このトピックでは、XL Fortran でサポートされている言語および言語拡張をリストします。

XL Fortran 言語は、次のもので構成されます。

- ISO 技術仕様書 29113 サポート (TS 29113 と呼びます)の一部。この言語は「*Information technology -- Further interoperability of Fortran with C, ISO/IEC TS 29113:2012*」に規定されています。
- ISO Fortran 2008 言語標準 (Fortran 2008 または F2008 と呼びます) の一部。この言語は、「*Information technology - Programming languages - Part 1: Base language, ISO/IEC 1539-1:2010*」に規定されています。
- 完全な ISO Fortran 2003 言語標準 (Fortran 2003 または F2003 と呼びます)。この言語は、「*Information technology - Programming languages - Part 1: Base language, ISO/IEC 1539-1:2004*」に規定されています。
- 完全な ISO Fortran 95 言語標準 (Fortran 95 または F95 と呼びます)。この言語は、「*Information technology - Programming languages - Fortran - Part 1: Base language, ISO/IEC 1539-1:1997*」に規定されています。
- 完全版の米国標準規格 Fortran 90 言語 (Fortran 90 または F90 と呼ばれます)。これは、資料「*American National Standard Programming Language Fortran 90*」、*ANSI X3.198-1992* および「*Information technology - Programming languages - Fortran*」、*ISO/IEC 1539-1:1991 (E)* で定義されています。この言語は、FORTRAN 77 標準の機能のスーパーセットを持っています。これは、さら

にエラー検査、配列処理、メモリー割り付けなどの作業の多くをプログラマーからコンパイラーに移行することを目的とする多くの機能が追加されています。

- Fortran 標準に対する拡張機能
  - IBM で定義された拡張機能に加えて、他のコンパイラー・ベンダーで定義された共通 Fortran 言語拡張機能
  - さまざまなコンパイラー・ベンダーの Fortran 製品に見られる業界用拡張機能
  - SAA Fortran で指定されている拡張機能
  - Vector Multimedia Extension (VMX) および Vector Scalar Extension (VSX) 組み込み関数の拡張機能

「*XL Fortran* ランゲージ・リファレンス」では、Fortran 2003 言語および Fortran 2008 言語に対する拡張機能には、『規則: エレメントの修飾』セクションの説明どおりのマークが付けられています。

ソース・コードの適合性検査

各種の FORTRAN 77、Fortran 90、Fortran 95、Fortran 2003、または Fortran 2008 標準との間でアプリケーションを移植する際に起こり得る問題を発見できるようにするために、XL Fortran コンパイラーは、特定の Fortran 定義に準拠しなくなった機能について警告するためのオプションを提供しています。

適切なコンパイラー・オプションを指定すると、XL Fortran コンパイラーは、ソース・ステートメントが次の Fortran 言語定義に準拠しているかどうかを検査します。

- 部分的な Fortran 2008 標準 (**-qlanglvl=2008std** オプション)
- 完全な Fortran 2003 標準 (**-qlanglvl=2003std** オプション)
- 完全な Fortran 95 標準 (**-qlanglvl=95std** オプション)
- 完全な American National Standard Fortran 90 標準 (**-qlanglvl=90std** オプション)
- 完全な American National Standard FORTRAN 77 標準 (**-qlanglvl=77std** オプション)
- Fortran 2008 から、廃止対象のすべての機能を除去したもの (**-qlanglvl=2008pure** オプション)
- Fortran 2003 から、廃止対象のすべての機能を除去したもの (**-qlanglvl=2003pure** オプション)
- Fortran 95 から、廃止対象のすべての機能を除去したもの (**-qlanglvl=95pure** オプション)
- Fortran 90 から、廃止対象のすべての機能を除去したもの (**-qlanglvl=90pure** オプション)
- C との相互運用性をさらに高めるための Technical Specification 29113 の部分的なサポート (**-qlanglvl=ts** オプション)
- Fortran 2008 に関する Technical Specification 29113 の補足から、廃止対象のすべての機能を除去したもの (**-qlanglvl=tspure** オプション)
- IBM SAA FORTRAN (**-qsaa** オプション)

また、**langlvl** オプションを使って、準拠しているかどうかをチェックすることもできます。

注: Fortran 2008 への準拠検査は、現在実装されているこの標準のサブセットを基にして行われます。

高度な構成が可能なコンパイラー

このトピックでは、コンパイラーの呼び出しに使用できるコマンドについて説明します。

コンパイラーは、次のいずれかのコマンドで呼び出すことができます。

- xlf
- xlf_r
- f77
- fort77
- xlf90
- xlf90_r
- f90
- xlf95
- xlf95_r
- f95
- xlf2003
- xlf2003_r
- f2003
- xlf2008
- xlf2008_r
- f2008
- CUDA Fortran xlcuf CUDA Fortran

次のコマンドは、XL Fortran Version 2 の動作や I/O 形式との最大の互換性を維持します。

- .f ファイル、.F ファイル、.f77 ファイル、および .F77 ファイルの場合: **xlf** および **xlf_r**
- 任意のソース・ファイルの場合: **f77** および **fort77**

表 4. 汎用性のない呼び出しコマンドの機能

コマンド	機能
f77 および **fort77**	XPG4 の動作との最大の互換性を提供
xlf90、**xlf90_r**、および **f90**	より高い Fortran 90 への準拠性と、効率と使いやすさの向上に役立つ実装の選択肢をいくつか提供
f95、**xlf95** および **xlf95_r**	より高い Fortran 95 への準拠性と、効率と使いやすさの向上に役立つ実装の選択肢をいくつか提供
xlf2003、**xlf2003_r**、および **f2003**	より高い Fortran 2003 への準拠性と、効率と使いやすさの向上に役立つ実装の選択肢をいくつか提供

表 4. 汎用性のない呼び出しコマンドの機能 (続き)

コマンド	機能
xlf2008、**xlf2008_r**、および **f2008**	より高い Fortran 2008 への準拠性と、効率と使いやすさの向上に役立つ実装の選択肢をいくつか提供
CUDA Fortran **xlcuf**	CUDA Fortran サポートを提供 CUDA Fortran

xlf_r、**xlf90_r**、**xlf95_r**、**xlf2003_r**、および **xlf2008_r** コマンドおよび一連の **xlf**、**xlf90**、**f90**、**xlf95**、**f95**、**xlf2003**、**f2003**、**xlf2008**、**f2008**、**f77**、および **fort77** のコマンド群との間に存在する主な違いは、前者はオブジェクト・ファイルをスレッド・セーフ・コンポーネント (ライブラリーなど) とリンクおよびバインドすることです。後者のコマンド群でこの動作をさせることも可能ですが、そのためには、以下のように **-F** コンパイラー・オプションを使って、使用する構成ファイル・スタンザを指定します。例えば の場合:

```
xlf -F/opt/ibm/xlf/16.1.0/etc/xlf.cfg:xlf_r
```

一連のオプションによって、コンパイラーの動作を制御できます。種々のカテゴリーのオプションを使用すれば、ソース・コードを変更せずに、デバッグ、プログラムのパフォーマンスの最適化と調整、他のプラットフォームのプログラムとの互換性を得るための拡張機能の選択、その他の一般的な作業が容易になります。

多様なコンパイラー・オプションを管理する作業を簡略化するために、別名またはシェル・スクリプトを別個に多数作成する代わりに、デフォルトの構成ファイルを編集、またはカスタマイズされた構成ファイルを使用することができます。

関連情報

- 12 ページの『カスタム・コンパイラー構成ファイルの使用』
- 31 ページの『XL Fortran プログラムのコンパイル』
- 61 ページの『第 5 章 機能カテゴリー別コンパイラー・オプションの要約』および 81 ページの『第 6 章 XL Fortran コンパイラー・オプションの詳細記述』

診断リスト作成

コンパイラー出力リストには、オプションで含めることも省略することもできるセクションがあります。

利用可能なコンパイラー・オプションや、リスト作成そのものについては、72 ページの『リスト、メッセージ、およびコンパイラー情報』および 351 ページの『第 11 章 XL Fortran コンパイラー・リストについて』を参照してください。

-S オプションを使用すると、アセンブラー・ソース・ファイルそのものが得られます。

シンボリック・デバッガーのサポート

さまざまなレベルの **-g** コンパイラー・オプションまたは **-qdbg** コンパイラー・オプションを使用することで、コンパイルしたオブジェクトにデバッグ情報を組み込むように、XL Fortran に指示できます。

詳しくは、**-g** または **-qdbg**を参照してください。

デバッグ情報は、**gdb**、 その他の任意のシンボリック・デバッガーによって調べることができ、プログラムのデバッグに役立ちます。

プログラムの最適化

XL Fortran コンパイラーは、プログラムの最適化をいくつかの方法で制御できるように支援します。

- さまざまなレベルのコンパイラーの最適化を選択できます。
- ループ、浮動小数点、その他のカテゴリーに対して別個に最適化を実施することができます。
- プログラムの実行場所に応じて、特定のクラスのマシンや非常に特殊なマシン構成に合うようにプログラムを最適化することができます。

「*XL Fortran* 最適化およびプログラミング・ガイド」では、これらの機能に対するロードマップおよび最適化戦略が提供されます。

第 3 章 XL Fortran のセットアップとカスタマイズ

このセクションでは、あらゆるユーザーに合わせて XL Fortran 設定をカスタマイズする方法を説明します。このセクションでは、インストール手順すべてを網羅しているわけではないので、その詳細についてはインストール手順を扱っている資料を参照してください。

さらにこのセクションは、コンパイラーのインストールまたは構成に関連した問題の診断に役立てるために参照することもできます。

指示の中には、スーパーユーザーでなければできないこと、つまりシステム管理者だけが適用できるものもあります。

インストール手順の指示が記載されている資料

コンパイラーをインストールするには、以下の資料を参照してください (掲載順での参照をお勧めします)。

1. 製品の tar ファイルまたは DVD のルートにある README.FIRST という名前のファイルを読んで、記述されている指示に従ってください。このファイルには、ユーザーが知っておく必要のある情報、および XL Fortran を使用する他の方々にも知らせる必要のある情報が入っています。
2. 「*XL Fortran* インストール・ガイド」をお読みになり、注意すべき重要な事項があるかどうか、あるいはインストール前にシステムに適用しなければならない更新があるかどうか確認してください。

デフォルトのインストール・ツールの使用

- この製品を SLES、RHEL、または CentOS をインストールするには、RPM Package Manager (RPM) を使い慣れている必要があります。 RPM の使用に関する情報については、URL http://www.rpm.org/ の RPM Web ページにアクセスするか、またはコマンド行に **rpm --help** と入力してください。

  ソフトウェアのインストール経験がある場合は、**rpm** コマンドを使用して配布メディアからすべてのイメージをインストールすることができます。
- Ubuntu にこの製品をインストールするには、Debian Package Manager (dpkg) を使い慣れている必要があります。 dpkg の使用に関する情報については、URL http://manpages.ubuntu.com/manpages/trusty/en/man1/dpkg.1.html の dpkg マニュアル・ページを表示するか、またはコマンド行に **dpkg --help** と入力します。

環境変数の正しい設定方法

オペレーティング・システムで使用するように設定してエクスポートできる環境変数は多数あります。以降の項では、XL Fortran コンパイラーとアプリケーション・プログラム、あるいはそのどちらか一方に特別に重要である環境変数を扱います。

環境変数の基礎

環境変数を、シェル・コマンド行から、またはシェル・スクリプト内で設定することができます。 (環境変数の設定について詳しくは、使用しているシェルの man ページ・ヘルプを参照してください。) どのシェルを使用しているかわからない場合は、**echo $SHELL** を発行して、現行シェルの名前を表示してください。

環境変数の内容を表示するには、**echo** *$var_name* コマンドを入力します。

注**:** 本書の残りの部分では、シェル・コマンドの例の多くで、すべてのシェルの構文を繰り返さずに **Bash** 表記を使用しています。

ライブラリー検索パスの設定

実行可能プログラムが共有ライブラリーにリンクされている場合は、ランタイム・ライブラリー検索パスを設定する必要があります。ランタイム・ライブラリー検索パスは、以下の 3 つの方法のいずれかで設定できます。

- 共有ライブラリーを実行可能プログラムにリンクする際に、**-R** (または **-rpath**) コンパイラー/リンク・オプションを使用する。
- 共有ライブラリーを実行可能プログラムにリンクする前に、**LD_RUN_PATH** 環境変数を設定する。
- **LD_LIBRARY_PATH** 環境変数を設定する。

例を以下に示します。

```
# Compile and link
xlf95 -L/usr/lib/mydir1 -R/usr/lib/mydir1 -L/usr/lib/mydir2 -R/usr/lib/mydir2
  -lmylib1 -lmylib2 test.f

# -L directories are searched at link time for both static and shared libraries.
# -R directories are searched at run time for shared libraries.
```

リンカー・オプション **-R** (または **-rpath**)、および環境変数 **LD_RUN_PATH** と **LD_LIBRARY_PATH** について詳しくは、**ld** コマンドの man ページを参照してください。

Profile-Directed Feedback 環境変数

-qpdf コンパイラー・オプションとともに使用できる Profile-Directed Feedback (PDF) 環境変数を以下に示します。

PDF_BIND_PROCESSOR

キャッシュ・ミス・プロファイルのために、アプリケーションを指定プロセッサーにバインドする場合は、PDF_BIND_PROCESSOR 環境変数を設定します。デフォルトではプロセッサー 0 が設定されます。

PDF_PM_EVENT

-qpdf1=level=2 を指定してコンパイルされたアプリケーションを実行し、各種レベルのキャッシュ・ミス・プロファイル情報を収集する場合は、PDF_PM_EVENT 環境変数を L1MISS、L2MISS、または L3MISS (使用可能な場合) に適宜設定します。

PDF_SIGNAL_TO_DUMP

実行中のスナップショット PDF プロファイル情報のファイルへのダンプを有効にしたい場合は、アプリケーションを実行する前に

PDF_SIGNAL_TO_DUMP 環境変数を定義する必要があります。この値は SIGRTMIN から SIGRTMAX の範囲内の整数でなければなりません。詳しくは、「*XL Fortran* 最適化およびプログラミング・ガイド」の『実行時にスナップショット PDF プロファイル作成情報をダンプ』を参照してください。

PDF_WL_ID

PDF_WL_ID 環境変数は、ユーザー・プログラムのトレーニング実行を何度も行うことによって生成される PDF カウンターのセットを区別するために使用されます。それぞれが異なる入力を受け取ります。

デフォルトでは、トレーニング実行の PDF カウンターは、最初のトレーニング実行が PDF カウンターの最初で唯一のセットに追加された後に実行されます。この動作は、各 PDF トレーニング実行の前に PDF_WL_ID 環境変数を設定することによって変更できます。 PDF_WL_ID は、1 から 65535 までの範囲の整数値に設定されます。すると、PDF ランタイムはこの数値を使用して、このトレーニング実行により生成される PDF カウンターのセットにタグ付けします。すべてのトレーニング実行が完了した後、PDF プロファイル・ファイルには、それぞれに ID 番号が設定された複数の PDF カウンター・セットが格納されます。

PDFDIR

-qpdf コンパイラー・オプションを使用して Fortran をコンパイルする場合、プロファイル情報を格納するディレクトリーの名前を PDFDIR 環境変数に設定することによって、そのディレクトリーを指定できます。コンパイラーは、プロファイル情報を保持するファイルを作成します。XL Fortran は、**-qpdf1** オプションを指定してコンパイルしたアプリケーションを実行したときに、ファイルを更新します。

プロファイル情報が誤った場所に格納されていたり、複数のアプリケーションによって更新されたりすると、問題が起きる可能性があります。このような問題を回避するために、以下の指針に従ってください。

- **-qpdf** オプションを使用する場合は、常に PDFDIR 環境変数を設定する。 PDFDIR 環境変数によって指定されたディレクトリーが存在することを確認してください。そうでない場合、コンパイラーは警告メッセージを発行します。
- 別のディレクトリーに各アプリケーションのプロファイル情報を保管します。あるいは **-qpdf1=pdfname**、**-qpdf1=exename** オプションを使用して提供されたテンプレートに従って、一時プロファイル・ファイルの名前を明示的に指定します。
- PDFDIR 環境変数の値は、そのアプリケーションについての PDF プロセス (コンパイル、実行、再コンパイル) が完了するまで変更しない。

TMPDIR: 一時ファイルのディレクトリーの指定

XL Fortran コンパイラーは、コンパイル時に使用するために多数の一時ファイルを作成し、 XL Fortran アプリケーション・プログラムは、 **STATUS='SCRATCH'** でオープンされるファイルの一時ファイルを実行時に作成します。デフォルトでは、これらのファイルは **/tmp** ディレクトリーに入れられます。

これらのファイルが入るディレクトリーを変更したい場合は、すべての一時ファイルを保持できるほど **/tmp** が大きくないので、コンパイラーまたはアプリケーション・プログラムを実行する前に、 **TMPDIR** 環境変数を設定してエクスポートしてください。

以下に示す **XLFSCRATCH_***unit* の方法を使用してスクラッチ・ファイルを明示的に指定した場合、 **TMPDIR** 環境変数はそのファイルに影響を与えません。

XLFSCRATCH_unit: スクラッチ・ファイルの名前の指定

スクラッチ・ファイルに特定の名前を指定するには、実行時オプションの **scratch_vars=yes** を設定し、次に **XLFSCRATCH_***unit* という形式の名前が付いた 1 つ以上の環境変数にファイル名を設定します。このファイル名は、それらのユニットをスクラッチ・ファイルとしてオープンするときに使用されます。例については、「*XL Fortran* 最適化およびプログラミング・ガイド」の『スクラッチ・ファイルの命名』を参照してください。

XLFUNIT_unit: 暗黙に接続されるファイルの名前の指定

暗黙に接続されるファイル、または **FILE=** 指定子なしにオープンされるファイルの名前を指定するには、まず実行時オプションの **unit_vars=yes** を指定し、次に **XLFUNIT_***unit* という形式の名前が付いた 1 つ以上の環境変数をファイル名に設定します。例については、「*XL Fortran* 最適化およびプログラミング・ガイド」の『明示的名前なしで接続されるファイルの命名』を参照してください。

カスタム・コンパイラー構成ファイルの使用

XL Fortran コンパイラーは、デフォルト構成ファイル /opt/ibm/xlf/16.1.0/etc/xlf.cfg.*$OSRelease.gcc$gccVersion*をインストール時に生成します。 (インストール中に構成ファイルの生成に使用できるさまざまなツールについての情報は、*XL Fortran* インストール・ガイド を参照してください。) 構成ファイルは、呼び出された時にコンパイラーが使用する情報を指定します。デフォルト構成ファイルの例は以下のとおりです。

- /opt/ibm/xlf/16.1.0/etc/xlf.cfg.sles.12.gcc.4.8.2
- /opt/ibm/xlf/16.1.0/etc/xlf.cfg.rhel.7.3.gcc.4.8.5
- /opt/ibm/xlf/16.1.0/etc/xlf.cfg.centos.7.gcc.4.8.3
- /opt/ibm/xlf/16.1.0/etc/xlf.cfg.ubuntu.16.04.gcc.4.8.2

単一ユーザー・システムで稼働している場合や、コンパイル・スクリプトや Make ファイルを持つコンパイル環境がすでにある場合は、デフォルトの構成ファイルをそのままにしておくことができます。

いくつかのコンパイラー・オプション・セットからユーザーが選択できるようにしたい場合は、特定の必要に合わせたカスタム構成ファイルを使用することもできます。例えば、**xlf** コンパイラー呼び出しコマンドを使用したコンパイルができるよう、**-qlist** をデフォルトで有効にすることもできます。これにより、**xlf** コマンドでコンパイラーを呼び出すたびに **-qnolist** が自動的に有効になるため、コンパイル時に毎回コマンド行でこのオプションを指定する必要がなくなります。

構成ファイルをカスタマイズする方法はいくつかあります。

- デフォルトの構成ファイルを直接編集する。 この場合、カスタマイズされたオプションは、すべてのコンパイルですべてのユーザーに対して適用されます。このオプションの欠点は、コンパイラーを更新するたびに提供される新規のデフォルト構成ファイルにカスタマイズ内容を再適用しなければならない点です。
- デフォルトの構成ファイルを、コンパイル時に **-F** オプションで指定するカスタマイズ・コピーの基盤として使用する。この場合、カスタム・ファイルは、コンパイル単位のデフォルト・ファイルをオーバーライドします。

  注**:** このオプションでは、コンパイラーにサービスを適用後、カスタマイズを再適用する必要があります。
- XLF_USR_CONFIG 環境変数を使用して、コンパイル時に指定するカスタムまたはユーザー定義の構成ファイルを作成する。この場合、カスタムのユーザー定義ファイルは、デフォルトの構成ファイルをオーバーライドするのではなく補完します。これらのファイルはコンパイル単位またはグローバル単位で指定することもできます。このオプションの利点は、更新中に新規のシステム構成ファイルをインストールする際、既存のカスタム構成ファイルを変更する必要がない点です。 カスタム、およびユーザー定義の構成ファイルの作成手順を以下に示します。

関連資料:

88 ページの『-F』

カスタム構成ファイルの作成

XLF_USR_CONFIG 環境変数を使用して、コンパイラーにカスタムのユーザー定義構成ファイルを使用するよう指示すると、コンパイラーはそのユーザー定義構成ファイルの設定を検討および処理してから、デフォルトのシステム構成ファイルの設定を確認します。

カスタムのユーザー定義構成ファイルを作成するには、**use** 属性の複数レベルを指定するスタンザを追加します。ユーザー定義の構成ファイルは、システム構成ファイルで指定された定義だけでなく、同じファイル内のいずれかで指定された定義も参照できます。特定のコンパイルの場合、コンパイラーは特定スタンザの検索をユーザー定義構成ファイルの先頭から開始してから、use 属性で指定された他のスタンザ (システム構成ファイルで指定されたスタンザを含む) を検索します。

use 属性で指定されたスタンザの名前が現在処理中のスタンザの名前と異なる場合、use スタンザの検索はユーザー定義構成ファイルの先頭から開始します。これは、次の例に示すスタンザ A、C、および D の場合です。例での 2 つの B スタンザのように、**use** 属性のスタンザの名前が現在処理中のスタンザの名前と同じ場合、**use** スタンザの検索は現行スタンザのその位置から開始します。

以下の例で、**use** 属性の複数レベルを使用する方法を示します。この例では、**options** 属性を使用して **use** 属性の機能方法を説明していますが、**libraries** などの他の属性を使用することもできます。

```
A: use =DEFLT
    options=<set of options A>
B: use =B
    options=<set of options B1>
B: use =D
    options=<set of options B2>
C: use =A
    options=<set of options C>
D: use =A
    options=<set of options D>
DEFLT:
    options=<set of options Z>
```

図 *1.* サンプル構成ファイル

この例では以下のとおりになります。

- スタンザ A ではオプション・セット *A* および *Z* が使用される
- スタンザ B ではオプション・セット *B1*、*B2*、*D*、*A*、および *Z* が使用される
- スタンザ C ではオプション・セット *C*、*A*、および *Z* が使用される
- スタンザ D ではオプション・セット *D*、*A*、および *Z* が使用される

属性は、スタンザと同じ順序で処理されます。オプションが指定される順序は、オプション解決にとって重要です。通常、あるオプションが複数回指定されている場合、そのオプションの最後に指定されたインスタンスが優先されます。

デフォルトで、構成ファイルのスタンザで定義された値は、前に処理されたスタンザで指定された値のリストに追加されます。例えば、XLF_USR_CONFIG 環境変数が ~/userconfig1 にあるユーザー定義のカスタム構成ファイルを指すように設定されているとします。以下の例で示す、ユーザー定義の構成ファイルおよびデフォルトの構成ファイルの場合、コンパイラーは、ユーザー定義構成ファイルの **xlf** スタンザを参照し、この構成ファイルで指定されたオプション・セットを *A1*、*A*、*D*、および *C* の順序で使用します。

```
xlf:  use=xlf
    options= <A1>

DEFLT: use=DEFLT
    options=<D>
```

図 *2.* カスタムのユーザー定義構成ファイル *~/userconfig1*

```
xlf:  use=DEFLT
    options=<A>

DEFLT:
    options=<C>
```

図 *3.* デフォルトの構成ファイル *xlf.cfg*

属性値のデフォルト順序のオーバーライド

属性値のデフォルト順序をオーバーライドするには、構成ファイルの属性の代入演算子 (=) を変更します。

表 *5.* 代入演算子および属性の順序

代入演算子	説明
-=	デフォルトの検索順序によって決定される値の前に、以下の値を付加します。
:=	デフォルトの検索順序によって決定される値を、以下の値で置き換えます。
+=	デフォルトの検索順序によって決定される値の後に、以下の値を付加します。

例えば、XLF_USR_CONFIG 環境変数が ~/userconfig2 にあるカスタムのユーザー定義構成ファイルを指すように設定されているとします。

カスタムのユーザー定義構成ファイル

~/userconfig2

```
xlf_prepend: use=xlf
             options-=<B1>
xlf_replace: use=xlf
             options:=<B2>
xlf_append: use=xlf
             options+=<B3>

DEFLT: use=DEFLT
       options=<D>
```

デフォルトの構成ファイル **xlf.cfg**

```
xlf: use=DEFLT
     options=<B>

DEFLT:
     options=<C>
```

上記の構成ファイル内のスタンザは、以下のオプション・セットを以下の順序で使用します。

1. スタンザ xlf は *B*、*D*、および *C* を使用する
2. スタンザ xlf_prepend は *B1*、*B*、*D*、*C* を使用する
3. スタンザ xlf_replace は *B2* を使用する
4. スタンザ xlf_append は *B*、*D*、*C* および *B3* を使用する

属性を 2 回以上指定する場合に、代入演算子を使用することもできます。例を以下に示します。

```
xlf:
    use=xlf
    options-=-Isome_include_path
    options+=some options
```

図 *4.* 追加の代入演算の使用

カスタム構成ファイルのスタンザの例

```
DEFLT: use=DEFLT
    options = -g
```

この例では、**-g** オプションをすべてのコンパイルで使用することを指定しています。

`xlf: use=xlf`
`  options+=-qlist`
 	
`xlf_r: use=xlf_r`
`  options+=-qlist`
	この例では、**xlf** および **xlf_r**コマンドにより呼び出されたすべてのコンパイルで、**-qlist** を使用することを指定します。このように **-qlist** を使用すると、システム構成ファイルで指定された **-qlist** のデフォルト設定がオーバーライドされます。

`DEFLT: use=DEFLT` `  libraries=-L/home/user/lib,-lmylib`	この例では、すべてのコンパイルが /home/user/lib/libmylib.a にリンクされることを指定しています。

Advance Toolchain との IBM XL Fortran for Linux, V16.1 の併用

IBM XL Fortran for Linux, V16.1 は、オープン・ソース開発ツールとランタイム・ライブラリーのセットである、IBM Advance Toolchain 9.0 をサポートします。IBM Advance Toolchain 9.0 を使用することで、最新の POWER® ハードウェア機構 (特に、チューニングされたライブラリー) を Linux 上で活用できます。Advance Toolchain 9.0 について詳しくは、Advance toolchain for Linux on Power Web サイトを参照してください。

IBM XL Fortran for Linux, V16.1 を Advance Toolchain と共に使用するには、以下の手順を実行します。

1. **at9.0** パッケージをデフォルトのインストール場所にインストールします。手順については、Advance toolchain for Linux on Power Web サイトの『Installation』を参照してください。
2. **xlf_configure** ユーティリティーを実行して **xlf.at.cfg**構成ファイルを作成します。**xlf.at.cfg** 構成ファイルで、XL Fortran コンパイラー以外の他のすべてのエンティティーは Advance Toolchain のエンティティーに送られます。このエンティティーには、リンカー、ヘッダー、およびランタイム・ライブラリーが含まれます。

   注**: xlf_configure** ユーティリティーを実行するには、root ユーザーになるか、`sudo` コマンドを使用する必要があります。

   ```
   xlf_configure -at
   ```
3. XL コンパイラーを Advance Toolchain サポートと共に呼び出します。
   - コンパイラーをデフォルトの場所にインストールした場合は、以下のコマンドを発行します。

     ```
     /opt/ibm/xlf/16.1.0/bin/xlf_at
     ```
   - コンパイラーを NDI の場所にインストールした場合は、以下のコマンドを発行します。

     ```
     $ndi_path/xlf/16.1.0/bin/xlf_at
     ```

注**:** XL コンパイラーを Advance Toolchain サポートと共に使用してアプリケーションを作成する場合、アプリケーションは Advance Toolchain のランタイム・ライブラリーに依存するため、Advance Toolchain 環境下でしか実行できなくなります。アプリケーションを他のマシン上で実行するためにコピーする場合は、

Advance Toolchain、または少なくとも Advance Toolchain のランタイム・ライブラリーが、それらのマシン上で使用可能であることを確認してください。

デフォルト構成ファイルの編集

構成ファイルは、呼び出された時にコンパイラーが使用する情報を指定します。XL Fortran は、インストール時にデフォルト構成ファイル `/opt/ibm/xlf/16.1.0/etc/xlf.cfg` を提供します。

多数のユーザーにいくつかの一連のコンパイラー・オプションの中から選択できるようにさせたい場合は、次のように新しく命名したスタンザを構成ファイルに追加して、既存のコマンドにリンクする新規コマンドを作成することもできます。例えば、以下と同様の方法で指定して、**xlf95** コマンドとのリンクを作成することができます。

```
ln -s /opt/ibm/xlf/16.1.0/bin/xlf95 /home/username/bin/xlf95
```

他の名前でコンパイラーを実行すると、コンパイラーは対応するスタンザにリストされているオプション、ライブラリーなどを使用します。

注**:**

- 構成ファイルには、リンクしたい他の名前付きスタンザが含まれています。
- 構成ファイルに変更を加えてから、別のシステムに makefiles を移動させたりコピーしたりする場合は、変更した構成ファイルをコピーすることも必要です。
- 構成ファイル内では、タブを区切り文字として使用することはできません。構成ファイルを修正する場合、字下げは必ずスペースで行ってください。

構成ファイルの属性

構成ファイルには、以下の属性が含まれています。

as_64 アセンブラーの絶対パス名。

asopt 例えば、コンパイラー・オプションとアセンブラー・オプションが同じ文字を使用している場合は、アセンブラー・オプションと見なして、オプションの名前をリストします。このリストは、連結されている単一文字フラグのセットです。引数を取るフラグの後にはコロンが続き、リスト全体が二重引用符で囲まれます。 **-W** コンパイラー・オプションによってアセンブラーにオプションを渡すよりも、この属性を設定した方が便利です。

bolt バインダーの絶対パス名。

code 最適化コード生成プログラムの絶対パス名。

cpp C プリプロセッサーの絶対パス名。特定のサフィックス (通常は **.F**) で終わっているファイルに対して自動的に呼び出されます。

cppoptions
コンマで区切られているオプションのストリング。**cpp** (C プリプロセッサー) は、これらのオプションが他のどのオプションよりも先にコマンド行に入力されたものと見なして処理します。この属性が必要な理由は、XL Fortran でコンパイルできる出力を作成するのに、通常 **cpp** オプションがいくつか必要になるからです。デフォルトは **-C** で、これは、出力に C ス

タイルのコメントを保持します。また、有効な **cpp** オプションに関しては、 156 ページの『-qfpp』 および 240 ページの『-qppsuborigarg』 オプションを参照してください。

注: コマンド行で **-C!** プリプロセッサー・オプションを指定して (**-WF**, **-C!**)、デフォルト設定をオーバーライドすることができます。

cppsuffix
XL Fortran でコンパイルする前に、 C プリプロセッサー (**cpp**) でファイルを前処理する必要があることを示すサフィックス。デフォルトでは **F** です。

crt_64 始動コードが入っているオブジェクト・ファイルのパス名。このオブジェクト・ファイルは、最初のパラメーターとしてリンケージ・エディターに渡されます。

CUDA Fortran **cuda_libdirs_64**
コンマ区切りの **-L** オプションによって指定された、CUDA Toolkit ライブラリーの検索パス。この属性は、コンパイラーが CUDA Fortran サポート付きで起動された場合のみ使用されます。 CUDA Fortran

CUDA Fortran **cuda_libraries_64**
CUDA Fortran プログラムのリンクに使用されるライブラリーを指定する、コンマ区切りの **-l** オプション。 CUDA Fortran

CUDA Fortran **cuda_path**
CUDA Toolkit ディレクトリーのベースの絶対パス名。 CUDA Fortran

defaultmsg
デフォルト・メッセージ・ファイルの絶対パス名。

fsuffix
Fortran ソース・ファイルに対して許可されているサフィックス。デフォルトは **f** です。コンパイラーは単一コンパイル内のすべてのソース・ファイルが同じサフィックスを持つことを要求します。したがって、他のサフィックスを持つファイル、例えば **f95** などをコンパイルするには、構成ファイル内のこの属性を変更するか、 **-qsuffix** コンパイラー・オプションを使用してください。 **-qsuffix** についての詳細は、 281 ページの『-qsuffix』を参照してください。

mcrt_64
crt_64 の場合と同じですが、オブジェクト・ファイルには **-p** オプションのプロファイル・コードがあります。

gcrt_64
crt_64 と同じですが、オブジェクト・ファイルには **-pg** オプションのプロファイル・コードがあります。

gcc_libs_64
GCC ライブラリーのパスを指定し、GCC ライブラリーをリンクするリンカー・オプション。

gcc_path_64
64 ビット・ツール・チェーンのパスを指定します。

hot 配列言語の変換を行うプログラムの絶対パス名。

include_64
コンパイル・インクルード・ファイル、モジュール・シンボル・ファイル、およびサブモジュール・シンボル・ファイルに使用する検索パスを指示します。

ipa　プロシージャー間最適化、ループ最適化、およびプログラムの並列化を実行するプログラムの絶対パス名。

ld_64　リンカーの絶対パス名。

ldopt　例えば、コンパイラー・オプションとリンカー・オプションが同じ文字を使用している場合は、リンカー・オプションと見なして、オプションの名前をリストします。このリストは、連結されている単一文字フラグのセットです。引数を取るフラグの後にはコロンが続き、リスト全体が二重引用符で囲まれます。

-W コンパイラー・オプションを使用してリンカーにオプションを渡すよりも、この属性を設定した方が便利な場合があります。しかし、いずれにしても、認識されないオプションの大部分はリンカーに渡されます。

GPU **llvm2ptx**
LLVM-IR から PTX への変換プログラムの絶対パス名。GPU

GPU **nvcc**
nvcc コンパイラーの絶対パス名。GPU

options
コンマで区切られているオプションのストリング。コンパイラーは、これらのオプションが他のどのオプションよりも先にコマンド行に入力されたものと見なして処理します。この属性を使用すると、通常使用されるオプションを中央の 1 か所に入れることができるため、コマンド行を短くできます。

osuffix
入力ファイルとして指定されているオブジェクト・ファイルを認識するのに使用されるサフィックス。デフォルトでは **o** です。

GPU パーティショナー
W-Code IR パーティショナーの絶対パス名。GPU

GPU **ptxas**
PTX アセンブラーの絶対パス名。GPU

slm_dir
SLM タグ・ファイルのディレクトリー。デフォルトは **/var/opt/ibm/xl-compiler/** (デフォルト・インストールの場合) または *$prefix*/**var/opt/ibm/xl-compiler/** (デフォルト以外のインストールの場合) です。 *$prefix* はデフォルト以外のインストール・パスです。

slm_period
各メトリックの対象となる秒数。 SLM デーモンは、定義された期間ごとに使用情報を出力します。デフォルトは **300** です。

slm_limit
各タグ・ファイルが占有可能な最大バイト数。デフォルトは **5000000** です。

slm_timeout
終了までデーモンが待機する必要がある最小秒数。デフォルトは **5** です。

slm_auth
許可ファイルのディレクトリー。デフォルトでは **/etc/XLAuthorizedUsers** です。

smplibraries
-qsmp コンパイラー・オプションを指定してコンパイルされたプログラムをリンクするのに使用するライブラリーを指定します。

ssuffix
入力ファイルとして指定されているアセンブラー・ファイルを認識するのに使用されるサフィックス。デフォルトでは **s** です。

use 属性の値は、指定した、もしくはローカルのスタンザから与えられます。単一値属性の場合は、ローカル・スタンザまたはデフォルト・スタンザに値が指定されていないと、 **use** 属性の値が適用されます。コンマで区切られているリストの場合は、 **use** 属性の値がローカル・スタンザの値に追加されます。単一レベルの **use** 属性だけがサポートされています。別の **use** 属性が入っているスタンザを指定する **use** 属性を指定してはなりません。

GPU **wc2llvm**
W-Code IR から LLVM-IR への変換プログラムの絶対パス名。 GPU

xlf メイン・コンパイラー実行可能ファイルの絶対パス名。コンパイラー・コマンドは、このファイルを実行するドライバー・プログラムです。

xlfopt 例えば、コンパイラー・オプションとリンカー・オプションが同じ文字を使用している場合は、コンパイラー・オプションと見なして、オプションの名前をリストします。このリストは、連結されている単一文字フラグのセットです。引数を取るフラグの後にはコロンが続き、リスト全体が二重引用符で囲まれます。

注:

- コンパイル・インクルード・ファイルに複数の検索パスを指定するには、次のように、それぞれのパス・ロケーションをコンマで区切ります。

```
include = -I/path1, -I/path2, ...
```

- 88 ページの『-F』 オプションを使用して、異なる構成ファイル、構成ファイル内の特定のスタンザ、またはその両方を選択するのに使用できます。

関連情報

- 23 ページの『XL Fortran 入力ファイル』
- 25 ページの『XL Fortran 出力ファイル』
- 337 ページの『第 9 章 コンパイラー・ライセンス使用状況の追跡』

インストールした XL Fortran のレベルの判別

特定のマシン上にインストールした XL Fortran のレベルが不明な場合があります。この情報はソフトウェア・サポートに連絡するときに必要になります。

システム・インストール・プロシージャーによって製品の最新レベルをインストールしたことを検査するには、次のコマンドを発行します。

SLES、**RHEL**、および **CentOS** 上

```
rpm -qa | grep xlf.16.1.0 | xargs rpm -qi
```

Ubuntu 上

```
dpkg -l xlf.16.1.0
```

この結果には、システム上にインストールされたコンパイラー・イメージのバージョン、リリース、モディフィケーション、修正レベルが含まれます。

また、**-qversion** コンパイラー・オプションを使用して、コンパイラーのバージョン、リリース、コンパイラーのレベル、およびそのコンポーネントを表示することもできます。

2 つのレベルの XL Fortran の実行

2 つの異なるレベルの XL Fortran コンパイラーを 1 つのシステムに共存させることができます。したがって、デフォルトで一方のレベルを呼び出し、明示的に選択すれば、いつでももう一方のレベルを呼び出すことができます。

これを行うための詳細については、「*XL Fortran* インストール・ガイド」を参照してください。

第 4 章 XL Fortran プログラムの編集、コンパイル、リンク、実行

ほとんどの Fortran プログラム開発は、編集、コンパイル/リンク (デフォルトは単一ステップ)、および実行のサイクルの繰り返しから構成されています。開発サイクルで発生した問題については、次のセクションを参考にしてください。

前提条件の情報:

1. 必須の Linux 設定 (例えば、ある一定の環境変数およびストレージの限界) すべてがユーザー ID に対して正しくなければ、コンパイラーを使用することはできません。詳しくは、 9 ページの『環境変数の正しい設定方法』を参照してください。
2. XL Fortran プログラムの書き方および最適化についてさらに学ぶには、「*XL Fortran* ランゲージ・リファレンス」および「*XL Fortran* 最適化およびプログラミング・ガイド」を参照してください。

入出力ファイル

このトピックでは、XL Fortran が認識するファイル・タイプについて説明します。

XL Fortran 入力ファイル

コンパイラーへの入力ファイルには次のものがあります。

ソース・ファイル **(**サフィックス **.f** または **.F** あるいはそのバリエーション**)**

.f、**.f77**、**.f90**、**.f95**、**.f03**、**.f08**、CUDA Fortran **.cuf** CUDA Fortran 、**.F**、**.F77**、**.F90**、**.F95**、**.F03**、**.F08**、および CUDA Fortran **.CUF** CUDA Fortran ファイルのすべてが、コンパイル対象のソース・ファイルです。コンパイラーは、指定されたソース・ファイルをコマンド行で指定された順序でコンパイルします。指定されたソース・ファイルが見つからない場合、コンパイラーはエラー・メッセージを作成し、次のファイルがあれば、そのファイルの処理に移ります。大文字 **F** を含むサフィックスを持つファイルは、コンパイルされる前に C プリプロセッサー (**cpp**) に渡されます。

インクルード・ファイルもソースを含んでいて、**.f** 以外のサフィックスを持っていることがしばしばあります。

関連情報: 39 ページの『C プリプロセッサーによる Fortran ファイルの引き渡し』を参照してください。

17 ページの『デフォルト構成ファイルの編集』および 281 ページの『-qsuffix』 に記載されている **fsuffix** および **cppsuffix** 属性を使用する場合は、別のサフィックスを選択します。

オブジェクト・ファイル **(.o** サフィックス**)**

.o ファイルはすべてオブジェクト・ファイルです。コンパイラーはソース・ファイルをコンパイルした後、その結果作成された **.o** ファイルと、入力ファイルとして指定した **.o** ファイル、およびプロダクト・ディレクトリ

ーやシステム・ライブラリー・ディレクトリーにあるいくつかの **.o** ファイルや **.a** ファイルを、**ld** コマンドを使用してリンク・エディットし、1 つの実行可能出力ファイルを作成します。

関連情報: 77 ページの『リンク』および 41 ページの『XL Fortran プログラムのリンク』を参照してください。

osuffix 属性 (17 ページの『デフォルト構成ファイルの編集』および 281 ページの『-qsuffix』 に説明されている) を使用して、別のサフィックスを選択することができます。

アセンブラー・ソース・ファイル **(.s** サフィックス**)**
コンパイラーは、指定された **.s** ファイルをアセンブラー (**as**) に送ります。アセンブラー出力は、リンク時にリンカーに送られるオブジェクト・ファイルから構成されます。

関連情報: **ssuffix** 属性 (17 ページの『デフォルト構成ファイルの編集』および 281 ページの『-qsuffix』 に説明されている) を使用して、別のサフィックスを選択することができます。

共有オブジェクトまたはライブラリー・ファイル **(.so** サフィックス**)**
実行時にマルチプロセスによってロードされ共用されることが可能なオブジェクト・ファイルです。リンク時に共有オブジェクトが指定されると、オブジェクトに関する情報は出力ファイルに記録されますが、共有オブジェクトからのコードは実際に出力ファイルには含まれません。

GPU **LLVM IR** ビットコード・ライブラリー **(**サフィックス **.bc)**
コンパイラーは、LLVM IR ビットコード・ライブラリーを NVVM-IR から PTX への変換プログラム (llvm2ptx) に渡します。 GPU

構成ファイル **(.cfg** サフィックス**)**
構成ファイルの内容は、コンパイル・プロセスの多くの面 (最も一般的なのは、コンパイラーのデフォルト・コンパイル・オプション) を決定します。構成ファイルによって、各種のデフォルト時コンパイラー・オプションをまとめたり、 1 つのシステム上に複数のレベルの XL Fortran コンパイラーを残すことができます。

デフォルトの構成ファイルは、`/opt/ibm/xlf/16.1.0/etc/xlf.cfg` です。

関連情報: 構成ファイルの選択に関する情報については、 12 ページの『カスタム・コンパイラー構成ファイルの使用』および 88 ページの『-F』を参照してください。

モジュール・シンボル・ファイル **(***modulename***.mod)**
モジュール・シンボル・ファイルは、モジュールのコンパイルから作成された出力ファイルであり、そのモジュールを使用するファイルの以降のコンパイル用の入力ファイルになります。個々のモジュールに対して **.mod** ファイルが 1 つずつ作成され、したがって、ソース・ファイルを 1 つコンパイルすると、複数の **.mod** ファイルが作成できます。

関連情報: 94 ページの『-I』および 219 ページの『-qmoddir』を参照してください。

F2008 サブモジュール・シンボル・ファイル:
ancestormodulename_submodulename.**smod**

サブモジュール・シンボル・ファイルは、サブモジュールのコンパイルから作成された出力ファイルであり、下位サブモジュールの以降のコンパイル用の入力ファイルになります。各サブモジュールごとに 1 つの **.smod** ファイルが作成されるので、単一のソース・ファイルをコンパイルすると、複数の **.smod** ファイルが作成されることがあります。

サブモジュール・シンボル・ファイルは、上位モジュールのコンパイルや、参照結合を通じて上位モジュールにアクセスするコンパイル単位のコンパイルには必要ありません。

関連情報: 219 ページの『-qmoddir』を参照してください。

F2008

プロファイル・データ・ファイル

-qpdf1 オプションは、以降のコンパイルで使用する、実行時プロファイル情報を作成します。この情報は、パターン 『**.*pdf***』 または 『**.*pdf_map***』 に一致する名前で 1 つ以上の隠しファイルに格納されます。

関連情報: 228 ページの『-qpdf1、-qpdf2』を参照してください。

XL Fortran 出力ファイル

XL Fortran が提供する出力ファイルは、以下のとおりです。

実行可能ファイル **(a.out)**

デフォルト時、 XL Fortran は現行ディレクトリーに **a.out** という名前の実行可能ファイルを作成します。

関連情報: 別の名前を選択することについての情報は 104 ページの『-o』を、オブジェクト・ファイルのみを生成することについての情報は 85 ページの『-c』をそれぞれ参照してください。

オブジェクト・ファイル **(***filename***.o)**

-c コンパイラー・オプションを指定すると、コンパイラーは実行可能ファイルを作成する代わりに、指定された個々のソース・ファイルに対してオブジェクト・ファイルを 1 つ作成し、アセンブラーは指定された個々のアセンブラー・ソース・ファイルに対してオブジェクト・ファイルを 1 つ作成します。デフォルト時には、オブジェクト・ファイルはソース・ファイルと同じファイル名プレフィックスを持ち、現行ディレクトリーに存在します。

関連情報:

- **-c** コンパイラー・オプションについては、 85 ページの『-c』および 41 ページの『XL Fortran プログラムのリンク』を参照してください。
- オブジェクト・ファイルの名前変更についての情報は、 104 ページの『-o』を参照してください。

アセンブラー・ソース・ファイル **(***filename***.s)**

-S コンパイラー・オプションを指定すると、XL Fortran コンパイラーは実行可能ファイルを作成する代わりに、指定された個々のソース・ファイルに

対して同等のアセンブラー・ソース・ファイルを 1 つ作成します。デフォルト時には、アセンブラー・ソース・ファイルはソース・ファイルと同じファイル名プレフィックスを持ち、現行ディレクトリーに存在します。

関連情報: 317 ページの『-S』および 41 ページの『XL Fortran プログラムのリンク』を参照してください。アセンブラー・ソース・ファイルの名前変更についての情報は、104 ページの『-o』を参照してください。

コンパイラー・リスト・ファイル **(*filename*.lst)**
デフォルト時には、 1 つ以上のリスト関連のコンパイラー・オプションを指定しない限り、リストは作成されません。リスト・ファイルは現行ディレクトリーに入れられ、ソース・ファイルと同じファイル名プレフィックスと、サフィックス **.lst** を持っています。

関連情報: 72 ページの『リスト、メッセージ、およびコンパイラー情報』を参照してください。

モジュール・シンボル・ファイル **(*modulename*.mod)**
個々のモジュールには、そのモジュールを使用するプログラム単位、サブプログラム、インターフェース本体によって必要とされる情報が含まれる関連のシンボル・ファイルがあります。デフォルト時には、これらのシンボル・ファイルは現行ディレクトリーに入っている必要があります。

関連情報: 別のディレクトリーに **.mod** ファイルを書き込むことに関する情報については、219 ページの『-qmoddir』を参照してください。

F2008 サブモジュール・シンボル・ファイル:
ancestormodulename_submodulename.smod
個々のモジュールには、下位サブモジュールによって必要とされる情報が含まれる関連のシンボル・ファイルがあります。デフォルト時には、これらのシンボル・ファイルは現行ディレクトリーに入っている必要があります。

サブモジュール・シンボル・ファイルは、上位モジュールのコンパイルや、参照結合を通じて上位モジュールにアクセスするコンパイル単位のコンパイルには必要ありません。

関連情報: 別のディレクトリーに **.smod** ファイルを書き込むことに関する情報については、219 ページの『-qmoddir』を参照してください。

F2008

cpp 前処理済みソース・ファイル **(F*filename*.f)**
サフィックス **.F** を持つファイルをコンパイルする時に **-d** オプションを指定すると、 C プリプロセッサー (cpp) によって作成された中間ファイルが削除されないで保管されます。

関連情報: 39 ページの『C プリプロセッサーによる Fortran ファイルの引き渡し』および 87 ページの『-d』を参照してください。

プロファイル・データ・ファイル **(.*pdf*, .*pdf_map*)**
これらのファイルは **-qpdf1** オプションによって生成され、それ以後のコンパイルで、実際の実行結果に基づく最適化を調整するために使用されます。

関連情報: 228 ページの『-qpdf1、-qpdf2』を参照してください。

従属関係ファイル: *filename*.**d**

従属関係ファイルには、ソース・ファイルの従属関係情報が含まれています。従属関係ファイルは **make** コマンドによって使用され、ファイルのコンパイル順序が特定されるとともに、いずれかのファイルが変更された場合に再コンパイルする必要のある最低限のファイル・セットが特定されます。 99 ページの『-MMD』 オプションまたは 210 ページの『-qmakedep』 オプションを指定して、従属関係ファイルを生成できます。

関連情報: 従属関係ファイルの名前の設定については、 97 ページの 『-MF』を参照してください。従属関係ファイル内のオブジェクト・ファイルのターゲット名の指定については、 99 ページの『-MT』を参照してください。

XL Fortran ソース・ファイルの編集

Fortran ソース・プログラムを作成するために、**vi** または **emacs** などの使用可能なテキスト・エディターを使用することができます。ソース・プログラムにはサフィックス **.f** またはそのバリエーションがなければなりません。ただし、構成ファイルの **fsuffix** 属性で異なるサフィックスを指定する場合や、**-qsuffix** コンパイラー・オプションを使用する場合を除きます。コンパイルを開始する前に処理しなければならない C プリプロセッサー (**cpp**) ディレクティブがプログラムの中に入っている場合は、サフィックス **.F** またはそのバリエーションも使用することができます。サフィックス **.f77**、**.f90**、**.f95**、**.f03**、**.f08**、または CUDA Fortran **.cuf** CUDA Fortran を持つソース・ファイルもまた有効です。

Fortran ソース・プログラムが有効なプログラムであるためには、「*XL Fortran* ランゲージ・リファレンス」で指定されている言語定義に従っていなければなりません。

コンパイラー・オプションの指定

コンパイラー・オプションは、コンパイラー特性の設定、作成されるオブジェクト・コードやコンパイラー出力の指定、いくつかのプリプロセッサー機能の実行など、さまざまな機能を実行します。コンパイラー・オプションは、次の 1 つ以上の方法で指定することができます。

- コマンド行
- .cfg 拡張子を持つファイルである、カスタム構成ファイル
- ソース・プログラム
- システム環境変数
- Make ファイル

オプション設定の有効範囲と優先順位

3 つの位置のいずれかにコンパイラー・オプションを指定することができます。有効範囲と優先順位は、使用する位置で定義されます。 (XL Fortran には、オプション設定を指定できるコメント・ディレクティブ (**SOURCEFORM** など) がありますす。そのようなディレクティブの有効範囲と優先順位に関する一般的な規則はありません。)

位置	有効範囲	優先順位
構成ファイルのスタンザの中	実際にそのスタンザでコンパイルされたすべてのファイル内のすべてのコンパイル単位。	下位
コマンド行	そのコマンドでコンパイルされたすべてのファイル内のすべてのコンパイル単位。	中間
@PROCESS ディレクティブ (XL Fortran は、**SOURCEFORM** などの、オプション設定を指定できるコメント・ディレクティブも持っています。そのようなディレクティブの有効範囲と優先順位に関する一般的な規則はありません。)	次のコンパイル単位	上位

異なる設定で複数回オプションが指定されると、通常は最後の設定が効力を発します。例外はどれも 81 ページの『第 6 章 XL Fortran コンパイラー・オプションの詳細記述』 の個々の説明に示され、「競合オプション」という索引が付けられています。

コマンド行でのオプションの指定

XL Fortran は、従来の UNIX によるコマンド行オプションの指定方法をサポートしています。この方法では、次のように、負符号 (-) の後に 1 つ以上の文字 (フラグといいます) を指定します。

```
xlf95 -c file.f
```

多くの場合、複数のフラグを連結することも、個々に指定することもできます。

```
xlf95 -cv file.f     # These forms
xlf95 -c -v file.f   # are equivalent
```

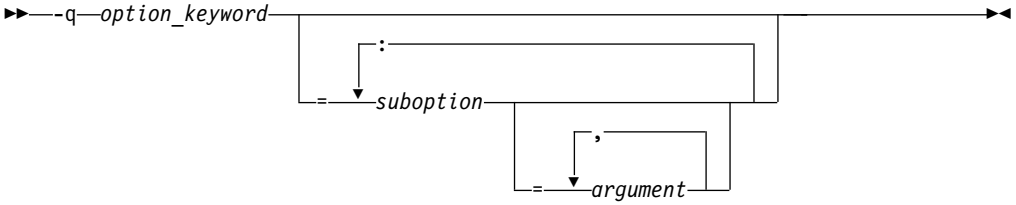
(例外がいくつかあります。例えば、**-pg**。これは単一オプションで、**-p -g** と同じではありません。)

フラグの中には、引数ストリングがさらに必要なものもあります。また、XL Fortran はそれらのフラグの解釈で柔軟性を持っています。最後に引数を指定したフラグであれば、複数のフラグを連結することができます。フラグを指定する方法について、以下の例で示します。

```
# All of these commands are equivalent.
  xlf95 -g -v -o montecarlo -p montecarlo.f
  xlf95 montecarlo.f -g -v -o montecarlo -p
  xlf95 -g -v montecarlo.f -o montecarlo -p
  xlf95 -g -v -omontecarlo -p montecarlo.f
# Because -o takes a blank-delimited argument,
# the -p cannot be concatenated.
  xlf95 -gvomontecarlo -p montecarlo.f
# Unless we switch the order.
  xlf95 -gvpomontecarlo montecarlo.f
```

他のコンパイラー、特に XL ファミリーのコンパイラーに精通していれば、既にこれらのフラグの多くにも精通していることでしょう。

覚えやすい形式で多数のコマンド行オプションを指定して、コンパイル・スクリプトおよび makefiles を理解しやすくすることができます。



この形式では、ブランクの挿入に関しては、より制限的です。それぞれの **-q** オプションはブランクで区切らなければならず、 **-q** オプションと、その後に続く引数ストリングとの間にブランクがあってはなりません。フラグ・オプションの名前とは異なり、**-q** オプション名には、 **q** が小文字でなければならないことを除いて、大文字と小文字の区別がありません。 **-q** オプションとこれが必要としている引数を分離するには等号を使用し、引数ストリング内のサブオプションを分離するにはコロンを使用してください。

例えば、次のようになります。

```
xlf95 -qddim -qXREF=full -qfloat=nomaf:rsqrt -O3 -qcache=type=c:level=1 file.f
```

ソース・ファイルでのオプションの指定

ソース・ファイルに **@PROCESS** コンパイラー指示を入れることによって、個々のコンパイル単位に影響を与えるようにコンパイラー・オプションを指定することができます。 **@PROCESS** コンパイラー指示によって、構成ファイル、デフォルト設定、またはコマンド行で指定したオプションをオーバーライドすることができます。

option　これは、**-q** を持たないコンパイラー・オプションの名前です。

suboption
　　コンパイラー・オプションのサブオプションです。

固定ソース形式では、**@PROCESS** は 1 桁目から、または 6 桁目より後に開始できます。自由ソース形式では、**@PROCESS** コンパイラー指示はどの桁からでも開始できます。

ステートメント・ラベルまたはインライン・コメントを **@PROCESS** コンパイラー指示と同じ行に入れることはできません。

デフォルト時には、**@PROCESS** コンパイラー指示で指定するオプション設定は、文が存在するコンパイル単位に対してのみ有効です。ファイルが複数のコンパイル単位を持っている場合は、オプション設定は、次の単位がコンパイルされる前に、

元の状態にリセットされます。 **DIRECTIVE** オプションによって指定されたトリガー定数は、ファイルの終わりまで (または **NODIRECTIVE** が処理されるまで) 有効です。

@PROCESS コンパイラー指示は、通常、コンパイル単位の最初の文の前になければなりません。唯一の例外は、**SOURCE** および **NOSOURCE** を指定する場合です。この 2 つは、コンパイル単位内のいかなる場所にある **@PROCESS** ディレクティブでも使用することができます。

コマンド行オプションの「ld」または「as」コマンドへの引き渡し

コンパイラーは、コンパイル中に必要に応じて他のコマンド (例えば **ld** および **as**) を自動的に実行するので、通常は、これらのコマンドのオプションにユーザーが関与する必要はありません。 これらの個々のコマンドに対してオプションを選択したい場合は、次のようにできます。

- コンパイラー・コマンド行にリンカー・オプションを入れます。コンパイラーが **-q** オプション以外のコマンド行オプションを認識しないと、そのオプションをリンカーに渡します。例:

  ```
  xlf95 --print-map file.f # --print-map is passed to ld
  ```

- コンパイラー・オプション **-W** または **-X** を使用して、コマンドの引数リストを作成してください。例:

  ```
  xlf95 -Wl,--print-map file.f # --print-map is passed to ld
  ```

  この例では、**ld** オプション **--print-map** はリンカー (**-Wl** オプションの **l** で指示される) の実行時にリンカーに渡されます。

  この形式は、前の形式よりも一般的です。なぜなら、**-W** オプションの後にさまざまな英字を使用することにより、**as** コマンドおよびコンパイル中に呼び出される他のコマンドに代わって機能するからです。

- 構成ファイル /opt/ibm/xlf/16.1.0/etc/xlf.cfgを編集するか、あるいは、独自の構成ファイルを作成してください。特定のスタンザをカスタマイズして、特定のコマンド行オプションをアセンブラーまたはリンカーに渡せるようにできます。

  例えば、/opt/ibm/xlf/16.1.0/etc/xlf.cfg の **xlf95** スタンザの中にこれらの行を組み込むには、次のようにします。

  ```
  asopt = "W"
  ldopt = "M"
  ```

  を入れて、次のコマンド

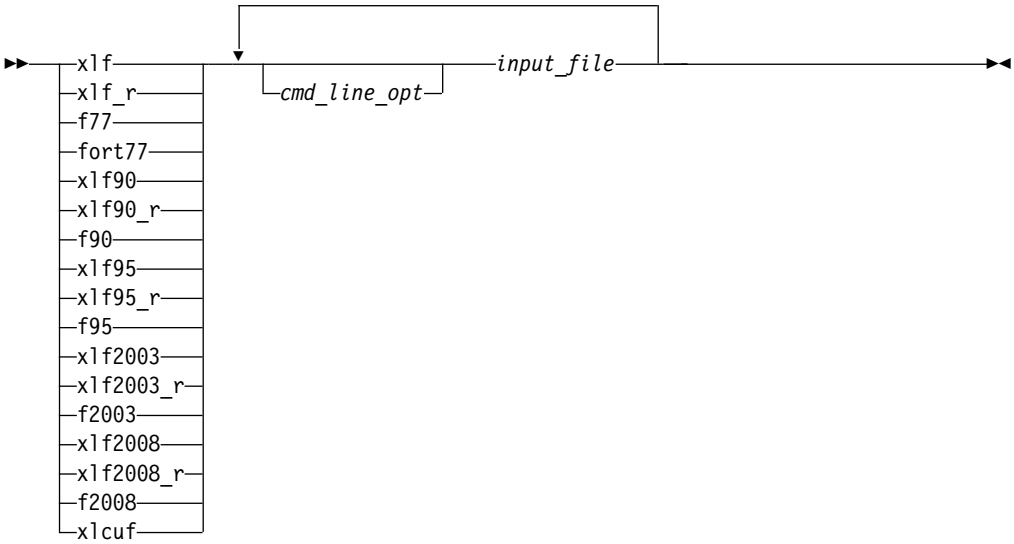
  ```
  xlf95 -Wa,-Z -Wl,-s -w produces_warnings.s uses_many_symbols.f
  ```

  を発行すると、**produces_warnings.s** ファイルはオプション **-W** と **-Z** (警告を出して、コンパイル・エラーがあってもオブジェクト・ファイルを作成する) でアセンブルされ、オプション **-s** と **-M** (最終実行可能ファイルを除去し、ロード・マップを作成する) でリンカーが呼び出されます。

関連情報: 323 ページの『-W、-X』および 12 ページの『カスタム・コンパイラー構成ファイルの使用』を参照してください。

XL Fortran プログラムのコンパイル

ソース・プログラムをコンパイルするには、コマンド **xlf**、**xlf_r**、 **xlf90**、**xlf90_r**、**f90**、**xlf95**、**xlf95_r**、 **f95**、 **xlf2003**、**xlf2003_r**、**f2003**、 **xlf2008**、**xlf2008_r**、**f2008**、および CUDA Fortran **xlcuf** CUDA Fortran のいずれかを使用してください。形式は次のとおりです。



これらのコマンドはすべて、本質的に同じ Fortran 言語を受け入れます。主要な違いは、別のデフォルト・オプション (構成ファイル `/opt/ibm/xlf/16.1.0/etc/xlf.cfg`) を使用していることです。

呼び出しコマンドは、Fortran ソース・ファイルをコンパイルし、すべての **.s** ファイルをアセンブルし、オブジェクト・ファイルとライブラリーをリンクして 1 つの実行可能プログラムを作成するのに必要なステップを実行します。特に、コマンド **xlf_r**、 **xlf90_r**、 **xlf95_r**、 **xlf2003_r**、 **xlf2008_r**、および CUDA Fortran **xlcuf** CUDA Fortran は、マルチスレッド用方式コンポーネント (ライブラリー、など) を使用して、オブジェクト・ファイルをリンクおよびバインドします。

以下に示す表では、使用できる呼び出しコマンドを要約します。

表 6. *XL Fortran* 呼び出しコマンド

ドライバー呼び出し [1]	主な機能	リンクされるライブラリー
xlf	選択された Fortran 言語レベル [3]	libxlf90.so
xlf_r	選択された言語レベルのスレッド・セーフ・バージョン [3]	libxlf90_r.so
f77, **fort77**	FORTRAN 77	libxlf90.so
xlf90, **f90**	Fortran 90	libxlf90.so
xlf90_r	スレッド・セーフ Fortran 90	libxlf90_r.so
xlf95, **f95**	Fortran 95	libxlf90.so
xlf95_r	スレッド・セーフ Fortran 95	libxlf90_r.so

表 6. *XL Fortran* 呼び出しコマンド (続き)

ドライバー呼び出し [1]	主な機能	リンクされるライブラリー
xlf2003	Fortran 2003	libxlf90.so
xlf2003_r	スレッド・セーフ Fortran 2003	libxlf90_r.so
f2003	Fortran 2003	libxlf90.so
xlf2008	Fortran 2008	libxlf90.so
xlf2008_r	スレッド・セーフ Fortran 2008	libxlf90_r.so
f2008	Fortran 2008	libxlf90.so
CUDA Fortran **xlcuf** [2]	CUDA Fortran サポート	libxlf90_r.so、libcudadevrt.a、libcudart.so、 libcuda.so、および libdevice*.bc CUDA Fortran

注:

1. これらの呼び出しコマンドは、`/opt/ibm/xlf/16.1.0/bin` ディレクトリーにあります。
2. CUDA Fortran **xlcuf** を指定することは、**xlf2008_r** および **-qcuda** オプションを指定することと同じです。
3. 呼び出しコマンド **xlf** および **xlf_r** は、ソース・ファイル名のサフィックスに従って適切な言語レベルを選択します。他の呼び出しコマンドは、Fortran ソース・ファイル名のサフィックスにかかわらず、一貫した動作を示します。次の例を参照してください。

   例 **1**

   ```
   xlf program1.f program2.f90 program3.f95 program4.f03 program5.f08
   ```

   動作は、次のようになります。

   - `program1.f` ファイルは、呼び出しコマンドが **f77** であった場合と同じようにコンパイルされます。
   - `program2.f90` ファイルは、呼び出しコマンドが **xlf90** であった場合と同じようにコンパイルされます。
   - `program3.f95` ファイルは、呼び出しコマンドが **xlf95** であった場合と同じようにコンパイルされます。
   - `program4.f03` ファイルは、呼び出しコマンドが **xlf2003** であった場合と同じようにコンパイルされます。
   - `program5.f08` ファイルは、呼び出しコマンドが **xlf2008** であった場合と同じようにコンパイルされます。

   CUDA Fortran

   例 **2**

   ```
   xlf_r program6.cuf
   ```

`program6.cuf` ファイルは、呼び出しコマンドが **xlcuf** であった場合と同じようにコンパイルされます。これにより、CUDA Fortran サポートが有効になります。

CUDA Fortran

libxlf90.so はスレッド化されたアプリケーションと非スレッド化されたアプリケーションの両方に提供されます。XL Fortran は、実行時にアプリケーションがスレッド化されているかどうかを判別します。

XL Fortran は、 **libxlf90_r.so** に加えて、ライブラリー **libxlf90_t.so** を提供します。 **libxlf90_t.so** は、 **libxlf90_r.so** がエクスポートするのと同じエントリー・ポイントをエクスポートします。 ライブラリー **libxlf90_r.so** は、**libxlf90_t.so** のスーパーセットです。ファイル **xlf.cfg** は、**xlf90_r**、**xlf95_r**、および **xlf_r** コマンドを使用すると自動的に **libxlf90_r.so** にリンクするようセットアップされます。 **libxlf90_t.so** は、一部スレッドをサポートするランタイム・ライブラリーです。 **libxlf90_r.so** と異なり、**libxlf90_t.so** はスレッド同期を提供しません。また、 **libxlf90_t.so** 内のルーチンは再入可能ではありません。そのため、入出力操作の実行または Fortran 組み込み機能の呼び出しができる Fortran スレッドは、一度に 1 つだけです。 1 つの Fortran スレッドしかないマルチスレッド・アプリケーションで、**libxlf90_r.so** の代わりに **libxlf90_t.so** を使用して、**libxlf90_r.so** におけるスレッド同期のオーバーヘッドを回避することができます。

マルチスレッドの実行可能プログラムを複数の Fortran スレッドとバインドする場合は、 **libxlf90_r.so** を使用する必要があります。呼び出しコマンド **xlf_r**、**xlf90_r**、 **xlf95_r**、 **xlf2003_r**、 **xlf2008_r**、または CUDA Fortran **xlcuf** CUDA Fortran を使用することによって、正しいリンクが保証されることに注意してください。

呼び出しコマンドまたはオプションによって暗黙指定されるディレクティブ・トリガーは、次のリストのとおりです。

- 呼び出しコマンド **f77**、 **fort77**、 **f90**、 **f95**、 **f2003**、 **xlf**、 **xlf90**、 **xlf95**、**xlf2003**、および **xlf2008** では、ディレクティブ・トリガーはデフォルトで **IBM*** です。
- その他のすべての呼び出しコマンドでは、ディレクティブ・トリガーはデフォルトで **IBM*** および **IBMT** です。
- **-qsmp** が有効であれば、コンパイラーはディレクティブ・トリガー **IBMP**、**SMP$**、および **$OMP** も認識します。
- **-qthreaded** が有効であれば、コンパイラーは **IBMT** ディレクティブ・トリガーも認識します。

関連情報:

61 ページの『第 5 章 機能カテゴリー別コンパイラー・オプションの要約』
81 ページの『第 6 章 XL Fortran コンパイラー・オプションの詳細記述』

コンパイルのシナリオ

このセクションでは、XL Fortran でコンパイルできるプログラムについて説明します。

CUDA Fortran プログラムのコンパイル

CUDA Fortran サポートは、以下のいずれかの方式で有効にできます。

- **xlcuf** 呼び出しコマンドを使用して任意の Fortran ソース・ファイルをコンパイルします。 **xlcuf** を指定することと、**xlf2008_r** と **-qcuda** オプションを指定することは同じことです。
- **xlf_r** 呼び出しコマンドを使用して、**.cuf** 接尾部または **.CUF** 接尾部を持つソース・ファイルをコンパイルします。 **xlf_r** 呼び出しコマンドは **.cuf** ファイルと **.CUF** ファイルを CUDA Fortran ファイルとして認識し、CUDA Fortran サポートを自動的に有効にします。
- 任意のスレッド・セーフ呼び出しコマンド (**xlf95_r** など) を使用して **-qcuda** オプションを指定し、任意の Fortran ソース・ファイルをコンパイルします。

Fortran 2008 プログラムのコンパイル

f2008、**xlf2008**、および **xlf2008_r** コマンドは、他の呼び出しコマンドよりも、ご使用のプログラムの Fortran 2008 標準 への準拠性を高めます。 Fortran 2008 コマンドはデフォルトで自由ソース形式を受け入れます。Fortran 2008 コマンドの I/O 形式は、**f95**、 **xlf95**、 **xlf95_r**、 **xlf95_r7**、 **f2003**、 **xlf2003**、および **xlf2003_r** の各コマンドと同様です。 Fortran 2008 コマンドは、無限大および NaN 浮動小数点の値を、Fortran 2003 コマンドと同じ方法でフォーマット設定します。Fortran 2008 コマンドにより、デフォルトでポリモアフィズムを使用可能に設定します。

デフォルトでは、**f2008**、**xlf2008**、および **xlf2008_r** コマンドは Fortran 2008 標準に完全には準拠していません。完全に準拠する必要がある場合には、次の追加のコンパイラー・サブオプションを使用してコンパイルします。

```
-qlanglvl=2008std -qnodirective -qnoescape -qfloat=nomaf:rndsngl:nofold
   -qnoswapomp -qstrictieeemod
```

次の実行時オプションもまた指定してください。

```
XLFRTEOPTS="err_recovery=no:langlvl=2008std:iostat_end=2003std:
            internal_nldelim=2003std"
```

関連情報

- 『Fortran 2003 プログラムのコンパイル』

Fortran 2003 プログラムのコンパイル

f2003、**xlf2003**、および **xlf2003_r** コマンドは、他の呼び出しコマンドよりも、ご使用のプログラムの Fortran 2003 標準 への準拠性を高めます。Fortran 2003 コマンドはデフォルトで自由ソース形式を受け入れます。Fortran 2003 コマンドの I/O 形式は、**f95**、**xlf95**、および **xlf95_r** コマンドと同様です。Fortran 2003 コマンドは、無限大および NaN 浮動小数点の値を、前のコマンドと異なる方法でフォーマット設定します。 Fortran 2003 コマンドにより、デフォルトでポリモアフィズムを使用可能に設定します。

デフォルトでは、**f2003**、**xlf2003**、および **xlf2003_r** コマンドは Fortran 2003 標準に完全には準拠していません。完全に準拠する必要がある場合には、次の追加のコンパイラー・サブオプションを使用してコンパイルします。

```
-qlanglvl=2003std -qnodirective -qnoescape -qfloat=nomaf:rndsngl:nofold
   -qnoswapomp -qstrictieeemod
```

次の実行時オプションもまた指定してください。

```
XLFRTEOPTS="err_recovery=no:langlvl=2003std:iostat_end=2003std:
            internal_nldelim=2003std"
```

Fortran 90 プログラムまたは Fortran 95 プログラムのコンパイル

コマンド **f90**、**xlf90**、および **xlf90_r** は、他の呼び出しコマンドより厳密にプログラムを Fortran 90 標準に準拠させます。 **f95**、**xlf95**、および **xlf95_r** コマンドは、他の呼び出しコマンドより厳密にプログラムを Fortran 95 標準に準拠させます。 のセットとで異なります。**f90**、**xlf90**、**xlf90_r**、 **f95**、**xlf95**、および **xlf95_r** は、新しいプログラムをコンパイルする場合の推奨コマンドです。これらのコマンドはどちらも Fortran 90 の自由ソース形式がデフォルトで使用できます。これを固定ソース形式に使用するには、**-qfixed** オプションを使用する必要があります。I/O 形式は、これらのコマンドとそれ以外のコマンドではわずかに異なっています。また、I/O 形式も、コマンド **f90**、 **xlf90**、 **xlf90_r** のセットと、コマンド **f95**、**xlf95**、 **xlf95_r** のセットで異なっています。できる限り、データ・ファイルに関しては Fortran 95 形式に切り替えることをお勧めします。

デフォルトでは、コマンド **f90**、**xlf90**、 および **xlf90_r** は、Fortran 90 標準に完全には準拠していません。また、デフォルトでは、コマンド **f95**、**xlf95**、 および **xlf95_r** は、Fortran 95 標準に完全には準拠していません。Fortran 90 または Fortran 95 への完全な準拠を必要とする場合は、次のコンパイラー・オプション (およびサブオプション) のいずれかを指定してコンパイルしてください。

```
-qnodirective -qnoescape -qfloat=nomaf:nofold -qnoswapomp
-qlanglvl=90std
-qlanglvl=95std
```

また、プログラムを実行する前に、次のいずれかのコマンドを使用して実行時オプションを指定してください。

```
export XLFRTEOPTS="err_recovery=no:langlvl=90std"
export XLFRTEOPTS="err_recovery=no:langlvl=95std"
```

デフォルト設定は、パフォーマンスとユーザビリティーの最善の組み合わせが得られるように設計されています。したがって、通常、デフォルト設定は必要な場合にだけ変更するようにしてください。上記のオプションの一部は、非常に特殊な状況で適合性を得るためにだけ必要です。

XL Fortran バージョン 2 プログラムのコンパイル

xlf は、.f ファイル、.F ファイル、.f77 ファイル、または .F77 ファイルのコンパイルに使用される場合、可能であれば必ず、以前のバージョンの XL Fortran と同じ I/O 形式、および一部の FORTRAN 77 互換の実装動作を使用することにより、既存のプログラムとの互換性を維持します。

構成ファイルがカスタマイズされていない場合、**xlf** を使用して .f ファイル、.F ファイル、.f77 ファイル、または .F77 ファイルをコンパイルするときに、**f77** は **xlf** と同様に機能します。

既存の makefiles および build 環境との互換性を保持するために、これらのコマンドを引き続き使用しなければならない場合があります。ただし、これらのコマンドでコンパイルされたプログラムは、細かい点で新しい標準に従っていない場合があることに留意してください。

ライブラリーのコンパイルとリンク

このセクションでは、ソース・ファイルをコンパイルしてオブジェクト・ファイルを作成し、ライブラリーに組み込む方法、ライブラリーをメインプログラムにリンクする方法、およびあるライブラリーを別のライブラリーにリンクする方法について説明します。

静的ライブラリーのコンパイル: 静的ライブラリーをコンパイルするには、次のようにします。

1. 各ソース・ファイルを、リンクを行わずにオブジェクト・ファイルへとコンパイルします。例を以下に示します。

   ```
   xlf -c bar.f example.f
   ```

2. **ar** コマンドを使用して、生成されたオブジェクト・ファイルをアーカイブ・ライブラリー・ファイルに追加します。例を以下に示します。

   ```
   ar -rv libfoo.a bar.o example.o
   ```

共有ライブラリーのコンパイル: 共有ライブラリーをコンパイルするには、**-qpic** オプションを使用する必要があります。

次の手順を実行して、共有ライブラリーをコンパイルします。

1. ソース・ファイルを、リンクを行わずに 1 つのオブジェクト・ファイルへコンパイルします。次に例を示します。

   ```
   xlf -qpic -c foo.f
   ```

2. 生成されたオブジェクト・ファイルから共有オブジェクトを作成するには、**-qmkshrobj** コンパイラー・オプションを使用します。例を以下に示します。

   ```
   xlf -qmkshrobj -o libfoo.so foo.o
   ```

「*XL Fortran* コンパイラー・リファレンス」内の関連情報


-qpic


-qmkshrobj

アプリケーションへのライブラリーのリンク: 同じコマンド・ストリングを使用して、静的ライブラリーまたは共有ライブラリーをメインプログラムにリンクすることができます。例を以下に示します。

```
xlf -o myprogram main.f -Ldirectory1:directory2 [-Rdirectory] -ltest
```

コンパイル時、**-L** オプションで指定された最初のディレクトリー内で `libtest.so` を検索するようにリンカーに指示します。`libtest.so` が見つからない場合、リンカーは `libtest.a` を検索します。いずれのファイルも見つからない場合、**-L** オプションで指定された次のディレクトリー内で検索が続行されます。

実行時、ランタイム・リンカーは **-R** オプションで指定された最初のディレクトリー内で `libtest.so` を検索します。`libtest.so` が見つからない場合、**-R** オプションで指定された次のディレクトリー内で検索が続行されます。**-R** オプションで指定されたパスは、実行時に LD_LIBRARY_PATH 環境変数によってオーバーライドできます。

その他のリンケージ・オプション (デフォルトの動作を変更するオプションを含む) については、オペレーティング・システムの **ld** に関する資料 を参照してください。

「*XL Fortran* コンパイラー・リファレンス」内の関連情報

-l

-L

共有ライブラリー間のリンク: モジュールをアプリケーションにリンクする場合と同様に、共有ライブラリー同士をリンクすることでそれらの間に依存関係を作成できます。例を以下に示します。

```
xlf -qmkshrobj -o mylib.so myfile.o -Ldirectory -Rdirectory -lfoo
```

「*XL Fortran* コンパイラー・リファレンス」内の関連情報

-qmkshrobj

-L

XL Fortran SMP プログラムのコンパイル

xlf_r、 **xlf90_r**、 **xlf95_r**、 **xlf2003_r**、 **xlf2008_r**、または CUDA Fortran **xlcuf** CUDA Fortran コマンドを使用して、XL Fortran SMP プログラムをコンパイルできます。 **xlf_r** コマンドは、**xlf** コマンドと同様です。 **xlf90_r** コマンドは、**xlf90** コマンドと同様です。 **xlf95_r** コマンドは、**xlf95** コマンドと同様です。 **xlf2003_r** コマンドは、**xlf2003** コマンドと同様です。 **xlf2008_r** コマンドは、**xlf2008** コマンドと同様です。 CUDA Fortran **xlcuf** を指定することと、**xlf2008_r** および **-qcuda** オプションを指定することは同じです。 CUDA Fortran 主な違いは、コマンド **xlf_r**、**xlf90_r**、**xlf95_r**、**xlf2003_r**、または **xlf2008_r** を指定した場合、マルチスレッド方式用コンポーネントがオブジェクト・ファイルのリンクおよびバインドに使用される点です。

これらのコマンドの 1 つを単独で使用すると、並列処理が行われないことに注意してください。 SMP ディレクティブを認識して並列化を活動化するコンパイラーの場合は、**-qsmp** も指定する必要があります。逆に、**-qsmp** オプションは、これらの呼び出しコマンドの 1 つとともに指定することのみ可能です。**-qsmp** を指定すると、ドライバーは構成ファイルのアクティブ・スタンザにある **smplibraries** 行で指定されたライブラリーにリンクします。

POSIX pthreads API サポート: XL Fortran は、IEEE 1003.1-2001 (POSIX) 標準 pthreads API を使用するスレッド・プログラミングをサポートします。

標準インターフェース・ライブラリーを指定してプログラムをコンパイルおよびリンクするには、**xlf_r**、**xlf90_r**、**xlf95_r**、**xlf2003_r**、**xlf2008_r**、または CUDA Fortran **xlcuf** CUDA Fortran コマンドを使用します。例えば、次のように指定します。

```
xlf95_r test.f
```

特定アーキテクチャーのためのコンパイル方法

-qarch および **-qtune** を使用して、特殊なアーキテクチャーにコードの生成と調整を行うようにコンパイラーに指示することができます。これにより、コンパイラー

は、マシン特定の命令を活用してパフォーマンスを向上させることができます。**-qarch** オプションは、コンパイル後のプログラムが実行できるアーキテクチャーを判別します。オプション **-qtune** と **-qcache** は、プラットフォーム固有の最適化の程度を改善します。

デフォルト時には、**-qarch** を設定すると、サポートされているすべてのアーキテクチャーに共通の命令のみを使用するコードが作成され、結果として **-qtune** と **-qcache** の設定値は、これに伴って一般的なものとなります。特定のプロセッサー・セットまたはアーキテクチャーのパフォーマンスを調整するために、これらのオプションの 1 つ以上に別の設定値を指定する必要がある場合もあります。通常の試行過程では、まず **-qarch** を使用して、次に **-qtune** を追加し、次に **-qcache** を追加します。 **-qarch** のデフォルト値は **-qtune** や **-qcache** のデフォルト値にも影響するため、 **-qarch** オプション以外は必要でない場合がしばしばあります。

コンパイル中のマシンがターゲット・アーキテクチャーでもある場合は、**-qarch=auto** によって、コンパイル中のマシンの設定値が自動的に検出されます。このコンパイラー・オプションの設定値の詳細については、 113 ページの『-qarch』を参照してください。 -O オプションの **-O4** と **-O5** も参照してください。

コンパイル中のマシンがターゲット・アーキテクチャーでもある場合は、**-qtune=auto** によって、コンパイル中のマシンの設定値が自動的に検出されます。このコンパイラー・オプションの設定値の詳細については、 292 ページの『-qtune』を参照してください。 -O オプションの **-O4** と **-O5** も参照してください。

プログラムのほとんどを、特定のアーキテクチャーで実行するようにしている場合は、これらのオプションのうちの 1 つ以上を構成ファイルに追加しておけば、それをすべてのコンパイルのデフォルトにすることができます。

Fortran プログラムのコンパイル順序

モジュールを使用するプログラム・ユニット、サブプログラム、またはインターフェース・ボディがある場合、先にモジュールをコンパイルする必要があります。モジュール、およびモジュールを使用するコードが別個のファイルに入っている場合、モジュールが入っているファイルを最初にコンパイルする必要があります。同じファイルに入っている場合は、モジュールは、ファイル内のモジュールを使用するコードの前になければなりません。モジュールにあるエンティティーを変更する場合、そのモジュールを使用するファイルをすべて再コンパイルする必要があります。

F2008 個別モジュール・プロシージャーの実装のみが変更されていても、インターフェースが同じままであれば、対応するモジュール・プロシージャー・インターフェース本体が宣言されているモジュールが含まれるファイルを再コンパイルする必要はありません。 F2008

コンパイルの取り消し

コンパイルの完了前にコンパイラーを停止するには、対話モードで **Ctrl+C** を入力するか、 **kill** コマンドを使用してください。

C プリプロセッサーによるコンパイル

C プリプロセッサーによる Fortran ファイルの引き渡し

一般的なプログラミングの慣例では、 C プリプロセッサー (**cpp**) によってファイルを引き渡します。 **cpp** は、ユーザーが指定した条件に基づいて出力ファイルに行を組み込んだり、出力ファイルから行を削除したり ("条件付きコンパイル") できます。また、ストリングを置換 ("マクロ展開") することも可能です。

XL Fortran は **cpp** を使用して、コンパイル前にファイルを前処理することができます。

特定ファイルについて **cpp** を呼び出すには、ファイル・サフィックス **.F**、**.F77**、**.F90**、**.F95**、**.F03**、 **.F08**、または CUDA Fortran **.CUF** CUDA Fortran を使用してください。このようなファイルは、中間ファイルに前処理されます。中間ファイルは、**-d** コンパイラー・オプションを指定することによって保管することができます。このオプションを指定しないと、ファイルは削除されます。 **-d** オプションを指定すると、中間ファイル名は **F***filename***.f*** または CUDA Fortran **F***filename***.cuf** CUDA Fortran になります。このオプションを指定しないと、中間ファイルの名前は */tmpdir/F8xxxxxx* になります。ここで、*x* は英数字です。*tmpdir* は、**TMPDIR** 環境変数に入れられている値であり、**TMPDIR** に値が指定されていない場合 **/tmp** になります。前処理は行いたいが、オブジェクト・ファイルや実行可能ファイルは作成したくない場合は、 **-qnoobject** オプションも指定してください。

XL Fortran がファイルに **cpp** を使用するとき、プリプロセッサーは **#line** ディレクティブを出力します。これは、**-d** オプションを指定しないかぎり行われます。**#line** ディレクティブは、**cpp** かそれ以外の Fortran ソース・コード・ジェネレーターにより作成されたコードと、作成した入力コードを関連づけます。プリプロセッサーによって、コードの行が挿入されたり削除されたりする場合があります。コードの行を出力する **#line** ディレクティブは、オリジナルのソースで使用された行番号をリストして、前処理されたコードに検出されるソース・ステートメントを識別するため、エラーの報告書作成およびデバッグの際に役に立ちます。

_OPENMP C プリプロセッサー・マクロを使用すれば、コードを条件付きで組み込めます。このマクロは、 **-qsmp=omp** コンパイラー・オプションが指定してあれば、 C プリプロセッサーが呼び出されるときに定義されます。このマクロの例を以下に示します。

```
      program par_mat_mul
        implicit none
        integer(kind=8)                ::i,j,nthreads
        integer(kind=8),parameter      ::N=60
        integer(kind=8),dimension(N,N) ::Ai,Bi,Ci
        integer(kind=8)                ::Sumi
#ifdef _OPENMP
         integer omp_get_num_threads
#endif

         common/data/ Ai,Bi,Ci
!$OMP   threadprivate (/data/)

!$omp   parallel
         forall(i=1:N,j=1:N) Ai(i,j) = (i-N/2)**2+(j+N/2)
         forall(i=1:N,j=1:N) Bi(i,j) = 3-((i/2)+(j-N/2)**2)
!$omp   master
#ifdef _OPENMP
```

```
         nthreads=omp_get_num_threads()
#else
        nthreads=8
#endif
!$omp   end master
!$omp   end parallel

!$OMP parallel default(private),copyin(Ai,Bi),shared(nthreads)
!$omp do
      do i=1,nthreads
         call imat_mul(Sumi)
      enddo
!$omp end do
!$omp end parallel

     end
```

条件付きコンパイルについて詳しくは、「*XL Fortran* ランゲージ・リファレンス」の『言語エレメント』セクションの 『条件付きコンパイル』 を参照してください。

cpp 前処理をカスタマイズできるようにするため、構成ファイルは属性 **cpp**、**cppsuffix**、および **cppoptions** を受け入れます。

文字 **F** は、オプション **-t** および **-W** を持つ C プリプロセッサーを表します。

関連情報:
- 87 ページの『-d』
- 318 ページの『-t』
- 323 ページの『-W、-X』
- 156 ページの『-qfpp』
- 240 ページの『-qppsuborigarg』
- 12 ページの『カスタム・コンパイラー構成ファイルの使用』

C プリプロセッサーへのオプションの引き渡し

コンパイラーは **-I** 以外の **cpp** オプションをコマンド行上で直接認識しないので、このようなオプションは、**-W** オプションまたは **-X** オプションを使用して渡す必要があります。例えば、**LNXV1** という名前のシンボルの有無をテストする **#ifdef** ディレクティブがプログラムに含まれている場合は、以下のどちらかのコマンドでコンパイルすることにより、このシンボルを **cpp** に定義することができます。

```
xlf95 conditional.F -WF,-DLNXV1
xlf95 conditional.F -Xpreprocessor -DLNXV1
```

XL Fortran プログラムに対する cpp ディレクティブ

マクロ展開は、予期しない結果 (例えば、**FORMAT** 文の変更や、固定ソース形式で 72 文字よりも長い行の作成など) を招いてデバッグが困難になる場合があるため、**cpp** は主に Fortran プログラムの条件付きコンパイルに使用することをお勧めします。条件付きコンパイルに最も頻繁に使用される **cpp** ディレクティブは、**#if**、**#ifdef**、**#ifndef**、**#elif**、**#else**、**#endif** です。

プリプロセッシングの問題の回避

Fortran と C では、一部の文字列の処理が異なるため、**/*** や ***/** を使用する場合は注意して使用してください。 (これらは C のコメント区切り文字として解釈される

場合があり、Fortran コメントの内部で使用した場合でも問題が起こる可能性があります。) また、**??** で始まる 3 文字の文字列にも注意が必要です。 (これは C の 3 文字表記と解釈される可能性があります。)

次の例を考慮します。

```
program testcase
character a
character*4 word
a = '?'
word(1:2) = '??'
print *, word(1:2)
end program testcase
```

プリプロセッサーが、ご使用の文字の組み合わせとそれに対応した 3 文字表記を突き合わせると、出力が予期したものとならない場合があります。

XL Fortran コンパイラー・オプション **-qnoescape** をコードで使用する必要がない場合は、解決策として、文字ストリングをエスケープ・シーケンス **word(1:2) = '¥?¥?'** に置き換えることが考えられます。しかし、**-qnoescape** コンパイラー・オプションを使用している場合は、この解決策は役に立ちません。その場合は、3 文字表記を無視する **cpp** が必要です。 XL Fortran は、コンパイラーの一部として出荷される **cpp** を使用します。これは **ISO C** に準拠しているため、3 文字表記シーケンスを認識します。

バイナリー・ファイル内部の情報の表示 (strings)

strings コマンドは、以下のようにいくつかのバイナリー・ファイルにエンコードされている情報を読み取ります。

- コンパイラー・バージョンに関する情報は、コンパイラーのバイナリー形式の実行可能ファイルおよびライブラリーの中にエンコードされています。
- 親モジュール、ビット・モード、**.mod** ファイルを作成したコンパイラー、**.mod** ファイルが作成された日時、およびソース・ファイルに関する情報は、各 **.mod** ファイル内にエンコードされています。

例えば、/opt/ibm/xlf/16.1.0/exe/xlfentry に組み込まれている情報を表示するには、次のコマンドを実行します。

```
strings /opt/ibm/xlf/16.1.0/exe/xlfentry | grep "@(#)"
```

XL Fortran プログラムのリンク

デフォルト時には、XL Fortran プログラムのリンクで特別に行うべきことは何もありません。コンパイラー呼び出しコマンドは、自動的にリンカーを呼び出し、実行可能出力ファイルを作成します。例えば、以下のコマンドを実行すると、オブジェクト・ファイル file1.o および file3.o がコンパイルされて作成されます。

```
xlf95 file1.f file2.o file3.f
```

次にすべてのオブジェクト・ファイルがリンカーへサブミットされ、1 つの実行可能ファイルが作成されます。

リンクが終了したら、 44 ページの『XL Fortran プログラムの実行』の指示に従ってプログラムを実行してください。

ライブラリーをリンクするには、 36 ページの『ライブラリーのコンパイルとリンク』の指示に従ってください。

注**:** デフォルト以外のリンカーを使用する場合は、以下のオプションのいずれかを使用できます。

- **-qpath** を使用して非デフォルト・リンカーを指定します。例えば、以下のように指定します。

  `-qpath=l:`*`linker_path`*

- コンパイラーの構成ファイルを、非デフォルト・リンカーを使用するようにカスタマイズします。構成ファイルのカスタマイズ方法について詳しくは、『カスタム・コンパイラー構成ファイルの使用』および『カスタム構成ファイルの作成』を参照してください。

別個のステップでのコンパイルおよびリンク

後でリンクできるオブジェクト・ファイルを作成するには、**-c** オプションを使用します。

```
xlf95 -c file1.f               # Produce one object file (file1.o)
xlf95 -c file2.f file3.f       # Or multiple object files (file1.o, file3.o)
xlf95 file1.o file2.o file3.o  # Link object files with appropriate libraries
```

コンパイラー呼び出しコマンドでリンカーを実行するのが最善な場合もあります。このコマンドは、余分な **ld** オプションおよびライブラリー名をリンカーに自動的に渡すからです。

ld コマンドへのオプションの引き渡し

ld コマンドへのオプションの引き渡しについて詳しくは、『コマンド行オプションの「ld」または「as」コマンドへの引き渡し』を参照してください。

動的および静的リンク

XL Fortran を使用すれば、ご使用のプログラムは、動的リンクと静的リンクの両方のためのオペレーティング・システム機能の利点を利用できるようになります。

- 動的リンクとは、プログラムが初めて実行されたときに、外部ルーチン用のコードが探し出されてロードされることです。共用ライブラリーを使用するプログラムをコンパイルすると、デフォルトではプログラムに動的にリンクされます。

  動的にリンクされたプログラムでは、共用ライブラリーのルーチンを複数のプログラムが使用していても、ディスク・スペースも仮想メモリーも少なくて済みます。リンク中には、ライブラリー・ルーチンまたは外部データ・オブジェクトとの命名の競合の可能性が少なくなります。これは、エクスポートされたシンボルのみが共有ライブラリーの外部で可視であるためです。 いくつかのプログラムが同時に同じ共有ルーチンを使用する場合は、静的にリンクされたプログラムよりも良好に動作する場合があります。 また、動的リンクを使用すれば、再リンクしないで共用ライブラリー内のルーチンをアップグレードすることができます。

  このリンク形式はデフォルトなので、これをオンにするのに追加のオプションは必要ありません。

- 静的リンクとは、プログラムによって呼び出されるすべてのルーチン用のコードが実行可能ファイルの一部になることを意味します。

  静的にリンクされたプログラムは、XL Fortran ライブラリーがないシステムに移動してそのシステム上で実行することができます。静的にリンクされたプログラムが、ライブラリー・ルーチンへの呼び出しを多数行ったり、多数の小さなルーチンを呼び出す場合、それらのプログラムは動的にリンクされたプログラムよりも良好に動作する場合があります。ライブラリー・ルーチンまたは外部データ・オブジェクトとの命名の競合の可能性が多くなります。これは、すべてのグローバル・シンボルが静的ライブラリーの外側で可視であるためです。また、それらのプログラムをあるシステム上でコンパイルした後、別のレベルのオペレーティング・システムを使用したシステム上で実行すると、機能しない場合があります。

  静的にリンクを行うには、**-qstaticlink** オプションをリンカー・コマンドに追加します。例を以下に示します。

  ```
  xlf95 -qstaticlink test.f
  ```

リンク中の命名競合の回避

実行時またはシステム・ライブラリー・ルーチンと同じ名前を持つ外部サブルーチン、外部関数、共通ブロックを定義すると、その名前の定義がその場所で使用されたり、リンク・エディット・エラーが発生する場合があります。

以下の一般的な解決方法を試行して、このような種類の名前の矛盾を回避するための参考にしてください。

- **-qextname** オプションを使用して、すべてのファイルをコンパイルできます。このオプションは、個々のグローバル・エンティティーの名前の終わりに下線を追加して、この名前とシステム・ライブラリー内の名前とを区別します。

  注**:** このオプションを使用する場合は、**dtime_** および **flush_** のようなサービスおよびユーティリティー・サブプログラムの名前では、最後の下線を使用する必要はありません。
- プログラムを動的にリンクすることができます。これがデフォルトです。

-qextname オプションを使用しない場合は、XL Fortran およびシステム・ライブラリー内の外部シンボルの名前との競合を回避するために、特別な予防措置をとる必要があります。

- サブルーチンまたは関数を **main** と命名しないでください。 XL Fortran が、プログラムの始動に入り口点 **main** を定義するからです。
- 下線で始まるグローバル名を一切 使用しないでください。特に、XL Fortran ライブラリーでは、**_xl** で始まるすべての名前が予約されています。
- XL Fortran ライブラリー、または、いずれかのシステム・ライブラリー内の名前と同じ名前を使用しないでください。プログラム内で安全に使用できない名前を判別するには、プログラム内にリンクされているすべてのライブラリー上で **nm** コマンドを実行して、プログラム内にも存在する可能性のある名前を出力から探すことができます。

プログラムに実際のルーチンを定義せずに、サブルーチン名または関数名を使用することがないように注意してください。その名前がいずれかのライブラリーの名前

と競合すると、プログラムはルーチンの間違ったバージョンを使用して、コンパイル時エラーまたはリンク時エラーを作成しない場合があります。

XL Fortran プログラムの実行

実行可能プログラムのデフォルトのファイル名は **a.out** です。 **-o** コンパイラー・オプションを指定して別の名前を選択することができます。誤ったコマンドをうっかり実行することがないように、システム・コマンドまたはシェル・コマンド (例えば、 **test** または **cp**) と同じ名前をプログラムに付けないようにする必要があります。名前の矛盾が発生した場合は、 **./test** などのパス名を指定することにより、プログラムを実行することができます。

実行可能ファイルのパス名とファイル名、実行時の引数をコマンド行に入力すれば、プログラムを実行できます。

実行の取り消し

プログラムの実行を中断するには、プログラムがフォアグラウンドにある間に **Ctrl+Z** キーを押してください。実行を再開するには、**fg** コマンドを使用してください。

プログラムの実行を中断するには、プログラムがフォアグラウンドにある間に **Ctrl+C** キーを押してください。

別のシステム上でのコンパイルと実行

XL Fortran 実行可能ファイルを別のシステム(同一または互換性のあるオペレーティング・システムを実行している)に移動して実行したい場合は、静的にプログラム (および任意で実行時メッセージ・カタログ) をリンクおよびコピーすることができます。また、プログラム (および、必要な場合は XL Fortran ライブラリーと任意で実行時メッセージ・カタログ) を動的にリンクしてコピーすることもできます。SMP 以外のプログラムの場合、**libxlf90.so**、**libxlfmath.so**、および **libxlomp_ser.so** は、通常、XL Fortran ライブラリーがなくてはなりません。SMP プログラムの場合、通常は少なくとも **libxlf90.so**、**libxlfmath.so**、および **libxlsmp.so** ライブラリーが必要です。 **libxlfpmt*.so** はそのプログラムが **-qautodbl** オプションでコンパイルされる場合のみ必要です。

動的にリンクしたプログラムが正しく動作するためには、実行システム上の XL Fortran ライブラリーおよびオペレーティング・システム・レベルがコンパイル・システム上のレベルと同じか、またはそれより新しいレベルでなければなりません。

静的にリンクしたプログラムが正しく動作するためには、実行システム上のオペレーティング・システム・レベルがコンパイル・システム上のレベルと同じでなければならない場合があります。

関連情報: 42 ページの『動的および静的リンク』を参照してください。

POSIX pthreads がサポートするランタイム・ライブラリー

POSIX のスレッド・サポートを使用して接続されたランタイム・ライブラリーが 2 種類あります。 **libxlf90_r.so** ライブラリーは、スレッド・セーフ版の Fortran ランタイム・ライブラリーです。**libxlsmp.so** ライブラリーは、SMP ランタイム・ライブラリーです。

呼び出しコマンド、またある場合は、コンパイラー・オプションによって、スレッドをサポートするのに適切なライブラリーのセットがバインドされています。例を以下に示します。

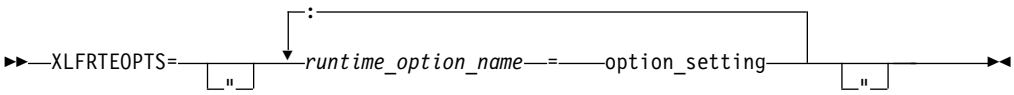
コマンド	使用されるライブラリー	インクルード・ディレクトリー
xlf90_r **xlf95_r** **xlf_r**	/opt/ibm/xlf/16.1.0/lib/libxlf90_r.so /opt/ibm/xlf/16.1.0/lib/libxlsmp.so	/opt/ibm/xlf/16.1.0/include

実行時オプションの設定

XL Fortran プログラム内の内部スイッチは、コンパイラー・オプションがコンパイル時の動作を制御する方法と似た方法で、実行時の動作を制御します。実行時オプションは、プログラム内の環境変数またはプロシージャー呼び出しによって設定することができます。 環境変数 **XLFRTEOPTS** および *XLSMPOPTS* を使用して、XL Fortran 実行時オプションの設定値を指定できます。

XLFRTEOPTS 環境変数

XLFRTEOPTS 環境変数を使用すると、ユーザーは、I/O、EOF エラー処理、乱数生成プログラムの指定などの項目の実行時の動作に影響を与えるオプションを指定することができます。XLFRTEOPTS は、次の **bash** コマンド形式を使用して宣言します。



オプション名と設定は、英大文字または小文字のどちらでも指定することができます。コロンおよび等号の前後にブランクを追加して、読みやすくすることができます。しかし、XLFRTEOPTS オプション・ストリングに組み込みブランクが含まれている場合は、オプション・ストリング全体を二重引用符 (**"**) で囲む必要があります。

プログラムが次のいずれかの状況を初めて検出したときに、環境変数がチェックされます。

- I/O 文が実行された。
- **RANDOM_SEED** プロシージャーが実行された。
- **ALLOCATE** 文がランタイム・エラー・メッセージを出す必要がある。
- **DEALLOCATE** 文がランタイム・エラー・メッセージを出す必要がある。
- **MATMUL** プロシージャーのマルチスレッド・インプリメンテーションが実行される。

プログラムの実行中に XLFRTEOPTS 環境変数を変更しても、プログラムには影響はありません。

SETRTEOPTS プロシージャー (「*XL Fortran* ランゲージ・リファレンス」で定義されています) は、環境変数 XLFRTEOPTS と同じ名前値のペアを含んでいる単一ストリング引数を受け入れます。これは環境変数をオーバーライドし、プログラムの実行中に設定を変更したいときに使用することができます。 **SETRTEOPTS** への別の呼び出しによって変更されない限り、プログラムの残りの部分には、新たな設定が引き続き有効です。プロシージャー呼び出しで指定された設定だけが変更されます。

次の実行時オプションは、環境変数 XLFRTEOPTS またはプロシージャー **SETRTEOPTS** で指定することができます。

aggressive_array_io={yes | no}
配列入出力操作を行うのに、より遅いアルゴリズムを適用するか、より速いアルゴリズムを適用するかを決める際に、XL Fortranランタイムで記述子情報を利用するかどうかを制御します。配列または配列セクションを連続として指定する記述子情報は、より速いアルゴリズムを適用するのに使用できます。配列または配列セクションが連続でないと、その情報はアンセーフになります。デフォルトでは、アグレッシブ配列入出力操作を実行します。

現行の XL Fortran ランタイム下で実行されたが、古い XL Fortran コンパイラーでコンパイルされたコードは、古いコンパイラーが XL Fortran 記述子情報を正しく設定していなかった場合、アグレッシブ配列入出力操作をアンセーフにしてしまいます。これは、もうサービスを受けていない古い XL Fortran コンパイラーでビルドされたコード、あるいは、最新のサービス・レベルではない XL Fortran コンパイラーでビルドされたコードに起こりやすい問題です。古いコードは、できるだけ、このオプションを使用せずに、現行のコンパイラーで再コンパイルしてください。

buffering={enable | disable_preconn | disable_all}
XL Fortran のランタイム・ライブラリーが、入出力操作で使用するバッファリングを実行するかどうかを判別します。

ライブラリーは、チャンクにあるファイル・システムからのデータの読み取りや、それに対するデータの書き込みを、少しずつ行うのではなく、**READ** 文や **WRITE** 文が来るたびに一括して行います。バッファリングを実行する主な利点は、パフォーマンスを向上させることができるということです。

Fortran のルーチンが他の言語のルーチンと一緒に作業するアプリケーションや、Fortran のプロセスが同じデータ・ファイル上の他のプロセスと一緒に作業するアプリケーションがある場合、Fortran ルーチンによって書かれたデータは、バッファリングが実行されるため、他のパーティーによってすぐには認識されない場合があります (その逆も言えます)。また、Fortran の **READ** 文は、I/O バッファーに必要以上のデータを読み込む場合があり、結果として次のデータ項目を読み取るはずの、他の言語で書かれたルーチンや他のプロセスによって実行される入力操作が失敗する可能性があります。このような場合、**buffering** 実行時オプションを使用して、XL Fortran のランタイム・ライブラリーのバッファリングを使用不能にすることができます。そうすれば、**READ**

文はファイルから必要とするデータを正確に読み取ることができ、 **WRITE** 文によるデータの書き込みも、文の完了時にファイル・システムへフラッシュされます。

注: I/O バッファリングは、順次アクセス装置 (パイプ、端末、ソケット など) 上のファイルでは常に使用可能です。 **buffering** オプションを設定しても、このようなタイプのファイルに影響を及ぼすことはありません。

論理装置で I/O バッファリングを使用不可にした場合は、**FLUSH** 文または Fortran のサービス・ルーチン **flush_** を使用してその論理装置用の I/O バッファーの内容をフラッシュする必要はありません。

buffering のサブオプションは、以下のとおりです。

enable
Fortran ランタイム・ライブラリーは、接続されている各論理装置ごとに入出力バッファーを保持します。ランタイム・ライブラリーが保持する現行の読み取り/書き込みファイル・ポインターは、ファイル・システムにある対応するファイルの読み取り/書き込みポインターとの同期を取らない場合があります。

disable_preconn
Fortran ランタイム・ライブラリーは、事前に接続されている各論理装置 (0、5、および 6) ごとに入出力バッファーを保持しません。ただし、接続されている他の論理装置の入出力バッファーはすべて保持します。ランタイム・ライブラリーが事前接続された装置用に保持する現行の読み取り/書き込みファイル・ポインターは、ファイル・システムにある対応するファイルの読み取り/書き込みポインターと同じです。

disable_all
Fortran ランタイム・ライブラリーは、どの論理装置にも入出力バッファーを保持しません。非同期 I/O を実行する Fortran プログラムでは、**buffering=disable_all** オプションを指定しないでください。

以下の例では、Fortran ルーチンと C ルーチンが、リダイレクトする標準入力からデータ・ファイルを読み取ります。最初に、メインの Fortran プログラムが整数を 1 つ読み取ります。それから、C ルーチンが整数を 1 つ読み取ります。最後に、メインの Fortran プログラムが別の整数を読み取ります。

Fortran のメインプログラム:

```
integer(4) p1,p2,p3
print *,'Reading p1 in Fortran...'
read(5,*) p1
call c_func(p2)
print *,'Reading p3 in Fortran...'
read(5,*) p3
print *,'p1 p2 p3 Read: ',p1,p2,p3
end
```

C のサブルーチン (c_func.c):

```
#include <stdio.h>
void
c_func(int *p2)
{
    int n1 = -1;

    printf("Reading p2 in C...¥n");
    setbuf(stdin, NULL);    /* Specifies no buffering for stdin */
```

```
    fscanf(stdin,"%d", &n1);
    *p2=n1;
    fflush(stdout);
}
```

入力データ・ファイル (infile):

```
11111
22222
33333
44444
```

メインプログラムは、リダイレクトする標準入力として infile を使用して実行します。次のようにします。

```
$ main < infile
```

buffering=disable_preconn をオンにすると、結果は次のようになります。

```
Reading p1 in Fortran...
Reading p2 in C...
Reading p3 in Fortran...
p1 p2 p3 Read:  11111 22222 33333
```

buffering=enable をオンにすると、結果は予想不能です。

buffer_size=*size*

装置のブロック・サイズを使用せずに、入出力バッファー・サイズをバイト単位で指定します。*size* は -1、または 4096 以上の整数値でなければなりません。デフォルトの -1 の場合、ファイルがある装置のブロック・サイズが使用されます。

このオプションを使用すると、装置のブロック・サイズが大きく、アプリケーションが同時に数多くのファイルをオープンするためにアプリケーションのメモリー不足が生じた場合、入出力バッファーで使用するメモリー量を減らすことができます。

この実行時オプションを使用する際には次のことに注意してください。

- 事前接続された装置は、このオプションの影響を受けません。事前接続された装置のバッファー・サイズは、バッファーが入っている装置のブロック・サイズと同じです。例外として、ブロック・サイズが 64 KB を超える場合、バッファー・サイズは 64 KB に設定されます。
- この実行時オプションは、磁気テープ装置または論理ボリューム上のファイルには適用されません。
- **SETRTEOPTS** プロシージャーでバッファー・サイズを指定すると、XLFRTEOPTS 環境変数または **SETRTEOPTS** プロシージャーで設定された値はオーバーライドされます。このオプションを再設定しても、オープン済みの装置は影響を受けません。

cnverr={yes | no}

この実行時オプションが **no** に設定されていると、プログラムは変換エラーを検出する I/O 文の **IOSTAT=** および **ERR=** 指定子に従いません。その代わり、デフォルトの回復処置を実行します (**err_recovery** の設定とは無関係です)。さらに、警告メッセージを出すこともあります (**xrf_messages** が設定されているかどうかによって決まります)。

関連情報: 変換エラーについて詳しくは、「*XL Fortran* ランゲージ・リファレンス」の 『データ転送ステートメント』を参照してください。**IOSTAT** 値について詳しくは、「*XL Fortran* ランゲージ・リファレンス」の『条件および *IOSTAT* 値』を参照してください。

cpu_time_type={usertime | systime | alltime | total_usertime | total_systime | total_alltime}

CPU_TIME(TIME) の呼び出しによって戻される時間の尺度を決定します。

cpu_time_type のサブオプションは、以下のとおりです。

usertime
プロセスのユーザー時間を戻します。

systime
プロセスのシステム時間を戻します。

alltime
プロセスのユーザーおよびシステム時間の合計を戻します。

total_usertime
プロセスのユーザー時間の合計を戻します。ユーザー時間の合計とは、プロセスのユーザー時間と、その子プロセス (ある場合) のユーザー時間の合計です。

total_systime
プロセスのシステム時間の合計を戻します。システム時間の合計とは、現行プロセスのシステム時間と、その子プロセス (ある場合) のシステム時間の合計です。

total_alltime
プロセスのユーザー時間とシステム時間の合計を戻します。ユーザー時間とシステム時間の合計とは、現行プロセスのユーザーおよびシステム時間と、その子プロセス (ある場合) のユーザーおよびシステム時間の合計です。

default_recl={64 | 32}

RECL= 指定子なしでオープンされた順次ファイル用のデフォルトのレコード・サイズを決定することができます。サブオプションは以下のとおりです。

64 デフォルトのレコード・サイズとして 64 ビット値を使用します。

32 デフォルトのレコード・サイズとして 32 ビット値を使用します。

32 ビット・プログラムを 64 ビット・モードに移植するときは、**default_recl** を使用してください。64 ビット・レコード長は指定された整変数に適合しません。以下を見てください。

```
INTEGER(4) I
OPEN (11)
INQUIRE (11, RECL=i)
```

default_recl=64 のとき、上記のコード・サンプルで実行時エラーが発生します。これは、デフォルト・レコード長 2**63-1 が 4 バイト整数 I に適合しないためです。**default_recl=32** を指定すると、I に適合するデフォルト・レコード・サイズ 2**31-1 が保証されます。

RECL= 指定子について詳しくは、「*XL Fortran* ランゲージ・リファレンス」の『**OPEN**』ステートメントを参照してください。

errloc={yes | no}
ランタイム・エラー状態が I/O、**ALLOCATE** または **DEALLOCATE** 文で起きた場合に、エラー・メッセージとともにファイル名および行番号を表示するかどうかを制御します。デフォルトでは、行番号およびファイル名が、ランタイム・エラー・メッセージの前に付加され表示されます。 **errloc=no** が指定されると、ランタイム・エラー・メッセージは、ソース・ロケーション情報なしで表示されます。

errloc 実行時オプションは、**SETRTEOPTS** プロシージャーでも指定することができます。

erroreof={yes | no}
ファイルの終わり条件が検出されたときに **END=** 指定子が存在しない場合は、**ERR=** 指定子によって指定されたラベルが分岐するかどうかを判別します。

err_recovery={yes | no}
この実行時オプションが **no** に設定されている場合、指定子 **IOSTAT=** または **ERR=** を持たない I/O 文の実行中に回復可能エラーが存在すると、プログラムが停止します。デフォルト時には、これらの文のいずれかが回復可能エラーを検出すると、プログラムは回復処置を行って作業を続行します。 **cnverr** を **yes** に設定し、**err_recovery** を **no** に設定すると、変換エラーが発生して、プログラムが停止する場合があります。

errthrdnum={yes | no}
errthrdnum=yes が有効な場合、XL Fortran は、**omp_get_thread_num** ルーチンによって指定された実行中スレッドのスレッド番号を、すべてのエラー・メッセージに付加します。単一スレッドのプログラムの場合、スレッド番号は 0 です。

errloc=yes を指定すると、ファイル名と行番号の前にスレッド番号が表示されます。**IOMSG=** 指定子が I/O 文に存在する場合、スレッド番号はエラー・メッセージの前に付けられ、メッセージの他の部分では標準エラーで表示されたのと同じフォーマットが使用されます。

errtrace={yes | no}
ランタイム・エラー状態が I/O、**ALLOCATE** または **DEALLOCATE** 文で起きた場合に、エラー・メッセージとともにトレースバックを表示するかどうかを制御します。**errtrace=no** の指定は、ランタイム・エラー・メッセージがトレースバックなしで表示されることを意味します。

より詳細な情報をトレースバックに表示するには、**-qlinedebug** オプションまたは **-g** オプションを指定してコンパイルしてください。

以下のいずれかの条件に該当する場合、トレースバックは表示されません。

- I/O 文で **IOSTAT=**、**ERR=**、**END=**、または **EOR=** 指定子を使用する場合。
- **ALLOCATE** または **DEALLOCATE** 文で **STAT=** 指定子を使用する場合。

例えば、以下のサンプル・コードでは、あるエラー処理コードに分岐するために **ERR=** 指定子が使用されます。エラーの原因である I/O 文で **ERR=** が指定されているため、トレースバックは生成されません。**errtrace=yes** が指定されていても、出力は `Open error.` です。

```
program open_error
open(unit=11, file='doesnotexist', status='old', err=200) ! no traceback
close(11)
200 print *, 'Open error.'
end
```

iostat_end={extended | 2003std}

ファイルの終わりおよびレコードの終わり条件が発生したときは、**IOSTAT** 値を、XL Fortran 定義または Fortran 2003 標準に基づいて設定します。サブオプションは以下のとおりです。

extended

IOSTAT 変数を、XL Fortran の値と条件の定義に基づいて設定します。

2003std

IOSTAT 変数を、Fortran 2003 の値と条件の定義に基づいて設定します。

例えば、**iostat_end=2003std** 実行時オプションを設定すると、ファイルの終わり条件に対して戻される拡張子と異なる **IOSTAT** 値となります。

```
export XLFRTEOPTS=iostat_end=2003std
character(10) ifl
integer(4) aa(3), ios
ifl = "12344321  "
read(ifl, '(3i4)', iostat=ios) aa ! end-of-file condition occurs and
                                  ! ios is set to -1 instead of -2.
```

IOSTAT 値の設定と使用について詳しくは、「*XL Fortran* ランゲージ・リファレンス」の『**READ**』、『**WRITE**』、および『条件および *IOSTAT* 値』セクションを参照してください。

intrinthds={num_threads}

MATMUL および **RANDOM_NUMBER** 組み込みプロシージャーの並列実行のスレッド数を指定します。 **MATMUL** 組み込みの使用時の **num_threads** のデフォルト値は、オンラインのプロセッサー数と同じです。
RANDOM_NUMBER 組み込みの使用時の **num_threads** のデフォルト値は、オンラインのプロセッサー数*2 に等しくなります。

MATMUL および **RANDOM_NUMBER** 組み込みプロシージャーで使用可能なスレッド数を変更すると、パフォーマンスに影響を及ぼす可能性があります。

langlvl={ | 90std | 95std | 2003std | 2008std | extended}

Fortran の標準および標準の拡張機能をサポートするレベルを判別します。サブオプションの値は、以下のようになります。

90std Fortran 90 標準の I/O 文および形式のすべての拡張機能にコンパイラーがエラーのフラグを付けるよう指示します。

95std Fortran 95 標準の I/O 文および形式のすべての拡張機能にコンパイラーがエラーのフラグを付けるよう指示します。

2003std

Fortran 2003 標準の I/O 文および形式のすべての拡張機能にコンパイラーがエラーのフラグを付けるよう指示します。

例えば、**langlvl=2003std** 実行時オプションを設定すると、ランタイム・エラー・メッセージが出されます。

```
integer(4) aa(100)
call setrteopts("langlvl=2003std")
    ...          ! Write to a unit without explicitly
    ...          ! connecting the unit to a file.
write(10, *) aa  ! The implicit connection to a file does not
    ...          ! comform with Fortran 2003 behavior.
```

2008std

Fortran 2003 標準が指定するすべての標準 I/O ステートメントと形式、および XL Fortran がサポートする Fortran 2008 形式をコンパイラーが受け入れるよう指示します。それ以外は、エラーとしてフラグが付けられます。

extended

Fortran 95 言語標準、Fortran 2003 機能、XL Fortran がサポートする Fortran 2008 機能、および拡張機能をコンパイラーが受け入れるよう指示し、言語レベルのチェックが実際上オフになるようにします。

Fortran 95 標準の一部であり、XL Fortran で使用できる項目 (名前リストのコメントなど) のサポートを取得するには、以下のサブオプションのいずれかを指定する必要があります。

- **95std**
- **2003std**
- **2008std**
- **extended**

以下の例には、Fortran 95 拡張機能 (*file* 指定子が **OPEN** 文で脱落している) が含まれています。

```
program test1

call setrteopts("langlvl=95std")
open(unit=1,access="sequential",form="formatted")

10 format(I3)

write(1,fmt=10) 123

end
```

langlvl=95std を指定すると、ランタイム・エラー・メッセージが作成されます。

以下の例には、Fortran 90 には含まれていない Fortran 95 の機能 (名前リストのコメント) が含まれています。

```
program test2

INTEGER I
LOGICAL G
NAMELIST /TODAY/G, I

call setrteopts("langlvl=95std:namelist=new")
```

```
open(unit=2,file="today.new",form="formatted", &
    & access="sequential", status="old")

read(2,nml=today)
close(2)

end

today.new:

&TODAY  ! This is a comment
I = 123, G=.true. /
```

langlvl=95std を指定すると、ランタイム・エラー・メッセージは作成されません。しかし、**langlvl=90std** を指定すると、ランタイム・エラー・メッセージが作成されます。

err_recovery 設定は、発生したエラーが回復可能なエラーであるか、それとも重大なエラーであるかを判別します。

multconn={yes | no}
複数の論理ユニットで同時に同じファイルにアクセスできるようにします。このオプションを使用すると、ファイルのコピーを作成せずにファイル内の同じ複数の位置を同時に読み取ることができます。

同じプログラム内の多重接続が許可されるのは、ディスク・ドライブなどのランダム・アクセス・デバイス上にあるファイルの場合だけです。次のような場合は、同じプログラム内の多重接続は許可されていません。

- 書き込み専用で接続されているファイル (**ACTION='WRITE'**)
- 非同期 I/O
- 順次アクセス装置 (パイプ、端末、ソケットなど) 上のファイル

ファイルに損傷を与えないようにするために、以下の点に注意してください。

- 同じファイルに対する 2 度目の **OPEN** 文および後続する **OPEN** 文が許可されるのは読み取りの場合だけです。
- もともと入力と出力の両用でファイルがオープン (**ACTION='READWRITE'**) された場合、最初の **OPEN** 文でファイルと接続された装置は次の装置の接続時に読み取り専用 (**ACCESS='READ'**) になります。ファイルに接続されているすべての装置をクローズし、それから最初の装置を再オープンしてその装置に対する書き込みアクセスを復元します。
- 2 つのファイルが同じデバイスと i ノード番号を共用している場合、その 2 つは同じファイルと見なされます。 したがって、リンクされたファイルは同じファイルと見なされます。

multconnio={tty | nulldev | combined | no }
デバイスで複数の論理装置に接続できるようにします。それにより、同じ装置に接続されている複数の論理装置に書き込んだり、その論理装置から読み取ったりすることができます。サブオプションは以下のとおりです。

combined
ヌル装置と TTY 装置の組み合わせを複数の論理装置に接続できるようにします。

nulldev
ヌル装置を複数の論理装置に接続できるようにします。

tty
TTY デバイスで複数の論理装置に接続できるようにします。

注記: このオプションを使用すると、予測不能な結果が生じる場合があります。

これでプログラムにおいて、**UNIT** パラメーターとは値が異なっていても、**FILE** パラメーターとは同じ値を含む **OPEN** 文を複数指定することができます。例えば、TTY デバイス **/dev/tty** にリンクされている **mytty** というシンボリック・リンクがある場合は、**multconnio=tty** オプションを指定する際に、以下のプログラムを実行することができます。

```
PROGRAM iotest
OPEN(UNIT=3, FILE='mytty', ACTION="WRITE")
OPEN(UNIT=7, FILE='mytty', ACTION="WRITE")
END PROGRAM iotest
```

Fortran は、装置 0、5、および 6 を TTY デバイスに事前に接続します。通常は、**OPEN** 文を使用して、装置 0、5、および 6 に接続された TTY デバイスに、追加の装置を接続することはできませんが、**multconnio=tty** オプションを指定すれば、それが可能です。例えば、装置 0、5、および 6 が TTY デバイス **/dev/tty** に事前に接続されている場合、**multconnio=tty** オプションを指定すれば、以下のプログラムを実行することができます。

```
PROGRAM iotest
   ! /dev/pts/2 is your current tty, as reported by the 'tty' command.
   ! (This changes every time you login.)
   CALL SETRTEOPTS ('multconnio=tty')
   OPEN (UNIT=3, FILE='/dev/pts/2')
   WRITE (3, *) 'hello'  ! Display 'hello' on your screen
END PROGRAM
```

namelist={new | old}
プログラムが入出力に XL Fortran の新しい **NAMELIST** 形式を使用するか、または古い () **NAMELIST** 形式を使用するかを判別します。Fortran 90 および Fortran 95 標準では、この新しい形式が要求されています。

注: **NAMELIST** 出力を含む既存のデータ・ファイルを読み取るには、古い設定が必要になる場合があります。ただし、新しいデータ・ファイルの書き込みには、標準に準拠している新しい形式を使用してください。

namelist=old では、**langlvl=90std**、**langlvl=95std**、または **langlvl=2003std** 設定はいずれも、非標準 **NAMELIST** 形式をエラーと見なしません。

関連情報: **NAMELIST** I/O について詳しくは、「*XL Fortran* ランゲージ・リファレンス」の『名前リストの形式設定』を参照してください。

naninfoutput={2003std | old | default}
IEEE 例外値の表示が Fortran 2003 標準に準拠しているか、古い XL Fortran 動作に戻すかを制御します。この実行時オプションでは、異なるコンパイル・コマンドで作成されたオブジェクト・ファイルが、古い動作または Fortran 2003 標準に基づいて、すべての IEEE 例外値を出力できるようにします。サブオプションは、次のとおりです。

default
例外値の出力は、プログラムのコンパイル方法によって異なります。

old
例外値の出力は、古い XL Fortran 動作に準拠します。

2003std
例外値の出力は、Fortran 2003 標準に準拠します。

nlwidth=*record_width*
デフォルト時には、**NAMELIST** 書き込み文は、書き込まれた **NAMELIST** 項目をすべて含むことができる長さの出力レコードを 1 つ作成します。出力レコード **NAMELIST** を指定の幅に制限するには、実行時オプション **nlwidth** を使用します。

注**:** このオプションは、順次ファイル用の **RECL=** 指定子を使用することによって、無効なオプションになります。プログラムは指定されたレコード長の範囲内に入るように **NAMELIST** 出力を合わせようとするからです。幅 **nlwidth** が宣言されているファイルのレコード長を超過しない限りは、依然として **nlwidth** を **RECL=** と組み合わせて使用することができます。

random={generator1 | generator2}
RANDOM_SEED が **GENERATOR** 引数を指定して呼び出されていない場合は、 **RANDOM_NUMBER** が使用する生成プログラムを指定します。 **generator1** (デフォルト) の値は **GENERATOR=1** に一致し、 **generator2** の値は **GENERATOR=2** に一致します。 **RANDOM_SEED** が **GENERATOR** 引数を指定して呼び出されている場合は、その時以降のプログラム内のランダム・オプションをオーバーライドします。 ランダム・オプションを変更するために、 **GENERATOR** オプションを指定して **RANDOM_SEED** を呼び出した後で SETRTEOPTS を呼び出しても、効果はありません。

scratch_vars={yes | no}
スクラッチ・ファイルに特定の名前を指定するには、実行時オプション **scratch_vars** を **yes** に設定し、環境変数 **XLFSCRATCH_***unit* を、指定した装置番号へ関連付けたいファイルの名前に設定します。 例については、「*XL Fortran* 最適化およびプログラミング・ガイド」の『スクラッチ・ファイルの命名』を参照してください。

ufmt_bigendian={*units_list***}**
ビッグ・エンディアン I/O が実行される不定形式データ・ファイルの装置番号を指定します。指定された不定形式ファイル内のビッグ・エンディアン形式データは、入出力操作中にオンザフライで、XL Fortran アプリケーションを実行中のマシンで使用されているリトル・エンディアン 形式との間で相互変換されます。

この実行時オプションは、内部ファイルを処理しません。内部ファイルは常にフォーマット済みです。指定された装置は、入出力の **UNFORMATTED** 形式の明示的または暗黙的な **OPEN** によって接続される必要があります。

このオプションの構文は以下のとおりです。

```
ufmt_bigendian=units_list
```

ここで、

```
units_list = units | units_list, units

units = unit | unit- | -unit | unit1-unit2 | newunit | *
```

装置番号は整数です。その値の範囲は 1 から 2 147 483 647 です。

unit 論理装置の数を指定します。

unit- 装置番号 *unit* から最大可能装置番号までの装置の範囲を指定します。

-unit 装置番号 1 から装置番号 *unit* までの装置の範囲を指定します。

unit1-unit2
装置番号 *unit1* から装置番号 *unit2* までの装置の範囲を指定します。

newunit
すべての **NEWUNIT** 値の範囲を指定します。 **NEWUNIT** 値に関して詳しくは、『NEWUNIT 値』を参照してください。

***** すべてのユニットを指定します。これは、1-,newunit と同等です。

注**:**

1. **CHARACTER** 型データのバイト・オーダーはスワップされません。
2. コンパイラーは、**REAL*4** または **REAL*8** 型の値の内部表記が IEEE 浮動小数点フォーマット準拠であることを前提としています。I/O は、異なる内部表記では正しく作動しません。
3. 異なるベンダー間では **REAL*16** 型の値の内部表記は不整合です。コンパイラーは、**REAL*16** 型の値の内部表記を XL Fortran の表記と同じように処理します。I/O は、異なる内部表記では正しく作動しません。
4. 派生型データの変換はサポートされていません。派生型の配置は、異なるベンダー間で不整合です。
5. 異なるベンダーからのインプリメンテーションの相違によって、Linux で実行中の XL Fortran アプリケーションと、ビッグ・エンディアン・システムで実行中の Fortran アプリケーション間で、ビッグ・エンディアン不定形式データを交換する際に問題が起こりやすくなります。XL Fortran は、ユーザーが自分のプログラムを XL Fortran に移植しやすくする多くのオプションを提供します。ビッグ・エンディアン・データ・ファイルの交換に問題がある場合は、その問題に取り組みやすくなるようにこれらのオプションを確認してください。
6. XL Fortran は、不定形式データ転送のバイト・オーダーを指定する複数の方式を提供します。**ufmt_bigendian** オプションの優先順位が最も高く、**CONVERT=** 指定子、および **-qufmt** オプションはその後になります。詳しくは、**OPEN** ステートメントの **CONVERT=** 指定子、および 294 ページの『-qufmt』 を参照してください。

unit_vars={yes | no}
暗黙的に接続されるファイル、または **FILE=** 指定子なしでオープンされるファイルに特定の名前を指定するには、実行時オプション **unit_vars=yes** を設定し、次に 1 つ以上の環境変数に、ファイル名を **XLFUNIT**_*unit* の形式で設定します。例については、「*XL Fortran* 最適化およびプログラミング・ガイド」の『明示的名前なしで接続されるファイルの命名』を参照してください。

uwidth={32 | 64}
不定形式順次ファイルのレコード長フィールドの幅を指定する場合、値をビット単位で指定します。不定形式順次ファイルのレコード長が (2**31 - 1) バイトから 8 バイトを引いた値 (データを囲むレコード終了文字を表す) より大きい場

合は、実行時オプションの **uwidth=64** を設定して、レコード長フィールドを 64 ビットに拡張する必要があります。このようにすると、レコード長は最高で (2**63 - 1) から 16 バイトを引いた値 (データを囲むレコード終了文字を表す) にすることができます。

xrf_messages={yes | no}
入出力操作、**RANDOM_SEED** 呼び出し、および **ALLOCATE** または **DEALLOCATE** 文の実行中に、プログラムがエラー状態に関する実行時メッセージを表示しないようにするには、実行時オプション **xrf_messages** を **no** に設定してください。**no** に設定しておかないと、変換エラーおよびその他の問題に関する実行時メッセージが、標準エラー・ストリームに送られます。

次の例は、実行時オプション **cnverr** を **yes** に設定し、 **xrf_messages** オプションを **no** に設定します。

```
# Basic format
  XLFRTEOPTS=cnverr=yes:xrf_messages=no
  export XLFRTEOPTS

# With imbedded blanks
  XLFRTEOPTS="xrf_messages = NO : cnverr = YES"
  export XLFRTEOPTS
```

SETRTEOPTS への呼び出しとして、上記の例は次のように記述できます。

```
  CALL setrteopts('xrf_messages=NO:cnverr=yes')
! Name is in lowercase in case -U (mixed) option is used.
```

OMP および SMP の実行時オプションの設定

XLSMPOPTS 環境変数を使用すると、ユーザーは SMP の実行に影響を与えるオプションを指定することができます。 OpenMP 環境変数、**OMP_DYNAMIC**、**OMP_NESTED**、**OMP_NUM_THREADS**、および **OMP_SCHEDULE**は、並列コードの実行を制御できます。これらの使用について詳しくは、「*XL Fortran* 最適化およびプログラミング・ガイド」のセクション『*XLSMPOPTS*』および『*OpenMP* 環境変数』を参照してください。

BLAS/ESSL 環境変数

デフォルトで、libxlopt ライブラリーは読者が XL Fortran でコンパイルするすべてのアプリケーションにリンクされます。ただし、サード・パーティーの基本線形代数サブプログラム (BLAS) ライブラリーを使用していたり、ESSL ルーチンを組み込むバイナリー ・ファイルを出荷したい場合、XL_BLAS_LIB 環境変数を使用してこれらを指定する必要があります。例えば、ご使用の BLAS ライブラリーが libblas と呼ばれる場合、環境変数を次のように設定します。

XL_BLAS_LIB=/usr/lib/libblas.a のエクスポート

コンパイラーが BLAS ルーチンへの呼び出しを生成するとき、libblas ライブラリーで定義されているサブルーチンが libxlopt で定義されているものの代わりに実行時に使用されます。

XLF_USR_CONFIG

XLF_USR_CONFIG 環境変数を使用して、コンパイラーが使用するカスタム構成ファイルのロケーションを指定します。ファイル名は、絶対パスとともに指定される必要があります。コンパイラーは、最初にこのファイルの定義を処理してから、デ

フォルトのシステム構成ファイルまたは **-F** オプションで指定されたカスタマイズ・ファイルの定義を処理します。詳細は、 12 ページの『カスタム・コンパイラー構成ファイルの使用』を参照してください。

実行時の動作に影響を与える他の環境変数

LD_LIBRARY_PATH、**LD_RUN_PATH**、および **TMPDIR** 環境変数は、 9 ページの『環境変数の正しい設定方法』で説明されているように、実行時に影響を与えます。これらの環境変数は、XL Fortran 実行時オプションではなく、**XLFRTEOPTS** と **XLSMPOPTS** のどちらにも設定することはできません。

XL Fortran 実行時例外

次のような操作を行うと、**SIGTRAP** シグナル形式で実行時例外が発生し、その結果として、通常『Trace/breakpoint トラップ』 メッセージが送出されます。

- コンパイル時に **-C** オプションを指定した後に、文字サブストリング式または配列添え字が限界を超えた。
- コンパイル時に **-C** オプションを指定した後に、文字ポインターとターゲットの長さが一致しなくなった。
- プログラム内の制御の流れが、プログラムのコンパイル時に重大度 **S** のセマンティクス・エラーが送出されるロケーションに達した。
- コンパイル時に **-qfloat=nanq** オプションを指定した後に、NaN 値を生成する浮動小数点演算および NaN 値のロードを行った。
- 固定小数点をゼロで割った。
- TRAP ハードウェア固有の組み込みプロシージャーを呼び出した。

次のような操作を行うと、**SIGFPE** シグナル形式で実行時例外が発生します。

- コンパイル時に適切な **-qflttrap** サブオプションを指定した場合は浮動小数点例外。

事前定義された XL Fortran 例外ハンドラーを例外が発生する前にインストールしておくと、例外が発生した後、診断メッセージおよびトレースバック (呼び出されて例外を発生する原因となった各ルーチン内でのオフセットを示す) が標準エラーに書き込まれます。ファイル・バッファーも、プログラムが終了される前にフラッシュされます。 **-g** オプションを指定してプログラムをコンパイルすると、トレースバックはアドレス・オフセットだけでなくソース行番号も表示します。

シンボリック・デバッガーを使用して、エラーを判別することができます。 **gdb** は、例外の原因を説明する特定のエラー・メッセージを提供します。

関連情報**:**

- 85 ページの『-C』
- 157 ページの『-qflttrap』
- 257 ページの『-qsigtrap』

また、「*XL Fortran* 最適化およびプログラミング・ガイド」の以下のトピックも参照してください。

- 上記の例外について詳しくは、『浮動小数点例外の検出とトラッピング』を参照してください。

- 例外ハンドラーのリストについては、『浮動小数点状況と制御レジスターの制御』を参照してください。

第 5 章 機能カテゴリー別コンパイラー・オプションの要約

Linux プラットフォームで使用可能な XL Fortran オプションは、以下のカテゴリーにグループ化されます。

- 『出力制御』
- 63 ページの『入力制御』
- 64 ページの『言語エレメント制御』
- 67 ページの『浮動小数点および整数制御』
- 68 ページの『オブジェクト・コード制御』
- 69 ページの『エラー・チェックおよびデバッグ』
- 72 ページの『リスト、メッセージ、およびコンパイラー情報』
- 74 ページの『最適化およびチューニング』
- 77 ページの『リンク』
- 78 ページの『移植性とマイグレーション』
- 79 ページの『コンパイラーのカスタマイズ』

オプションが等価の @PROCESS ディレクティブをサポートする場合、これは指示されます。リストされたオプションの詳細情報を得るには、そのオプションの説明の全ページを参照してください。

-q で始まるコンパイラー・オプション、サブオプション、そして **@PROCESS** ディレクティブを、大文字または小文字のいずれかで入力できます。ただし、**-qmixed** オプションを指定してある場合、 **-qextern** オプションに指定するプロシージャー名は、大文字と小文字を区別することに注意してください。

本書全般で、**-q** コンパイラー・オプションおよびサブオプションには小文字を使用し、**@PROCESS** ディレクティブには大文字を使用するという規則を使用しています。

使用するオプションの重要度を理解し、代わりに使用できるオプションがわかれば、プログラムを正しく効率的に機能させるために費やす時間と努力を節減することができます。

各コンパイラー・オプションについて詳しくは、 81 ページの『第 6 章 XL Fortran コンパイラー・オプションの詳細記述』を参照してください。

出力制御

このカテゴリーのオプションは、コンパイラーが作成するファイル出力のタイプ、および出力のロケーションを制御します。これらは、呼び出されるコンパイラー・コンポーネント、行われる (行われない) 前処理、コンパイル、リンクのステップ、生成される出力の種類を判別する基本オプションです。

表 7. コンパイラー出力オプション

オプション名	@PROCESS ディレクティブ	説明
85 ページの『-c』	なし。	コンパイラーに、ソース・ファイルをコンパイルまたはアセンブルすることのみを指示し、リンクすることは指示しない。このオプションでは、ソース・ファイルごとに .o ファイルが出力されます。
87 ページの『-d』	なし。	**cpp** によって生成されるプリプロセスされたソース・ファイルが、削除されるのではなく保持されるようにする。
99 ページの『-MMD』	なし。	**make** コマンドの記述ファイルの組み込みに適したターゲットを含む従属関係出力ファイルを生成する。**-MMD** は 210 ページの『-qmakedep』 の短い形式です。
97 ページの『-MF』	なし。	**-qmakedep** オプションまたは **-MMD** オプションによって生成される従属関係出力ファイルの名前または場所を指定する。
99 ページの『-MT』	なし。	**-qmakedep** オプションまたは **-MMD** オプションによって生成される従属関係出力ファイル内の **make** 規則のオブジェクト・ファイルのターゲット名を指定する。
104 ページの『-o』	なし。	出力オブジェクト、アセンブラー、または実行可能ファイルの名前を指定する。
210 ページの『-qmakedep』	なし。	**make** コマンドの記述ファイルの組み込みに適したターゲットを含む従属関係出力ファイルを生成する。**-qmakedep** は、 99 ページの『-MMD』 の長い形式です。
217 ページの『-qmkshrobj』	なし。	生成されたオブジェクト・ファイルから共有オブジェクトを作成する。
219 ページの『-qmoddir』	なし。	コンパイラーが書き込むモジュール (.mod) または F2008 サブモジュール F2008 (.smod) ファイルの場所を指定する。
291 ページの『-qtimestamps』	なし。	暗黙的なタイム・スタンプがオブジェクト・ファイルに挿入されるようにするかどうかを制御する。

表 7. コンパイラー出力オプション (続き)

オプション名	@PROCESS ディレクティブ	説明
317 ページの『-S』	なし。	ソース・ファイルごとにアセンブラー言語ファイルを生成する。

入力制御

このカテゴリーのオプションは、ソース・ファイルのタイプと場所を指定します。

表 8. コンパイラー入力オプション

オプション名	@PROCESS ディレクティブ	説明
94 ページの『-I』	なし。	ディレクトリーをインクルード・ファイル、.mod ファイル、および .smod ファイルの検索パスに追加する。
140 ページの『-qdlines』	DLINES	コンパイラーが、列 1 に D がある固定ソース形式の行をコンパイルするか、またはそれをコメントとして扱うかを指定する。
123 ページの『-qcclines』	CCLINES	コンパイラーが、固定ソース形式および F90 自由ソース形式の条件付きコンパイル行を認識するかどうかを判別する。このオプションは、IBM 自由ソース形式ではサポートされません。
127 ページの『-qci』	CI	処理する **INCLUDE** 行の識別番号 (1 から 255) を指定する。
129 ページの『-qcr』	なし。	コンパイラーが CR (復帰) 文字を解釈する方法を制御する。
138 ページの『-qdirective』	DIRECTIVE	コメント行をコンパイラーのコメント・ディレクティブとして識別する、トリガー定数と呼ばれる文字のシーケンスを指定する。
150 ページの『-qfixed』	FIXED	入力ソース・プログラムが固定ソース形式であることを示し、オプションで行の最大長を指定する。
156 ページの『-qfpp』	なし。	C プリプロセッサーにおける Fortran 固有のプリプロセッシングを制御する。 これは C プリプロセッサー・オプションであるため、**-WF** オプションを使用して指定する必要があります。

表 8. コンパイラー入力オプション (続き)

オプション名	@PROCESS ディレクティブ	説明
160 ページの『-qfree』	FREE	ソース・コードが自由ソース形式であることを示す。
216 ページの『-qmixed』	MIXED	コンパイラーが名前の大文字小文字を区別するようにする。
240 ページの『-qppsuborigarg』	なし。	さらにマクロ展開を行う前に、C プリプロセッサーにオリジナルのマクロ引数を置換するよう指示します。 これは C プリプロセッサー・オプションであるため、**-WF** オプションを使用して指定する必要があります。
244 ページの『-qpreprocess』	なし。	**cpp** を呼び出して、有効な Fortran ファイル接尾部 (.f や .f90 など) を持つファイルのプリプロセスを行います。
281 ページの『-qsuffix』	なし。	コマンド行でソース・ファイルの接尾部を指定する。
313 ページの『-qxlines』	XLINES	列 1 に X がある固定ソース形式の行をコンパイルするか、またはコメントとして扱うかを指定する。

言語エレメント制御

このカテゴリーのオプションによって、ソース・コードの特性を指定することができます。また、このオプションを使用すると言語制限を強制または緩和したり、言語拡張を使用可能にしたり使用不可にすることもできます。

表 9. 言語エレメント制御オプション

オプション名	@PROCESS ディレクティブ	説明
86 ページの『-D』	なし。	マクロ `#define` プリプロセッサー・ディレクティブ内と同様に定義する。
112 ページの『-qaltivec』	ALTIVEC	これは、ベクトル・レジスターにおけるベクトル・エレメントの順序を指定します。
131 ページの『-qcuda (CUDA Fortran)』	なし。	CUDA Fortran に対するコンパイラー・サポートを使用可能にする。

表 *9.* 言語エレメント制御オプション (続き)

オプション名	**@PROCESS** ディレクティブ	説明
179 ページの『-qinit』	INIT(F90PTR)	ポインターの初期関連状況を関連付け解除にする。
198 ページの『-qlanglvl』	LANGLVL	非適合の検査を行う言語標準 (または標準のスーパーセットまたはサブセット) を決定する。これは、非準拠のソース・コードと、そのような非準拠を許容するオプションを識別します。
216 ページの『-qmbcs』	MBCS	文字リテラル定数、ホレリス定数、H 編集記述子、および文字ストリング編集記述子にマルチバイト文字セット (MBCS) と Unicode 文字のどちらを含めることができるかをコンパイラーに指示する。
220 ページの『-qnullterm』	NULLTERM	仮引数として渡される各文字定数式に NULL 文字を付加して、ストリングを C 関数に渡しやすくする。
224 ページの『-qonetrip』, 83 ページの『-1』	ONETRIP	**DO** 文が実行される場合に、繰り返し回数が 0 であっても、コンパイルされたプログラム内の各 **DO** ループを少なくとも 1 回実行する。このオプションにより、FORTRAN 66 との互換性が実現します。
239 ページの『-qposition』	POSITION	**POSITION=** 指定子および対応する **STATUS=** 値 (**OLD** または **UNKNOWN**) を持たない **OPEN** ステートメントが指定された後、データを書き込む際にファイル・ポインターをファイルの末尾に配置する。
244 ページの『-qqcount』	QCOUNT	**Q** 文字カウント編集記述子 (**Q**)、および拡張精度 **Q** 編集記述子 (**Q***w.d*) を受け入れる。
251 ページの『-qsaa』	SAA	SAA FORTRAN 言語定義に準拠しているかどうかを検査する。これは、非準拠のソース・コードと、そのような非準拠を許容するオプションを識別します。

表 9. 言語エレメント制御オプション (続き)

オプション名	@PROCESS ディレクティブ	説明
251 ページの『-qsave』	SAVE	ローカル変数のデフォルト・ストレージ・クラスを指定する。
256 ページの『-qsclk』	なし。	**SYSTEM_CLOCK** 組み込みプロシージャーがプログラム内で使用する解決を指定する。
295 ページの『-qundef』, 320 ページの『-u』	UNDEF	変数名の暗黙型定義が許可されないことを指定する。 **-qundef** は 320 ページの『-u』オプションの長形式です。
303 ページの『-qxlf77』	XLF77	変更された言語セマンティクスと I/O データ形式の FORTRAN 77 の特質との互換性を確保する。
306 ページの『-qxlf90』	XLF90	Fortran 言語の特質に関する Fortran 90 標準との互換性を確保する。
308 ページの『-qxlf2003』	XLF2003	前の Fortran 標準に準拠するコンパイラー呼び出しによってコンパイルする際に、Fortran 2003 標準固有の言語機能を使用できるようにする機能、および Fortran 2003 標準に準拠するコンパイラー呼び出しによってコンパイルする際に、それらの機能を使用不可にする機能を提供する。
312 ページの『-qxlf2008』	XLF2008	前の Fortran 標準に準拠するコンパイラー呼び出しによってコンパイルする際に、Fortran 2008 標準固有の言語機能を使用できるようにする機能、および Fortran 2008 標準に準拠するコンパイラー呼び出しによってコンパイルする際に、それらの機能を使用不可にする機能を提供する。
316 ページの『-qzerosize』	ZEROSIZE	サイズがゼロの文字ストリングおよび配列を処理する可能性のあるプログラム内で、このようなオブジェクトの有無についての検査を行うかどうかを決定する。

表 9. 言語エレメント制御オプション (続き)

オプション名	**@PROCESS** ディレクティブ	説明
319 ページの『-U』	なし。	コンパイラーまたは **-D** コンパイラー・オプションによって定義されたプリプロセッサー・マクロ名の定義を解除する。

浮動小数点および整数制御

アプリケーションがどのように計算を行うかを詳細に指定すると、システムの浮動小数点のパフォーマンスや精度 (丸めの送信方法など) をより効率的に活用することができます。IEEE 浮動小数点の仕様に順守しすぎると、アプリケーションのパフォーマンスに影響を及ぼす可能性があるので注意してください。以下の表のオプションを使用して、浮動小数点パフォーマンスと IEEE 標準への順守間のトレードオフを制御することができます。これらのオプションのうち、整数計算の特定のアスペクトを制御できるものもあります。

表 10. 浮動小数点と整数の制御オプション

オプション名	**@PROCESS** ディレクティブ	説明
117 ページの『-qautodbl』	AUTODBL	単精度浮動小数点計算を倍精度へ変換する操作、および倍精度計算を拡張精度へ変換する操作が自動的に実行されるようにする。
141 ページの『-qdpc』	DPC	実定数を **DOUBLE PRECISION** 変数に割り当てるときに最大限の正確性を実現できるよう、実定数の精度を高める。
142 ページの『-qenum』	なし。	列挙子定数の範囲を指定し、ストレージ・サイズを決定できるようにする。
153 ページの『-qfloat』	FLOAT	浮動小数点の計算を高速化したり、精度を上げるためのさまざまなストラテジーを選択する。
173 ページの『-qieee』, 325 ページの『-y』	IEEE	コンパイル時に定数浮動小数点式を評価する場合にコンパイラーが使用する丸めモードを指定する。
188 ページの『-qintlog』	INTLOG	式およびステートメントで整数と論理データ・エンティティーを混用できることを指定する。

表 10. 浮動小数点と整数の制御オプション (続き)

オプション名	**@PROCESS** ディレクティブ	説明
190 ページの『-qintsize』	INTSIZE	長さまたは種類が指定されていないデフォルトの **INTEGER** および **LOGICAL** データ・エンティティーのサイズを設定する。
245 ページの『-qrealsize』	REALSIZE	**REAL**、**DOUBLE PRECISION**、**COMPLEX**、および **DOUBLE COMPLEX** の値のデフォルトのサイズを設定します。
279 ページの『-qstrictieeemod』	STRICTIEEEMOD	コンパイラーが、**ieee_arithmetic** および **ieee_exceptions** 組み込みモジュールに関する Fortran 2003 IEEE 算術計算規則に従うかどうかを指定する。

オブジェクト・コード制御

これらのオプションは、オブジェクト・コードの特性、前処理されたコード、またはコンパイラーが生成したその他の出力に影響を与えます。

表 11. オブジェクト・コード制御オプション

オプション名	**@PROCESS** ディレクティブ	説明
184 ページの『-qinlglue』	INLGLUE	**-O2** 以上の最適化で使用すると、アプリケーション内の外部関数呼び出しを最適化するグルー・コードをインライン化する。
235 ページの『-qpic』	なし。	共有ライブラリーでの使用に必須の位置独立コードを生成する。
253 ページの『-qsaveopt』	なし。	ソース・ファイルのコンパイルに使用するコマンド行オプション、構成ファイルで指定されたユーザーの構成ファイル名とオプション、コンパイル中に呼び出される各コンパイラー・コンポーネントのバージョンとレベル、およびその他の情報を対応するオブジェクト・ファイルに保管する。
285 ページの『-qtbtable』	なし。	オブジェクト・ファイルに組み込まれるデバッグ・トレースバック情報の量を制御する。

表 11. オブジェクト・コード制御オプション (続き)

オプション名	@PROCESS ディレクティブ	説明
GPU -qtgtarch	なし。	コードが実行される可能性がある、実 GPU アーキテクチャーまたは仮想 GPU アーキテクチャーを指定します。これにより、GPU アーキテクチャーに固有であるか、または仮想アーキテクチャーに共通である機能またはマシン命令を、コンパイラーが最大限に活用できるようになります。
290 ページの『-qthreaded』	なし。	スレッド・セーフ・コードを生成する必要があるかどうかをコンパイラーに指示する。

エラー・チェックおよびデバッグ

以下の表のオプションによって、ご使用のソース・コードの問題点を検出し、修正することができます。場合によっては、これらのオプションを使用することでオブジェクト・コードが変更されたり、コンパイル時間が長くなったり、アプリケーションの実行速度を落とすことがあるランタイム検査を導入することがあります。オプションの説明には、余分な検査がパフォーマンスにどのような影響を及ぼす可能性があるのかが示されています。

ご使用のアプリケーションの動作およびパフォーマンスに関して受信される情報の量および種類を制御するには、 72 ページの『リスト、メッセージ、およびコンパイラー情報』を参照してください。

最適化されたコードをデバッグする方法について詳しくは、「*XL Fortran* 最適化およびプログラミング・ガイド」を参照してください。

表 12. エラー・チェックおよびデバッグ・オプション

オプション名	@PROCESS ディレクティブ	説明
82 ページの『-#』	なし。	実際にコンパイラー・コンポーネントを呼び出さずに、コマンド行に指定されたコンパイルのステップをプレビューする。
85 ページの『-C』, 124 ページの『-qcheck』	CHECK	特定タイプのランタイム検査を実行するコードを生成する。
93 ページの『-gsplit-dwarf』	なし。	DWARF デバッグ情報を、1 つ以上の異なる .dwo ファイルに生成できるようにします。

表 12. エラー・チェックおよびデバッグ・オプション (続き)

オプション名	**@PROCESS** ディレクティブ	説明
132 ページの『-qcudaerr (CUDA Fortran)』	CUDAERR	コンパイラーが自動的に CUDA API 呼び出しのエラー・チェック・コードを挿入するかどうかを制御します。
90 ページの『-g』, 134 ページの『-qdbg』	DBG	シンボリック・デバッガーが使用するデバッグ情報を生成し、選択したソース・ロケーションでのデバッグ・セッションでプログラムの状態を使用できるようにします。
157 ページの『-qfltrap』	FLTTRAP	実行時に検出する浮動小数点例外条件のタイプを決定する。
161 ページの『-qfullpath』	なし。	このオプションは、**-g** オプションまたは **-qlinedebug** オプションと使用した場合、デバッグ情報付きでコンパイルされたオブジェクト・ファイルのソース・ファイルおよびインクルード・ファイルのフルパス名 (絶対パス名) を記録して、デバッグ・ツールが正確にソース・ファイルの場所を検索できるようにします。
162 ページの『-qfunctrace』	なし。	ご使用のプログラムでプロシージャーの入り口点および出口点をトレースする。ご使用のプログラムに C++ コンパイル単位が含まれていると、このオプションは C++ catch ブロックもトレースする。
164 ページの『-qfunctrace_xlf_catch』	なし。	catch トレース・サブルーチンの名前を指定します。
165 ページの『-qfunctrace_xlf_enter』	なし。	entry トレース・サブルーチンの名前を指定します。
166 ページの『-qfunctrace_xlf_exit』	なし。	exit トレース・サブルーチンの名前を指定します。
167 ページの『-qhalt』	HALT	コンパイル時のメッセージの最大重大度が指定した重大度と同じかそれを超える場合は、オブジェクト・ソース・ファイル、実行可能ソース・ファイル、またはアセンブラー・ソース・ファイルのいずれかを作成する前に、コンパイルを停止する。

表 12. エラー・チェックおよびデバッグ・オプション (続き)

オプション名	**@PROCESS** ディレクティブ	説明
168 ページの『-qhaltonmsg』	HALTONMSG	指定されたエラー・メッセージが生成された場合、オブジェクト・ファイル、実行可能ファイル、またはアセンブラー・ソース・ファイルを作成せずにコンパイルを停止する。
174 ページの『-qinfo』	なし。	通知メッセージのグループを作成または抑制する。
180 ページの『-qinitalloc』	INITALLOC	デバッグ目的で、割り振られているが特定の値に初期化されていない割り振り可能な変数およびポインター変数を初期化します。
182 ページの『-qinitauto』	なし。	デバッグのために、未初期化の自動変数を特定の値に初期化する。
198 ページの『-qkeepparm』	なし。	**-O2** 以上の最適化で使用した場合に、プロシージャー・パラメーターをスタックに保管するかどうかを指定します。
203 ページの『-qlinedebug』	なし。	デバッガー用に行番号およびソース・ファイル名の情報のみを生成する。
222 ページの『-qobject』	OBJECT	ソース・ファイルの構文検査の後に、オブジェクト・ファイルを作成するか、即時に停止するかを指定する。
257 ページの『-qsigtrap』	なし。	メインプログラムを含むファイルのコンパイル時に、**SIGTRAP** および **SIGFPE** 例外をキャッチするために指定されたトラップ・ハンドラーをセットアップする。
269 ページの『-qstackprotect』	なし。	スタックを上書きまたは破壊する悪意のある入力データやプログラミング・エラーから保護する。
301 ページの『-qwarn64』	なし。	32 ビットから 64 ビットへのマイグレーションで問題を起こす可能性のあるステートメントを識別する通知メッセージを表示する。
302 ページの『-qxflag=dvz』	なし。	コンパイラーが浮動小数点のゼロ除算演算を検出するためのコードを生成できるようにする。

リスト、メッセージ、およびコンパイラー情報

次の表にあるオプションを使用すれば、リスト・ファイルを制御したり、コンパイラー・メッセージを表示する方法とタイミングを制御したりできます。そのオプションを、 69 ページの『エラー・チェックおよびデバッグ』で説明されているオプションと一緒に使用すれば、エラーおよび予期しない振る舞いを検査するときに提供される、ご使用のアプリケーションに関する概要をより堅固なものにすることができます。

表 *13.* リスト・オプションとメッセージ・オプション

オプション名	**@PROCESS** ディレクティブ	説明
116 ページの『-qattr』	ATTR	リストの属性および相互参照セクションの属性コンポーネントを組み込むコンパイラー・リストを作成する。
151 ページの『-qflag』	FLAG	診断メッセージを指定した重大度レベルまたはそれより高い重大度レベルのものに制限する。
169 ページの『-qhelp』	なし。	コンパイラーの man ページを表示する。
204 ページの『-qlist』	LIST	これは、オブジェクトおよび定数域セクションが含まれるコンパイラー・リスト・ファイルを生成します。
205 ページの『-qlistfmt』	なし。	これは、最適化の機会を見いだすときに役立つレポートを XML 形式または HTML 形式で作成します。
208 ページの『-qlistopt』	なし。	コンパイラー呼び出し時に有効なすべてのオプションを含むコンパイラー・リスト・ファイルを作成する。
213 ページの『-qmaxerr』	MAXERR	指定した重大度レベル以上のエラー・メッセージの数が指定した数に到達すると、コンパイルを停止する。
234 ページの『-qphsinfo』	PHSINFO	各コンパイル・フェーズで標準出力にかかった時間を報告する。
219 ページの『-qnoprint』	なし。	他のリスト・オプションの設定に関係なく、コンパイラーがリスト・ファイルを作成しないようにする。

表 13. リスト・オプションとメッセージ・オプション (続き)

オプション名	**@PROCESS** ディレクティブ	説明
248 ページの『-qreport』	なし。	コードのセクションをどのように最適化したかを示すリスト・ファイルを作成する。
260 ページの『-qslmtags』	なし。	これは、SLM タグ・ロギングがコンパイラー・ライセンス使用状況をトラッキングするかどうかを制御します。
267 ページの『-qsource』	SOURCE	リストのソース・セクションを含むコンパイラー・リスト・ファイルを作成し、エラー・メッセージの印刷時にソース情報を追加して提供する。
282 ページの『-qsuppress』	なし。	特定の通知メッセージ、または警告メッセージが生成された場合、表示されたりリスト・ファイルに追加されるのを防ぐ。
298 ページの『-qversion』	なし。	呼び出しているコンパイラーのバージョンおよびリリースを表示する。
315 ページの『-qxref』	XREF	リストの属性および相互参照の相互参照コンポーネントを含むコンパイラー・リストを作成する。
322 ページの『-V』	なし。	コマンドを作成するためにディスプレイから直接カット・アンド・ペーストできる点を除き、**-v** と同じ。
321 ページの『-v』	なし。	呼び出されているプログラムと各プログラムに指定されているオプションの名前を戻すことによって、コンパイルの進行を報告する。
322 ページの『-w』	なし。	警告メッセージを抑制する (-qflag=e:e と同等)。

最適化およびチューニング

以下の表内のオプションを使用して、最適化および調整の処理を制御し、ランタイムでのアプリケーションのパフォーマンスを高めることができます。すべてのオプションがすべてのアプリケーションに役立つというわけではありません。コンパイル時での増加、デバッグ機能の縮小、最適化によって提供される向上の間でトレードオフが行われることがあります。このセクションのオプション記述に加えて、最適化および調整の処理、最適化による容易なソース・コードの書き込みについての詳細は、「*XL Fortran* 最適化およびプログラミング・ガイド」を参照してください。

67 ページの『浮動小数点および整数制御』に記載されているオプションの中にも、パフォーマンスを向上させるものがあります。ただし、これらのオプションを使用する際は、アプリケーションが必要とする浮動小数点セマンティクスが保存されるように注意してください。

表 *14.* 最適化およびチューニング・オプション

オプション名	**@PROCESS** ディレクティブ	説明
101 ページの『-O』	OPTIMIZE	コンパイル中にコードを最適化するかどうかを指定し、最適化する場合は、レベルを指定する。
106 ページの『-qalias』	ALIAS(*argument_list*)	これは、特定のカテゴリーの別名割り当てがプログラムに含まれているのか、またはプログラムが標準の Fortran 別名割り当て規則に準拠していないのかを示します。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。
113 ページの『-qarch』	なし。	コードを実行する可能性のある、プロセッサー・アーキテクチャーまたはアーキテクチャーのファミリーを指定します。これによって、コンパイラーはアーキテクチャーに固有か、またはアーキテクチャーのファミリーに共通のマシン命令を最大限に活用することができます。
114 ページの『-qassert』	ASSERT	コンパイラーの最適化の微調整に役立つコードの特性に関する情報を提供する。
121 ページの『-qcache』	なし。	特定の実行マシンに対して、キャッシュ構成を指定する。

表 *14.* 最適化およびチューニング・オプション (続き)

オプション名	**@PROCESS** ディレクティブ	説明
128 ページの『-qcompact』	COMPACT	コード・サイズを増やす最適化を回避する。
140 ページの『-qdirectstorage』	なし。	特定のコンパイル単位がライトスルーを使用可能にしたストレージまたはキャッシュ禁止ストレージを参照する可能性があることをコンパイラーに通知する。
145 ページの『-qessl』	なし。	コンパイラーが、Engineering and Scientific Subroutine Library (ESSL) ルーチンを Fortran 90 組み込みプロシージャーに代えて使用できるようにする。
149 ページの『-qfdpr』	なし。	オブジェクト・ファイルに IBM Feedback Directed Program Restructuring (FDPR®) パフォーマンス・チューニング・ユーティリティーが結果の実行可能ファイルを最適化するために必要とする情報を提供する。
170 ページの『-qhot』	HOT(サブオプション)	最適化中に上位ループ分析および変換 (HOT) を実行する。
-qinline	なし。	パフォーマンスを改善するために、プロシージャーへの呼び出しを生成する代わりに、それらのプロシージャーのインライン化を試行する。
192 ページの『-qipa』	なし。	プロシージャー間分析 (IPA) と呼ばれる最適化のクラスを使用可能にしたり、カスタマイズする。
201 ページの『-qlibansi』	なし。	ANSI C ライブラリー関数の名前のすべての関数が、実際にライブラリー関数であり、異なるセマンティクスを持つユーザー関数でないことを想定する。
-qlibmpi	なし。	Message Passing Interface (MPI) 名を持つすべての関数が事実上 MPI 関数であり、異なるセマンティクスを持つユーザー関数でないことを表明する。

表 14. 最適化およびチューニング・オプション (続き)

オプション名	@PROCESS ディレクティブ	説明
214 ページの『-qmaxmem』	MAXMEM	メモリーを消費する、指定したキロバイト数になるまでの特定の最適化の実行中に、コンパイラーによって割り振られるメモリーの量を制限する。
GPU 222 ページの『-qoffload』	なし。	OpenMP ターゲット領域の NVIDIA GPU へのオフロードのサポートを有効にする。
105 ページの『-p』	なし。	コンパイラーが作成するオブジェクト・ファイルをプロファイル用に準備する。
228 ページの『-qpdf1、-qpdf2』	なし。	*profile-directed feedback* (PDF) を介して最適化を調整する。サンプル・プログラムの実行から得られた結果を使用して、条件付き分岐の付近や頻繁に実行されるコード・セクションでの最適化を改善します。
242 ページの『-qprefetch』	なし。	コード・パフォーマンスを改善する機会がある場所に、プリフェッチ命令を自動的に挿入する。
256 ページの『-qshowpdf』	なし。	コンパイル・ステップとリンク・ステップで **-qpdf1** および最小最適化レベル **-O2** と共に使用すると、アプリケーション内のすべてのプロシージャーについて、追加プロファイル情報を含む PDF マップ・ファイルを作成する。
258 ページの『-qsimd』	なし。	ベクトル命令をサポートするプロセッサーでコンパイラーがベクトル命令を自動的に利用できるかどうかを制御する。
261 ページの『-qsmallstack』	なし。	可能な場合にスタック使用量を最小にする。
262 ページの『-qsmp』	なし。	プログラム・コードの並列化を使用可能にする。
270 ページの『-qstacktemp』	なし。	実行時に特定の XL Fortran コンパイラー一時ファイルを割り振る場所を決定する。

表 14. 最適化およびチューニング・オプション (続き)

オプション名	@PROCESS ディレクティブ	説明
274 ページの『-qstrict』	STRICT	これは、デフォルトでは最適化レベル **-O3** 以上で行われ、オプションでは **-O2** で行われる最適化によって、厳密な IEEE 浮動小数点準拠に大きく関連する特定のプログラムのセマンティクスが変更されないようにします。
280 ページの『-qstrict_induction』	なし。	コンパイラーが帰納 (ループ・カウンター) 変数の最適化を実行しないようにする。このような最適化は、帰納変数に関係する整数オーバーフロー操作が発生した場合に問題となる可能性があります。
292 ページの『-qtune』	なし。	特定のハードウェア・アーキテクチャーで最も効率よく実行できるように、命令選択、スケジューリング、およびその他のアーキテクチャー依存のパフォーマンスの強化をチューニングする。ターゲット SMT モードの指定で、そのモードで最高のパフォーマンスが得られる最適化を指示できるようにする。
295 ページの『-qunroll』	なし。	プログラム内で **DO** ループのアンロールが許可されるかどうかを指定する。アンロールは外部および内部の **DO** ループで許可されます。
297 ページの『-qunwind』	なし。	プロシージャー呼び出し中に、コンパイラーが揮発性レジスターへの保存および復元のデフォルトの振る舞いを保持することを指定する。
300 ページの『-qvisibility』	VISIBILITY(*suboption*)	オブジェクト・ファイル内の外部リンケージ・シンボルの可視性属性を指定します。

リンク

リンク作成は自動的に起こりますが、以下の表のオプションでは、入出力をリンカーに送信し、リンカーがどのようにオブジェクト・ファイルを処理するかを制御します。

コンパイラーは認識されないオプションをリンカーに渡すので、コンパイラー・コマンド行に実際に **ld** オプションを入れることができます。

表 15. リンク・オプション

オプション名	**@PROCESS** ディレクティブ	説明
87 ページの『-e』	なし。	**-qmkshrobj** オプションと共に使用する場合は、共用オブジェクトのエントリー・ポイントを指定します。
95 ページの『-L』	なし。	**-l** オプションによって指定されたライブラリー・ファイルのディレクトリー・パスをリンク時に検索する。
96 ページの『-l』	なし。	指定されたライブラリー・ファイルを検索します。リンカーは *libkey.so* を検索し、*libkey.so* が見つからない場合は *libkey.a* を検索します。
271 ページの『-qstaticlink』	なし。	静的または共有のランタイム・ライブラリーをアプリケーションにリンクするかどうかを制御する。

移植性とマイグレーション

このカテゴリーのオプションは、過去、現在、そして未来のハードウェア、オペレーティング・システム、およびコンパイラーでのアプリケーション動作互換性を保守し、最小の変更でご使用のアプリケーションを XL コンパイラーに移行するのに役立ちます。

表 16. 移植性とマイグレーション

オプション名	**@PROCESS** ディレクティブ	説明
109 ページの『-qalign』	ALIGN	ストレージ内でデータ・オブジェクトの位置合わせを指定する。これによって、誤って配置されたデータが原因で起こるパフォーマンス上の問題を回避できます。
119 ページの『-qbindcextname』	BINDCEXTNAME	**-qextname** オプションが **BIND(C)** エンティティーに影響するかどうか制御する。
130 ページの『-qctyplss』	CTYPLSS	型なし定数が使用される可能性がある場合に必ず、文字定数式を使用できるかどうかを指定する。
135 ページの『-qddim』	DDIM	ポインティング先配列の境界を、その配列が参照されるたびに再評価することを指定し、ポインティング先配列の境界式に対するいくつかの制限を除去する。

表 16. 移植性とマイグレーション (続き)

オプション名	@PROCESS ディレクティブ	説明
136 ページの『-qdescriptor』	なし。	コンパイルされたアプリケーションでオブジェクト指向でないエンティティー用に使用する XL Fortran 内部記述子データ構造フォーマットを指定する。
143 ページの『-qescape』	ESCAPE	文字ストリング、ホレリス定数、H 編集記述子、および文字ストリング編集記述子におけるバックスラッシュの取り扱い方法を指定する。
146 ページの『-qextern』	なし。	XL Fortran 組み込み関数の代わりにユーザー作成のプロシージャーを呼び出せるようにする。
147 ページの『-qextname』	EXTNAME	すべてのグローバル・エンティティーの名前に下線を追加する。
209 ページの『-qlog4』	LOG4	論理オペランドを使用する論理演算の結果が **LOGICAL(4)** であるか、または最大長のオペランドを使用する **LOGICAL** であるかを指定する。
236 ページの『-qport』	PORT	プログラムを XL Fortranに移植する際に、他の Fortran 言語の拡張機能に対応できるようにするためのオプションを提供する。
284 ページの『-qswapomp』	SWAPOMP	XL Fortran プログラムでコンパイラーが OpenMP ルーチンを認識し、置換しなければならないことを指定する。
294 ページの『-qufmt』	UFMT	不定形式データ・ファイルで入出力操作のバイト・オーダーを設定する。
303 ページの『-qxflag=oldtab』	XFLAG(OLDTAB)	列 1 から 5 のタブを 1 文字として解釈する (固定ソース形式プログラムの場合)。

コンパイラーのカスタマイズ

以下の表のオプションによって、コンパイラー・コンポーネント、構成ファイル、標準インクルード・ディレクトリー、および内部コンパイラー操作の代替ロケーションを指定することができます。これらのオプションは、特殊なインストール済み環境、テスト・シナリオ、および追加のコマンド行オプションの指定に役立ちます。

表 17. コンパイラー・カスタマイズ・オプション

オプション名	**@PROCESS** ディレクティブ	説明
83 ページの『-B』	なし。	アセンブラー、C プリプロセッサー、およびリンカーなどの XL Fortran コンポーネントの代替パス名を指定します。
88 ページの『-F』	なし。	代替構成ファイルを指定するか、構成ファイル内で使用するスタンザを指定するか、またはこれらを両方とも指定する。
101 ページの『-NS』, 268 ページの『-qspillsize』	SPILLSIZE	レジスター・スピル・スペース (溢れたレジスターをストレージに退避させるために最適化プログラムが使用する内部プログラム・ストレージ域) のサイズをバイト単位で指定する。
224 ページの『-qoptfile』	なし。	コンパイルで使用する追加コマンド行オプションのリストが入った応答ファイルを指定する。応答ファイルは一般的に .rsp 接尾部を持ちます。
227 ページの『-qpath』	なし。	これは、XL Fortran コンポーネント (アセンブラー、C プリプロセッサー、リンカーなど) の代替パス名を指定します。
318 ページの『-t』	なし。	**-B** オプションで指定したプレフィックスを、指定したコンポーネントに適用する。
323 ページの『-W、-X』	なし。	1 つ以上のオプションをコンパイル中に実行されるコンポーネントに渡す。

第 6 章 XL Fortran コンパイラー・オプションの詳細記述

このセクションには、XL Fortran で使用可能な個々のオプションについての説明が含まれます。

各オプションでは、以下の情報が提供されます。

カテゴリー
: オプションが属する機能カテゴリーはここに一覧表示されます。

@PROCESS
: 多くのコンパイラー・オプションでは、等価の @PROCESS ディレクティブを使用して、オプションのアプリケーションの範囲を単一のソース・ファイルまたはコンパイル単位、または選択したコードのセクションに制限して、ソース・コード内のオプションの機能を適用することができます。

目的
: このセクションでは、オプション (および等価のディレクティブ) の効果、およびその使用の利点を簡単に説明します。

構文
: このセクションでは、コマンド行オプションおよび等価の @PROCESS ディレクティブ (使用可能な場合) の構文を提供します。構文がまずコマンド行形式で示され、次に @PROCESS 形式で示されています。コマンド行構文を表すのに使用される表記の説明については、 viii ページの『規則』を参照してください。

  オプションの最小文字数を示すために大文字が使用されることがあります。例えば、**-qassert=CONTIGuous** における大文字の **CONTIG** は、このオプションに使用する必要がある最小文字数を示しています。したがって、**-qassert=contig** と **-qassert=contigu** のどちらを使用しても、コンパイラーは有効と認識します。

  @PROCESS 構文では、以下の表記が使用されます。

  - 各オプションのデフォルトは、下線が引かれた太文字で示されています。
  - 個々の必須引数は、特殊表記を付けずに記述されます。
  - 選択肢のセットから選択する必要がある場合、そのセットが { 記号と } 記号で囲まれています。
  - オプションの引数は [記号と] 記号で囲まれています。
  - 選択項目のグループから選択できる場合は、それぞれが | 文字で区切られています。
  - 繰り返せる引数の後には省略符号 (...) が続いています。

デフォルト
: 多くの場合、デフォルトのオプション設定は、構文図に明確に示されます。ただし、多くのオプションでは、効果のある他のコンパイラー・オプションに応じて、複数のデフォルト設定があります。このセクションでは、適用する場合のさまざまなデフォルト設定を示します。

パラメーター
このセクションでは、このオプションで使用可能なサブオプションについて説明します。

使用法
このセクションでは、留意が必要な規則または使用上の考慮事項について説明します。これらには、オプションの適用可能性、複数のオプション仕様の優先順位規則などの制限が含まれます。

例　必要に応じて、コマンド行構文の例とその使用例がこのセクションで提供されます。

-#

カテゴリー

エラー・チェックおよびデバッグ

@PROCESS

なし。

目的

実際にコンパイラー・コンポーネントを呼び出さずに、コマンド行に指定されたコンパイルのステップをプレビューする。

このオプションは、さまざまなコンパイル・ステップのためにコンパイラーがコマンドを実行するポイントで、これらのアクションを実際には実行しないで、実行するコマンドおよび渡すシステム引数リストのシミュレーションを表示します。

構文

オプション:

デフォルト

適用されません。

使用法

このオプションの出力を調べると、特定のコンパイルに関して、次の事項を迅速かつ確実に判別することができます。
- どのファイルが関係があるか
- 個々のステップで、どのオプションが有効になっているか

ソース・コードのコンパイルのオーバーヘッドを回避して、 **.lst** ファイルなどの既存のファイルの上書きも回避します。(ユーザーが **make** コマンドに精通している場合、これは **make -n** に似ています。)

このオプションでは、**-v** および **-V** と同じ出力を作成しますが、コンパイルは実行しません。

-qipa とともにこのオプションを指定する場合、コンパイラーは IPA リンク・ステップの後にリンカー情報を表示しないことにご注意ください。これは、コンパイラーが IPA を実際に呼び出していないためです。

関連情報

- 321 ページの『-v』
- 322 ページの『-V』

-1

カテゴリー

言語エレメント制御

目的

DO 文が実行される場合に、繰り返し回数が 0 であっても、コンパイルされたプログラム内の各 **DO** ループを少なくとも 1 回実行する。このオプションにより、FORTRAN 66 との互換性が実現します。

-qonetrip は、**-1** の長い形式です。

構文

オプション:

@PROCESS:

@PROCESS ONETRIP | **NOONETRIP**

デフォルト

デフォルトは、その後の Fortran 標準の動作に従うことで、この標準では反復回数が 0 の場合は **DO** ループは実行されません。

制約事項

このオプションは **FORALL** 文や **FORALL** 構文、あるいは配列コンストラクターによる **DO** ループには影響しません。

-B

カテゴリー

コンパイラーのカスタマイズ

@PROCESS

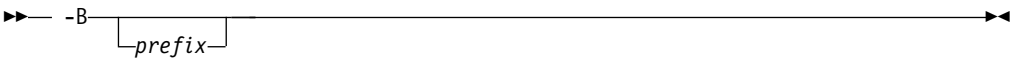
なし。

目的

アセンブラー、C プリプロセッサー、およびリンカーなどの XL Fortran コンポーネントの代替パス名を指定します。

これは **-t** オプションと組み合わせて使用することができ、これらのコンポーネントのどれが **-B** の影響を受けるかを決定します。しかし同じ機能として、代わりに **-qpath** オプションを使用することをお勧めします。

構文



デフォルト

コンパイラー・コンポーネントのデフォルトのパスは、コンパイラー構成ファイルで定義されています。

パラメーター

prefix
コンポーネントが存在するディレクトリーの名前です。これは **/** (斜線) で終わっていなければなりません。

使用法

個々のコンポーネントの完全なパス名を形成するために、ドライバー・プログラムは標準プログラム名に *prefix* を追加します。 1 つ以上の **-t***mnemonic* オプションを組み込むことによっても、このオプションの影響を受けるコンポーネントを制限することができます。

これらのコマンドのデフォルト・パス名を構成ファイルに指定することもできます。

このオプションを使用すると、コンポーネントの一部または全部の XL Fortran コンポーネントの複数レベルを保持したり、アップグレードされたコンポーネントを永続的にインストールする前に、試してみることができます。複数レベルの XL Fortran を使用可能にしておく場合は、適切な **-B** オプションおよび **-t** オプションを構成ファイルのスタンザに入れてから、**-F** オプションを使って、使用するスタンザを選択する必要があります。

関連情報

- 227 ページの『-qpath』
- 318 ページの『-t』
- 88 ページの『-F』
- 12 ページの『カスタム・コンパイラー構成ファイルの使用』
- 21 ページの『2 つのレベルの XL Fortran の実行』

-C

カテゴリー

エラー・チェックおよびデバッグ

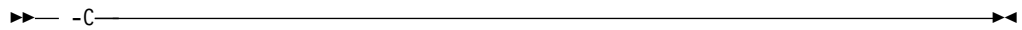
目的

特定タイプのランタイム検査を実行するコードを生成する。

-qcheck は、**-C** の長い形式です。

構文

オプション:



@PROCESS:

@PROCESS CHECK | **NOCHECK**

デフォルト

-qnocheck

-c

カテゴリー

オブジェクト・コードの制御

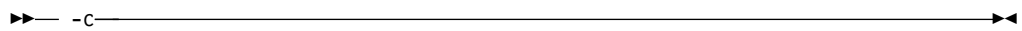
@PROCESS

なし。

目的

コンパイラーに、ソース・ファイルをコンパイルまたはアセンブルすることのみを指示し、リンクすることは指示しない。このオプションでは、ソース・ファイルごとに .o ファイルが出力されます。

構文



デフォルト

適用されません。

使用法

-c と組み合わせて **-o** オプションを使用すると、**.o** ファイルの代わりに別の名前が選択されます。この場合、一度にコンパイルできるソース・ファイルは 1 つだけです。

関連情報

- 104 ページの『-o』

-D

カテゴリー

言語エレメント制御

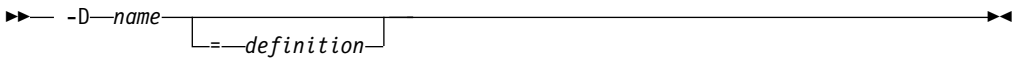
@PROCESS

なし。

目的

マクロ `#define` プリプロセッサー・ディレクティブ内と同様に定義する。

構文



デフォルト

適用されません。

パラメーター

name
定義するマクロです。-D*name* は `#define` *name* と同等です。例えば、-DCOUNT は `#define COUNT` と同等です。

definition
name に割り当てる値です。 -D*name=definition* は、 `#define` *name definition* と同等です。例えば、-DCOUNT=100 は `#define COUNT 100` と同等です。

使用法

`#define` ディレクティブを使用して、既に **-D** オプションによって定義されているマクロ名を定義すると、エラー状態となります。

-D オプションによって定義されるマクロの定義解除に使用される **-U***name* オプションは、**-D***name* オプションより高い優先順位を持ちます。

15.1.6 より前のバージョンの XL Fortran では、**-D** オプションは **-qdlines** オプションの短縮形でした。

例

myprogram.f で、COUNT という名前のインスタンスをすべて 100 に置換するには、以下のように入力します。

```
xlf myprogram.f -DCOUNT=100
```

関連情報

- 140 ページの『-qdlines』
- 244 ページの『-qpreprocess』
- 319 ページの『-U』

-d

カテゴリー

出力制御

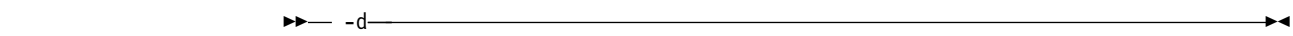
@PROCESS

なし。

目的

cpp によって生成されるプリプロセスされたソース・ファイルが、削除されるのではなく保持されるようにする。

構文



デフォルト

適用されません。

結果

このオプションによって生成されるファイルには、元のソース・ファイルの名前から派生した **F***filename*.**f*** という形式の名前が付きます。例えば、ソース・ファイル test.F03 は前処理されて Ftest.f03 というファイルに変換されます。

関連情報

- 39 ページの『C プリプロセッサーによる Fortran ファイルの引き渡し』

-e

カテゴリー

リンク

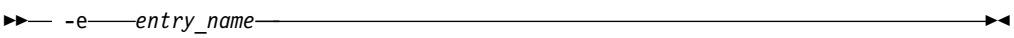
@PROCESS

なし。

目的

-qmkshrobj オプションと共に使用する場合は、共用オブジェクトのエントリー・ポイントを指定します。

構文



デフォルト

なし。

パラメーター

name
共有実行可能ファイルのエントリー・ポイントの名前です。

使用法

-e オプションは、**-qmkshrobj** オプションと組み合わせてのみ指定します。

注**:** オブジェクト・ファイルをリンクする場合は、**-e** オプションを使用しないでください。実行可能出力のデフォルト・エントリー・ポイントは、`__start` です。このラベルを **-e** フラグで変更すると、エラーが生成される可能性があります。

関連情報

- 217 ページの『-qmkshrobj』

-F

カテゴリー

コンパイラーのカスタマイズ

@PROCESS

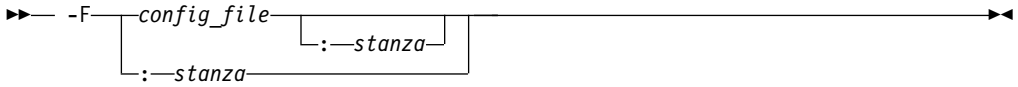
なし。

目的

代替構成ファイルを指定するか、構成ファイル内で使用するスタンザを指定するか、またはこれらを両方とも指定する。

構成ファイルは、さまざまな種類のデフォルト、例えば特定のコンパイル・ステップのためのオプション、コンパイラーが必要とするさまざまなファイルの位置を指定します。

構文



デフォルト

デフォルトでは、コンパイラーは、インストール時に構成される構成ファイル、および現在使用されている呼び出しコマンド用にそのファイル内で定義されたスタンザを使用します (例えば、**xlf2003**、**xlf90_r**、**xlf90**、など)。

パラメーター

config_file
使用する代替コンパイラー構成ファイルの絶対パス名。

stanza
コンパイルで使用する構成ファイル・スタンザの名前。これは、使用される呼び出しコマンドに関係なく、コンパイラーがその *stanza* でエントリーを使用するように指示します。例えば、**xlf2003** でコンパイルし、**xlf95** スタンザを指定した場合、コンパイラーは、**xlf95** スタンザに指定されたすべての設定を使用します。

使用法

複雑なコンパイル・スクリプトを書く代替として、コンパイラーの基本機能をカスタマイズできる簡単な方法は、/opt/ibm/xlf/16.1.0/etc/xlf.cfg に新しいスタンザを追加して、個々のスタンザに異なる名前と異なるセットのデフォルト・コンパイラー・オプションを指定することです。また、デフォルトの構成ファイルを編集するのではなく、XLF_USR_CONFIG 環境変数を使用してユーザー定義の構成ファイルを指定するという方法もあります。多数の分散したコンパイル・スクリプトや makefile よりも、一か所に集めた単一ファイルの方が維持しやすいことがわかります。

適切な **-F** オプションを指定してコンパイラーを実行することにより、使用するオプションのセットを選択することができます。完全な最適化のために 1 セットのオプションを持ち、完全なエラー・チェックなどのためにもう 1 セットのオプションを持つことなどができます。ユーザー定義の構成ファイルの設定が **-F** オプションが指定した設定より前に処理されるように注意してください。

制約事項

新しいコンパイラーのリリースがインストールされるたびにデフォルトの構成ファイルが置き換えられるので、新しいスタンザや新しいコンパイラー・オプションを確実に保管するようにしてください。

代わりに、カスタマイズされた設定を XLF_USR_CONFIG 環境変数で指定されたユーザー定義の構成ファイルに格納することができます。このファイルは、再インストール中に置き換えることはありません。

例

```
 # Use stanza debug in default xlf.cfg.
   xlf95 -F:debug t.f

# Use stanza xlf95 in /home/fred/xlf.cfg.
   xlf95 -F/home/fred/xlf.cfg t.f

# Use stanza myxlf in /home/fred/xlf.cfg.
   xlf95 -F/home/fred/xlf.cfg:myxlf t.f
```

関連情報

- 13 ページの『カスタム構成ファイルの作成』 では、カスタムのユーザー定義構成ファイルの内容と、**-F** オプションを使用せずにファイル内の別のスタンザを選択する方法が説明されています。
- 17 ページの『デフォルト構成ファイルの編集』 では、**-F** オプションを使用して構成ファイルの内容を編集する方法が説明されています。
- 83 ページの『-B』
- 318 ページの『-t』
- 323 ページの『-W、-X』

-g

カテゴリー

エラー・チェックおよびデバッグ

@PROCESS

DBG

目的

シンボリック・デバッガーが使用するデバッグ情報を生成し、選択したソース・ロケーションでのデバッグ・セッションでプログラムの状態を使用できるようにします。

多様な **-g** レベルを使用することで、デバッグ機能とコンパイラー最適化の間のバランスを取ることができます。H**-g** レベルを高くすると、デバッグのサポートが充実しますが、実行時または場合によってはコンパイル時のパフォーマンスが低下します。**-g** レベルを低くすると、実行時のパフォーマンスは向上しますが、デバッグ・セッションで一部の機能を使用できなくなります。

-O2 最適化レベルが有効な場合は、デバッグ機能が完全にサポートされます。

-O2 より高いレベルの最適化が有効な場合は、デバッグ機能が制限されます。

-g は **-qdbg** の短い形式です。

構文

@PROCESS:

@PROCESS DBG | **NODBG**

デフォルト

-g を指定しない場合は、**-g0(-qnodbg** に相当) が有効になります。これは、コンパイラーがデバッグ情報を生成しない (プログラム状態を保持しない) ことを意味します。

-g が指定されている場合、デフォルト値は以下のようになります。

- 最適化が無効な場合 (**-qnoopt**)、**-g** は **-g9** や **-qdbg=level=9** と同等です。
- **-O2** 最適化レベルが有効な場合、**-g** は **-g2** や **-qdbg=level=2** と同等です。

パラメーター

-g0 デバッグ情報を生成しません。プログラム状態は保存されません。

-g1 行番号およびソース・ファイル名に関する最小限の読み取り専用のデバッグ情報を生成します。プログラム状態は保存されません。このオプションは、**-qlinedebug** と同じです。

-g2 行番号、ソース・ファイル名、および変数に関する読み取り専用のデバッグ情報を生成します。

-O2 以上の最適化レベルが有効な場合、プログラム状態は保持されません。

-g3、**-g4**

行番号、ソース・ファイル名、および変数に関する読み取り専用のデバッグ情報を生成します。

-O2 最適化レベルが有効な場合は、以下のようになります。

- プログラム状態は保存されません。
- デバッガーでは、プロシージャーのパラメーター値を各プロシージャーの先頭で使用できます。

-g5、**-g6**、**-g7**

行番号、ソース・ファイル名、および変数に関する読み取り専用のデバッグ情報を生成します。

-O2 最適化レベルが有効な場合は、以下のようになります。

- デバッガーでは、**IF** 構文、ループ構文、プロシージャー定義、およびプロシージャー呼び出しでプログラム状態を使用できます。詳しくは、『使用法』を参照してください。
- デバッガーでは、プロシージャーのパラメーター値を各プロシージャーの先頭で使用できます。

-g8 行番号、ソース・ファイル名、および変数に関する読み取り専用のデバッグ情報を生成します。

-O2 最適化レベルが有効な場合は、以下のようになります。

- デバッガーでは、すべての実行可能ステートメントの先頭でプログラム状態を使用できます。
- デバッガーでは、プロシージャーのパラメーター値を各プロシージャーの先頭で使用できます。

-g9 行番号、ソース・ファイル名、および変数に関するデバッグ情報を生成します。変数の値をデバッガーで変更できます。

-O2 最適化レベルが有効な場合は、以下のようになります。

- デバッガーでは、すべての実行可能ステートメントの先頭でプログラム状態を使用できます。
- デバッガーでは、プロシージャーのパラメーター値を各プロシージャーの先頭で使用できます。

使用法

最適化が無効なときに **-g2** 以上のレベルを指定すると、常にデバッグ情報が使用可能になります。**-O2** 最適化レベルが有効なときに **-g5** 以上のレベルを指定すると、選択したソース・ロケーションでデバッグ情報が使用可能になります。

-O2 とともに **-g8** または **-g9** を指定した場合は、実行可能ステートメントが存在するすべてのソース行でデバッグ情報が使用可能になります。

-O2 とともに **-g5**、**-g6**、または **-g7** を指定した場合は、以下の言語構造体でデバッグ情報が使用可能になります。

- **IF** 構文

  すべての **IF** ステートメント (**IF** キーワードが指定されている行) の先頭でデバッグ情報が使用可能です。また、**IF** 構文の直後に存在する次の実行可能ステートメントの先頭でも使用可能です。
- ループ構成体

  すべての **DO** ステートメント (**DO** キーワードが指定されている行) の先頭でデバッグ情報が使用可能です。また、**DO** 構文の直後に存在する次の実行可能ステートメントの先頭でも使用可能です。
- プロシージャー定義

  プロシージャーの本体に存在する最初の実行可能ステートメントでデバッグ情報が使用可能です。
- プロシージャー呼び出し

ユーザー定義プロシージャーを呼び出す各ステートメントの先頭でデバッグ情報が使用可能です。また、プロシージャー呼び出しを含むステートメントの直後に存在する次の実行可能ステートメントの先頭でも使用可能です。

例

myprogram.f をコンパイルし、デバッグ用に testing という実行可能プログラムを生成するには、以下のコマンドを使用します。

```
xlf myprogram.f -o testing -g
```

以下のコマンドは、特定の **-g** レベルを **-O2** と併用して myprogram.f をコンパイルし、デバッグ情報を生成します。

```
xlf myprogram.f -O2 -g8
```

関連情報

- SNAPSHOT
- 349 ページの『Fortran プログラムのデバッグ』
- 7 ページの『シンボリック・デバッガーのサポート』
- 134 ページの『-qdbg』
- 203 ページの『-qlinedebug』
- 161 ページの『-qfullpath』
- 101 ページの『-O』
- 198 ページの『-qkeepparm』

-gsplit-dwarf

カテゴリー

エラー・チェックおよびデバッグ

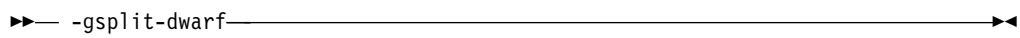
@PROCESS

なし。

目的

DWARF デバッグ情報を、1 つ以上の異なる .dwo ファイルに生成できるようにします。

構文



デフォルト

適用されません。

使用法

-gsplit-dwarf オプションを使用すると、リンク時間が短くなるよう、デバッグ情報が入っているファイルへのリンクを避けることができます。

注:

- GPU プログラムが **-qcuda** または **-qoffload** オプションを使用することによって NVIDIA GPU 装置をターゲットにする場合、**-gsplit-dwarf** はホスト・コードのデバッグ情報にのみ影響します。これは、装置コードのデバッグ情報がオブジェクト・ファイルまたは実行可能ファイルの特別なセクションに保管されるので、分割して取り出せないためです。 GPU
- このオプションには、.dwo ファイルを読み取ることができるデバッガーが必要です。

-I

カテゴリー

入力制御

@PROCESS

なし。

目的

ディレクトリーをインクルード・ファイル、.mod ファイル、および .smod ファイルの検索パスに追加する。

構文

デフォルト

適用されません。

パラメーター

path_name
有効なパス名 (例えば、**/home/dir**、**/tmp**、または **./subdir**)。

使用法

XL Fortran が **cpp** を呼び出す場合、このオプションを指定しておくと、**#include** ファイルの検索パスにディレクトリーが追加されます。コンパイラーは、インクルードするデフォルト・ディレクトリー、.mod ファイル、および .smod ファイルを検査する前に、検索パス内の個々のディレクトリーを検査します。インクルード・ファイルの場合は、**INCLUDE** 行のファイル名が絶対パスで示されていない場合にのみ、このパスが使用されます。 **#include** ファイルの **-I** オプションについての詳細は、**cpp** の資料を参照してください。

規則

検索を行う前に、コンパイラーは *path_name* に / を追加して、ファイル名と連結します。複数の **-I** オプションがコマンド行に指定されると、ファイルはコマンド行上の *path_name* 名の順序で検索されます。

-I オプションで指定した任意のパスを検索した後、以下のディレクトリーが次のような順序で検索されます。

1. 現行ディレクトリー (ここでコンパイラーが実行されます)
2. ソース・ファイルがあるディレクトリー (上記の 1 と違う場合)
3. `/usr/include`

また、コンパイラーは、コンパイラーに付属のインクルード・ファイルおよび **.mod** ファイルが入っている `/opt/ibm/xlf/16.1.0/include` の中も検索します。

関連情報

- 219 ページの『-qmoddir』
- 161 ページの『-qfullpath』

-k

カテゴリー

入力制御

目的

ソース・コードが自由ソース形式であることを示す。

このオプションは **-qfree=f90** の短い形式です。

構文

オプション:

@PROCESS:

`@PROCESS FREE(F90)`

関連情報

- 160 ページの『-qfree』
- 「*XL Fortran* ランゲージ・リファレンス」の『自由ソース形式』

-L

カテゴリー

リンク

@PROCESS

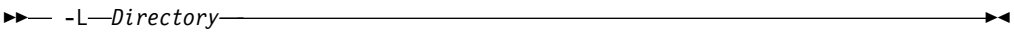
なし。

目的

-l オプションによって指定されたライブラリー・ファイルのディレクトリー・パスをリンク時に検索する。

構文

オプション:



デフォルト

適用されません。

使用法

Directory を **-l** フラグ (**L** の小文字) で指定されたライブラリーの検索に使用する検索ディレクトリーのリストに追加します。/opt/ibm/xlf/16.1.0/lib で指定されているデフォルト・ライブラリー以外のライブラリーを使用する場合は、その他のライブラリーの場所を示す **-L** オプションを 1 つ以上指定します。

規則

このオプションは **ld** コマンドに直接渡され、XL Fortran によって処理されることは絶対にありません。

関連情報

- 77 ページの『リンク』
- 41 ページの『XL Fortran プログラムのリンク』

-l

カテゴリー

リンク

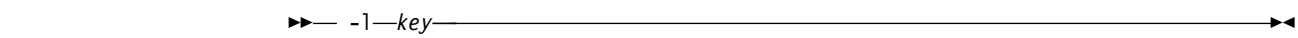
@PROCESS

なし。

目的

指定されたライブラリー・ファイルを検索します。リンカーは *libkey.so* を検索し、*libkey.so* が見つからない場合は *libkey.a* を検索します。

構文



デフォルト

コンパイラーのデフォルト設定では、一部のコンパイラーのランタイム・ライブラリーしか検索できません。デフォルトの構成ファイルは **-l** コンパイラー・オプションを使用してデフォルトのライブラリー名を指定し、**-L** コンパイラー・オプションを使用してライブラリーのデフォルト検索パスを指定します。

パラメーター

key
lib と *.a* または *.so* 文字を除去したライブラリー名です。

規則

このオプションは **ld** コマンドに直接渡され、XL Fortran によって処理されることは絶対にありません。

例

myprogram.f をコンパイルし、/usr/mylibdirディレクトリーにあるライブラリー libmylibrary.so または libmylibrary.a とリンクするには、以下のコマンドを入力します。libmylibrary.a よりも libmylibrary.so が優先されます。

```
xlc myprogram.f -lmylibrary -L/usr/mylibdir
```

関連情報

- 77 ページの『リンク』
- 41 ページの『XL Fortran プログラムのリンク』

-MF

カテゴリー

出力制御

@PROCESS

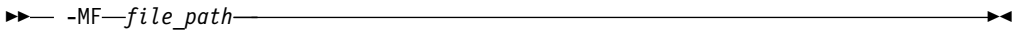
なし。

目的

-qmakedep オプションまたは **-MMD** オプションによって生成される従属関係出力ファイルの名前または場所を指定する。

-qmakedep および **-MMD** オプションについて詳しくは、210 ページの『-qmakedep』 および 99 ページの『-MMD』 を参照してください。

構文



デフォルト

-MF が指定されていない場合、従属関係出力ファイルは現行作業ディレクトリー内に作成されます。このファイルは、ベース名はオブジェクト・ファイルと同じですが、.d サフィックスが付きます。

パラメーター

file_path
生成される従属関係出力ファイルの名前または場所です。*file_path* には、ファイルのパス名またはディレクトリーを指定できます。相対または絶対のいずれかにできます。

file_path が既存のディレクトリー名である場合、生成される従属関係出力ファイルは指定ディレクトリーに置かれます。このファイルは、ベース名は生成されるオブジェクト・ファイルと同じですが、.d サフィックスが付きます。それ以外の場合、*file_path* は生成される従属関係出力ファイルの名前になります。詳しくは、『例』のセクションを参照してください。

使用法

-MF オプションによって指定されたファイルが既に存在する場合は、上書きされます。

複数のソース・ファイルをコンパイルするときに **-MF** オプションに対して単一ファイル名を指定すると、単一の従属関係ファイルのみが生成されます。従属関係ファイルには、コマンド・ラインで最後に指定されたファイルの **make** 規則が含まれます。

例

source.f をコンパイルして、mysource.d という名前の従属関係出力ファイルを作成するには、以下を入力します。

```
xlf -c -qmakedep source.f -MF mysource.d
```

source1.f および source2.f をコンパイルして、それぞれ source1.d および source2.d という名前の 2 つの従属関係出力ファイルを /project/deps/ ディレクトリー内に作成するには、以下を入力します。

```
xlf -c -qmakedep source1.f source2.f -MF /project/deps/
```

source.f をコンパイルして、mysource.d という名前の従属関係出力ファイルを deps/ ディレクトリー内に作成するには、以下を入力します。

```
xlf -c -qmakedep source.f -MF deps/mysource.d
```

source.f をコンパイルして、object.o という名前のオブジェクト・ファイルおよび mysource.d という名前の従属関係出力ファイルを作成するには、以下を入力します。

```
xlf -c -qmakedep source.f -o object.o -MF mysource.d
```

関連情報

- 210 ページの『-qmakedep』
- 『-MMD』
- 『-MT』

-MMD

カテゴリー

出力制御

@PROCESS

なし。

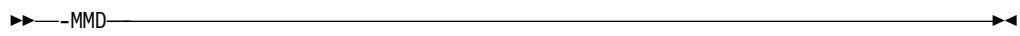
目的

make コマンドの記述ファイルの組み込みに適したターゲットを含む従属関係出力ファイルを生成する。

従属関係出力ファイルには .d サフィックス付きの名前が付けられます。このファイルは、メイン・ソース・ファイルの従属関係それぞれに対して個別の規則を指定します。

-MMD は 210 ページの『-qmakedep』 の短い形式です。

構文



デフォルト

適用されません。

関連情報

- 210 ページの『-qmakedep』

-MT

カテゴリー

出力制御

@PROCESS

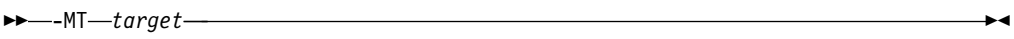
なし。

目的

-qmakedep オプションまたは **-MMD** オプションによって生成される従属関係出力ファイル内の **make** 規則のオブジェクト・ファイルのターゲット名を指定する。

-qmakedep および **-MMD** オプションについて詳しくは、210 ページの『-qmakedep』 および 99 ページの『-MMD』 を参照してください。

構文



デフォルト

-MT が指定されていない場合、ターゲット名はオブジェクト・ファイルのベース名になります。

パラメーター

ターゲット ***(target)***
: 生成された従属関係ファイル内のオブジェクト・ファイルに対して指定する名前。

使用法

指定できるターゲットは 1 つのみです。複数の **-MT** オプションを指定した場合、最後の **-MT** オプションからのターゲットが使用されます。

例

mysource.f という名前のソース・ファイルには、以下のコードが入っています。

```
#include "options.h"
MODULE m
  USE n

CONTAINS
  SUBROUTINE sub
    IMPLICIT NONE
    INCLUDE 'constants.h'
    CALL my_print(pi)
  END SUBROUTINE
END MODULE
```

mysource.f をコンパイルして、mysource.d という名前の従属関係出力ファイルを作成し、同時にパス情報 /home/user/sample/ をオブジェクト・ファイルのターゲット名として mysource.d ファイルに組み込むには、以下を入力します。

```
xlf -c -qmakedep mysource.f -MT '/home/user/sample/mysource.o'
```

その場合、以下のような mysourced ファイルが生成されます。

```
/home/user/sample/mysource.o m.mod: n.mod
/home/user/sample/mysource.o m.mod: option.h
/home/user/sample/mysource.o m.mod: mysource.f
/home/user/sample/mysource.o m.mod: constants.h
```

関連情報
- 210 ページの『-qmakedep』
- 99 ページの『-MMD』
- 97 ページの『-MF』

-NS

カテゴリー

コンパイラーのカスタマイズ

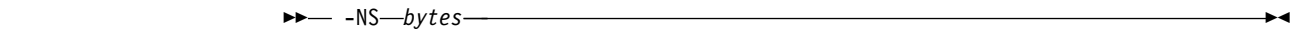
目的

レジスター・スピル・スペース (溢れたレジスターをストレージに退避させるために最適化プログラムが使用する内部プログラム・ストレージ域) のサイズをバイト単位で指定する。

-qspillsize は **-NS** の長い形式です。

構文

オプション:



@PROCESS:

```
@PROCESS SPILLSIZE(bytes)
```

デフォルト

デフォルト時には、個々のサブプログラムのスタックには、512 バイトの予備空間が確保されます。

このオプションが必要な場合は、コンパイル時メッセージでその旨をユーザーに通知します。

パラメーター

bytes
: レジスターに保持する変数の数が多すぎて、プログラムがレジスターの内容用の一時ストレージを必要とする場合に備えて、個々のサブプログラムに確保するスタック空間のバイト数。

関連情報
- 268 ページの『-qspillsize』

-O

カテゴリー

最適化およびチューニング

目的

コンパイル中にコードを最適化するかどうかを指定し、最適化する場合は、レベルを指定する。

-qOPTimize は **-O** の長い形式です。

構文

オプション:

@PROCESS:

@PROCESS OPTimize[(*0*|*2*|*3*)] | **NOOPTimize**

デフォルト

nooptimize または **-O0** または **optimize=0**

パラメーター

指定しない場合
: ほぼすべての最適化が使用不可になります。これは、**-O0** または **-qnoopt** を指定するのと同じです。

-O
: XL Fortran の各リリースで、**-O** は、コンパイル速度と実行時のパフォーマンスの最良のトレードオフを表す最適化のレベルを使用可能にします。特定のレベルの最適化が必要な場合は、適切な数値を指定してください。 **-O** は、**-O2** と同等です。

-O0
: ほぼすべての最適化が使用不可になります。このオプションは、**-qnoopt** と同じです。

-O1
: 将来の利用に備えて予約されています。このフォームは無視され、コンパイルの結果には影響を与えません。

-O2
: コンパイルに必要な時間やストレージを不適切に増やすことなく、改善されたパフォーマンスを提供することを意図した一連の最適化を行います。

-O3 | **-Ofast**
: メモリーとコンパイル時間を大量に使用してさらに最適化を行います。その結果、**-qstrict** を指定していない場合は、プログラムのセマンティクスがわずかに変更される場合があります。コンパイル時にリソースに制限が加わっても、実行時の速度の向上の方を重視したいときに、この最適化を使用してください。

また、このレベルの最適化は、 **rsqrt** サブオプションをデフォルトでオンにすることで **-qfloat** オプションの設定に影響を及ぼし、**-qmaxmem=-1** を設定します。

-O3 または **-Ofast** を指定すると、**-qhot** が暗黙指定されます。

-O4 積極的にソース・プログラムを最適化しますが、生成コードの改善のためにコンパイル時間が余分にかかります。このオプションは、コンパイル時、またはリンク時に指定することができます。このオプションをリンク時に指定すると、少なくともメインプログラムを含んでいるファイルのコンパイル時にも指定しない限り、効果はありません。

-O4 は、次のオプションを暗黙指定します。

- **-qipa**
- **-O3** (および、このオプションが暗黙指定するすべてのオプションと設定)
- **-qarch=auto**
- **-qtune=auto**
- **-qcache=auto**

-qarch、**-qtune**、**-qcache** の「**auto**」設定は、実行環境がコンパイル環境と同じであることを暗黙指定することに注意してください。

このオプションは「最後のオプションが選択される」という競合解決規則に従っており、そのため **-O4** によって変更されるオプションを後から変更できます。

-O5 **-O4** オプションの機能をすべて使用できるだけでなく、 **-qipa=level=2** オプションの機能も使用できます。

注**:**

最適化したアプリケーションと最適化していないアプリケーションで同じ浮動小数点の正確度を得るには、**-qfloat=nomaf** コンパイラー・オプションを指定しなければなりません。 **-qfloat=nomaf** を指定した後でも、依然として浮動小数点の正確度に差がある場合は、**-qstrict** コンパイラー・オプションを使用すると、最適化が原因で浮動小数点のセマンティクスで行われる変更を制御できます。

使用法

普通は、コンパイルとリンク・ステップの両方に同じ最適化レベルを使用します。**-O4** あるいは **-O5** 最適化レベルを使用して最良の実行時パフォーマンスを得るには、こうすることが重要です。 **-O5** レベルの場合、すべてのループ変換 (**-qhot** オプションを介して指定したように) は、リンク・ステップで行われます。

最適化のレベルを高くすることで、パフォーマンスがさらに改善される場合と、されない場合があります。これは、分析を加えてさらに最適化の機会を見い出せるかどうかによって決まります。

-O3 以上の最適化レベルでは、プログラムの動作を変更することがあり、その結果、通常は起こらない例外が起こる可能性があります。 **-qstrict** オプションを使用することによって、最適化の代わりに、**-O2** と同じプログラム動作を保持します。使用不可になる最適化のリストについては、**-qstrict** オプションを参照してください。

-O 以上の最適化を使用する場合は、**-qtbtable=small** が有効になります。生成されるトレースバック・テーブルには関数名やパラメーターの情報は含まれません。

@PROCESS 文で **-O** オプションを使用する場合は、最適化レベル 0、2、または 3 のみを使用できます。コマンド行呼び出しでの **-O3** の使用とは異なって、**@PROCESS OPT(3)** の指定は、**-qhot** を暗黙指定しません。 **-O3** 以上の最適化レベルがコマンド行に指定されている場合、その最適化レベルによって設定された **-qhot** オプションおよび **-qipa** オプションは、**@PROCESS OPT(0)** または **@PROCESS OPT(2)**によってオーバーライドできません。

最適化とともにコンパイルを行うと、時間とマシン・リソースが他のコンパイルよりも必要になる場合があります。

コンパイラーがプログラムを最適化すればするほど、プログラムをシンボリック・デバッガーでデバッグするのが難しくなります。

関連情報

- 285 ページの『-qtbtable』 は、オブジェクト・ファイルに含まれるデバッグ・トレースバック情報の量を制御します。
- 274 ページの『-qstrict』には、プログラムのセマンティクスを変える場合のある **-O3** の影響を取り去る方法が示されています。
- 192 ページの『-qipa』、 228 ページの『-qpdf1、-qpdf2』 と 170 ページの『-qhot』は、一部のプログラムについて、パフォーマンスを向上させる可能性がある追加の最適化をオンにします。
- 「*XL Fortran* 最適化およびプログラミング・ガイド」の『アプリケーションの最適化』には、コンパイラーが使用する最適化手法の技術的な詳細と、コードから最大限のパフォーマンスを得るために使用できるいくつかの方法が説明されています。

-o

カテゴリー

出力制御

@PROCESS

なし。

目的

出力オブジェクト、アセンブラー、または実行可能ファイルの名前を指定する。

構文

デフォルト

実行可能ファイルのデフォルト名は、**a.out** です。 オブジェクト・ソース・ファイルまたはアセンブラー・ソース・ファイルのデフォルト名はソース・ファイルと同じですが、拡張子**.o** または **.s** が付きます。

使用法

オブジェクト・ファイルの名前を選択するには、このオプションを **-c** オプションと組み合わせて使用してください。アセンブラー・ソース・ファイルの場合は、これを **-S** オプションと組み合わせて使用してください。

規則

オプション **-c** または **-S** が指定されている場合を除き、**-o** オプションは XL Fortran で処理されないで、**ld** c コマンドに直接渡されます。

例

```
xlf95 t.f                   # Produces "a.out"
xlf95 -c t.f                # Produces "t.o"
xlf95 -o test_program t.f   # Produces "test_program"
xlf95 -S -o t2.s t.f        # Produces "t2.s"
```

-p

カテゴリー

最適化およびチューニング

@PROCESS

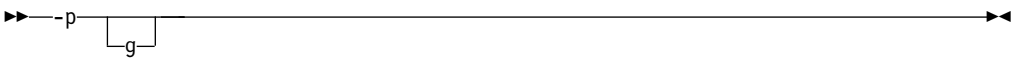
なし。

目的

コンパイラーが作成するオブジェクト・ファイルをプロファイル用に準備する。

コンパイラーは個々のルーチンが呼び出される回数をカウントするモニター・コードを作成します。コンパイラーは、各サブプログラムの開始時にモニター・サブルーチンへの呼び出しを挿入します。

構文



デフォルト

適用されません。

使用法

プログラムを **-p**または **-pg** でコンパイルして正常に終了すると、プロファイル情報を使用する**gmon.out** ファイルが作成されます。これで、**gprof** コマンドを使用して実行時プロファイルを作成することができます。

例

```
$ xlf95 -pg needs_tuning.f
$ a.out
$ gprof
 .
 .
 .

detailed and verbose profiling data
 .
 .
 .
```

関連情報

- プロファイルと、**gprof** コマンドについて詳しくは、ご使用のオペレーティング・システムの資料を参照してください。

-qalias

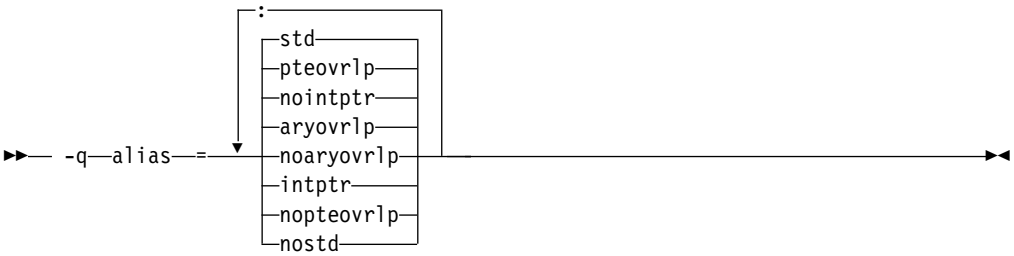
カテゴリー

最適化およびチューニング

目的

これは、特定のカテゴリーの別名割り当てがプログラムに含まれているのか、またはプログラムが標準の Fortran 別名割り当て規則に準拠していないのかを示します。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。

構文



@PROCESS:

```
@PROCESS  ALIAS( {ARGUMENT_LIST} )
```

デフォルト

-qalias=aryovrlp:nointptr:pteovrlp:std

パラメーター

aryovrlp | **noaryovrlp**
コンパイル単位にストレージ関連配列間の配列割り当てが含まれているかどうかを示します。含まれていない場合は、**noaryovrlp** を指定してパフォーマンスを改善します。

intptr | **nointptr**
コンパイル単位に整数 **POINTER** 文が含まれているかどうかを示します。含まれている場合、**intptr** を指定してください。

pteovrlp | **nopteovrlp**
ポインティング先変数でないデータ・オブジェクトを参照するためにポインティング先変数を使用できるかどうか、または 2 つのポインティング先変数が同じストレージ・ロケーションを参照できるかどうかを示します。含まれていない場合は、 **nopteovrlp** を指定してください。

std | **nostd**
コンパイル単位に非標準別名付け (以下を参照) が含まれているかどうかを示します。含まれている場合、**nostd** を指定してください。

使用法

ストレージ内の項目が複数の名前で参照できる場合は、別名が存在します。 Fortran 90、Fortran 95、Fortran 2003、および Fortran 2008 標準では、別名付けを許可する型と許可しない型があります。以下の状況のように非標準の別名付けが存在すると、XL Fortran コンパイラーが実行する非常に複雑な最適化によって、好ましくない結果が生じる可能性があります。

- 同じサブプログラム参照で、実引数として同じデータ・オブジェクトが複数回渡されます。実引数のいずれかが定義済み、未定義、または再定義になった場合は、別名付けは無効です。
- サブプログラム参照は、参照されたサブプログラム内部でアクセス可能なオブジェクトと仮引数を関連付けます。仮引数と関連付けられたオブジェクトの何らかの部分が、仮引数への参照によってではなく、定義済み、未定義、再定義になる場合は、別名付けは無効です。
- 仮引数以外の何か別の方法で、仮引数が、その呼び出されたサブプログラム内で定義済み、未定義、または再定義になります。
- 共通ブロック内の配列の境界を超えて添え字を付けています。

制約事項

このオプションはいくつかの変数の最適化を禁止しているので、それを使用するとパフォーマンスが低下します。

ある非標準または整数 **POINTER** 別名付けが含まれているプログラムは、正しい **-qalias** 設定でコンパイルしないと、誤った結果を作成する場合があります。呼び出しコマンド **xlf** または **xlf_r** を使用して .f ファイル、.F ファイル、.f77 ファイル、または .F77 ファイルをコンパイルする場合や、呼び出しコマンド f77 または fort77 を使用する場合、コンパイラーは、整数 **POINTER** が存在する可能性があると想定します (**-qalias=aryovrlp:pteovrlp:std:intptr**)。他のすべての呼び出しコマン

ドは、プログラムに標準の別名付け (**-qalias=aryovrlp:pteovrlp:std:nointptr**) のみが含まれていると想定します。

例

-qalias=nopteovrlp を使って以下のサブルーチンをコンパイルすると、さらに効率のよいコードをコンパイラーが生成できる場合があります。整数ポインター (**ptr1** および **ptr2**) が、動的に割り振られたメモリーだけを指しているため、**-qalias=nopteovrlp** を使ってこのサブルーチンをコンパイルすることができます。

```
  subroutine sub(arg)
      real arg
      pointer(ptr1, pte1)
      pointer(ptr2, pte2)
      real pte1, pte2

      ptr1 = malloc(%val(4))
      ptr2 = malloc(%val(4))
      pte1 = arg*arg
      pte2 = int(sqrt(arg))
      arg = pte1 + pte2
      call free(%val(ptr1))
      call free(%val(ptr2))
  end subroutine
```

コンパイル単位内のほとんどの配列の割り当てが、オーバーラップしない配列に関与しており、少数の割り当てがストレージ関連配列に関与している場合、オーバーラップした割り当てを追加ステップでコーディングすれば **NOARYOVRLP** サブオプションを安全に使用することができます。

```
@PROCESS ALIAS(NOARYOVRLP)
! The assertion that no array assignments involve overlapping
! arrays allows the assignment to be done without creating a
! temporary array.
      program test
        real(8) a(100)
        integer :: j=1, k=50, m=51, n=100

        a(1:50) = 0.0d0
        a(51:100) = 1.0d0

        ! Timing loop to achieve accurate timing results
        do i = 1, 1000000
           a(j:k) = a(m:n)    ! Here is the array assignment
        end do

        print *, a
      end program

! We cannot assert that this unit is free
! of array-assignment aliasing because of the assignments below.
      subroutine sub1
      integer a(10), b(10)
      equivalence (a, b(3))
      a = b          ! a and b overlap.
      a = a(10:1:-1) ! The elements of a are reversed.
      end subroutine

! When the overlapping assignment is recoded to explicitly use a
! temporary array, the array-assignment aliasing is removed.
! Although ALIAS(NOARYOVRLP) does not speed up this assignment,
! subsequent assignments of non-overlapping arrays in this unit
! are optimized.
@PROCESS ALIAS(NOARYOVRLP)
```

```
subroutine sub2
integer a(10), b(10), t(10)
equivalence (a, b(3))
t = b; a = t
t = a(10:1:-1); a = t
end subroutine
```

SUB1 が呼び出される場合は、**J** と **K** の間に別名が存在します。 **J** と **K** は、ストレージ内の同じ項目を参照します。Fortran では、**J** または **K** が更新される場合は別名付けは許可されず、検出されないままであると、予測不能な結果が起きることがあります。

```
CALL SUB1(I,I)
...
SUBROUTINE SUB1(J,K)
```

以下の例では、別名が存在する可能性があることを **-qalias=nostd** が示さない限り、プログラムは 6 の代わりに 5 を **J** に保管します。

```
INTEGER BIG(1000)
INTEGER SMALL(10)
COMMON // BIG
EQUIVALENCE(BIG,SMALL)
...
BIG(500) = 5
SMALL (I) = 6   ! Where I has the value 500
J = BIG(500)
```

関連情報

- 検討すべき別名割り当て方法については、「*XL Fortran* 最適化およびプログラミング・ガイド」の『アプリケーションの最適化』を参照してください。

-qalign

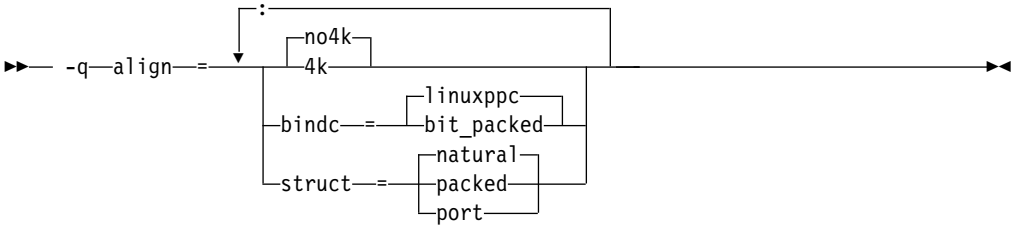
カテゴリー

移植性とマイグレーション

目的

ストレージ内でデータ・オブジェクトの位置合わせを指定する。これによって、誤って配置されたデータが原因で起こるパフォーマンス上の問題を回避できます。

フォーマット



@PROCESS:

```
@PROCESS ALIGN({[NO]4K|STRUCT{(suboption)}|BINDC{(suboption)}})
```

デフォルト

-qalign= no4k:struct=natural:bindc=linuxppc.

パラメーター

[no]4k、**bindc**、および **struct** オプションを一緒に指定することができ、しかも互いに排他的ではありません。**[no]4k** オプションは、基本的には論理ボリューム I/O とディスク・ストライピングの組み合わせに有効です。

[no]4k

データ・ストライプ I/O でパフォーマンスを向上させるため、大きなデータ・オブジェクトをページ (4 KB) 境界へ位置合わせするかどうかを指定します。オブジェクトはオブジェクト・ファイル内でどのように表されるかによって影響を受けます。影響を受けるオブジェクトは、大きさが 4 KB 以上で静的ストレージまたは bss ストレージにある配列と構造体、それに、8 KB 以上の CSECT (通常は **COMMON** ブロック) です。大きな **COMMON** ブロック、配列が入っている等価グループ、構造体は、ページ境界に位置合わせされるため、配列の位置合わせはそれを含んでいるオブジェクト内の配列の位置によって異なります。非シーケンス派生型の構造体の中では、コンパイラーは埋め込みを追加して大きな配列をページ境界に位置合わせします。

bindc={suboption}

BIND(C) 属性を持つ XL Fortran 派生型の位置合わせと埋め込みが、対応する XL C 位置合わせオプションを使用してコンパイルされる C 構文型と互換性を持つことを指定します。互換性のある位置合わせオプションは次のとおりです。

XL Fortran オプション	対応する **XL C** オプション
-qalign=bindc=bit_packed	**-qalign=bit_packed**
-qalign=bindc=linuxppc	**-qalign=linuxppc**

struct={suboption}

struct オプションは、レコード構造体を使用して、宣言された派生型のオブジェクトあるいは配列の保管方法と埋め込みをそれらのコンポーネント間で使用するかどうかを指定します。すべてのプログラム単位は、**-qalign=struct** オプションと同じ設定を使用してコンパイルする必要があります。使用できるサブオプションは、以下の 3 つです。

packed

packed サブオプションを **struct** オプションで指定すると、派生型のオブジェクトは、**%FILL** コンポーネントで表される任意の埋め込みを除いて、コンポーネント間で埋め込みを行わずに保管されます。ストレージ形式は、標準派生型宣言を使用して宣言されていた派生型を持つシーケンス構造体の結果と同じです。

natural

natural サブオプションを **struct** オプションで指定すると、派生型のオブジェクトは、その他にストレージの関連付けを要求していな

ければ、コンポーネントが自然の位置合わせ境界で保管されるように、十分埋め込みを行って、保管されます。下の表の左側の列にあるオブジェクト・タイプの自然の位置合わせ境界は、テーブルの右側の列にある対応する項目のバイト数の倍数を示しています。

型	自然の位置合わせ (バイトの倍数)
INTEGER(1), LOGICAL(1), BYTE, CHARACTER	1
INTEGER(2), LOGICAL(2)	2
INTEGER(4), LOGICAL(4), REAL(4)	4
INTEGER(8), LOGICAL(8), REAL(8), COMPLEX(4)	8
REAL(16), COMPLEX(8), COMPLEX(16)	16
派生	そのコンポーネントの最大位置合わせ

natural サブオプションを **struct** オプションで指定すると、派生型の配列はその他にストレージの関連付けを要求していない限り、自然の位置合わせ境界で各エレメントの各コンポーネントが保管されるように保管されます。

port

port サブオプションを **struct** オプションで指定すると、

- ストレージ埋め込みは、上記の **natural** サブオプションで説明したのと同じになります。ただし、型 complex の位置合わせが、同じ種類の型 real のコンポーネントの位置あわせと同じ場合を除きます。
- 直後に共用体が続くオブジェクトの埋め込みは、その共用体内の各マップで、最初のマップ・コンポーネントの始めに挿入されます。

制約事項

port サブオプションは **AUTOMATIC** 属性を持つ配列または構造体、あるいは動的に割り振られる配列には影響を及ぼしません。このオプションは非シーケンス派生型のレイアウトを変更する場合があるので、不定形式ファイルを使用してそのようなオブジェクトを読み書きするプログラムをコンパイルする場合は、すべてのソース・ファイルについてこのオプションには同じ設定を使用してください。

配列が **AUTOMATIC** 属性を持つかどうか、また、そのために **-qalign=4k** によって影響を受けないかどうかは、 116 ページの『-qattr』のリストで **AUTOMATIC** キーワードまたは **CONTROLLED AUTOMATIC** キーワードを探せば知ることができます。このリストには、データ・オブジェクトのオフセットも示されています。

-qaltivec

カテゴリー

言語エレメント制御

@PROCESS

ALTIVEC(LE | BE)

目的

これは、ベクトル・レジスターにおけるベクトル・エレメントの順序を指定します。

構文

デフォルト

-qaltivec=le

パラメーター

be ビッグ・エンディアン・エレメント順序を指定します。ベクトルは、ベクトル・レジスターに左から右に向かって配置されます。このとき、エレメント 0 がレジスター内の左端のエレメントになるように配置されます。

le リトル・エンディアン・エレメント順序を指定します。ベクトルは、ベクトル・レジスターに右から左に向かって配置されます。このとき、エレメント 0 がレジスター内の右端のエレメントになるように配置されます。

使用法

-qaltivec オプションは、以下のカテゴリーのプロシージャーに影響します。

- Vector Multimedia Extension (VMX) のロードおよび保管組み込みプロシージャー
- Vector Scalar Extension (VSX) のロードおよび保管組み込みプロシージャー
- ベクトル・エレメント順序を参照する非ロードおよび非保管組み込みプロシージャー

以下のリストに、影響を受けるすべてのプロシージャーを示します。

- ロード・プロシージャー
  - VMX ロード・プロシージャー: **VEC_LD**、**VEC_LDE**、および **VEC_LDL**
  - VSX ロード・プロシージャー: **VEC_XLD2**、**VEC_XLW4**、および **VEC_XL**
- 保管プロシージャー
  - VMX 保管プロシージャー: **VEC_ST**、**VEC_STE**、および **VEC_STL**
  - VSX 保管プロシージャー: **VEC_XSTD2**、**VEC_XSTW4**、および **VEC_XST**

- 非ロード・プロシージャーおよび非保管プロシージャー: **VPERMXOR**、**VEC_EXTRACT**、**VEC_INSERT**、**VEC_MERGEH**、**VEC_MERGEL**、**VEC_MERGEE**、**VEC_MERGEO**、**VEC_MULE**、**VEC_MULO**、**VEC_PACK**、**VEC_PERM**、**VEC_PROMOTE**、**VEC_SPLAT**、 **VEC_SUM2S**、**VEC_SUMS**、**VEC_UNPACKH**、および **VEC_UNPACKL**

例

- レジスターでベクトル・エレメント順序をビッグ・エンディアン・エレメント順序に変更するには、以下のコマンドを入力します。

```
xlf95 myprogram.f -qaltivec=be
```

関連情報

- 『-qarch』
- ベクトル組み込みプロシージャー (IBM 拡張)
- ベクトル・データ型
- レジスター内のベクトル・レイアウト (ビッグ・エンディアンまたはリトル・エンディアン・エレメント順序)
- 258 ページの『-qsimd』
- 「*AltiVec Technology Programming Interface Manual*」(http://www.freescale.com/files/32bit/doc/ref_manual/ALTIVECPIM.pdf から入手可能)

-qarch

カテゴリー

最適化およびチューニング

@PROCESS

なし。

目的

コードを実行する可能性のある、プロセッサー・アーキテクチャーまたはアーキテクチャーのファミリーを指定します。これによって、コンパイラーはアーキテクチャーに固有か、またはアーキテクチャーのファミリーに共通のマシン命令を最大限に活用することができます。

構文

デフォルト

- **-qarch=pwr8**
- **-O4** または **-O5** が有効な場合は **-qarch=auto**

パラメーター

auto
コンパイルを実行しているマシンの特定のアーキテクチャーを自動的に検出します。ランタイム環境はコンパイル環境と同じであると見なされます。**-O4** または **-O5** オプションが設定されている、または暗黙指定されている場合、このオプションは暗黙指定されます。

pwr8
POWER8 または POWER9™ ハードウェア・プラットフォームで実行する命令を含んだオブジェクト・コードを作成します。

pwr9
POWER9 テクノロジーを利用する命令を含んだオブジェクト・コードを作成します。

使用法

どの **-qarch** 設定についても、コンパイラーは特定のマッチングする **-qtune** 設定をデフォルトとして使用します。それにより、さらにパフォーマンスが改善されます。**-qarch** と **-qtune** の同時使用について詳しくは、 292 ページの『-qtune』を参照してください。

例

`myprogram.f` からコンパイルされた実行可能プログラム `testing` が、VSX 命令サポート (例えば power8) を使用したコンピューターで実行されるように指定するには、以下を入力します。

```
xlf -o testing myprogram.f -qarch=pwr8
```

関連情報

- -qprefetch
- -qfloat
- 292 ページの『-qtune』
- 37 ページの『特定アーキテクチャーのためのコンパイル方法』

-qassert

カテゴリー

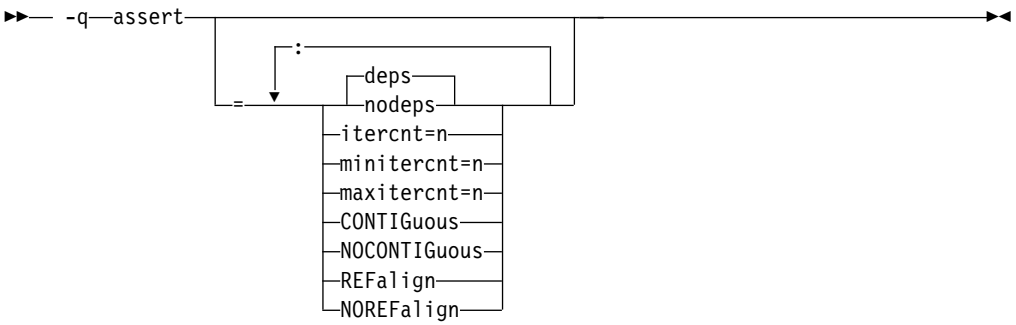
最適化およびチューニング

目的

コンパイラーの最適化の微調整に役立つコードの特性に関する情報を提供する。

構文

オプション:



@PROCESS:

@PROCESS ASSERT(suboptions)

デフォルト

-qassert=deps:norefalign:nocontig

パラメーター

deps | **nodeps**
ループ送り依存性があるかどうかを指定します。

itercnt=*n*
ループ反復数を静的に判別できないときに最適化プログラムが使用するループの反復カウント値を指定します。*n* は正の整数でなければなりません。

minitercnt=*n*
プログラム内で予期されるループの最小反復カウントを指定します。*n* は正の整数でなければなりません。

maxitercnt=*n*
プログラム内で予期されるループの最大反復カウントを指定します。*n* は正の整数でなければなりません。

CONTIGuous | **NOCONTIGuous**
すべてのコンパイル単位について、次の隣接性を指定します。

- すべての配列ポインターは、隣接したターゲットに関連付けられたポインターである。
- すべての想定形状配列は、隣接した実引数に関連付けられた引数である。

-qassert=contig が指定されると、コンパイラーはメモリーの隣接したブロックを占有するオブジェクトのメモリー・レイアウトに従って、最適化を実行できます。

-qassert=contig の使用では、配列ポインターおよび想定形状配列に対して **CONTIGUOUS** 属性を指定するのと同じ効果は得られません。**-qassert=contig** では隣接性アサーションの妥当性検査は行われません。 F2008 Fortran 2008 セマンティクスに適合するようにするには、**CONTIGUOUS** 属性を使用してください。 F2008

注:

- **-qassert=contig** は、**ASSERT** ディレクティブではサポートされません。
- このサブオプションを使用すると、警告なしに予期しない結果が発生する場合があります。

REFalign | **NOREFalign**
コンパイル単位内のすべてのポインターが、ポインター型の長さに従って自然に位置合わせされたデータのみを指すように指定します。このアサーションを行うと、コンパイラーがより効率のよいコードを生成する可能性があります。このアサーションが特に有用なのは、SIMD アーキテクチャーをターゲットとして、**-qhot=level=0** または **-qhot=level=1** を **-qsimd=auto** と一緒に指定した場合です。

使用法

itercnt、**minitercnt**、および **maxitercnt** の値は、正確である必要はありません。これらの値は、パフォーマンスのみに影響し、正確性に影響することはありません。**minitercnt** <= **itercnt** <= **maxitercnt** のルールに従って、値を指定します。そうでない場合は、値が不整合であることを示すメッセージが発行され、不整合な値は無視されます。

関連情報
- 「*XL Fortran* 最適化およびプログラミング・ガイド」の『上位変換』 (アサーションの使用に関する背景情報と説明)
- 「*XL Fortran* ランゲージ・リファレンス」の **ASSERT** ディレクティブ
- F2008 CONTIGUOUS 属性 F2008

-qattr

カテゴリー

リスト、メッセージ、およびコンパイラー情報

目的

リストの属性および相互参照セクションの属性コンポーネントを組み込むコンパイラー・リストを作成する。

構文

@PROCESS:

@PROCESS ATTR[(FULL)] | **NOATTR**

デフォルト

-qnoattr

パラメーター

full
プログラム内のすべての識別子を、参照されているかどうかにかかわらず報告します。このサブオプションを使用しないで **-qattr** を指定すると、使用される識別子だけが報告されます。

使用法

-qattr=full の後に **-qattr** が指定されると、完全な属性リストが依然として作成されます。

属性リストを使用して、正しく指定されていない属性が起こす問題のデバッグを支援することができます。あるいは、新しいコードを書いている際に各オブジェクトの属性の覚え書きとして使用することもできます。

関連情報

- 72 ページの『リスト、メッセージ、およびコンパイラー情報』
- 356 ページの『属性および相互参照セクション』

-qautodbl

カテゴリー

浮動小数点および整数のコントロール

目的

単精度浮動小数点計算を倍精度へ変換する操作、および倍精度計算を拡張精度へ変換する操作が自動的に実行されるようにする。

構文

@PROCESS:

`@PROCESS AUTODBL(`*setting*`)`

デフォルト

-qautodbl=none

パラメーター

-qautodbl サブオプションを使用して、プロモートまたは埋め込みが行われるオブジェクト間、あるいはプロモートまたは埋め込みが行われないオブジェクト間のストレージの関係を保持するための別の方法を選択します。

使用できる設定は次のとおりです。

none ストレージを共用しているオブジェクトのプロモートまたは埋め込みを行いません。この設定はデフォルトです。

dbl4 単精度の浮動小数点オブジェクト (サイズは 4 バイト)、または同様なオブジェクトから構成されている浮動小数点オブジェクト (例えば **COMPLEX** または配列オブジェクト) をプロモートします。
- **REAL(4)** は、**REAL(8)** にプロモートされます。
- **COMPLEX(4)** は、**COMPLEX(8)** にプロモートされます。

このサブオプションは、リンク中に **libxlfpmt4.a** ライブラリーを必要とします。

dbl8 倍精度の浮動小数点オブジェクト (サイズは 8 バイト)、または同様なオブジェクトから構成されている浮動小数点オブジェクトをプロモートします。
- **REAL(8)** は、**REAL(16)** にプロモートされます。
- **COMPLEX(8)** は、**COMPLEX(16)** にプロモートされます。

このサブオプションは、リンク中に **libxlfpmt8.a** ライブラリーを必要とします。

dbl **dbl4** が実行するプロモーションと **dbl8** が実行するプロモーションを結合します。

このサブオプションは、リンク中に **libxlfpmt4.a** および **libxlfpmt8.a** ライブラリーを必要とします。

使用法

ストレージの関係が重要で、XL Fortran のデフォルトとは異なっている場合に、コードを移植する際にこのオプションを使用すると便利なことがわかります。例えば、IBM VS FORTRAN コンパイラー用に書かれたプログラムは、そのコンパイラーの同等のオプションを使用することができます。

リンク中に適切な **-qautodbl** オプションが指定されると、プログラムは必要なエクストラ・ライブラリーと自動的にリンクされます。自動的にリンクされない場合は、手操作でリンクする必要があります。

同一プログラムに **REAL(4)** 計算と **REAL(8)** 計算の両方があり、**REAL(8)** 演算のスピードを落とさずに **REAL(4)** 演算をスピードアップしたい場合は、**dbl4** を使用してください。

すべての結果の精度を最高にする場合、**dbl** を使用できます。**dbl4** および **dbl8** サブオプションは、各自オプションの精度が高められている実数型のサブセットを選択します。

dbl4 を使用すると、**REAL(8)** オブジェクトを **REAL(16)** にしなくても、**REAL(4)** オブジェクトのサイズを大きくできます。**REAL(16)** 計算では、**REAL(8)** 計算の場合より効率が悪くなります。

-qautodbl オプションは、プロモートされる引数を持つ組み込み機能への呼び出しを処理します。必要な場合は、正しい高精度の組み込み関数が代わりに使用されます。例えば、単精度項目がプロモートされている場合は、プログラム内で **SIN** を呼び出すと、それが自動的に **DSIN** への呼び出しになります。

ご使用のプログラムにベクトル・タイプが含まれている場合、 **-qautodbl** オプションを指定してはなりません。

制約事項

- 文字データはプロモートも埋め込みもされないので、プロモートまたは埋め込みが行われるストレージ関連の項目との関係は維持することができません。
- ポインティング先用のストレージ・スペースがシステム・ルーチン **malloc** によって取得される場合、それがプロモートまたは埋め込みされるならば、**malloc** に対して指定されたサイズには、ポインティング先を表すのに必要な余分な空間を考慮に入れなければなりません。
- 高精度の特定の名前が存在しないために組み込み関数をプロモートできない場合は、元の組み込み関数が使用されて、コンパイラーが警告メッセージを表示します。
- プログラム内のすべてのコンパイル単位は、同一の **-qautodbl** 設定でコンパイルする必要があります。

関連情報

プロモーション、埋め込み、ストレージ/値の関係に関する背景情報を得たり、ソースの例を参照するには、 360 ページの『-qautodbl のプロモーションと埋め込みの実装の詳細』を参照してください。

245 ページの『-qrealsize』には、**-qautodbl** のように機能しますが、デフォルトの kind 型の項目にのみ影響を与え、埋め込みはまったく行わない、別のオプションについて説明してあります。 **-qrealsize** および **-qautodbl** オプションの両方を指定する場合、**-qautodbl** だけが有効になります。また、**-qautodbl** は、**-qdpc** オプションをオーバーライドします。

-qbindcextname

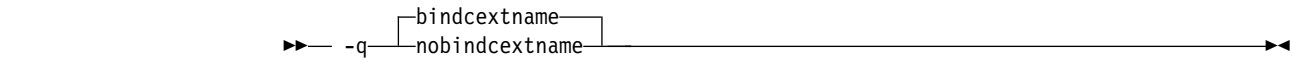
カテゴリー

移植性とマイグレーション

目的

-qextname オプションが **BIND(C)** エンティティーに影響するかどうか制御する。

構文



@PROCESS:

@PROCESS **BINDCEXTNAME** | NOBINDCEXTNAME

デフォルト

-qbindcextname

使用法

-qextname オプションおよび **BIND(C)** 属性は、C での使用を可能にするため、Fortran のグローバル・エンティティーの名前を変更する 2 つの方法です。

NAME= 指定子を使用して、インターフェース・ブロックに **BIND(C)** バインディング・ラベルを明示的に指定した場合、コンパイラーは、**-qextname** および **-qbindcextname** オプションに関係なく、プロシージャーへの呼び出しでこのバインディング・ラベルを使用します。

NAME= 指定子を使用して、**BIND(C)** バインディング・ラベルをインターフェースが明示的に指定しない場合、コンパイラーは暗黙的なバインディング・ラベルを作成します。**-qextname** オプションも指定した場合、コンパイラーは **-qbindcextname** オプションが有効な場合のみ、暗黙的なバインディング・ラベルに下線を付加します。

BIND(C) プロシージャーを宣言するコンパイル単位に対して **-qextname** および **-qbindcextname** オプションを指定した場合、バインディング・ラベルが明示的に指定されていても、コンパイラーはそのバインディング・ラベルに下線を付加します。

注**:**

- **BIND(C)** エンティティーの名前が同じであるかどうか確認する必要があります。これによって、同じ **BIND(C)** エンティティーで明示的に指定されているバインディング・ラベルのないものに 2 つのコンパイル単位がアクセスした場合、1 つの単位に **-qbindcextname** オプションを指定し、もう一方に **-qnobindcextname** オプションを指定してコンパイルしてはなりません。
- **-q[no]bindcextname** オプションは、 **-qextname** オプションも指定された場合にのみ有効です。**-qextname** オプションが名前付きエンティティーのリストで指定された場合、**-q[no]bindcextname** オプションは、これらの名前付きエンティティーのみに影響を及ぼします。

例

```
interface
    integer function foo() bind(c)
    end funciton
    integer function bar()
    end function
end interface

print *, foo()
print *, bar()
end

xlf90 x.f -qextname -qbindcextname         # calls "foo_", and "bar_"
xlf90 x.f -qextname -qnobindcextname       # calls "foo", and "bar_"
xlf90 x.f -qextname=foo -qbindcextname     # calls "foo_", and "bar"
xlf90 x.f -qextname=foo -qnobindcextname   # calls "foo", and "bar"
xlf90 x.f                                  # calls "foo", and "bar"
xlf90 x.f -qnobindcextname                 # calls "foo", and "bar"
```

関連情報

- 147 ページの『-qextname』
- 『BIND (Fortran 2003)』

- 「バインディング・ラベル」

-qcache

カテゴリー

最適化およびチューニング

@PROCESS

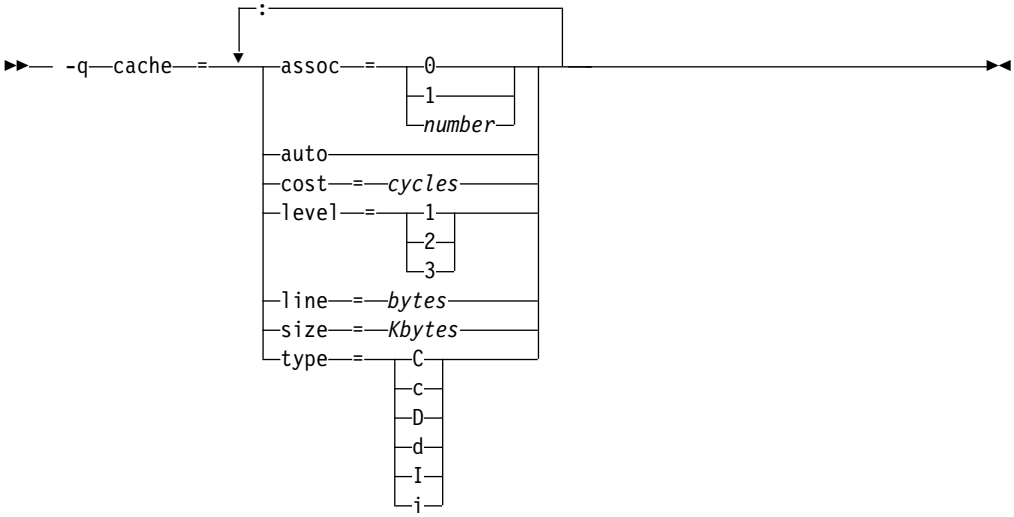
なし。

目的

特定の実行マシンに対して、キャッシュ構成を指定する。

コンパイラーはこの情報を使用して、特に、データ・キャッシュに適合するデータ量に限定して処理するように構造化 (あるいはブロック化) 可能なループ演算の場合に、プログラムのパフォーマンスを調整します。

構文



デフォルト

適用されません。

パラメーター

assoc=*number*

キャッシュのセット連想度を指定します。

0 直接マップされたキャッシュ

1 完全結合のキャッシュ

n **> 1** *n* 方式のセット連想キャッシュ

auto コンパイルを実行しているマシンの特定のキャッシュ構成を自動的に検出します。実行環境はコンパイル環境と同じであると見なされます。

cost=*cycles*
余分なキャッシュを脱落させる最適化を実行するかどうかをコンパイラーが判別できるように、キャッシュ・ミスの結果生じるパフォーマンス・ペナルティーを指定します。

level=*level*
どのレベルのキャッシュが影響を受けるかを指定します。

1 基本キャッシュ

2 マシンがレベル 2 のキャッシュを持っていない場合は、レベル 2 のキャッシュ、またはテーブル・ルックアサイド・バッファー (TLB)

3 レベル 2 のキャッシュを持っているマシンでは TLB

他のレベルも使用できますが、現在のところ未定義です。システムに複数のレベルのキャッシュがある場合は、別の **-qcache** オプションを使用して各レベルを説明してください。

line=*bytes*
キャッシュの行サイズを指定します。

size=*Kbytes*
このキャッシュの合計サイズを指定します。

type={C|c| D|d|I|i}
設定が適用されるキャッシュのタイプを指定します。
- 結合されているデータおよび命令キャッシュの場合、**C** または **c**
- データ・キャッシュの場合、**D** または **d**
- 命令キャッシュの場合、I または **i**

使用法

プログラムの実行場所となるシステムの種類が明確で、かつこのシステムの命令/データ・キャッシュの構成が、デフォルトの構成 (**-qtune** 設定による) とは異なる場合は、キャッシュの特性を正確に指定することにより、コンパイラーは、特定のキャッシュ関連最適化によって得られる利点をさらに正確に算出できるようになります。

-qcache オプションを有効にするには、**level** および **type** サブオプションを組み込み、**-qhot** オプションまたは **-qhot** を暗黙指定するオプションを指定する必要があります。

- すべてではないがいくつかの値が分かっている場合は、分かっている値を指定してください。
- システムに複数のレベルのキャッシュがある場合は、別の **-qcache** オプションを使用して各レベルを説明してください。このオプションでテストする時間が限定されている場合は、命令キャッシュ特性を指定するよりもデータ・キャッシュ特性を指定する方が重要です。
- ターゲット・システムの正確なキャッシュ・サイズがわからない場合は、比較的小さな推定値を使用してください。未使用のキャッシュ・メモリーがある方が、

ターゲット・システムのキャッシュ・サイズより大きな値を指定することによってキャッシュ・ミスあるいはページ不在が起きるよりも良い状態です。

キャッシュ構成に対して誤った値を指定したり、構成の異なるマシン上でプログラムを実行した場合は、プログラムの実行速度は遅くなりますが、正しく機能します。キャッシュ・サイズの正確な値がわからない場合は、無難な推定値を使用してください。

例

システムが、命令用とデータ・レベル 1 の結合キャッシュを持ち、キャッシュが双方向連結で、サイズが 8 KB で、64 バイトのキャッシュ・ラインを持つ場合にシステムのパフォーマンスを調整するには、次のようにします。

```
xlf95 -O3 -qhot -qcache=type=c:level=1:size=8:line=64:assoc=2 file.f
```

2 つのレベルのデータ・キャッシュを持つシステムのパフォーマンスを調整するには、次のように **-qcache** オプションを 2 つ使用します。

```
xlf95 -O3 -qhot -qcache=type=D:level=1:size=256:line=256:assoc=4 ¥
      -qcache=type=D:level=2:size=512:line=256:assoc=2 file.f
```

2 つのタイプのキャッシュを持つシステムのパフォーマンスを調整する場合も、次のように **-qcache** オプションを 2 つ使用します。

```
xlf95 -O3 -qhot -qcache=type=D:level=1:size=256:line=256:assoc=4 ¥
      -qcache=type=I:level=1:size=512:line=256:assoc=2 file.f
```

関連情報

- 113 ページの『-qarch』
- 170 ページの『-qhot』
- 292 ページの『-qtune』

-qcclines

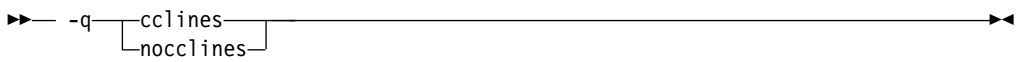
カテゴリー

入力制御

目的

コンパイラーが、固定ソース形式および F90 自由ソース形式の条件付きコンパイル行を認識するかどうかを判別する。このオプションは、IBM 自由ソース形式ではサポートされません。

構文



@PROCESS:

```
@PROCESS CCLINES | NOCCLINES
```

デフォルト

デフォルトは、**-qsmp=omp** オプションをオンにした場合は **-qcclines** です。オフにした場合、デフォルトは **-qnocclines** です。

関連情報

- 「*XL Fortran* ランゲージ・リファレンス」の『条件付きコンパイル』

-qcheck

カテゴリー

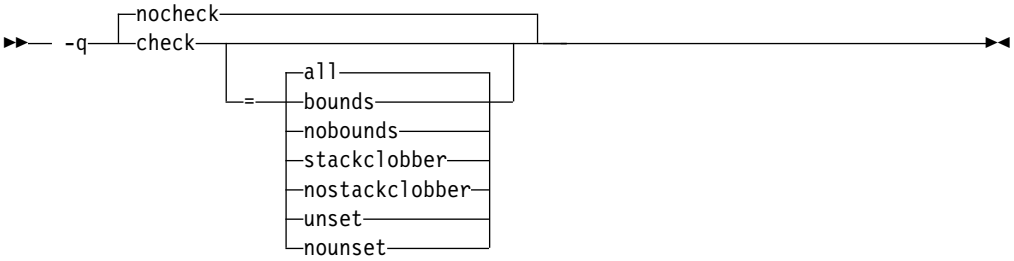
エラー・チェックおよびデバッグ

目的

特定タイプのランタイム検査を実行するコードを生成する。

-qcheck は、**-C** オプションの長い形式です。

構文



@PROCESS:

@PROCESS CHECK[(*suboptions*)] | **NOCHECK**

デフォルト

-qnocheck

パラメーター

all
すべてのサブオプションを使用可能にします。

bounds | nobounds
配列エレメント、配列セクション、文字サブストリングなどへの個々の参照をチェックし、定義されたエンティティーの境界内で参照が行われているかを確認します。

stackclobber | nostackclobber
ユーザー・プログラムの保存域内の不揮発性レジスターのスタック破損を検出します。このタイプの破損は、スタックの保存域内の不揮発性レジスターが変更されている場合にのみ発生します。

-qstackprotect オプションとこのサブオプションの両方がオンの場合、このサブオプションが最初にスタック破壊をキャッチします。

unset | nounset
使用する自動変数を、設定する前にチェックします。使用する前に自動変数が設定されていない場合は、トラップが実行時に行われます。

-qinitauto オプションは、自動変数を初期化します。その結果、**-qinitauto** オプションは、**-qcheck=unset** オプションから未初期化の変数を非表示にします。

-qsave オプションは、自動変数のストレージ・クラスを STATIC に変更します。その結果、**-qsave** オプションが指定されていると、設定される前に使用されている変数が **-qcheck=unset** オプションで検知されなくなります。

-qsigtrap オプションを指定すれば、そうした変数の使用につながるプロシージャー呼び出しを示すトレースバックを取得できます。**-g** オプションも指定した場合は、そのトレースバックには行番号とプロシージャー名も含まれます。

サブオプションなしで **-qcheck** オプションを指定することは、**-qcheck=all** を指定することと同等です。

使用法

-qcheck オプションは、複数回指定することができます。サブオプションの設定は累積されますが、後の方サブオプションによって前の方サブオプションがオーバーライドされます。

all サブオプションは、フィルターとして他の 1 つ以上のオプションの **no...** 形式と共に使用することができます。例えば、次のコマンドは、境界を越える参照を除くすべてのものを検査します。

```
xlf myprogram.f -qcheck=all:nobounds
```

no... 形式のサブオプションとともに **all** を使用する場合は、 **all** は通常最初のサブオプションでなければなりません。

コンパイル時に、参照が境界外に及ぶことをコンパイラーが判別できる場合、**-qcheck=bounds** が使用可能になっていると、報告されるエラーの重大度が **S** (重大) に上がります。

実行時に、参照が境界を越える場合、または特定のタイプのスタック破壊が検出される場合、プログラムは **SIGTRAP** シグナルを生成します。デフォルトでは、このシグナルはプログラムを停止させて、コア・ダンプを作成します。これは予想どおりの動作であり、コンパイラー製品に欠陥があることを示すものではありません。

実行時検査を行うと実行速度が遅くなることがあるので、個々のプログラムにとって重要な要因は何か (パフォーマンスへの影響、または、エラーが検出されない場合に間違った結果が出る可能性など) をユーザー側で判別する必要があります。このオプションを使用するのは、プログラムのテスト中またはデバッグ中のみにする (パフォーマンスがより重要な場合) か、あるいは、プロダクション・バージョンをコンパイルするときだけ (安全性がより重要な場合) にしてください。

-qcheck オプションは、いくつかの最適化を回避します。コードのデバッグが完了した後に、**-qcheck** オプションを削除してから、パフォーマンスを高めるために最適化オプションを適宜追加することをお勧めします。

文字サブストリング式の有効な境界は、**-qzerosize** オプションの設定によって異なります。

例

以下のコード例は、**-qcheck=bounds** によって検出された、割り振り可能な配列の境界外に及ぶアクセスを示しています。

```
program outofbounds
  real, allocatable :: x(:)
  allocate(x(5:10))
  x(1) = 3.0  ! Out of bound access.
  print *, x
end
```

以下の例では、整数ポインターのランダム割り当てにより、呼び出しスタックの保存域が破壊されます。**-qcheck=stackclobber** がこの問題を検出し、トラップを発生させます。

```
subroutine update(i, off, value)
  implicit none
  integer, value :: i, off
  character, value :: value
  integer pointee
  pointer(p, pointee)
  p = i + off
  pointee = ichar(value)
end subroutine

subroutine sub1()
  implicit none
  interface
    subroutine update(i, off, value)
      integer, value :: i, off
      character, value :: value
    end subroutine
  end interface
  character(9) buffer
  buffer = ""
  ! This call to update will corrupt the call stack.
  ! Offset 48 is within the save area for this program.
  call update(loc(buffer), 48, 'a')
  print *, buffer
end subroutine

program main
  call sub1
end program
```

注: 呼び出しスタックの保存域のオフセットは、プログラムのソースおよび最適化レベルによって異なります。-O2 以下の最適化レベルが有効な場合、オフセット 48 は保存域内に収まります。

次の関数 `factorial` では、`n<=1` の場合は `temp` が初期化されず、`n>1` の場合は `result` が設定前にアクセスされます。`factorial.f` をコンパイルするときに、実行時のこうした問題および原因となるトラップが検出されるようにするには、次のコマンドを入力します。

```
xlf95 -qcheck=unset -O -g -qsigtrap factorial.f
```

factorial.f には、次のコードが入っています。

```
module m
contains
  recursive function factorial(n) result(result)
    integer, value :: n
    integer result, temp

    if (n > 1) then
      temp = n * factorial(n - 1)
      print *, result ! line 9
    endif

    result = temp ! line 12
  end function
end module

use m
integer x
x = factorial(1)
end
```

コンパイラーは、次の通知メッセージを発行します。実行時にトラップが行 12 と行 18 の付近で発生しています。

```
1500-098 (I) "factorial.f", line 9: "result" is used before it is set.
1500-098 (I) "factorial.f", line 12: "temp" might be used before it is set.
1501-510  Compilation successful for file factorial.f.
$ ./a.out

  Signal received: SIGTRAP - Trace trap
    Fortran language trap: reference to unset memory location

  Traceback:
    Offset 0x00000014 in procedure __m_NMOD_factorial, near line 12 in file factorial.f
    Offset 0x00000028 in procedure _main, near line 18 in file factorial.f
    --- End of call chain ---
```

注**: noopt** で **-qcheck=unset** を設定すると、コンパイラーはコンパイル時に通知メッセージを発行しません。

関連情報

- 269 ページの『-qstackprotect』
- 170 ページの『-qhot』
- 316 ページの『-qzerosize』
- 257 ページの『-qsigtrap』および「*XL Fortran* 最適化およびプログラミング・ガイド」の『例外ハンドラーのインストール』 では、**SIGTRAP** シグナルの検出とその状態からの回復を、プログラムを終了させずに行う方法について説明しています。

-qci

カテゴリー

入力制御

目的

処理する **INCLUDE** 行の識別番号 (1 から 255) を指定する。

構文

@PROCESS:

@PROCESS CI(*number*,...,*number*)

デフォルト

適用されません。

使用法

このオプションを使用すると、一種の条件付きコンパイルができます。あまり使用しないコード (例えば **WRITE** 文のデバッグ、追加のエラー・チェック・コード、XLF 固有のコード) を別のファイルに入れて、それらを処理するかどうかを個々のコンパイルに対して決定することができるからです。

INCLUDE 行の終わりに数字が入っている場合は、 **-qci** オプションでその番号が指定されている場合のみ、そのファイルが含まれます。認識される識別番号のセットは、**-qci** オプションのすべてのオカレンスに対して指定されているすべての識別番号の集合です。

注**:**

1. **INCLUDE** 行の任意の数字は広く行き渡っている XL Fortran 機能ではないので、それを使用すると、プログラムの移植性が制限される場合があります。
2. このオプションは、#include C プリプロセッサー・ディレクティブではなく、XL Fortran INCLUDE ディレクティブによってのみ作動します。

例

```
REAL X /1.0/
INCLUDE 'print_all_variables.f' 1
X = 2.5
INCLUDE 'print_all_variables.f' 1
INCLUDE 'test_value_of_x.f' 2
END
```

この例では、**-qci** オプションを指定しないでコンパイルすると、単に **X** が宣言されて、それに値が割り当てられます。 **-qci=1** を指定してコンパイルすると、インクルード・ファイルの 2 つのインスタンスが含まれ、 **-qci=1:2** を指定してコンパイルすると、両方のインクルード・ファイルが含まれます。

関連情報

- 「*XL Fortran* ランゲージ・リファレンス」の『**INCLUDE**』 ディレクティブ

-qcompact

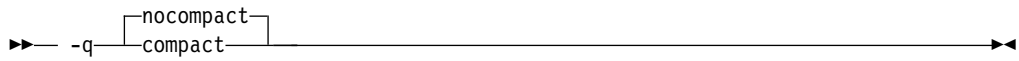
カテゴリー

最適化およびチューニング

目的

コード・サイズを増やす最適化を回避する。

構文



@PROCESS:

@PROCESS COMPACT | **NOCOMPACT**

デフォルト

-qnocompact

使用法

デフォルトでは、ループ・アンロールおよび配列ベクトル化など、パフォーマンスの改善のために最適化プログラムが使用する手法によって、プログラムが大きくなってしまう場合があります。ストレージが限られているシステムの場合は、**-qcompact** を使用して、発生する拡張を少なくすることができます。プログラムに多数のループや配列言語構文がある場合、**-qcompact** オプションを使用すると、アプリケーション全体のパフォーマンスに影響が出ます。このオプションの使用を、最適化による影響が出ないプログラム部分だけに制限することができます。

規則

-qcompact を有効にしても、その他の最適化オプションは依然として機能しています。コード・サイズは、最適化中に自動的に行われるコードの複製を制限することで縮小されます。

このオプションが有効になるのは、**-O2** 以上の最適化レベルで指定された場合のみです。

-qcr

カテゴリー

入力制御

@PROCESS

なし。

目的

コンパイラーが CR (復帰) 文字を解釈する方法を制御する。

このオプションにより、Mac OS または DOS/Windows エディターを使用して作成したコードをコンパイルできます。

構文

デフォルト

デフォルトでは、CR (16 進値 X'0d') または LF (16 進値 X'0a') 文字、あるいは CRLF (16 進値 X'0d0a') の組み合わせは、ソース・ファイルでの行の終了を示します。

使用法

-qnocr を指定した場合、コンパイラーは LF 文字のみを行の終了文字として認識します。 CR 文字を行の終了以外の目的に使用する場合は、**-qnocr** を指定しなければなりません。

-qctyplss

カテゴリー

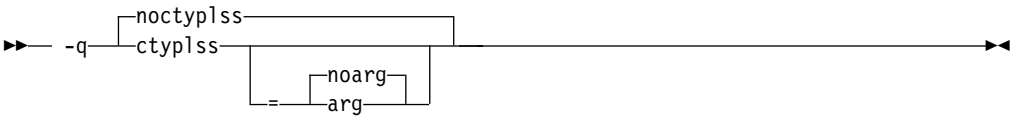
移植性とマイグレーション

目的

型なし定数が使用される可能性がある場合に必ず、文字定数式を使用できるかどうかを指定する。

この言語拡張機能は、プログラムを他のプラットフォームから移植するときに必要になる場合があります。

構文



@PROCESS:

@PROCESS CTYPLSS[([NO]ARG)]| **NOCTYPLSS**

デフォルト

-qnoctyplss

パラメーター

arg | noarg
-qctyplss の動作を保存するサブオプション。さらに **arg** は、実引数として使用されるホレリス定数が、整数の実引数として扱われることを指定します。

使用法

-qctyplss を指定すると、文字定数式はホレリス定数であるかのように扱われ、したがって論理式および演算式で使用することができます。

- **-qctyplss** オプションを指定して、引数リストのキーワード **%VAL** とともに文字定数式を使用すると、ホレリス定数と文字定数との区別ができます。文字定数はレジスターの右端バイトに置かれて左はゼロで埋め込まれます。一方、ホレリス定数は、左端バイトに置かれて右はブランクで埋め込まれます。その他のすべての **%VAL** 規則が適用されます。
- このオプションは、定数配列または定数配列のサブオブジェクトに関連のある文字式にはいかなる点でも適用されません。

例

例 **1 :** 次の例では、コンパイラー・オプション **-qctyplss** を指定すると、文字定数式を使用することができます。

```
@PROCESS CTYPLSS
      INTEGER I,J
      INTEGER, PARAMETER :: K(1) = (/97/)
      CHARACTER, PARAMETER :: C(1) = (/'A'/)

      I = 4HABCD          ! Hollerith constant
      J = 'ABCD'          ! I and J have the same bit representation

! These calls are to routines in other languages.
      CALL SUB(%VAL('A')) ! Equivalent to CALL SUB(97)
      CALL SUB(%VAL(1HA)) ! Equivalent to CALL SUB(1627389952)

! These statements are not allowed because of the constant-array
! restriction.
!     I = C // C
!     I = C(1)
!     I = CHAR(K(1))
      END
```

例 **2:** 次の例では、変数 *J* は、参照用に渡されます。サブオプション **arg** は、ホレリス定数が整数の実引数であるかのように渡されることを指定します。

```
@PROCESS CTYPLSS(ARG)
      INTEGER :: J

      J = 3HIBM
! These calls are to routines in other languages.
      CALL SUB(J)
      CALL SUB(3HIBM)   ! The Hollerith constant is passed as if
                        ! it were an integer actual argument
```

関連情報

- 「*XL Fortran* ランゲージ・リファレンス」の『ホレリス定数』
- 「*XL Fortran* 最適化およびプログラミング・ガイド」の『参照または値による引数の引き渡し』

-qcuda (CUDA Fortran)

カテゴリー

言語エレメント制御

@PROCESS

なし。

目的

CUDA Fortran に対するコンパイラー・サポートを使用可能にする。

構文

デフォルト

- **-qcuda** は **xlcuf** 呼び出しコマンドのデフォルト設定です。 **-qnocuda** は他の呼び出しコマンドのデフォルト設定です。
- コンパイラーの呼び出しに **xlf_r** を使用する場合は、**-qcuda** が **.cuf** ファイルと **.CUF** ファイルのデフォルト設定となり、**-qnocuda** が他のファイル拡張子のデフォルト設定となります。

使用法

-qcuda は、_r 接尾部を持つ呼び出しコマンドと一緒の場合にのみ指定できます。そのようなコマンドでは必ずスレッド・セーフ・コンパイルが行われます。

-qcuda は、最適化レベルが **-O2**、**-O3**、および **-Ofast** の場合に使用できます。

-qcuda は **-qsmp=omp** と一緒に使用できますが、それ以外のいずれの **-qsmp** サブオプションとも一緒には使用できません。 **-qsmp=omp** が指定されている場合は、ホスト・コードにのみ OpenMP 並列化を含めることができます。

-qcuda は、**-O4**、**-O5**、**-qipa**、**-qpdf1**、**-qpdf2**、**-qrealsize=8** のいずれとも一緒には使用できません。

例

CUDA Fortran プログラムのコンパイルには、以下のいずれのコマンドも使用できます。

```
xlf_r mycudaprogram1.cuf
xlf95_r -qcuda mycudaprogram2.f
xlcuf mycudaprogram3.f
xlcuf mycudaprogram4.cuf
```

関連情報

- *XL Fortran* を使用した *CUDA Fortran* プログラミングの概要

-qcudaerr (CUDA Fortran)

カテゴリー

エラー・チェックおよびデバッグ

目的

コンパイラーが自動的に CUDA API 呼び出しのエラー・チェック・コードを挿入するかどうかを制御します。

構文

オプション:

@PROCESS:

```
@PROCESS CUDAERR(STMTS | ALL | NONE)
```

デフォルト

-qcudaerr=stmts

パラメーター

stmts これは、以下の項目に対してエラー検査コードを挿入するようにコンパイラーに指示します。

- ALLOCATE、CALL、代入などの Fortran ステートメントで行われる CUDA ランタイム API 呼び出し
- 同期点 (cudaDeviceSynchronize API 関数に対する呼び出しなど)

all これは、すべての CUDA ランタイム API 呼び出し (自分のプログラムにおける明示的な呼び出しを含む) に対してエラー検査コードを挿入するようにコンパイラーに指示します。

none これは、いずれの CUDA ランタイム API 呼び出しに対してもエラー検査コードを挿入しないようにコンパイラーに指示します。

使用法

挿入されたエラー検査コードで、失敗した CUDA ランタイム API 呼び出しが検出されると、コンパイラーはエラー・メッセージを発行します。このエラー・メッセージには以下の情報が含まれています。

- 失敗した CUDA ランタイム API 関数の名前
- エラー・コード
- `cudaGetErrorString` API 関数によって返される説明

挿入されたエラー検査コードによって、検出された CUDA エラーはクリアされません。 **-qcudaerr** オプションはユーザー作成のエラー検査コードに影響しません。

CUDA ランタイム API エラーが検出されたときにコンパイラーにプログラムを終了させるには、以下のいずれかの方法でランタイム・エラー・リカバリーを無効にします。

- `setrteopts` サブルーチンを、`err_recovery=no` を指定して呼び出します。
- XLFRTEOPTS 環境変数を次のように設定します。

```
export XLFRTEOPTS=err_recovery=no
```

例

サンプル・プログラム out_of_bounds.cuf は次のようになっています。

```
INTEGER, DEVICE, ALLOCATABLE :: device_arr(:)
INTEGER :: host_arr(10)
ALLOCATE(device_arr(5))
device_arr = host_arr
END
```

これを、**xlcuf** を指定してコンパイルし、実行可能ファイルを実行します。コンパイラーからは次のメッセージが発行されます。

```
"out_of_bounds.cuf", line 4: 1525-244 API function cudaMemcpy
failed with error code 11:  invalid argument.
```

out_of_bounds.cuf に、ホスト配列と非標準化デバイス配列との間の代入が含まれているため、その代入は cudaMemcpy API 呼び出しに変換されます。しかし、device_arr が小さすぎることが CUDA ランタイムで検出されるため、この API は失敗します。 **-qcudaerr=stmts** がデフォルトで有効になっているため、コンパイラーは cudaMemcpy の結果を検査し、CUDA ランタイム API エラーのコードと説明が含まれたメッセージを表示します。

関連情報

- *XL Fortran* を使用した *CUDA Fortran* プログラミングの概要

-qdbg

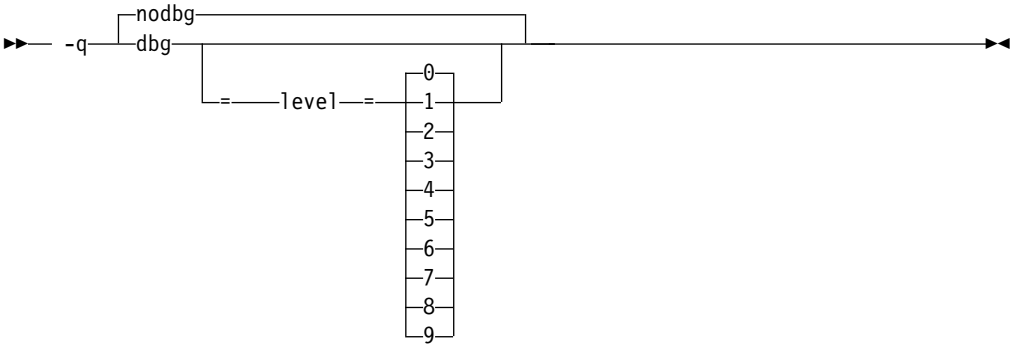
カテゴリー

エラー・チェックおよびデバッグ

目的

-qdbg は **-g** の長い形式です。

構文



@PROCESS:

@PROCESS DBG | **NODBG**

デフォルト

-qnodbg、または **-qdbg=level=0**

パラメーター

-qdbg=level=0
-qnodbg または **-g0** と同等です。

-qdbg=level=1
-g1 または **-qlinedebug** と同等です。

-qdbg=level=2
-g2 と同等です。

-qdbg=level=3
-g3 と同等です。

-qdbg=level=4
-g4 と同等です。

-qdbg=level=5
-g5 と同等です。

-qdbg=level=6
-g6 と同等です。

-qdbg=level=7
-g7 と同等です。

-qdbg=level=8
-g8 と同等です。

-qdbg=level=9
-g9 と同等です。

関連情報

- 90 ページの『-g』
- 203 ページの『-qlinedebug』

-qddim

カテゴリー

移植性とマイグレーション

目的

ポインティング先配列の境界を、その配列が参照されるたびに再評価することを指定し、ポインティング先配列の境界式に対するいくつかの制限を除去する。

構文

@PROCESS:

@PROCESS DDIM | **NODDIM**

デフォルト

-qnoddim

使用法

デフォルト時には、ポインティング先配列のみが変数名を含む次元宣言子を持つことができ (その配列がサブプログラム内にある場合)、次元宣言子の変数は仮引数、共通ブロックのメンバー、使用関連付けまたはホスト関連付けでなければなりません。次元のサイズは、サブプログラムに対する入り口で計算され、サブプログラムの実行中は一定の状態に保たれます。

-qddim オプションでは、次のとおりです。

- ポインティング先が参照されるたびに、そのポインティング先配列の境界が再評価されます。このプロセスは動的次元設定 と呼ばれています。宣言子内の変数は配列が参照されるたびに評価されるので、変数の値を変更すると、ポインティング先配列のサイズも変更されます。
- 配列宣言子内に存在できる変数に関する制限は取り除かれます。したがって、通常のローカル変数をこれらの式で使用することができます。
- メインプログラム内のポインティング先配列は、その配列宣言子の中に変数を持つこともできます。

例

```
@PROCESS DDIM
INTEGER PTE, N, ARRAY(10)
POINTER (P, PTE(N))
DO I=1, 10
   ARRAY(I)=I
END DO
N = 5
P = LOC(ARRAY(2))
PRINT *, PTE   ! Print elements 2 through 6.
N = 7          ! Increase the size.
PRINT *, PTE   ! Print elements 2 through 8.
END
```

-qdescriptor

カテゴリー

移植性とマイグレーション

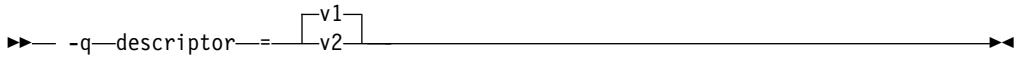
@PROCESS

なし。

目的

コンパイルされたアプリケーションでオブジェクト指向でないエンティティー用に使用する XL Fortran 内部記述子データ構造フォーマットを指定する。

構文



デフォルト

- **-qdescriptor=v1**

パラメーター

v1 内部記述子のデータ構造フォーマットを使用します。これはコンパクトですが、Fortran 言語の新機能の一部 (オブジェクト指向など) を表現できません。

v2 拡張可能な、内部記述子のデータ構造フォーマットを使用します。この設定によって、プログラムが Fortran のオブジェクト指向機能を、パラメータ化された派生型と同様に利用できるようになります。

使用法

実行中の **-qdescriptor** 設定にかかわらず、オブジェクト指向の構造体またはパラメータ化された派生型を含むアプリケーションは、その構造体用の **v2** データ構造フォーマットを使用します。

-qdescriptor 設定の選択は、配布用のライブラリーまたはモジュールのビルド時における重要な考慮事項です。これらのライブラリーとモジュールのユーザーは、**-qdescriptor** 設定を認識し、互換可能な方法でその設定を使用するコードをコンパイルする必要があります。オブジェクト自体が、ユーザーが読み取り可能な書式でコンパイル・オプションをエンコードするように、そのようなライブラリーおよびモジュールを **-qsaveopt** オプションでビルドすることをお勧めします。

ユーザー可視の派生型を含むモジュールをビルドする場合、**-qxlf2003=polymorphic** サブオプションでのビルドを検討してください。これにより、モジュールのユーザーは、ポリモアフィズムを使用する Fortran オブジェクト指向コンテキストで派生型を使用または拡張することができます。

Fortran 2003 のオブジェクト指向のプログラミング・モデルでは、XL Fortran コンパイラーが、**-qxlf2003=polymorphic** を指定してコンパイルされなかったモジュールに定義された型から、型や型拡張の使用をサポートしています。これは、型がポリモアフィズムを必要とするコンテキストで使用されない場合に限ります。ただし、ポリモアフィズムを必要とするコンテキストで **-qxlf2003=polymorphic** を指定してコンパイルされなかったモジュールから、型や型拡張を使用しようとするのを、コンパイラーが検出した場合、エラー・メッセージが発行され、コンパイルが停止します。

-qdescriptor=v1 設定を指定してビルドされたモジュール、**-qdescriptor=v2** が指定されたコンパイルで使用された場合、コンパイラーは、この不一致を診断し、エラー・メッセージを発行後コンパイルを停止します。

-qdescriptor=v2 オプションを使用するとき、コンパイラーは、**v2** 設定を指定してビルドされたオブジェクトが、**v1** 設定で、あるいは、XL Fortran 10.1 以前のコンパイラーでビルドされたモジュールと混合されるという、安全でない使用法を診断

することはできません。ご使用のプログラムが正しく機能しているようにみえても、この使用法はサポートされていません。記述子のフォーマットは、特定の構成体で使用されても異なるサイズになり、データ・レイアウトは、未定義のサポートされていない動作に変化してしまいます。例えば、派生型内の割り当て可能なエンティティーおよびポインター・エンティティーのサイズは、派生型自身が異なるサイズとなるので異なってしまいます。

関連情報

- 253 ページの『-qsaveopt』
- 308 ページの『-qxlf2003』

-qdirective

カテゴリー

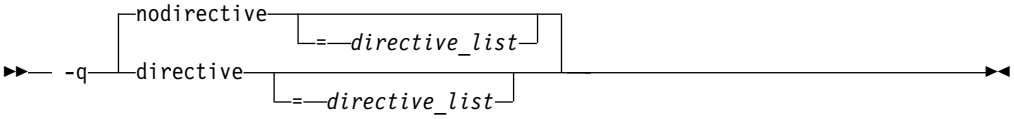
入力制御

目的

コメント行をコンパイラーのコメント・ディレクティブとして識別する、トリガー定数と呼ばれる文字のシーケンスを指定する。

コンパイラーのコメント・ディレクティブは、Fortran ステートメントではない行ですが、コンパイラーがそのように認識し、それに従って動作します。

フォーマット



@PROCESS:

@PROCESS DIRECTIVE[(*directive_list*)] | **NODIRECTIVE**[(*directive_list*)]

デフォルト

コンパイラーは、デフォルト時にはトリガー定数 **IBM*** を認識します。

-qsmp オプションは **-qdirective=SMP¥$:¥$OMP:IBMP** を暗黙指定します。これにより、コンパイラーはトリガー定数 **SMP$**、**$OMP**、および **IBMP** も認識できるようになります。

-qthreaded オプションは **-qdirective=ibmt** を暗黙指定します。これにより、コンパイラーは **IBMT** トリガー定数も認識できるようになります。

パラメーター

directive_list を持たない **-qnodirective** オプションは、以前に指定したディレクティブ識別子をすべてオフにします。 *directive_list* を持っている場合は、選択された識別子だけをオフにします。

directive_list を持たない **-qdirective** は以前の **-qnodirective** によってオフにされている場合でも、デフォルトのトリガー定数 **IBM*** をオンにします。

使用法

次のことに注意してください。

- 複数の **-qdirective** および **-qnodirective** オプションは付加オプションです。つまり、ディレクティブ識別子を複数回オンにしたり、オフにしたりできます。
- 1 つ以上の *directive_list* は、特定のファイルまたはコンパイル単位に適用することができます。 *directive_list* 内のいずれかのストリングで始まっているコメント行は、コンパイラーのコメント・ディレクティブであると見なされます。
- トリガー定数は、大文字と小文字の区別をしません。
- 文字 **(**、**)**、**'**、**"**、**:**、**=**、コンマ、ブランクは、トリガー定数の一部にすることはできません。
- これらのオプションとともに使用するトリガー定数内のワイルドカードの拡張を回避するために、コマンド行上で一重引用符で囲むことができます。例えば、次のようになります。

  ```
  xlf95 -O3 -qhot -qcache=type=D:level=1 -qdirective='dbg*' -qnodirective='IBM*' directives.f
  ```

- このオプションは、XL Fortran コンパイラーによって出されたディレクティブにのみ影響を与え、プリプロセッサーによって出されたディレクティブには影響しません。
- 誤ったトリガー定数を使用すると、警告メッセージまたはエラー・メッセージ、あるいはその両方が生成されることがあります。適切な関連するトリガー定数については、特定のディレクティブ・ステートメントを確認してください。

例

```
! This program is written in Fortran free source form.
PROGRAM DIRECTV
INTEGER A, B, C, D, E, F
A = 1 ! Begin in free source form.
B = 2
!OLDSTYLE SOURCEFORM(FIXED)
! Switch to fixed source form for this include file.
      INCLUDE 'set_c_and_d.inc'
!IBM* SOURCEFORM(FREE)
! Switch back to free source form.
E = 5
F = 6
END
```

この例の場合は、**-qdirective=oldstyle** オプションを指定してコンパイルし、**INCLUDE** 行の前の **SOURCEFORM** ディレクティブをコンパイラーが必ず認識するようにします。インクルード・ファイル行を処理すると、**SOURCEFORM(FREE)** 文の後は、プログラムは自由ソース形式に戻ります。

- 「*XL Fortran* ランゲージ・リファレンス」の 『**SOURCEFORM**』 ディレクティブ
- 「*XL Fortran* ランゲージ・リファレンス」の『ディレクティブ 』セクション

関連資料:

262 ページの『-qsmp』

290 ページの『-qthreaded』

-qdirectstorage

カテゴリー

最適化およびチューニング

@PROCESS

なし。

コンテキスト

なし。

目的

特定のコンパイル単位がライトスルーを使用可能にしたストレージまたはキャッシュ禁止ストレージを参照する可能性があることをコンパイラーに通知する。

フォーマット

デフォルト

-qnodirectstorage

使用法

このオプションは慎重に使用してください。メモリーとキャッシュ・ブロックの作業に精通し、最適なパフォーマンスを得るためにアプリケーションをチューニングすることができるプログラマーを対象としています。すべての Power のキャッシュ編成のインプリメンテーションでプログラムが正しく実行されるためには、プログラマーは、命令キャッシュとデータ・キャッシュが別々に存在することを想定し、その別個のキャッシュ・モデルに対してプログラムを作成する必要があります。

注**: -qdirectstorage** オプションを **CACHE_ZERO** ディレクティブとともに使用すると、プログラムに障害が生じるか、または間違った結果が生成される可能性があります。

関連情報

- 「*XL Fortran* ランゲージ・リファレンス」の CACHE_ZERO。

-qdlines

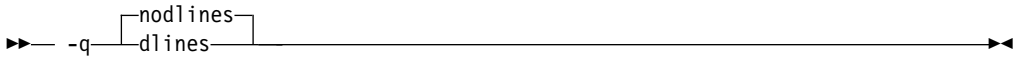
カテゴリー

入力制御

目的

コンパイラーが、列 1 に D がある固定ソース形式の行をコンパイルするか、またはそれをコメントとして扱うかを指定する。

フォーマット



@PROCESS:

@PROCESS DLINES | **NODLINES**

デフォルト

-qnodlines

使用法

-qdlines を指定すると、桁 1 に **D** がある固定ソース形式行がコンパイルされます。デフォルトの動作は、これらの行をコメント行と見なして処理します。これらは通常、オン/オフにする必要のあるデバッグ・コードの部分に使用されます。

-qdpc

カテゴリー

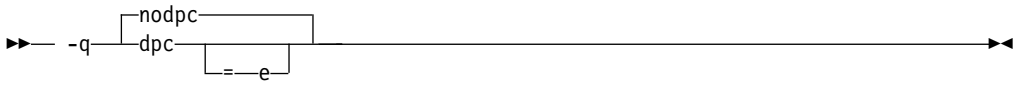
浮動小数点および整数のコントロール

目的

実定数を **DOUBLE PRECISION** 変数に割り当てるときに最大限の正確性を実現できるよう、実定数の精度を高める。

この言語拡張機能は、プログラムを他のプラットフォームから移植するときに必要になる場合があります。

フォーマット



@PROCESS:

@PROCESS DPC[(E)] | **NODPC**

デフォルト

-qnodpc

使用法

-qdpc を指定すると、すべての基本実定数 (例えば 1.1) が倍精度定数として処理されます。コンパイラーは、これを指定しないと **DOUBLE PRECISION** 変数への割り当て中に失われてしまう精度を持つ数字を保存します。 **-qdpc=e** を指定すると、指数 e を持つ定数も含め、すべての単精度定数が倍精度定数として処理されます。

このオプションは、kind 型パラメーターが指定されている定数には影響を与えません。

-qautodbl と **-qrealsize** は、さらに汎用的なオプションで、**-qdpc** が実行することも実行できます。これらのオプションのいずれかが指定されている場合は、**-qdpc** は効力を持ちません。

例

```
@process nodpc
        subroutine nodpc
        real x
        double precision y
        data x /1.000000000001/  ! The trailing digit is lost
        data y /1.000000000001/  ! The trailing digit is lost

        print *, x, y, x .eq. y  ! So x is considered equal to y
        end

@process dpc
        subroutine dpc
        real x
        double precision y
        data x /1.000000000001/  ! The trailing digit is lost
        data y /1.000000000001/  ! The trailing digit is preserved

        print *, x, y, x .eq. y  ! So x and y are considered different
        end

        program testdpc
        call nodpc
        call dpc
        end
```

コンパイルされると、このプログラムは次のように印刷して、

```
   1.000000000       1.00000000000000000     T
   1.000000000       1.00000000000100009     F
```

-qdpc によって余分な精度が保持されていることを示します。

- 117 ページの『-qautodbl』
- 245 ページの『-qrealsize』

-qenum

カテゴリー

浮動小数点および整数のコントロール

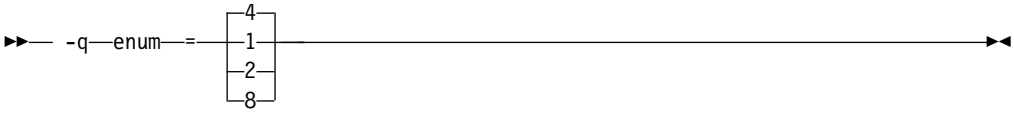
@PROCESS

なし。

目的

列挙子定数の範囲を指定し、ストレージ・サイズを決定できるようにする。

構文



デフォルト

-qenum=4

使用法

ストレージ・サイズに関わりなく、列挙型定数の値は *value* に対応する範囲によって制限されます。列挙型定数の値が指定された範囲を越える場合、警告メッセージが表示され、必要に応じて切り捨てが実行されます。

各 *value* に対応する範囲の限度および kind 型パラメーターは次のとおりです。

表 *18.* 列挙型定数のサイズおよびタイプ

値	列挙型定数値の有効な範囲	**kind** 型 パラメーター
1	-128 から 127 まで	4
2	-32768 から 32767 まで	4
4	-2147483648 から 2147483647 まで	4
8	-9223372036854775808 から 9223372036854775807 まで	8

関連情報

- 「*XL Fortran* ランゲージ・リファレンス」の『*ENUM/END ENUM*』ステートメント

-qescape

カテゴリー

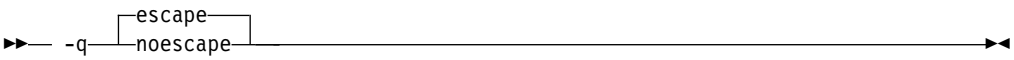
移植性とマイグレーション

目的

文字ストリング、ホレリス定数、H 編集記述子、および文字ストリング編集記述子におけるバックスラッシュの取り扱い方法を指定する。

円記号は、エスケープ文字または円記号文字として扱うことができます。この言語拡張機能は、プログラムを他のプラットフォームから移植するときに必要になる場合があります。

構文



@PROCESS:

@PROCESS **ESCAPE** | NOESCAPE

デフォルト

-qescape

使用法

-qescape を指定すると、円記号はこれらのコンテキスト内のエスケープ文字であると解釈されます。**-qnoescape** を指定した場合、円記号は円記号文字として扱われます。

デフォルト設定は、次のようなことを行う場合に便利です。

- エスケープ文字として円記号を使用する別の Fortran コンパイラーからコードを移植する。
- 「特殊な」文字、例えば、タブ文字または改行文字を文字データに入れる。このオプションを使用しない場合、代わりにプログラム内で直接 ASCII 値 (またはメインフレーム・システム上で EBCDIC 値) をエンコードするため、移植が一層困難になります。

変更されないままで渡される円記号文字に依存するコードを書いたり移植したりする場合は、**-qnoescape** を指定して、特殊な解釈が行われないようにします。また、デフォルト設定下の単一の円記号文字を表すのに、**¥¥** を書くこともできます。

例

```
$ # Demonstrate how backslashes can affect the output
$ cat escape.f
       PRINT *,'a¥bcde¥fg'
       END
$ xlf95 escape.f
** _main === End of Compilation 1 ===
1501-510  Compilation successful for file escape.f.
$ a.out
cde
    g
$ xlf95 -qnoescape escape.f
** _main === End of Compilation 1 ===
1501-510  Compilation successful for file escape.f.
 $ a.out
  a¥bcde¥fg
```

デフォルト設定 **-qescape** による最初のコンパイルで、バックスペース文字として **¥b** が印刷され、用紙送り文字として **¥f** が印刷されます。

-qnoescape オプションを指定すると、他の文字と同じように円記号が印刷されます。

関連情報

XL Fortran が認識するエスケープ・シーケンスのリストは、「*XL Fortran* 最適化およびプログラミング・ガイド」の『文字ストリングのエスケープ・シーケンス』に記載されています。

-qessl

カテゴリー

最適化およびチューニング

@PROCESS

なし。

目的

コンパイラーが、Engineering and Scientific Subroutine Library (ESSL) ルーチンを Fortran 90 組み込みプロシージャーに代えて使用できるようにする。

ESSL は、サブルーチンの集まりで、各種科学技術計算アプリケーション用に幅広い数学関数を提供します。これらのサブルーチンでは、特定のアーキテクチャーでパフォーマンス調整が行われます。 Fortran 90 組み込みプロシージャーの中には ESSL と類似のものがあります。これらの Fortran 90 組み込みプロシージャーを ESSL とリンクするとパフォーマンスが向上します。この場合、Fortran 90 組み込みプロシージャーのインターフェースを保持することができ、 ESSL を使用してパフォーマンスを向上させる追加の利点を得ることができます。

構文

デフォルト

-qnoessl

使用法

-lessl でリンクするときは、ESSL シリアル・ライブラリーを使用します。
-lesslsmp でリンクするときは、ESSL SMP ライブラリーを使用します。

-qessl でコードをコンパイルするときは常に、**-lessl** または **-lesslsmp** を使用する必要があります。

また、**libessl.so** および **libesslsmp.so** は、**libxlf90_r.so** に依存しているため、**libxlf90_r.so** をリンクのデフォルトとして使用する **xlf_r**、**xlf90_r**、または **xlf95_r** でコンパイルします。リンクするために直接リンカーを使用するか、他のコマンドを使用する場合、リンク・コマンド行で **-lxlf90_r** を指定することもできます。

次の MATMUL 関数呼び出しでは、**-qessl** を使用可能にすると、ESSL ルーチンを使用することができます。

```
real a(10,10), b(10,10), c(10,10)
  c=MATMUL(a,b)
```

関連情報

ESSL ライブラリーは、XL Fortran コンパイラーと一緒に出荷されることはありません。これらのライブラリーについて詳しくは、Engineering and Scientific Subroutine Library (ESSL) and Parallel ESSL の Web ページを参照してください。

-qextern

カテゴリー

移植性とマイグレーション

@PROCESS

なし。

目的

XL Fortran 組み込み関数の代わりにユーザー作成のプロシージャーを呼び出せるようにする。

構文

デフォルト

適用されません。

パラメーター

names
プロシージャー名をコロンで区切ったリストです。

使用法

プロシージャー名は、コンパイル中の個々のコンパイル単位の **EXTERNAL** 文内にあるかのように扱われます。プロシージャー名が XL Fortran 組み込みプロシージャーと競合する場合は、このオプションを使用して組み込みプロシージャーの代わりにソース・コード内のプロシージャーを呼び出します。

Fortran 90 および Fortran 95 は組み込み関数およびサブルーチンを多数持っているので、 FORTRAN 77 プログラムではこのオプションが必要なかった場合でも、このオプションを使用しなければならない場合があります。

例

```
subroutine matmul(res, aa, bb, ext)
  implicit none
  integer ext, i, j, k
  real aa(ext, ext), bb(ext, ext), res(ext, ext), temp
  do i = 1, ext
    do j = 1, ext
      temp = 0
      do k = 1, ext
        temp = temp + aa(i, k) * bb(k, j)
      end do
      res(i, j) = temp
    end do
  end do
end subroutine

implicit none
integer i, j, irand
integer, parameter :: ext = 100
real ma(ext, ext), mb(ext, ext), res(ext, ext)

do i = 1, ext
  do j = 1, ext
    ma(i, j) = float(irand())
    mb(i, j) = float(irand())
  end do
end do

call matmul(res, ma, mb, ext)
end
```

オプションを指定しないでこのプログラムをコンパイルすると、**MATMUL** への呼び出しが実際には組み込みサブルーチンを呼び出していて、プログラムに定義されているサブルーチンを呼び出さないため、コンパイルが失敗します。
-qextern=matmul を指定してコンパイルを行うと、プログラムを正しくコンパイルして実行することができます。

-qextname

カテゴリー

移植性とマイグレーション

目的

すべてのグローバル・エンティティーの名前に下線を追加する。

構文

@PROCESS:

@PROCESS EXTNAME[(*name1, name2,...*)] | **NOEXTNAME**

デフォルト

-qnoextname

パラメーター

name
特定のグローバル・エンティティー (または複数のエンティティー) を判別します。名前付きエンティティーのリストの場合、それぞれの名前をコロンで区切ってください。例: *name1: name2:...*。

メインプログラムの名前は影響を受けません。

使用法

-qextname オプションは、XL Fortran に混合言語プログラムを変更しないで移植する一助となります。

このオプションを使用して以下が原因となって発生する命名の問題を回避します。
- **main** または **MAIN** と命名されているか、またはシステム・サブルーチンと同じ名前を持っている Fortran サブルーチン、関数、共通ブロック。
- Fortran から参照される Fortran 以外のルーチンで、ルーチン名の終わりに下線が入っています。

  注**: flush_** および **dtime_** などのような XL Fortran サービスおよびユーティリティー・プロシージャーは、名前の中にすでに下線が付いています。 **-qextname** オプションを指定してコンパイルすることにより、後続の下線を付けずに、これらのプロシージャーの名前をコーディングすることができます。
- Fortran プロシージャーを呼び出して、Fortran 名の終わりに下線が付いている Fortran 以外のルーチン。
- データ名の終わりに下線が付いていて、Fortran プロシージャーと共用される Fortran 以外の外部データ・オブジェクトまたはグローバル・データ・オブジェクト。

プログラムのすべてのソース・ファイルは、必須モジュール・ファイルのソース・ファイルも含め、同じ **-qextname** 設定でコンパイルする必要があります。

xlfutility モジュールを使用してサービスおよびユーティリティー・サブプログラムが正しく宣言されていることを確認する場合は、**-qextname** を指定してコンパイルする際に名前を **xlfutility_extname** に変更する必要があります。

コンパイル単位内に参照される複数のサービスおよびユーティリティー・サブプログラムがある場合、名前が指定されていない **-qextname** と **xlfutility_extname** モジュールを使用すると、プロシージャー宣言検査が正しく機能しない可能性があります。

また、このオプションは、**-qextern**、**-qinline**、および **-qsigtrap** オプションで指定される名前にも影響を及ぼします。コマンド行上の名前に下線を入れる必要はありません。

例

```
@PROCESS EXTNAME
        SUBROUTINE STORE_DATA
        CALL FLUSH(10)  ! Using EXTNAME, we can drop the final underscore.
        END SUBROUTINE
```

```
@PROCESS(EXTNAME(sub1))
program main
  external :: sub1, sub2
  call sub1()               ! An underscore is added.
  call sub2()               ! No underscore is added.
end program
```

関連情報

- 146 ページの『-qextern』
- 185 ページの『-qinline』
- 257 ページの『-qsigtrap』
- 119 ページの『-qbindcextname』

-qfdpr

カテゴリー

最適化およびチューニング

@PROCESS

なし。

目的

オブジェクト・ファイルに IBM Feedback Directed Program Restructuring (FDPR) パフォーマンス・チューニング・ユーティリティーが結果の実行可能ファイルを最適化するために必要とする情報を提供する。

-qfdpr が有効な場合は、最適化データがオブジェクト・ファイルに保管されます。

構文

デフォルト

-qnofdpr

使用法

最高の結果を得るために、プログラム内ですべてのオブジェクト・ファイルに **-qfdpr** を使用します。FDPR は、静的にリンクされていても、ライブラリー・コードではなく **-qfdpr** でコンパイルされたファイルでのみ最適化を実行します。

FDPR ユーティリティーが実行する最適化は、**-qpdf** オプションが実行する最適化と似たものです。

FDPR パフォーマンス調整ユーティリティーには独自の制限がいくつかあるため、このユーティリティーを使用しても、どのプログラムの実行時間も必ず短縮されるとは限りません。また、オリジナル・プログラムとまったく同じ結果が得られる実行可能プログラムが必ず生成されるとも限りません。

例

FDPR ユーティリティーが要求するデータを組み込むように `myprogram.f` をコンパイルするには、以下を入力します。

```
xlf myprogram.f -qfdpr
```

関連情報

- 228 ページの『-qpdf1、-qpdf2』

-qfixed

カテゴリー

入力制御

目的

入力ソース・プログラムが固定ソース形式であることを示し、オプションで行の最大長を指定する。

構文

@PROCESS:

```
@PROCESS FIXED[(right_margin)]
```

デフォルト

-qfixed=72 は、呼び出しコマンド **xlf** および **xlf_r** のデフォルトです (これらのコマンドを使用して .f ファイル、.F ファイル、.f77 ファイル、または .F77 ファイルがコンパイルされる場合)。

-qfixed=72 は、呼び出しコマンド **f77** および **fort77** のデフォルトでもあります。

注:

-qfree=f90 は、 コマンド **f90**、**xlf90**、**xlf90_r**、**f95**、**xlf95**、**xlf95_r**、**f2003**、**xlf2003**、**xlf2003_r**、**f2008**、**xlf2008**、**xlf2008_r**、および CUDA Fortran **xlcuf** CUDA Fortran のデフォルトです。

使用法

FREE ディレクティブまたは **FIXED @PROCESS** ディレクティブを使用してコンパイル単位の形式を切り換えたり、**SOURCEFORM** 注釈ディレクティブを使用し

て (コンパイル単位内部でも) ファイルの残りの形式を切り換えることはできますが、コンパイラーの実行時に指定されたソース形式は、すべての入力ファイルに適用されます。

他のシステムのソース・コードの場合、デフォルトよりも大きい右マージンを指定しなければならない場合もあります。このオプションを使用すれば、最大右マージン 132 を指定することができます。

関連情報

- 160 ページの『-qfree』
- 「*XL Fortran* ランゲージ・リファレンス」の『固定ソース形式』 を参照してください。

-qflag

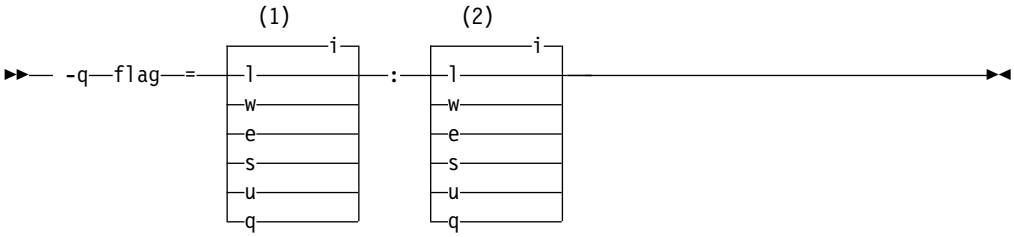
カテゴリー

リスト、メッセージ、およびコンパイラー情報

目的

診断メッセージを指定した重大度レベルまたはそれより高い重大度レベルのものに制限する。

構文



注:

1 リストに報告されたメッセージの最小の重大度レベル

2 端末に報告されたメッセージの最小の重大度レベル

@PROCESS:

```
@PROCESS FLAG(listing_severity,terminal_severity)
```

デフォルト

-qflag=i:i (すべてのコンパイラー・メッセージを表示する)

パラメーター

重大度レベル (最低から最高) は次のとおりです。

i 通知メッセージ。知る必要のある情報を説明しますが、通常、ユーザー側にはアクションを要求しません。

l 言語レベル・メッセージ (**-qlanglvl** オプションの使用時に生成されるものなど)。これは、移植不可能な言語構造体がある可能性を示します。

w 警告メッセージ。ユーザー側のアクションを要求するエラー条件を示しますが、依然として正しいプログラムです。

e エラー・メッセージ。ユーザー側にプログラムを訂正するアクションを要求するエラー条件を示しますが、結果プログラムは依然として実行可能な場合があります。

s 重大エラー・メッセージ。ユーザー側にプログラムを訂正するアクションを要求するエラー条件を示し、エラー位置に達すると結果プログラムに障害が起きます。コンパイラーがこの種類のエラーを検出した場合にオブジェクト・ファイルを生成するよう、**-qhalt** 設定を変更する必要があります。

u 回復不能エラー・メッセージ。コンパイラーが続行できなくなるエラー条件を示します。プログラムのコンパイルを行う前に、ユーザー側のアクションが必要です。

q メッセージなし。定義済みのエラー条件では生成されることがない重大度レベル。これを指定すると、回復不能エラーが検出されても、コンパイラーはメッセージを表示しません。

使用法

listing_severity と *terminal_severity* の両方を指定する必要があります。

listing_severity またはそれ以上の重大度を持つメッセージだけがリスト・ファイルに書き込まれます。 *terminal_severity* またはそれ以上の重大度を持つメッセージだけが端末装置に書き込まれます。

-qflag オプションは、指定された **-qlanglvl**、**-qsaa** などのオプションをオーバーライドします。

-w は、**-qflag=e:e** の短い形式です。

-qhaltonmsg オプションは、**-qflag** オプションよりも優先されます。**-qhaltonmsg** と **-qflag** が両方とも指定された場合は、**-qflag** が選択しないメッセージも出力され、コンパイルが停止します。

注**:** **-qflag=u:u** または **-qflag=q:q** が指定された場合は、**-qhaltonmsg** によって指定されたメッセージは表示されません。

関連情報

- 167 ページの『-qhalt』
- 198 ページの『-qlanglvl』
- 213 ページの『-qmaxerr』
- 168 ページの『-qhaltonmsg』
- 251 ページの『-qsaa』
- 282 ページの『-qsuppress』
- 322 ページの『-w』
- 341 ページの『XL Fortran エラー・メッセージに関する情報』

-qfloat

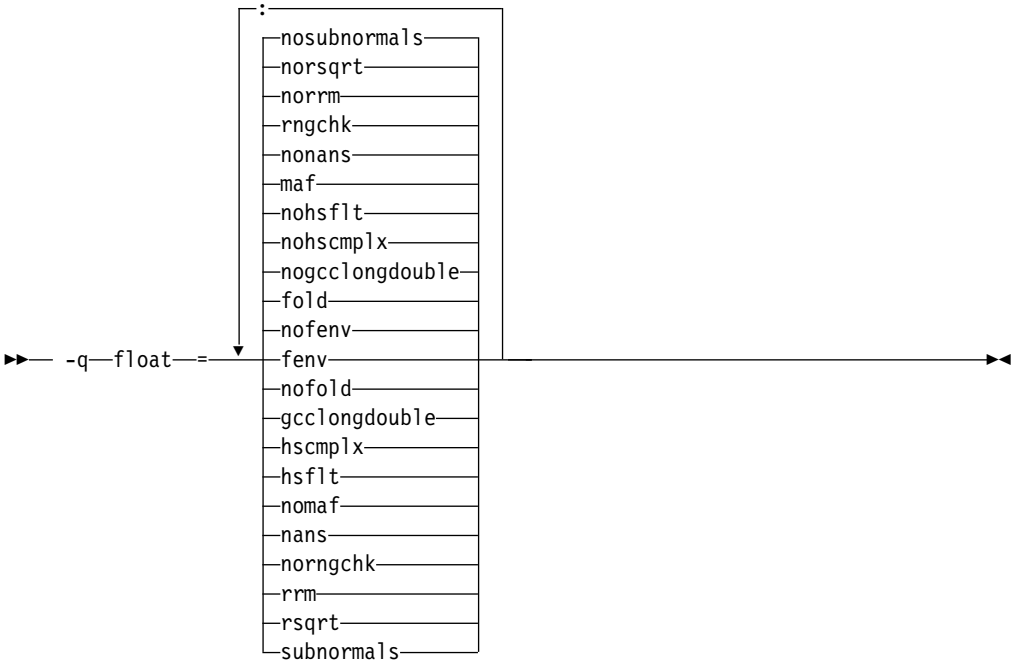
カテゴリー

浮動小数点および整数のコントロール

目的

浮動小数点の計算を高速化したり、精度を上げるためのさまざまなストラテジーを選択する。

構文



@PROCESS:

@PROCESS FLOAT(*suboptions*)

デフォルト

- **-qfloat=nofenv:fold:nogclongdouble:nohscmplx:nohsflt:maf:nonans:rngchk:norrm:norsqrt:nosubnormals**
- **-qfloat=rsqrt:norngchk** (**-qnostrict**、**-qstrict=nooperationprecision:noexceptions**、または **-O3** 以上の最適化レベルが有効な場合)

パラメーター

fenv | **nofenv**

コードが、ハードウェア環境に依存するかどうか、この依存関係によって予期せぬ結果が生じる可能性がある最適化を抑制するかどうかを指定します。

特定の浮動小数点操作は、浮動小数点状況および制御レジスター (FPSCR) に依存して、例えば、丸めモードを制御またはアンダーフローを検出します。特に、多くのコンパイラー組み込み関数は、FPSCR から値を直接読み取ります。

nofenv が有効なとき、コンパイラーは、プログラムがハードウェア環境に依存していないこと、浮動小数点操作のシーケンスを変更するアグレッシブなコンパイラー最適化が許可されていることを前提とします。**fenv** が有効なとき、そのような最適化は抑制されます。

予期せぬ動作を起こす可能性がある最適化から保護するために、ハードウェア浮動小数点環境を読み取る、または設定する記述を含むコードに **fenv** を使用してください。

fold | nofold
コンパイル時に浮動小数点の定数式を評価します。これは、実行時に評価する場合とは多少異なる結果を出す場合があります。 **nofold** が指定されていても、コンパイラーは常に仕様ステートメント内の定数式を評価します。

gcclongdouble | nogcclongdouble
コンパイラーが、128 ビットの REAL(16) および COMPLEX(32) 演算で GCC 提供のライブラリー関数と IBM 提供のライブラリー関数のどちらを使用するかを指定します。

gcclongdouble は、数値計算で GCC とのバイナリー互換性を確実にします。この互換性がご使用のアプリケーションで重要でない場合は、パフォーマンスを高めるために **nogcclongdouble** を使用してください。

注**:** 結果を、**nogcclongdouble** でコンパイルされたモジュールから **gcclongdouble** でコンパイルされたモジュールに引き渡すと、異なる数字結果が作成されます (Inf、NaN、および他のまれなケースなど)。そのような非互換を回避するために、コンパイラーは、IBM long double 型を GCC long double 型に変換する組み込み関数を提供しています ()。

hscmplx | nohscmplx
複素数の除算および複素数の絶対値を含む演算のスピードアップを行います。**hsflt** サブオプションの最適化のサブセットを提供するこのサブオプションは、複素数計算では優先されます。

hsflt | nohsflt
単精度式の丸めを防止して、浮動小数点部を除数の逆数を掛ける乗算と置き換えることによって、計算をスピードアップします。 **hsflt** は、**hscmplx** を暗黙に示します。

hsflt サブオプションは、**nans** および **spnans** サブオプションをオーバーライドします。

注**:** 浮動小数点計算で特性が分っている場合は、複素数の除算および浮動小数点の変換を行うアプリケーションで **-qfloat=hsflt** を使用します。特に、浮動小数点結果はすべて、単精度表示の定義範囲内になければなりません。このオプションは、警告なしで予期せぬ結果を起こすおそれがあるので、注意して使用してください。複素数計算では、**hscmplx** サブオプション (上述) を使用することをお勧めします。これは、**hsflt** の予期しない結果を起こさずに同じ程度のスピードアップをはかることができます。

maf | nomaf
適切な箇所で浮動小数点乗算・加算命令を使用することによって、より高速でより正確に浮動小数点計算を行います。 この結果は、コンパイル時に行われる類似の計算の結果または他のタイプのコンピューターでの結果と正確に同じになら

ない場合があります。 負のゼロの結果が作成される可能性があります。これらの演算では、負の無限大または正の無限大に丸めると、反転します。 このサブオプションは、浮動小数点の中間結果の精度に影響を与える可能性があります。**-qfloat=nomaf** が指定されると、乗加算命令が正確さを必要としない場合、乗加算命令は生成されません。

nans | nonans

-qfltrap=invalid:enable オプションを使用してシグナル NaN (非数字) 値を含む例外条件の検出および処理を行うことを可能にします。シグナル NaN 値は、他の浮動小数点演算からは出てこないため、このサブオプションは、プログラムがこの値を明示的に作成する場合にのみ使用してください。

rngchk | norngchk

-O3 以上の最適化レベルで **-qstrict** を指定しないと、範囲のチェックが、ソフトウェアの割り算演算とインライン化された平方根演算の入力引数で実行されるかどうかを制御します。**norngchk** の指定は、コンパイラーに範囲検査をスキップするように指示し、ループ内で除算演算および平方根演算を繰り返し行う状況でのパフォーマンスを向上させることができます。

norngchk を有効にして、以下の制限を適用します。

- 割り算演算の被除数は、+/-INF にしないでください。
- 割り算演算の除数は、0.0、+/- INF、または非正規数にしないでください。
- 被除数の商と除数は、+/-INF にしないでください。
- 平方根演算の入力は、INF にしないでください。

これらの条件が満たされない場合、不正な結果が生じる可能性があります。例えば、割り算演算の除数が 0.0 または非正規化数 (倍精度の場合は、絶対値 < 2^{-1022}、単精度の場合は、絶対値 < 2^{-126}) の場合、INF ではなく NaN になります。除数が +/- INF のとき、0.0 の代わりに NaN になります。入力が、sqrt 演算で +INF になる場合、INF ではなく NaN となります。

norngchk は、**-qnostrict** が有効なときのみ許可されます。**-qstrict**、**-qstrict=infinities**、**-qstrict=operationprecision**、または **-qstrict=exceptions** が有効なときは、**norngchk** は無視されます。

rrm | norrm

実行時に丸めモードがデフォルト (最も近い値に丸める) にならなければならない浮動小数点の最適化を、浮動小数点の丸めモードが変更されるか、実行時に最も近い値に丸めないようにコンパイラーに通知することで、防ぎます。プログラムが実行時の丸めモードを変更する場合は、**rrm** を使用してください。そうでない場合、プログラムは、間違った計算結果を出してしまいます。

rsqrt | norsqrt

平方根の結果で割る除算を、平方根の逆数を掛ける乗算と置き換えることによって、一部の計算をスピードアップします。

-O3 以上の最適化レベルでコンパイルする場合、**rsqrt** は自動的に使用可能になります。使用不可にするには、**-qstrict**、**-qstrict=nans**、**-qstrict=infinities**、**-qstrict=zerosigns**、または **-qstrict=exceptions** も指定します。

subnormals | nosubnormals

これは、非正規化浮動小数点値 (正規化されていない浮動小数点値とも呼ばれる) がコードで使用されるかどうかを指定します。このサブオプションを指定し

ても指定しなくても、プログラムの振る舞いは変わりませんが、コンパイラーはこの情報を使用して、可能な限りパフォーマンスを向上させます。

注: **-qfloat** サブオプションとそれらに対応する **-qstrict** の関係について詳しくは、274 ページの『-qstrict』を参照してください。

使用法

デフォルト設定以外の **-qfloat** サブオプションを使用すると、指定のサブオプションのすべての必須条件が満たされない場合に、浮動小数点計算で正しくない結果をもたらす可能性があります。したがって、IEEE 浮動小数点値に関連した浮動小数点の計算を扱う際に、プログラムにエラーが混入する可能性を適切に評価できる場合にのみ、このオプションを使用してください。

-qstrict | **-qnostrict** および **float** サブオプションが競合する場合は、後に指定された設定が使用されます。

例

コンパイル時に定数浮動小数点式が評価され、乗加算命令が生成されないように、`myprogram.f` をコンパイルするには、以下を入力します。

```
xlf myprogram.f -qfloat=fold:nomaf
```

関連情報

- 113 ページの『-qarch』
- 157 ページの『-qflttrap』
- 274 ページの『-qstrict』
- 『XL Fortran 浮動小数点処理のインプリメンテーションの詳細』(「*XL Fortran* 最適化およびプログラミング・ガイド」)

-qfpp

カテゴリー

入力制御

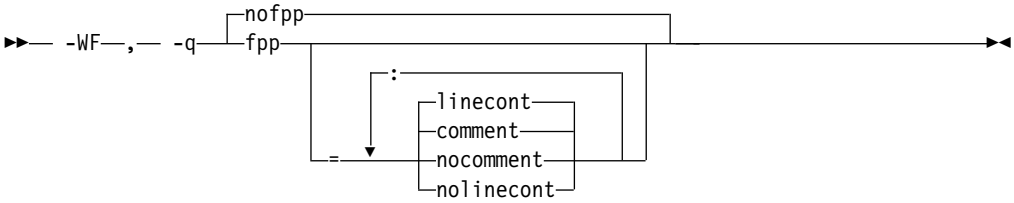
@PROCESS

なし。

目的

C プリプロセッサーにおける Fortran 固有のプリプロセッシングを制御する。

構文



デフォルト

- -qnofpp

パラメーター

comment | nocomment
C プリプロセッサー (**cpp**) に、マクロ展開のコメント区切りとして ! 文字を認識するよう指示します。このサブオプションが有効な場合、! 文字と、その行で ! 文字に続くすべての文字は、マクロ展開の実行時に **cpp** で無視されます。

linecont | nolinecont
cpp に、行継続文字として & 文字を認識するよう指示します。このサブオプションが有効な場合、**cpp** は、& 文字および C スタイルの ¥ 行継続文字を同等に扱います。

サブオプションなしで **-qfpp** を指定すると、**-qfpp=comment:linecont** と等価になります。

使用法

-qfpp は、C プリプロセッサー・オプションであり、**-WF** オプションを使用して指定する必要があります。

関連情報

- 323 ページの『-W、-X』
- 240 ページの『-qppsuborigarg』
- 39 ページの『C プリプロセッサーによる Fortran ファイルの引き渡し』

-qfltttrap

カテゴリー

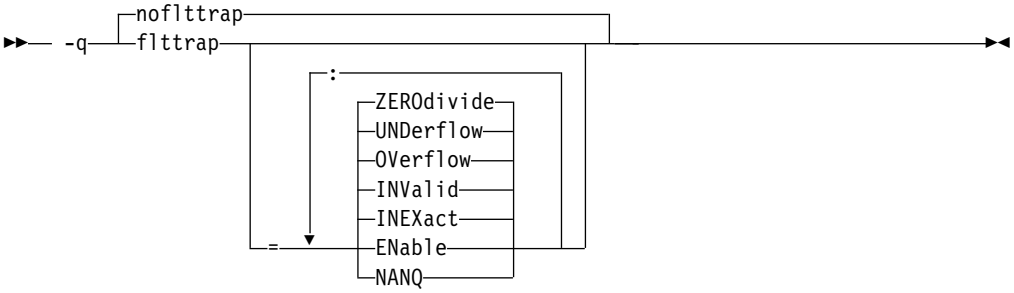
エラー・チェックおよびデバッグ

目的

実行時に検出する浮動小数点例外のタイプを決定する。

該当する例外が発生すると、プログラムは **SIGFPE** シグナルを受信します。

構文



@PROCESS:

FLTTRAP[(*suboptions*)] | **NOFLTTRAP**

デフォルト

-qnoflttrap

サブオプションを指定せずに **-qflttrap** オプションを指定することは、**-qflttrap=invalid:inexact:overflow:undflow:zerodivide** を指定することと同等です。

パラメーター

ENable
例外で **SIGFPE** シグナルが生成されるように、メインプログラム内での指定された例外のチェックをオンにします。ソース・コードを変更しないで例外トラッピングをオンにしたい場合は、このサブオプションを指定する必要があります。

INEXact
例外チェックが使用可能な場合は、浮動小数点の不正確さを検出してトラップします。浮動小数点計算では不正確な結果はよくあることなので、この種の例外を常にオンにしておく必要はありません。

INValid
例外チェックが使用可能な場合は、浮動小数点無効操作を検出してトラップします。

NANQ
すべての静止非数値 (NaNQ) およびシグナル非数値 (NaNS) を検出およびトラップします。トラッピング・コードは、**enable** サブオプションの指定にかかわらず生成されます。このサブオプションは、無効演算によって作成されなかったものを含め、浮動小数点命令によって処理または生成されたすべての NaN 値を検出します。このオプションは、パフォーマンスに影響を与える可能性があります。

OVerflow
例外チェックが使用可能な場合は、浮動小数点オーバーフローを検出してトラップします。

UNDerflow
例外チェックが使用可能な場合は、浮動小数点アンダーフローを検出してトラップします。

ZEROdivide
例外チェックが使用可能な場合は、浮動小数点ゼロ割り算を検出してトラップします。

使用法

例外はハードウェアによって検出されますが、トラッピングは有効ではありません。このデフォルトには **enable** が含まれないため、ソースで既に **fpsets** または類似のサブルーチンを使用していると便利です。

サブオプションの有無に関わらず **-qflttrap** を 2 回以上指定すると、サブオプションなしの **-qflttrap** は無視されます。

-qflttrap オプションは IPA とのリンク中に認識されます。リンク・ステップでオプションを指定するとコンパイル時の設定値をオーバーライドします。

注**:** 実行された変換および一部のベクトル命令の例外処理サポートが原因で、**-qsimd=auto** を使用すると、例外が catch されるロケーションが変更されるか、またはコンパイラーが例外の catch に失敗することもあります。

『浮動小数点例外の検出とトラッピング』トピック(「*XL Fortran* 最適化およびプログラミング・ガイド」) では、**-qflttrap** オプションの使用方法やこのオプションを使用すべき場合について、特に初心者向けに詳しく説明しています。

例

```
REAL :: x, y, z
DATA x /5.0/, y /0.0/
z = x / y
PRINT *, z
END
```

以下のコマンドを使用してこのプログラムをコンパイルした場合は、プログラムは除算が実行されると停止します。

```
xlf -qflttrap=zerodivide:enable -qsigtrap divide_by_zero.f
```

zerodivide サブオプションが、保護対象とする例外のタイプを識別します。**enable** サブオプションを指定した場合は、例外が発生すると **SIGFPE** シグナルが生成されます。**-qsigtrap** オプションを使用すると、シグナルがプログラムを停止したときに、情報出力が作成されます。

関連情報

- 浮動小数点例外の検出とトラッピング
- 153 ページの『-qfloat』
- 113 ページの『-qarch』
- 257 ページの『-qsigtrap』

-qfree

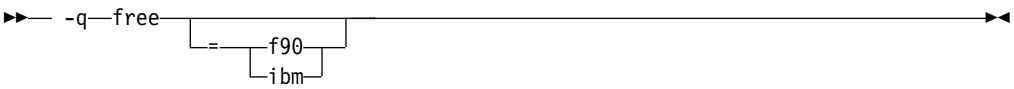
カテゴリー

入力制御

目的

ソース・コードが自由ソース形式であることを示す。

構文



@PROCESS:

```
@PROCESS FREE[({F90|IBM})]
```

デフォルト

-qfree そのものは、 Fortran 90 自由ソース形式を指定します。

-qfixed=72 は、呼び出しコマンド **xlf** および **xlf_r** のデフォルトです (これらのコマンドを使用して .f ファイル、.F ファイル、.f77 ファイル、または .F77 ファイルがコンパイルされる場合)。 **-qfixed=72** は、呼び出しコマンド **f77** および **fort77** のデフォルトでもあります。

-qfree=f90 は、 コマンド **f90**、**xlf90**、**xlf90_r**、**f95**、**xlf95**、**xlf95_r**、**f2003**、**xlf2003**、**xlf2003_r**、**f2008**、**xlf2008**、**xlf2008_r**、および CUDA Fortran **xlcuf** CUDA Fortran のデフォルトです。

パラメーター

ibm
VS FORTRAN に対して定義されている自由ソース形式との互換性を指定します。

f90
Fortran 90 に対して定義されている自由ソース形式との互換性を指定します。

Fortran 90 用に定義した自由ソース形式は、Fortran 95、Fortran 2003、および Fortran 2008 にも適用されることに注意してください。

使用法

FREE または **FIXED @PROCESS** ディレクティブを使用すると、ある特定のコンパイル単位の形式を切り換えることができ、**SOURCEFORM** コメント・ディレクティブを使用すると、(あるコンパイル単位内で使用した場合であっても) そのファイルの残りの部分の形式を切り換えられますが、コンパイラーの実行時に指定するソース形式は、すべての入力ファイルに適用されます。

Fortran 90 自由ソース形式は、現在および将来 Fortran 90 および Fortran 95 機能をサポートするコンパイラーに最大の移植性を与えるために使用される形式です。

IBM 自由ソース形式は、IBM VS FORTRAN コンパイラーの自由形式と同等で、z/OS® プラットフォームからのプログラムの移植を支援するためのものです。

-k は **-qfree=f90** と同等です。

関連情報

- 150 ページの『-qfixed』
- 95 ページの『-k』
- 自由ソース形式 (*XL Fortran* ランゲージ・リファレンス内)

-qfullpath

カテゴリー

エラー・チェックおよびデバッグ

@PROCESS

なし。

目的

このオプションは、**-g** オプションまたは **-qlinedebug** オプションと使用した場合、デバッグ情報付きでコンパイルされたオブジェクト・ファイルのソース・ファイルおよびインクルード・ファイルのフルパス名 (絶対パス名) を記録して、デバッグ・ツールが正確にソース・ファイルの場所を検索できるようにします。

構文

デフォルト

デフォルトでは、コンパイラーは元のソース・ファイルの相対パス名をそれぞれの **.o** ファイルの中に記録します。また、インクルード・ファイルの相対パス名が記録される場合もあります。

使用法

実行可能ファイルをデバッグの前に別のディレクトリーへ移動する必要がある場合、または複数のバージョンのソース・ファイルがあってデバッガーが必ず元のソース・ファイルを使用するようにしたい場合は、 **-qfullpath** オプションを **-g** または **-qlinedebug** オプションと組み合わせて使用すると、ソース・レベル・デバッガーは正しいソース・ファイルを見つけることができます。

-qfullpath は **-g** または **-qlinedebug** オプションがなくても機能しますが、 **-g** または **-qlinedebug** オプションを一緒に指定しない限りソース・レベルのデバッグはできません。

例

この例では実行可能ファイルは作成後に移動されますが、デバッガーは元のソース・ファイルを引き続き見つけることができます。

```
$ xlf95 -g -qfullpath file1.f file2.f file3.f -o debug_version
...
$ mv debug_version $HOME/test_bucket
$ cd $HOME/test_bucket
$ gdb debug_version
```

関連情報

- 90 ページの『-g』
- 203 ページの『-qlinedebug』

-qfunctrace

カテゴリー

エラー・チェックおよびデバッグ

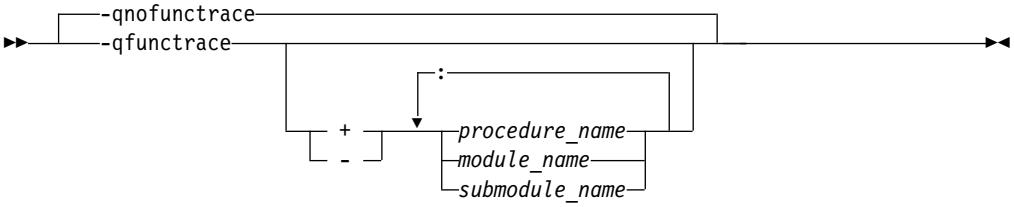
@PROCESS

なし。

目的

ご使用のプログラムでプロシージャーの入り口点および出口点をトレースする。ご使用のプログラムに C++ コンパイル単位が含まれていると、このオプションは C++ catch ブロックもトレースする。

構文



デフォルト

-qnofunctrace

パラメーター

+ 指定したプログラム、プロシージャー、またはモジュール・プロシージャーをトレースするようにコンパイラーに指示します。これらのすべての内部プロシージャーはデフォルトでトレースされます。

- 指定したプログラム、プロシージャー、モジュール・プロシージャー、およびそれらの内部プロシージャーのいずれもトレースしないようにコンパイラーに指示します。

procedure_name
プログラム、外部プロシージャー、またはモジュール・プロシージャーの名前です。名前は **-qmixed** が有効であると、大文字と小文字を区別します。
BIND(C) バインディング・ラベルおよびマングルされたモジュール・プロシージャー名が使用できます。ただし、大文字と小文字を正しく指定していなければなりません。 **-qextname** が有効である場合、*procedure_name* は、追加の下線なしのプロシージャー名です。

module_name
モジュールの名前。名前は **-qmixed** が有効であると、大文字と小文字を区別します。

F2008 ***submodule_name***
サブモジュールの名前。**-qmixed** が有効な場合、この名前の大文字と小文字は区別されます。 F2008

使用法

-qfunctrace により、ご使用のプログラムに含まれるすべてのプロシージャーがトレースできるようになります。**-qnofunctrace** を指定すると、 **-qfunctrace** で使用可能に設定されたトレースを使用不可に設定します。

-qfunctrace+ および **-qfunctrace-** サブオプションにより、プロシージャーの特定のリストをトレースすることができるようになります。これらのサブオプションは **-qnofunctrace** の影響は受けません。プロシージャーのリストは累積されます。モジュール・プロシージャーとそれを含むモジュール F2008 またはサブモジュール F2008 の両方が指定された場合、プロシージャーの指定が優先されます。

このオプションは、定義したトレースプロシージャーへの呼び出しを挿入します。これらのプロシージャーは、リンク・ステップで指定しておく必要があります。トレース・プロシージャーのインターフェース、およびそれらを呼び出すタイミングについて詳しくは、「*XL Fortran* 最適化およびプログラミング・ガイド」の 『コード内のトレース・プロシージャー』のセクションを参照してください。

例

以下の表に、さまざまな目的を達成するために **-qfunctrace** オプションを使用する例をいくつか示します。

目的	使用例
すべてのプロシージャーのトレース	`-qfunctrace`
プロシージャー *x*、*y*、および *z* のトレース	`-qfunctrace+x:y:z`
x 以外のすべてのプロシージャーのトレース	`-qfunctrace -qfunctrace-x` または `-qfunctrace-x -qfunctrace`

プロシージャー *x* および *y* のみのトレース	`-qfunctrace+x -qfunctrace+y` または
	`-qfunctrace+x -qnofunctrace`
	`-qfunctrace+y`

目的	使用例
プロシージャー *y* のみのトレース	`-qfunctrace+y -qnofunctrace` または `-qfunctrace+y`
モジュール *y* の、プロシージャー *x* 以外のすべてのモジュール・プロシージャーのトレース	`-qfunctrace-x -qfunctrace+y` または `-qfunctrace+y -qfunctrace-x`

関連情報

- 『-qfunctrace_xlf_catch』
- 165 ページの『-qfunctrace_xlf_enter』
- 166 ページの『-qfunctrace_xlf_exit』
- トレース・プロシージャーの名前を指定するために使用できるディレクティブの詳細については、「*XL Fortran* ランゲージ・リファレンス」の **FUNCTRACE_XLF_CATCH**、『**FUNCTRACE_XLF_ENTER**』、『**FUNCTRACE_XLF_EXIT**』 セクションを参照してください。
- **NOFUNCTRACE** ディレクティブを使用する規則の詳細については、「*XL Fortran* ランゲージ・リファレンス」の **NOFUNCTRACE**を参照してください。
- ご使用のコードにプロシージャー・トレース・ルーチンを実装する方法の詳細と、詳細例およびこれらを使用するための規則のリストについては、「*XL Fortran* 最適化およびプログラミング・ガイド」の『コード内のトレース・プロシージャー』を参照してください。

-qfunctrace_xlf_catch

カテゴリー

エラー・チェックおよびデバッグ

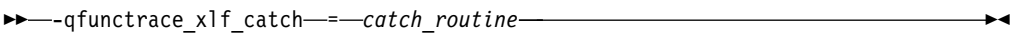
@PROCESS

なし。

目的

catch トレース・サブルーチンの名前を指定します。

構文



デフォルト

適用されません。

パラメーター

catch_routine
　catch トレース・サブルーチンの名前を示します。

使用法

-qfunctrace_xlf_catch オプションを使用して、コンパイルされている外部プロシージャーまたはモジュール・プロシージャーを catch トレース・プロシージャーとして使用する必要があることを指定します。

注**:**
- トレース・サブルーチンを書き込む場合、プログラムに `__func_trace_catch` というユーザー・プロシージャーを含まないようにしてください。
- **-qfunctrace_xlf_catch** オプションを使用する際に、内部サブルーチンの名前を指定してはなりません。

関連情報
- 「*XL Fortran* ランゲージ・リファレンス」の **FUNCTRACE_XLF_CATCH** ディレクティブ。
- 162 ページの『-qfunctrace』
- 『-qfunctrace_xlf_enter』
- 166 ページの『-qfunctrace_xlf_exit』
- ご使用のコードにトレース・プロシージャーを実装する方法について詳しくは、「*XL Fortran* 最適化およびプログラミング・ガイド」の 『コード内のトレース・プロシージャー』を参照してください。

-qfunctrace_xlf_enter

カテゴリー

エラー・チェックおよびデバッグ

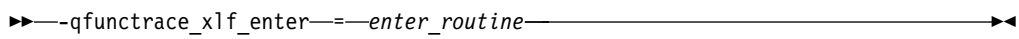
@PROCESS

なし。

目的

entry トレース・サブルーチンの名前を指定します。

構文



デフォルト

適用されません。

パラメーター

enter_routine
entry トレース・サブルーチンの名前を示します。

使用法

-qfunctrace_xlf_enter オプションを使用して、コンパイルされている外部プロシージャーまたはモジュール・プロシージャーを enter トレース・プロシージャーとして使用する必要があることを指定します。

注**:**
- トレース・サブルーチンを書き込む場合、プログラムに `__func_trace_enter` というユーザー・プロシージャーを含まないようにしてください。
- **-qfunctrace_xlf_enter** オプションを使用する際に、内部サブルーチンの名前を指定してはなりません。

関連情報
- 「*XL Fortran* ランゲージ・リファレンス」の**FUNCTRACE_XLF_ENTER** ディレクティブ。
- 162 ページの『-qfunctrace』
- 164 ページの『-qfunctrace_xlf_catch』
- 『-qfunctrace_xlf_exit』
- ご使用のコードにトレース・プロシージャーを実装する方法について詳しくは、「*XL Fortran* 最適化およびプログラミング・ガイド」の 『コード内のトレース・プロシージャー』を参照してください。

-qfunctrace_xlf_exit

カテゴリー

エラー・チェックおよびデバッグ

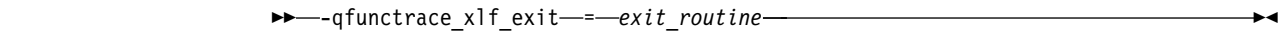
@PROCESS

なし。

目的

exit トレース・サブルーチンの名前を指定します。

構文



デフォルト

適用されません。

パラメーター

exit_routine
exit トレース・サブルーチンの名前を示します。

使用法

-qfunctrace_xlf_exit オプションを使用して、コンパイルされている外部プロシージャーまたはモジュール・プロシージャーを exit トレース・プロシージャーとして使用する必要があることを指定します。

注**:**

- トレース・サブルーチンを書き込む場合、プログラムに `__func_trace_exit` というユーザー・プロシージャーを含まないようにしてください。
- **-qfunctrace_xlf_exit** オプションを使用する際に、内部サブルーチンの名前を指定してはなりません。

関連情報

- 「*XL Fortran* ランゲージ・リファレンス」の**FUNCTRACE_XLF_EXIT** ディレクティブ。
- 162 ページの『-qfunctrace』
- 164 ページの『-qfunctrace_xlf_catch』
- 166 ページの『-qfunctrace_xlf_exit』
- ご使用のコードにトレース・プロシージャーを実装する方法について詳しくは、「*XL Fortran* 最適化およびプログラミング・ガイド」の 『コード内のトレース・プロシージャー』を参照してください。

-qhalt

カテゴリー

エラー・チェックおよびデバッグ

目的

コンパイル時のメッセージの最大重大度が指定した重大度と同じかそれを超える場合は、オブジェクト・ソース・ファイル、実行可能ソース・ファイル、またはアセンブラー・ソース・ファイルのいずれかを作成する前に、コンパイルを停止する。

構文

注**:**

1 オブジェクト・ファイルを作成しないようにするメッセージの最小の重大度レベル

@PROCESS:

`@PROCESS HALT(`*severity*`)`

デフォルト

-qhalt=s。この場合コンパイラーはコンパイルが失敗してもオブジェクト・ファイルを生成しません。

パラメーター

重大度レベル (最低から最高) は次のとおりです。

i 通知メッセージ。知る必要のある情報を説明しますが、通常、ユーザー側にはアクションを要求しません。

l 言語レベル・メッセージ (**-qlanglvl** オプションの使用時に生成されるものなど)。これは、移植不可能な言語構造体がある可能性を示します。

w 警告メッセージ。ユーザー側のアクションを要求するエラー条件を示しますが、依然として正しいプログラムです。

e エラー・メッセージ。ユーザー側にプログラムを訂正するアクションを要求するエラー条件を示しますが、結果プログラムは依然として実行可能な場合があります。

s 重大エラー・メッセージ。ユーザー側にプログラムを訂正するアクションを要求するエラー条件を示し、エラー位置に達すると結果プログラムに障害が起きます。コンパイラーがこの種類のエラーを検出した場合にオブジェクト・ファイルを生成するよう、**-qhalt** 設定を変更する必要があります。

u 回復不能エラー・メッセージ。コンパイラーが続行できなくなるエラー条件を示します。プログラムのコンパイルを行う前に、ユーザー側のアクションが必要です。

使用法

-qhalt オプションは **-qobject** オプションを、そして **-qnoobject** オプションは **-qhalt** オプションをオーバーライドできます。

関連情報

- 151 ページの『-qflag』
- 『-qhaltonmsg』
- 213 ページの『-qmaxerr』
- 222 ページの『-qobject』

-qhaltonmsg

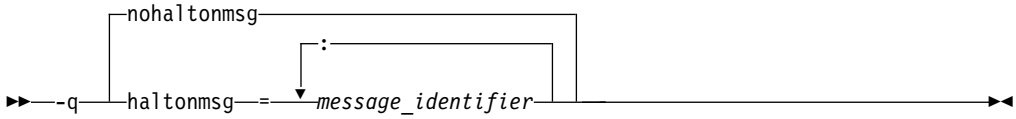
カテゴリー

エラー・チェックおよびデバッグ

目的

指定されたエラー・メッセージが生成された場合、オブジェクト・ファイル、実行可能ファイル、またはアセンブラー・ソース・ファイルを作成せずにコンパイルを停止する。

構文



@PROCESS:

@PROCESS HALTONMSG(*message_identifier*[, *message_identifier*[, ...]]) | **NOHALTONMSG**

デフォルト

-qnohaltonmsg

パラメーター

message_identifier[:message_identifier ...]

指定したエラー・メッセージ (*nnnn-mmm*) またはメッセージのリスト (*nnnn-mmm[:nnnn-mmm ...]*) が生成された場合、オブジェクト・ファイル、実行可能ファイル、またはアセンブラー・ソース・ファイルを作成する前にコンパイルを停止します。メッセージのリストを指定するには、それぞれのメッセージ番号をコロンで区切ってください。

nnnn-mmm は、メッセージ番号です。ここで、

- *nnnn* は、1500 から 1585 の範囲にある 4 桁の整数でなければなりません。 XL Fortran メッセージ番号はこの範囲内にあります。
- *mmm* は、3 桁の任意の整数です (必要であればゼロを先行させます)。

使用法

-qhaltonmsg オプションの結果としてコンパイラーが停止した場合、コンパイラーの戻りコードはゼロ以外です。元の重大度レベルが S よりも低い場合、**-qhaltonmsg** によって指定されたメッセージの重大度レベルは S に変更されます。

-qflag=u:u または **-qflag=q:q** が指定された場合は、**-qhaltonmsg** によって指定されたメッセージは表示されません。

-qhaltonmsg は、**-qsuppress** および **-qflag** よりも優先されます。

関連情報

- 167 ページの『-qhalt』
- 151 ページの『-qflag』
- 282 ページの『-qsuppress』

-qhelp

カテゴリー

リスト、メッセージ、およびコンパイラー情報

@PROCESS

なし。

目的

コンパイラーの man ページを表示する。

構文

使用法

-qhelp オプションを指定すると、入力ファイルを指定したかどうかに関係なく、コンパイラーの man ページが表示され、コンパイルは停止します。

関連情報

- 298 ページの『-qversion』

-qhot

カテゴリー

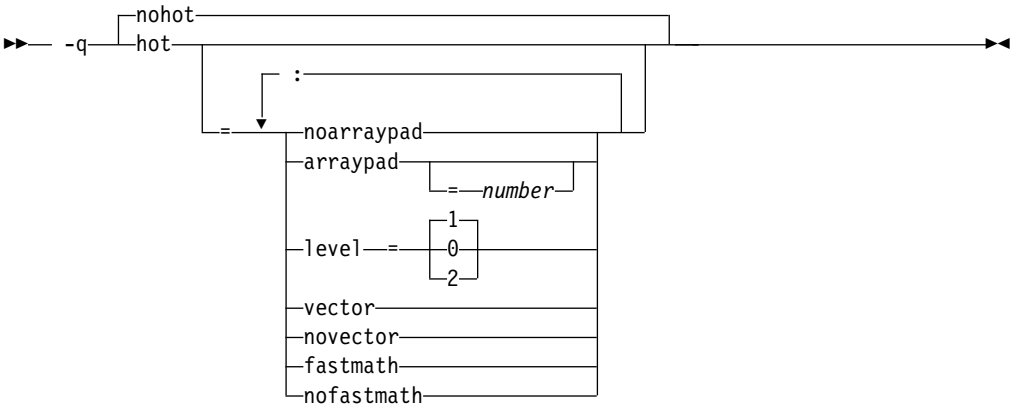
最適化およびチューニング

目的

最適化中に上位ループ分析および変換 (HOT) を実行する。

-qhot コンパイラー・オプションは、ループと配列言語を最適化するためのチューニングを助ける強力な代替手段です。このコンパイラー・オプションは、指定されたサブオプションに関係なく、常にループの最適化を試行します。

構文



@PROCESS:

@PROCESS HOT[=suboptions] | **NOHOT**

デフォルト

- **-qnohot**
- **-qsmp**、**-O3**、**-O4**、**-O5**、または **-Ofast** が有効な場合、**-qhot=noarraypad:level=1:vector:fastmath**。
- **-qhot** をサブオプションなしで指定することは、**-qhot=noarraypad:level=1:vector:fastmath** と等価です。

パラメーター

arraypad | noarraypad
コンパイラーは配列処理ループの効率を高められそうな配列次元を増やすことができます。(キャッシュ・アーキテクチャーのインプリメンテーションのため、2 の累乗である配列次元がキャッシュの使用効率の低下を招く可能性があります。)**-qhot=arraypad** を指定して、ソースに 2 の累乗である次元を持つ大きな配列が含まれる場合は、配列処理プログラムを遅らせるキャッシュのミスとページ障害を削減することができます。これは特に、最初の次元が 2 の累乗である場合に効果的です。このサブオプションを *number* なしで使用する場合、コンパイラーは、有効であると推測した配列を埋め込んだり、選択した量だけを埋め込んだりします。すべての配列で必ずしも埋め込みが行われるわけではなく、また異なる配列ごとに異なる分量で埋め込みが行われます。 *number* を指定すると、コンパイラーは、コード内にすべての配列を埋め込みます。

注**: arraypad** の使用は危険です。埋め込みが起きた場合、再シェーピングや等価性のチェックを行わないため、コードを中断してしまう可能性があります。

number
それぞれの配列ごとにソース内に埋め込まれる要素の数を表す正整数値。埋め込み数は、正の整数値でなければなりません。キャッシュの使用効率を高めるために、埋め込み値は、最大配列エレメント・サイズの倍数 (通常は、4、8、または 16) にすることをお勧めします。

level=0
上位変換のサブセットを実行し、デフォルトを **novector:noarraypad:fastmath** に設定します。

level=1
上位変換のデフォルト設定を実行します。

level=2
上位変換のデフォルト・セットと、より積極的なループ変換をいくつか実行します。このオプションを指定すると、積極的なループの分析と変換が実行されて、キャッシュの再利用が強化されるとともに、ループ並列化の機会が活用されます。

vector | novector
-qnostrict および 、または **-O3** 以上の最適化レベルを指定したとき、**vector** は、コンパイラーに、配列の連続的要素においてループ内で実行されるある種の操作 (例えば、平方根、逆数平方根) を libxlopt の Mathematical Acceleration Subsystem (MASS) ライブラリーのルーチンへの呼び出しに変換させます。

vector サブオプションは、単精度および倍精度の浮動小数点計算をサポートします。このサブオプションは、大量の計算処理需要のあるアプリケーションに役立ちます。

novector は、ループ配列演算の MASS ライブラリー・ルーチン呼び出しへの変換を使用不可にします。

ベクトル化はプログラムの結果の精度に影響を及ぼす可能性があるため、**-03** 以上を使用する場合に、精度の変更を受け入れることができないのであれば、**-qhot=novector** を指定します。

fastmath | nofastmath

このサブオプションを使用すると、数学関数の高速スカラー・バージョンとデフォルトのバージョンのいずれかを使用するよう、アプリケーションをチューニングできます。

-qhot=fastmath を指定すると、**-qstrict=nolibrary** が使用可能に設定されている場合にのみ、数学ルーチンを、XLOPT ライブラリーから取得可能な数学ルーチンに置き換えることができます。

-qhot=nofastmath は、XLOPT ライブラリーによる数学ルーチンの置き換えを使用不可にします。**-qhot=fastmath** は、**-qhot** が指定されている場合、ホット・レベルに関係なくデフォルトで使用可能になります。

使用法

コマンド行で **-qhot** を指定したとき、最適化レベルが指定されない場合は、コンパイラーは **-O2** と見なします。

-qsmp、**-O3**、**-O4** または **-O5** を使用して、デフォルトの **level** 設定 **1** を指定する場合、他のオプションの後に **-qhot=level=0** または **-qhot=level=2** を指定してください。

-O2、**-qnohot**、または **-qnoopt** がコマンド行で使用される場合、@PROCESS ディレクティブでの HOT オプションの指定は、コンパイル単位では有効ではなくなります。

-C オプションは、いくつかの配列最適化をオフにします。

-qreport オプションを **-qhot** オプションまたは **-qhot** を暗黙指定するいずれかの最適化オプションと一緒に使用して、ループがどのように変換されたかを示す疑似 Fortran レポートを生成できます。**-qreport** オプションまたは **-qlistfmt** オプションも指定されている場合、ループ変換はリスト・レポートに組み込まれます。このリスト・ファイルの LOOP TRANSFORMATION SECTION には、データ・プリフェッチ挿入ロケーションに関する情報も入っています。さらに、**-qprefetch=assistthread** を使用してプリフェッチ支援スレッドを生成する場合、リスト・ファイルの LOOP TRANSFORMATION SECTION に「データ・プリフェッチの支援スレッドが生成されました。」というメッセージも表示されます。**-qprefetch=assistthread** を指定すると、コンパイラーは最適化レベル **-O3** 以上で積極的なデータ・プリフェッチを生成します。 詳しくは、 248 ページの『-qreport』を参照してください。

関連情報

- 113 ページの『-qarch』
- 85 ページの『-C』
- 258 ページの『-qsimd』
- 248 ページの『-qreport』
- 205 ページの『-qlistfmt』
- 101 ページの『-O』
- 274 ページの『-qstrict』
- 262 ページの『-qsmp』
- 「*XL Fortran* 最適化およびプログラミング・ガイド」の『*Mathematical Acceleration Subsystem (MASS)* の使用』
- 「*XL Fortran* ランゲージ・リファレンス」の『ループ最適化のためのディレクティブ』
- 「*XL Fortran* 最適化およびプログラミング・ガイド」の上位変換

-qieee

カテゴリー

浮動小数点および整数のコントロール

目的

コンパイル時に定数浮動小数点式を評価する場合にコンパイラーが使用する丸めモードを指定する。

構文

@PROCESS:

```
@PROCESS IEEE({Near | Minus | Plus | Zero})
```

デフォルト

Near。これは、最も近い値に丸めます。

パラメーター

Near 最も近い値に丸めます。

Minus
負の無限大の方向に丸める。

Plus 正の無限大の方向に丸める。

Zero ゼロ方向に丸めます。

使用法

このオプションは、XL Fortran サブルーチン **fpsets** など実行時に丸めモードを変更する方法と組み合わせて使用してください。このオプションは、コンパイル時の演算 (例えば、 **2.0/3.5** などのような定数式の計算) に使用される丸めモードを設定します。

コンパイル時の演算と実行時の演算に同じ丸めモードを指定することにより、浮動小数点結果に矛盾が生じることを回避します。

注**: -O** オプションも指定すると、コンパイル時の算術計算はかなり長くなります。

実行時のデフォルトから丸めモード (最も近い値への丸め) を変更する場合は、必ず **-qfloat=rrm** も指定して、デフォルトの丸めモードでのみ適用される最適化をオフにしてください。

プログラムが、実数 (16) 値を含む演算を含む場合、丸めモードは、最も近い値の **-qieee=near** に設定される必要があります。

関連情報

- 「*XL Fortran* 最適化およびプログラミング・ガイド」の『丸めモードの選択』
- 101 ページの『-O』
- 153 ページの『-qfloat』

-qinfo

カテゴリー

エラー・チェックおよびデバッグ

@PROCESS

なし。

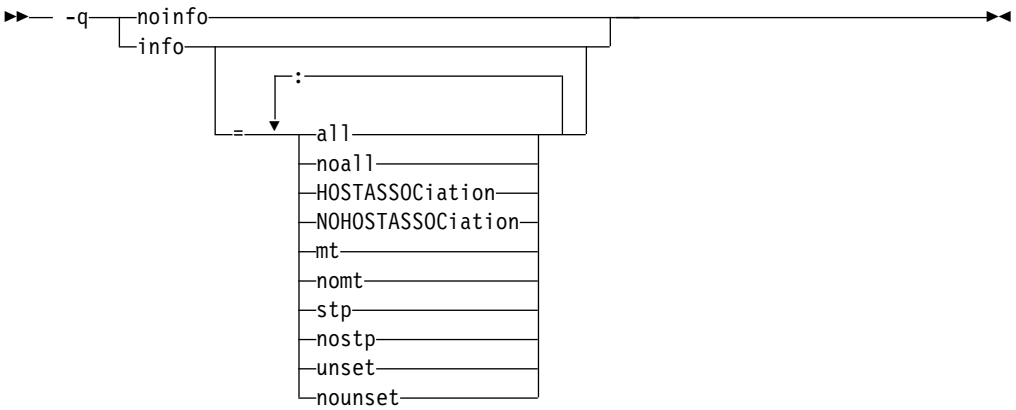
目的

通知メッセージのグループを作成または抑制する。

メッセージは標準出力に出力されますが、リスト・ファイルが生成される場合はオプションでそのファイルにも出力されます。

構文

オプション構文



@PROCESS:

@PROCESS INFO[(*suboptions*)] | **NOINFO**

デフォルト

-qnoinfo

パラメーター

all
すべてのグループ (**mt** を除く) の診断メッセージを使用可能にします。

noall (オプションのみ)
すべてのグループのすべての診断メッセージを無効にします。

mt | nomt

並列コード内の潜在的な同期の問題をレポートします。このサブオプションは、1 つのスレッドが共用揮発性フラグ変数を使用しているグローバル・スレッド・フラグ・パターンを検出し、計算が完了してその結果をメモリーに格納したことを他のスレッドに通知します。他のスレッドは、このフラグ変数を変更しないループ内にあるフラグ変数を確認します。このフラグ変数の値が変わると、待機スレッドはメモリーからの計算結果にアクセスします。PowerPC ストレージ・モデルでは、このフラグ変数が最初のスレッドに設定される前、およびこのフラグ変数が待機スレッドで確認された後に同期を行う必要があります。同期は **LIGHT_SYNC** ディレクティブまたは **ISYNC** ディレクティブによって実行できます。

使用する必要がある同期ディレクティブのタイプは、コードによって変わります。通常は、システム・メモリーへのストレージ・アクセス順序が保持されるため、**LIGHT_SYNC** ディレクティブを使用すれば充分です。ただし、命令プリフェッチによって、このフラグ変数が更新される前に計算結果にアクセスするコードの実行を開始する可能性があるような方法で待機スレッド内のループが作成されている場合、**ISYNC** ディレクティブを使用して順序を保持する必要があります。このようなパターンは通常、以下のようになります。

```
10 CALL sleep_(value)
   IF (.NOT. flag) GOTO 10
   ! The SYNC directive is needed here.
   x = shared_computation_result
```

同期が必要ない一部のパターンは、上述のパターンと同様です。このサブオプションによって生成されるメッセージは、潜在的な同期の問題に関する提案のみです。

-qinfo=mt サブオプションを使用するには、**-qthreaded** オプションを使用可能にして、以下のオプションを少なくとも 1 つ指定する必要があります。

- **-O3**
- **-O4**
- **-O5**
- **-qipa**
- **-qhot**
- **-qsmp**

デフォルト・オプションは **-qinfo=nomt** です。

HOSTASSOCiation | NOHOSTASSOCiation
親子結合によって最初にアクセスされるエンティティーに関する情報メッセージを出します。ただし、エンティティーが **IMPORT** ステートメントによってアクセスされる場合、情報メッセージは出されません。

-qinfo=all / noall を使用して、**-qinfo=HOSTASSOCiation** オプションを使用可能にするかどうかを制御できます。デフォルト・オプションは **-qinfo=NOHOSTASSOCiation** です。

stp | nostp
スタック破壊から保護されていないプロシージャーに関する警告を発行します。**-qstackprotect** オプションを同時に有効にしない限り、**-qinfo=stp** は有効となりません。他の **-qinfo** オプションと同様に、**-qinfo=stp** は **-qinfo=all / noall** を通じて有効または無効にします。デフォルトのオプションは **-qinfo=nostp** です。

unset | nounset
設定される前に使用されている自動変数を検知して、コンパイル時にこれらの変数に通知メッセージをフラグ付けします。

-qinfo=unset は、最適化時に使用できるプログラム情報 (制御フロー情報など) を使用します。その結果として、検知精度は最適化レベルとともに向上します。例えば、-00 ではフラグ付けされない未設定変数の使用に関して、-02 ではフラグ付けされることがあります。-04 や -05 などの検知精度の高い最適化レベルでは、通知メッセージ内の行番号が不正確になる可能性があります。ごくまれに、これらの最適化レベルでプログラムが十分なほどに再配列され、静的分析の結果として誤検出 (false positive) メッセージが発行されることがあります。最適化レベル -02 で最高の結果が得られます。

-qinitauto オプションは、自動変数を初期化します。その結果、**-qinitauto** オプションが指定されていると、設定される前に使用されている変数が **-qinfo=unset** オプションで検知されなくなります。

-qsave オプションは、自動変数のストレージ・クラスを STATIC に変更します。その結果、**-qsave** オプションが指定されていると、設定される前に使用されている変数が **-qinfo=unset** オプションで検知されなくなります。

使用法

サブオプションなしで **-qinfo** を指定することは、**-qinfo=all** を指定することと同等です。

-qnoinfo を指定することは、**-qinfo=noall** を指定することと同等です。

例

プログラムをコンパイルして、スタック保護に関する通知メッセージを生成するには、次のコマンドを入力します。

```
xlf90 myprogram.f -qinfo=stp -qstackprotect
```

t.f をコンパイルして、親子結合されている変数に関する通知メッセージを生成するには、以下のコマンドを入力します。

```
xlf2008 t.f -qinfo=HOSTASSOCiation
```

t.f には以下のコードが含まれているとします。

```
PROGRAM p
  IMPLICIT none

  INTEGER :: var

  var = 3
  CALL sub()

  CONTAINS
    SUBROUTINE sub()
      PRINT *, var ! The compiler issues an information message when
                   ! entity 'var' is accessed by host association for
                   ! the first time.

      PRINT *, var ! No message is issued here.
    END

    INTEGER FUNCTION func()
      func = var   ! Entity 'var' is in a different scope. The compiler
                   ! issues an information message when the entity is
                   ! accessed by host association for the first time in
                   ! each scope.

    END
END
```

コンパイラーは、以下の情報メッセージを出します。

```
"t.f", line 11.20: 1521-004 (I) Entity var is accessed by host association.
"t.f", line 16.18: 1521-004 (I) Entity var is accessed by host association.
```

sync.f をコンパイルして、並列コード内の潜在的な同期の問題に関する通知メッセージを生成するには、以下のコマンドを入力します。

```
xlf95_r -qinfo=mt -O3 sync.f
```

sync.f には以下のコードが含まれているとします。

```
MODULE m
  IMPLICIT NONE
  LOGICAL, VOLATILE :: done    ! shared flag
  INTEGER, VOLATILE :: shared_result
CONTAINS
  SUBROUTINE setter(id)
```

```
    IMPLICIT NONE
    INTEGER, INTENT(IN) :: id

    CALL sleep_(5)
    shared_result = 7
    ! !ibm* light_sync
    done = .TRUE.              ! line 13
  END SUBROUTINE

  SUBROUTINE waiter(id)
    IMPLICIT NONE
    INTEGER, INTENT(IN) :: id

    DO WHILE (.NOT. done)
      CALL sleep_(1)
    END DO                     ! line 22
    ! !ibm* light_sync
    PRINT *, shared_result
  END SUBROUTINE
END MODULE

PROGRAM MAIN
  USE m, ONLY: waiter, setter
  USE f_pthread
  IMPLICIT NONE

  TYPE(f_pthread_t) threads(2)
  TYPE(f_pthread_attr_t) attr
  INTEGER(4) flag, result

  ! Initialization
  result = f_pthread_attr_init(attr)
  IF (result /= 0) ERROR STOP 1
  flag = FLAG_DEFAULT

  ! Create threads
  result = f_pthread_create(threads(1), attr, flag, waiter, 1)
  IF (result /= 0) ERROR STOP 2

  result = f_pthread_create(threads(2), attr, flag, setter, 2)
  IF (result /= 0) ERROR STOP 3

  result = f_pthread_join(threads(1))
  result = f_pthread_join(threads(2))
END PROGRAM
```

コンパイラーは、以下の通知メッセージを出します。

```
** m === End of Compilation 1 ===
** main === End of Compilation 2 ===

1586-669 (I) "sync.f", line 22: If this loop is used as a synchronization
point, additional synchronization via a directive or built-in function might
be needed.

1586-670 (I) "sync.f", line 13: If this statement is used as a synchronization
point, additional synchronization via a directive or built-in function might
be needed.

1501-510 Compilation successful for file sync.f.
```

次の関数 factorial.f は、n<=1 の場合は temp を初期化しません。n>1 の場合、factorial.f は設定される前の result にもアクセスします。 **-qnoopt** で **-qinfo=unset** が指定されている場合は、この問題は検知されません。factorial.f をコンパイルして、初期化されていない変数に関する通知メッセージを生成するには、次のコマンドを入力します。

```
xlf95 -qinfo=unset -O factorial.f
```

factorial.f には、次のコードが入っています。

```
module m
contains
  recursive function factorial(n) result(result)
    integer, value :: n
    integer result, temp

    if (n > 1) then
      temp = n * factorial(n - 1)
      print *, result ! line 9
    endif

    result = temp ! line 12
  end function
end module

use m
integer x
x = factorial(1)
end
```

コンパイラーは、以下の通知メッセージを出します。

```
1500-098 (I) "factorial.f", line 9: "result" is used before it is set.
1500-099 (I) "factorial.f", line 12: "temp" might be used before it is set.
1501-510  Compilation successful for file factorial.f.
```

関連情報

- 248 ページの『-qreport』
- 「*XL Fortran* ランゲージ・リファレンス」のisync
- 「*XL Fortran* ランゲージ・リファレンス」のlight_sync
- 同期および PowerPC ストレージ・モデルについて詳しくは、http://www.ibm.com/developerworks/systems/articles/powerpc.html の記事を参照してください。

-qinit

カテゴリー

言語エレメント制御

目的

ポインターの初期関連状況を関連付け解除にする。

このオプションは Fortran 90 以降に当てはまることに注意してください。

構文

@PROCESS:

```
@PROCESS INIT(F90PTR)
```

デフォルト

適用されません。

使用法

このオプションを使用して、ポインターを定義する前に使用することによって生じた問題の発見および修正を行うことができます。

関連情報

- 「*XL Fortran* ランゲージ・リファレンス」の 『ポインター関連付け』を参照してください。

-qinitalloc

カテゴリー

エラー・チェックおよびデバッグ

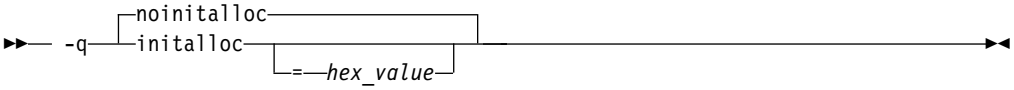
目的

デバッグ目的で、割り振られているが特定の値に初期化されていない割り振り可能な変数およびポインター変数を初期化します。

-qinitalloc オプションは、以下の初期化されていない変数に適用されます。

- **ALLOCATABLE** 属性を持ち、**ALLOCATE** ステートメントを使用して割り振られる変数
- **POINTER** 属性を持ち、**ALLOCATE** ステートメントを使用して割り振られる変数

構文



@PROCESS:

```
@PROCESS INITALLOC[(hex_value)] | NOINITALLOC
```

デフォルト

-qnoinitalloc

デフォルトでは、コンパイラーは割り振られたストレージを特定の値には初期化しません。

パラメーター

hex_value
1 から 8 桁の 16 進数。

- *hex_value* を指定しない場合、コンパイラーは割り振られたストレージの各バイトの値をゼロに初期化します。
- ストレージの各バイトを特定の値に初期化するには、 *hex_value* に 1 桁か 2 桁で指定してください。 1 桁だけを指定すると、コンパイラーは左側の *hex_value* にゼロを埋め込みます。

- ストレージの各ワードを特定の値に初期化するには、 *hex_value* に 3 桁から 8 桁で指定してください。 2 桁より大きいが、8 桁よりも少なく指定すると、コンパイラーは *hex_value* の左側にゼロを埋め込みます。
- ワードの初期化の場合、割り振り可能な変数の長さが 4 バイトの倍数でなければ、*hex_value* は適切な長さになるように、左側が切り捨てられます。例えば、割り振り可能な変数が 1 バイトのみである場合、*hex_value* に 5 桁で指定しても、コンパイラーは左側 3 桁を切り捨て、残りの右側 2 桁を変数に割り当てます。
- 英字桁の指定は、大文字でも小文字でも構いません。
- デフォルトの初期化が行われる派生型変数の場合、*hex_value* での初期化はデフォルトの初期化の前に実行されます。『例 2』を参照してください。

使用法

-qinitalloc オプションには、以下の利点があります。

- *hex_value* をゼロ (デフォルト値) に設定すると、割り振り可能な変数はすべて使用前にクリアされます。
- このオプションを使用して、実数型または複素数型の変数をシグナル NaN または静止 NaN に初期化できます。これは、プログラム内で未初期化の変数を検出するのに役立ちます。

このオプションの使用法は、**-qinitauto** オプションの使用法に類似しています。詳しくは、 182 ページの『-qinitauto』を参照してください。

制約事項**:** 等価なオブジェクト、構造体コンポーネント、そして配列エレメントは、別々に初期化されることはありません。代わりに、ストレージのシーケンス全体が集合的に初期化されます。

例

例 **1:**

-qinitalloc オプションの使用法の例を以下に示します。

```
SUBROUTINE Sub()
  REAL(4), ALLOCATABLE :: a, b
  CHARACTER, ALLOCATABLE :: c
  REAL(8), ALLOCATABLE :: d

  ALLOCATE(a)                     ! a is allocated but not initialized.
  ALLOCATE(b, SOURCE = 3.0)       ! b is allocated and initialized to 3.0.
  ALLOCATE(c)
  ALLOCATE(d)
END SUBROUTINE
```

例えば、**-qinitalloc=0cf** を指定してプログラムをコンパイルする場合、コンパイラーは以下の初期化を実行します。

- `0cf` に 5 個のゼロを埋め込み、`a` を `000000CF` に初期化します
- `b` の元の初期化を維持します
- `0cf` の 1 桁目を切り捨て、`c` を `CF` に初期化します
- `0cf` に 5 個のゼロを埋め込み、値を繰り返し、`d` を `000000CF000000CF` に初期化します。

例 **2:**

デフォルトの初期化が行われるコンポーネントが派生型に含まれる場合の **-qinitalloc** オプションの使用法の例を以下に示します。

```
TYPE dt
  INTEGER :: i = 1     ! i has default initialization
  INTEGER :: j
END TYPE
TYPE(dt), ALLOCATABLE :: dt1
ALLOCATE(dt1)
```

-qinitalloc を指定してプログラムをコンパイルする場合、コンパイラーは i のデフォルトの初期化を維持し、j をゼロに初期化します。

関連情報

- 「*XL Fortran* ランゲージ・リファレンス」の『**ALLOCATABLE**』属性
- 「*XL Fortran* ランゲージ・リファレンス」の『**ALLOCATE**』ステートメント
- 「*XL Fortran* ランゲージ・リファレンス」の『**POINTER**』ステートメント

-qinitauto

カテゴリー

エラー・チェックおよびデバッグ

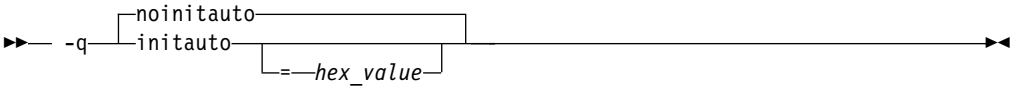
@PROCESS

なし。

目的

デバッグのために、未初期化の自動変数を特定の値に初期化する。

構文



デフォルト

-qnoinitauto

デフォルトでは、コンパイラーは自動ストレージの値を特定の値に初期化していません。しかし、ストレージの領域をすべてゼロで満たすことは可能です。

パラメーター

hex_value
1 から 8 桁の 16 進数。

- *hex_value* の数値を指定しない場合、コンパイラーは自動ストレージの各バイトの値をゼロに初期化します。

- ストレージの各バイトを特定の値に初期化するには、 *hex_value* に 1 桁か 2 桁で指定してください。 1 桁だけを指定すると、コンパイラーは左側の *hex_value* にゼロを埋め込みます。
- ストレージの各ワードを特定の値に初期化するには、 *hex_value* に 3 桁から 8 桁で指定してください。 2 桁より大きいが、8 桁よりも少なく指定すると、コンパイラーは *hex_value* の左側にゼロを埋め込みます。
- ワードの初期化の場合、自動変数の長さが 4 バイトの倍数でなければ、*hex_value* は適切な長さになるように、左側が切り捨てられる場合があります。例えば、*hex_value* に 5 桁で指定しても、自動変数の長さが 1 バイトである場合、コンパイラーは *hex_value* の左側 3 桁を切り捨て、その変数の右側に 2 桁を組み込みます。
- 英字桁の指定は、大文字でも小文字でも構いません。

使用法

このオプションにより、定義前に参照される変数を見つけることができます。例えば、**REAL** 変数を シグナル NAN 値に初期化するための **-qinitauto** オプションと、**-qflttrap** オプションの両方を使用することにより、実行時に初期化されていない **REAL** 変数を参照していないかどうかを識別することができます。

hex_value をゼロに設定すると、自動変数はすべて使用前にクリアされます。プログラムの中には、変数がゼロに初期化され、ゼロに初期化されないと機能しない、と想定するものがあります。また、最適化されなければ機能し、最適化されると障害が発生する、と想定するプログラムもあります。一般に、変数をすべてゼロ・バイトに設定すれば、そのような実行時エラーは回避されます。実行時エラーを回避するためには、このオプションに依存するよりも、ゼロへのリセットを必要とする変数を見つけてプログラムにコードを挿入するほうが良い方法です。このオプションを使用すると、通常、必要以上の数をゼロにするので、プログラムが遅くなる可能性があります。

それらのエラーを見つけて修正するには、正しくない結果が常に再現されるようバイトの値をゼロ以外に設定します。この方法は、デバッグ・ステートメントを追加したり、シンボリック・デバッガーにプログラムをロードしてエラーを排除する場合に、特に価値があります。

hex_value を **FF** (255) に設定すると、"負の非数値"、つまり 静止 NAN の初期値が **REAL** および **COMPLEX** 変数に与えられます。これらの変数で演算を行っても、結果は 静止 NAN 値になり、初期化されていない変数が計算で使用されたことが明らかになります。

このオプションは、サブプログラム内に初期化されていない変数を含んでいるプログラムをデバッグするときに役立ちます。例えば、シグナル NAN 値を使用して **REAL** 変数を初期化するときに使用できます。繰り返したときに倍精度の シグナル NAN 値を持つ 8 桁の 16 進数を指定することにより、8 バイトの **REAL** 変数を倍精度の シグナル NAN 値に初期化することができます。例えば、7FBFFFFF のような数値を指定することができます。これは、**REAL(4)** 変数に入れられると、単精度の シグナル NAN 値を持つことになります。 7FF7FFFF は、**REAL(4)** 変数に入れられると、単精度の 静止 NAN 値を持つことになります。 **REAL(8)** 変数に

同じ数値を 2 回入れる (7FF7FFFF7FF7FFFF) と、倍精度のシグナル NAN 値を持つようになります。

制約事項

同等な変数、構造体のコンポーネント、そして配列エレメントは、別々に初期化されることはありません。代わりに、ストレージのシーケンス全体が集合的に初期化されます。

例

次の例では、自動変数のワード初期化を実行する方法が示されています。

```
subroutine sub()
integer(4), automatic :: i4
character, automatic :: c
real(4), automatic :: r4
real(8), automatic :: r8
end subroutine
```

次のオプションを指定してコードをコンパイルする場合、*hex_value* が 2 桁より長くなったら、コンパイラーはワード初期化を実行します。

```
-qinitauto=0cf
```

コンパイラーは、i4、r4、および r8 変数の場合は、 *hex_value* にゼロを埋め込み、c 変数の場合は最初の 16 進数字を切り捨てることにより、変数を初期化します。

変数	値
i4	000000CF
c	CF
r4	000000CF
r8	000000CF000000CF

関連情報

- 157 ページの『-qfltttrap』
- 「*XL Fortran* ランゲージ・リファレンス」の**AUTOMATIC** 属性

-qinlglue

カテゴリー

オブジェクト・コード制御

目的

-O2 以上の最適化で使用すると、アプリケーション内の外部関数呼び出しを最適化するグルー・コードをインライン化する。

グルー・コード やプロシージャー・リンケージ・テーブル・コードは、リンカーによって生成され、2 つの外部関数の間で制御を渡すために使用されます。 **-qinlglue** が有効である場合、最適化プログラムは、パフォーマンスを向上させるためにグル

ー・コードをインライン化します。**-qnoinlglue** が有効である場合、グルー・コードのインライン化は行われません。

構文

@PROCESS:

@PROCESS **INLGLUE** | NOINLGLUE

デフォルト

- **-qinlglue**

使用法

グルー・コードをインライン化すると、コード・サイズが大きくなる場合があります。**-qcompact** を指定すると、コードの増大を防止するために **-qinlglue** 設定はオーバーライドされます。**-qinlglue** を有効にしたい場合は、**-qcompact** を指定しないでください。

-qnoinlglue または **-qcompact** を指定すると、パフォーマンスが低下する場合があります。これらのオプションは注意して使用してください。

関連情報

- 128 ページの『-qcompact』
- 292 ページの『-qtune』
- 「*XL Fortran* 最適化およびプログラミング・ガイド」のインライン化
- 「*XL Fortran* 最適化およびプログラミング・ガイド」のコード・サイズの管理

-qinline

カテゴリー

最適化およびチューニング

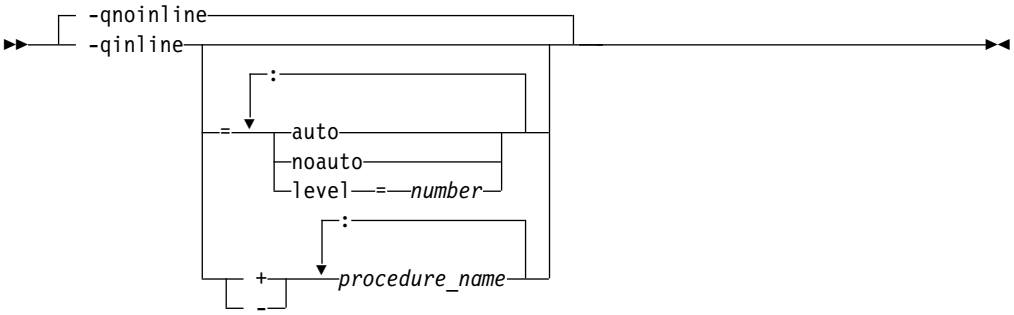
@PROCESS

なし。

目的

パフォーマンスを改善するために、プロシージャーへの呼び出しを生成する代わりに、それらのプロシージャーのインライン化を試行する。

構文



デフォルト

-qinline が指定されていない場合、デフォルト・オプションは以下のようになります。

- **-qnoinline** (**-O0** または **-qnoopt** の最適化レベル)
- **-qinline=noauto:level=5** (**-O2** の最適化レベル)
- **-qinline=auto:level=5** (**-O2 -qipa**、**-O3**、またはそれ以上の最適化レベル)

-qinline をサブオプションなしで指定する場合、デフォルト・オプションは **-qinline=auto:level=5** です。

パラメーター

auto | noauto
自動インライン化を有効または無効にします。オプション **-qinline=auto** が有効になっているときは、コンパイラーはすべてのプロシージャーをインライン化対象とみなします。オプション **-qinline=noauto** が有効になっているときは、コンパイラーは、インライン化に適しているとみなした小さなプロシージャーのみをインライン化します。

level=*number*
これは、インライン化の相対度数を示します。 *number* の値は 0 から 10 までの範囲にある整数でなければなりません。 *number* のデフォルト値は 5 です。 *number* の値が大きくなるほど、コンパイラーはより積極的なインライン化を行います。

procedure_name
procedure_name が **-qinline+** オプションの後に指定されている場合、指定されたプロシージャーはインライン化する必要があります。*procedure_name* が **-qinline-** オプションの後に指定されている場合、指定されたプロシージャーはインライン化してはなりません。

使用法

最適化レベル **-O2**、**-O3**、**-O4**、または **-O5** とともに **-qinline** を指定して、プロシージャーのインライン化を有効にすることができます。

-qinline が有効な場合、コンパイラーは、特定のプロシージャーをインライン化することによってパフォーマンスを向上可能かどうかを判別します。つまり、プロシージャーがインライン化に適切であるかどうかは、インライン化される呼び出し数の限度、および結果として生じるコード・サイズの増加量の限度という 2 つの要因

に左右されます。したがって、プロシージャーのインライン化を使用可能にしても、プロシージャーがインライン化されるとは保証されません。

インライン化によって必ずしもランタイムのパフォーマンスが向上するわけではないため、使用するコードでこのオプションの効果を検査する必要があります。再帰的または相互に再帰的なプロシージャーはインライン化しないでください。

-qinline+<*procedure_name***>** オプションまたは **-qinline-<***procedure_name***>** オプションを使用すれば、インライン化する必要がある/インライン化してはならないプロシージャーを指定できます。

-qnoinline を指定すると、**-qipa** オプションを指定した高水準最適化プログラムによって実行されたインライン化などのすべてのインライン化が使用不可になります。

デバッグ情報を生成する目的で **-g** オプションを指定すると、**-qinline** のインライン化効果が抑制される場合があります。

コード・サイズを増加させる最適化を避ける目的で **-qcompact** オプションを指定すると、**-qinline** のインライン化効果が抑制される場合があります。

デフォルトでは、**-qinline** は内部プロシージャーまたはモジュール・プロシージャーにのみ影響を及ぼします。さまざまなスコープでのプロシージャーの呼び出しのために、インライン拡張をオンにするには、**-qipa** オプションも使用する必要があります。

さまざまなコンパイル単位に適用される **@PROCESS** ディレクティブまたはコンパイル・オプションが競合すると、インライン化の効率に影響する場合があります。例えば、あるプロシージャーについてインライン化を指定した場合、一部の **@PROCESS** コンパイラー・ディレクティブが有効に実施されないことがあります。インライン化と IPA について詳しくは、「*XL Fortran* 最適化およびプログラミング・ガイド」を参照してください。

プロシージャーに対してインライン化を指定した場合、以下の @PROCESS コンパイラー指示である、ALIAS、ALIGN、ATTR、COMPACT、DBG、EXTCHK、EXTNAME、FLOAT、FLTTRAP、HALT、IEEE、LIST、MAXMEM、OBJECT、OPTIMIZE、PHSINFO、SPILLSIZE、STRICT、および XREF は、ファイル内の最初のコンパイル単位の前にある場合にのみ有効になります。

例

例 **1**

プロシージャーが一切インライン化されないように `myprogram.f` をコンパイルするには、以下のコマンドを使用します。

```
xlf myprogram.f -O2 -qnoinline
```

自動インライン化を使用可能にしたい場合、`auto` サブオプションを使用します。

```
-O2 -qinline=auto
```

6 から 10 までのインライン化レベルを指定して、より積極的な自動インライン化を実現できます。例を以下に示します。

```
-O2 -qinline=auto:level=7
```

自動インライン化が既にデフォルトで使用可能に設定されており、インライン化レベル 7 を指定したい場合、次のように入力します。

```
-O2 -qinline=level=7
```

例 **2**

myprogram.f に salary、taxes、expenses、および benefits プロシージャー がある場合、以下のコマンドを使用して myprogram.f をコンパイルし、これらのプロシージャーをインライン化します。

```
xlf myprogram.f -O2 -qinline+salary:taxes:expenses:benefits
```

プロシージャー salary、taxes、expenses、および benefits をインライン化せずに myprogram.f をコンパイルしたい場合、以下のコマンドを入力します。

```
xlf myprogram.f -O2 -qinline-salary:taxes:expenses:benefits
```

自動インライン化を使用不可にし、**-qinline+** オプションを指定することで、特定のプロシージャーをインライン化することもできます。次の例を検討してみます。

```
-O2 -qinline=noauto -qinline+salary:taxes:benefits
```

この場合、プロシージャー salary、taxes、および benefits がインライン化されます。 inline 指定子を使用して宣言されているプロシージャーもインライン化されます。他のプロシージャーはインライン化されません。

+ と **-** のサブオプションを互いに、または **-qinline** の他のサブオプションと混用することはできません。例えば、以下のオプションは、無効なサブオプションの組み合わせです。

```
-qinline+increase-decrease    // Invalid
-qinline=level=5+increase     // Invalid
```

ただし、複数の **-qinline** オプションを別々に使用することはできます。次の例を参照してください。

```
-qinline+increase -qinline-decrease -qinline=noauto:level=5
```

関連情報

- 90 ページの『-g』
- 192 ページの『-qipa』
- 「*XL Fortran* 最適化およびプログラミング・ガイド」の『プロシージャー間分析』
- -qinline インライン化オプション (「*XL Fortran* 最適化およびプログラミング・ガイド」)

-qintlog

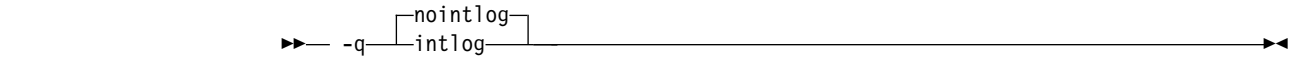
カテゴリー

浮動小数点および整数のコントロール

目的

式およびステートメントで整数と論理データ・エンティティーを混用できることを指定する。

構文



@PROCESS:

@PROCESS INTLOG | **NOINTLOG**

デフォルト

-qnointlog

使用法

-qintlog が指定されると、整数オペランドで指定する論理演算子は、それらの整数に対してビット単位で操作し、整数演算子は論理オペランドの内容を整数と見なします。

次の演算では、論理変数を使用することができません。
- **ASSIGN** 文変数
- 割り当てられた **GOTO** 変数
- **DO** ループ・インデックス変数
- **DATA** 文内の暗黙の **DO** ループ・インデックス変数
- 入出力コンストラクター内または配列コンストラクター内のいずれかでの暗黙の **DO** ループ・インデックス変数
- **FORALL** 構文内にあるインデックス変数

組み込み関数 **IAND**、**IOR**、**IEOR**、および **NOT** を使用して、ビット単位の論理演算を行うこともできます。

MOVE_ALLOC 組み込み関数は、1 つの整数と 1 つの論理引数を取ることはできません。

例

```
INTEGER I, MASK, LOW_ORDER_BYTE, TWOS_COMPLEMENT
I = 32767
MASK = 255
! Find the low-order byte of an integer.
LOW_ORDER_BYTE = I .AND. MASK
! Find the twos complement of an integer.
TWOS_COMPLEMENT = (.NOT. I) + 1
END
```

関連情報
- **-qport=clogicals** オプション。

-qintsize

カテゴリー

浮動小数点および整数のコントロール

目的

長さまたは種類が指定されていないデフォルトの **INTEGER** および **LOGICAL** データ・エンティティーのサイズを設定する。

このオプションは、データ・エンティティーのサイズを大きくするための一般的な方法として機能させるためのものではありません。用途は、他のシステム用に作成されたコードとの互換性を維持することに限定されています。

構文

@PROCESS:

@PROCESS INTSIZE(*bytes*)

デフォルト

-qintsize=4

パラメーター

bytes
許可されたサイズは 2、4、または 8 です。

使用法

このオプションは、データのデフォルト・サイズが異なるシステムから、プログラムを変更せずに移植できるようにするためのものです。例えば、16 ビットのマイクロプロセッサー用に書かれたプログラムには **-qintsize=2** が必要で、 CRAY コンピューター用に書かれたプログラムには **-qintsize=8** が必要です。

PARAMETER 文を追加して、引数として渡す定数に明示的な長さを指定する必要がある場合があります。

指定されたサイズ[1] 以下のようなデータ・エンティティーに適用されます。
- 長さまたは種類が指定されていない **INTEGER** および **LOGICAL** 仕様ステートメント。
- 長さまたは種類が指定されていない **FUNCTION** 文。

1. Fortran 90 または 95 の用語では、これらの値は *kind* 型付きパラメーターで参照されます。

- デフォルトの **INTEGER** 引数、**LOGICAL** 引数、戻り値などの授受を行う組み込み関数 (**INTRINSIC** 文に長さや種類が指定されていない場合)。指定されている長さまたは種類は、戻り値のデフォルト・サイズと一致しなければなりません。
- 暗黙の整数または論理値である変数。
- 種類が指定されていない整数および論理リテラル定数。指定されているバイト数で表せないほど値が長い場合は、コンパイラーは十分に長いサイズを選択します。 2 バイト整数の範囲は -(2**15) から 2**15-1、 4 バイト整数の範囲は -(2**31) から 2**31-1、 8 バイト整数の範囲は -(2**63) から 2**63-1 です。
- 整数または論理コンテキスト内の型なし定数。
- **-qintsize** は、**INTEGER** 型と **LOGICAL** 型に加えて、**vector(integer)** に対しても機能します。**-qintsize=2** を指定することは、**vector(integer*2)** を指定することと同等です。同様に、**-qintsize=4** を指定することは **vector(integer*4)** を指定することと同等です。**-qintsize=8** を指定することは、**vector(integer*8)** を指定することと同等です。

例

次の例を見れば、変数、リテラル定数、組み込み関数、算術演算子、入出力操作が、変更されたデフォルト整数サイズをどのように処理するかが理解できます。

```
@PROCESS INTSIZE(8)
      PROGRAM INTSIZETEST
      INTEGER I
      I = -9223372036854775807    ! I is big enough to hold this constant.
      J = ABS(I)                  ! So is implicit integer J.
      IF (I .NE. J) THEN
        PRINT *, I, '.NE.', J
      END IF
      END
```

次の例は、整数のデフォルト・サイズでのみ機能します。

```
      CALL SUB(17)
      END

      SUBROUTINE SUB(I)
      INTEGER(4) I                ! But INTSIZE may change "17"
                                  ! to INTEGER(2) or INTEGER(8).
      ...
      END
```

デフォルト値を変更する場合は、**INTEGER(4)** の代わりに **INTEGER** として I を宣言するか、以下のように、実引数に長さを指定する必要があります。

```
@PROCESS INTSIZE(8)
      INTEGER(4) X
      PARAMETER(X=17)
      CALL SUB(X)      ! Use a parameter with the right length, or
      CALL SUB(17_4)   ! use a constant with the right kind.
      END
```

関連情報

- 245 ページの『-qrealsize』
- 「*XL Fortran* ランゲージ・リファレンス」の『型宣言: 型パラメーターおよび指定子』

-qipa

カテゴリー

最適化およびチューニング

@PROCESS

なし。

目的

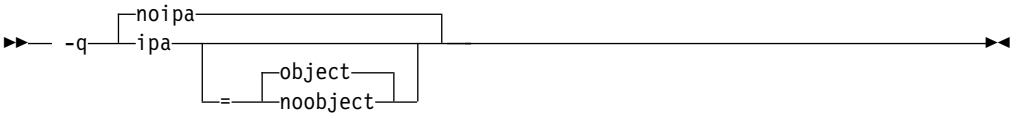
プロシージャー間分析 (IPA) と呼ばれる最適化のクラスを使用可能にしたり、カスタマイズする。

IPA は 2 ステップのプロセスです。コンパイル中に実行される最初のステップは、初期分析の実行およびプロシージャー間の分析情報のオブジェクト・ファイルへの格納で構成されます。リンク実行中に行われる 2 つ目のステップは、アプリケーション全体の完全な再コンパイルを促し、プログラム全体を最適化します。

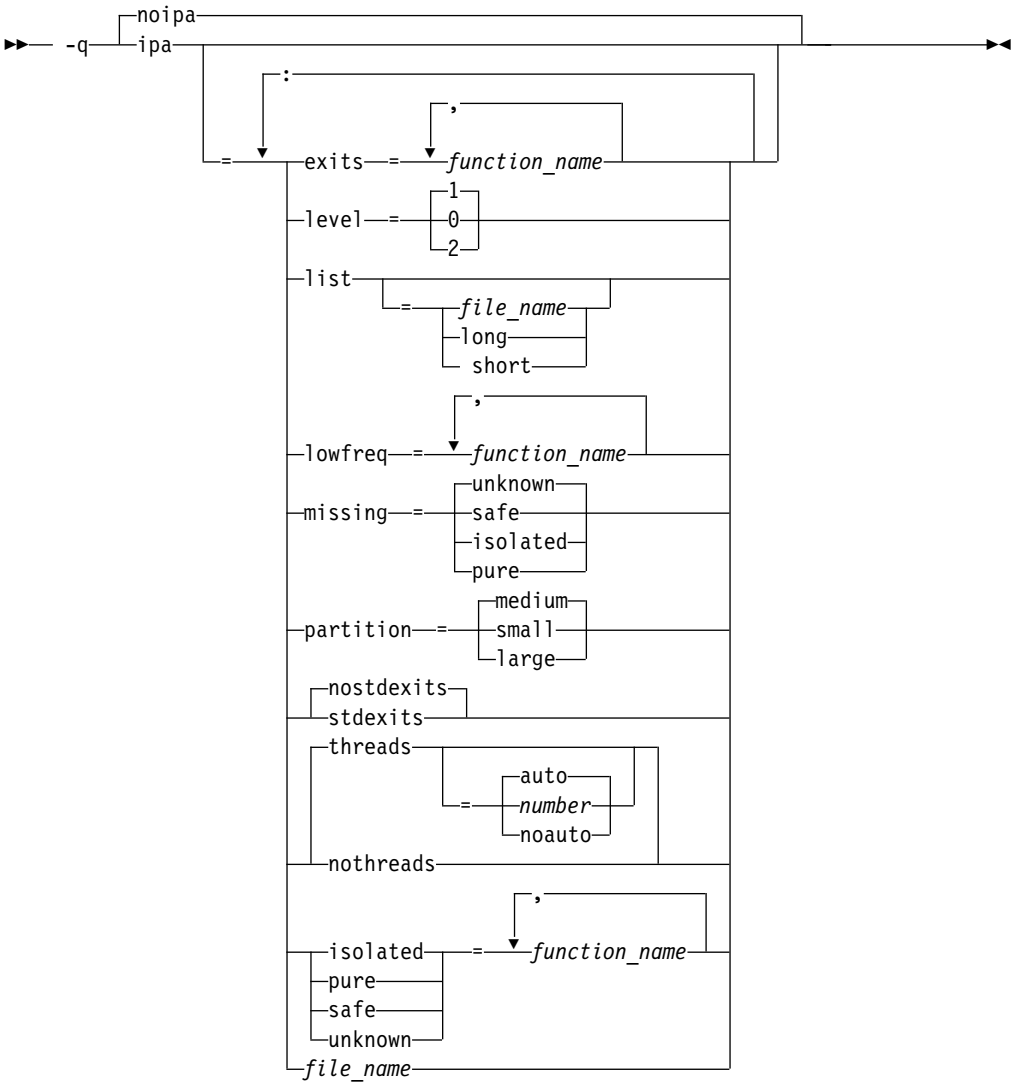
コンパイル・ステップまたはリンク・ステップ、あるいは両ステップの実行中に **-qipa** を使用できます。単一のコンパイラー呼び出しでコンパイルおよびリンクする場合、リンク時のサブオプションのみが関連性を持ちます。別のコンパイラー呼び出しでコンパイルおよびリンクする場合、コンパイル・ステップ実行中にはコンパイル時のサブオプションのみが関連性を持ち、リンク・ステップ実行中にはリンク時のサブオプションのみが関連性を持ちます。

構文

-qipa コンパイル時構文



-qipa リンク時構文



デフォルト

- **-qnoipa**

パラメーター

以下のパラメーターを指定できるのは、個別のコンパイル・ステップでのみです。

object | noobject
標準のオブジェクト・コードを出力オブジェクト・ファイルに入れるかどうかを指定します。

noobject を指定すると、最初の IPA フェーズ実行中にオブジェクト・コードが生成されないため、全体のコンパイル時間を大幅に短縮することができます。**noobject** で **-S** を指定する場合、**noobject** は無視されます。

コンパイルもリンクも同じステップで実行し、 **-S** もリスト・オプションもまったく指定されない場合は、 **-qipa=noobject** が暗黙指定されます。

コンパイル・ステップでサブオプションなしで **-qipa** を指定することは、**-qipa=object** と同等です。

以下のパラメーターは、同じコンパイラー呼び出しでのコンパイルおよびリンクの結合ステップで、あるいは単独のリンク・ステップでのみ指定できます。

exits
プログラム出口を表すプロシージャーの名前を指定します。プログラム出口とは、決して戻ることができず、また IPA パス 1 でコンパイルされたプロシージャーを呼び出すことができない呼び出しのことです。呼び出しはプログラムに戻らないため、コンパイラーはこれらのプロシージャーへの呼び出しを最適化できます (例えば、保存/復元シーケンスを除去して)。これらのプロシージャーは、**-qipa** でコンパイルされたプログラムの他の部分を呼び出すことはできません。

isolated
-qipa でコンパイルされないコンマ区切りされたプロシージャーリストを指定します。*isolated* または呼び出しチェーン内のプロシージャーとして指定するプロシージャーは、グローバル変数を直接参照できません。

level
プロシージャー間分析の最適化レベルを指定します。有効なサブオプションは、以下のサブオプションのいずれかです。

0 最小限のプロシージャー間分析および最適化しか行いません。

1 インライン化、限定された別名分析、および限定された呼び出し位置の調整を使用可能にします。

2 完全なプロシージャー間のデータ・フローおよび別名分析を実行します。

レベルを指定しないと、デフォルト値は 1 になります。

データ再編成情報を生成するには、**-qreport** とともに最適化レベル **-qipa=level=2** または **-O5** を指定します。IPA リンク・フェーズ時に、プログラム変数データのデータ再編成メッセージが、リスト・ファイルのデータ再編成セクションに書き込まれます。再編成には、共通ブロック分割、配列分割、配列転置、メモリー割り振りマージ、配列インターリービング、および配列合体が含まれます。

list
リンク・フェーズ中にリスト・ファイルを生成するように指定します。リスト・ファイルには、 IPA によって実行される変換および分析をはじめ、区画ごとのオプションのオブジェクト・リストに関する情報が入ります。

list_file_name を指定しないと、リスト・ファイル名はデフォルトで a.lst になります。リスト・ファイルを生成する他のオプションと一緒に **-qipa=list** を指定すると、IPA はすべての既存 a.lst ファイルを上書きする a.lst ファイルを生成します。a.f という名前のソース・ファイルがあれば、IPA リストは通常のコンパイラー・リスト a.lst を上書きします。**-qipa=list=***list_file_name* サブオプションを使用すると、代わりのリスト・ファイル名を指定できます。

その他のサブオプションは、以下のサブオプションのいずれかです。

short リスト・ファイルの必要情報が少なくなります。 リストのオブジェクト・ファイル・マップ、ソース・ファイル・マップ、およびグローバル・シンボル・マップの各セクションが生成されます。

long リスト・ファイルの必要情報が多くなります。 **short** サブオプションで

生成されるすべてのセクションに加えて、オブジェクト解決警告、オブジェクト参照マップ、インライナー・レポート、および区画マップのセクションが生成されます。

lowfreq
まれにしか呼び出されないと考えられるプロシージャーを指定します。これらは一般的に、エラー処理、トレース、または初期化のプロシージャーです。これらのプロシージャーの呼び出しの最適化を軽くすることによって、コンパイラーが、プログラムのその他の部分の実行を高速化できる場合があります。

missing
-qipa でコンパイルされず、**unknown**、**safe**、**isolated**、および **pure** サブオプションのいずれによっても明示的に指定されていないプロシージャーのプロシージャー間の振る舞いを指定します。

有効なサブオプションは、以下のサブオプションのいずれかです。

safe これは、指定されていないプロシージャーが、明示された (指定された) 関数を、直接呼び出しまたはプロシージャー・ポインターによって間接的に呼び出すことがないように指定します。

isolated
欠落プロシージャーが、可視のプロシージャーにアクセスできるグローバル変数を直接参照しないことを指定します。共有ライブラリーからバインドされたプロシージャーは分離された ものと見なされます。

pure 欠落プロシージャーは、安全 *(safe)* かつ分離されて おり、可視のプロシージャーへアクセスできるストレージを間接的に変更しないことを指定します。また、純粋な *(pure)* プロシージャーには、監視できる内部状態はありません。

unknown
欠落プロシージャーが安全 *(safe)*、分離された、または純粋な *(pure)* ものとして認識されないことを指定します。このサブオプションは、欠落プロシージャーの呼び出しに対するプロシージャー間の最適化の量を大幅に制限します。

デフォルトでは **unknown** と想定されます。

partition
受け渡し 2 の間に IPA によって作成された各プログラム区画のサイズを指定します。有効なサブオプションは、以下のサブオプションのいずれかです。

- **small**
- **medium**
- **large**

より大きな区画には、より多くのプロシージャーを組み込めるため、プロシージャー間分析が向上しますが、最適化するにはさらに大きなストレージが必要になります。ページングのためにコンパイルに時間がかかりすぎる場合は、区画サイズを縮小してください。

pure
-qipa でコンパイルされない純粋な *(pure)* プロシージャーを指定します。*pure* として指定されるプロシージャーは、*isolated* および *safe* でなくてはなりませ

ん。また、内部状態を変更したり、呼び出し元から可視のデータを潜在的に変更するよう定義されている副次作用があってはいけません。

safe
-qipa を使用してコンパイルされず、プログラムの他の部分を呼び出さない安全な *(safe)* プロシージャーを指定します。安全なプロシージャーは、グローバル変数および仮引数を変更できますが、**-qipa** でコンパイルされたプロシージャーを呼び出すことはできません。

stdexits | nostdexits
特定の事前定義ルーチンを、**exits** サブオプションを使用したように最適化できることを指定します。プロシージャーは、abort、exit、_exit、および _assert です。

threads | nothreads
パス 2 中の IPA 最適化プロセスの一部を並列スレッドで実行します。これによって、マルチプロセッサー・システムのコンパイル・プロセスを高速化できます。 **threads** サブオプションの有効なサブオプションは次のとおりです。

auto | noauto
auto が有効な場合、コンパイラーは、マシン負荷に基づいてヒューリスティックに複数のスレッドを選択します。 **noauto** が有効な場合、コンパイラーは、マシン・プロセッサー 1 つにつき 1 つのスレッドを作成します。

number
特定数のスレッドを使用するようコンパイラーに命令します。 *number* 1 から 32 767 までの範囲の整数値に設定できます。ただし、*number* は事実上、システムで使用可能なプロセッサーの数に限定されます。

サブオプションなしの **threads** は **-qipa=threads=auto** を暗黙指定します。

unknown
-qipa でコンパイルされない不明な *(unknown)* プロシージャーを指定します。不明 *(unknown)* として指定されたプロシージャーは、**-qipa** でコンパイルされたプログラムの他の部分を呼び出したり、グローバル変数や仮引数を変更できます。

file_name
特別な形式のサブオプション情報を含むファイルの名前を指定します。

ファイル・フォーマットは以下のとおりです。

```
# ... comment
attribute{, attribute} = name{, name}

missing = attribute{, attribute}
exits = name{, name}
lowfreq = name{, name}
list [ = file-name | short | long ]
level = 0 | 1 | 2
partition = small | medium | large
```

ここで、*attribute* は以下のいずれかです。
- exits
- lowfreq
- unknown
- safe

- isolated
- pure

使用法

-qipa を指定すると、最適化レベルが **-O2** に自動的に設定されます。パフォーマンスをさらに改善するために、**-qinline** オプションを指定することもできます。**-qipa** オプションは、最適化の実行中に検査される領域、単一プロシージャーから複数プロシージャー (ソース・ファイルが異なる可能性あり) へのインライン化とそれらの間のリンクを継承します。

-qipa とリンクして使用されるオブジェクト・ファイルが、**-qipa=noobject** オプションを指定して作成されると、エントリー・ポイントを含むファイルは (実行可能プログラムの場合は主プログラム、またはライブラリー場合はエクスポートされた関数) **-qipa** を指定してコンパイルされる必要があります。

異なるリリースのコンパイラーで作成されたオブジェクトをリンクすることはできますが、新しいリリースのコンパイラーが使用されて、リンクされたオブジェクトを作成しているので、少なくとも同じリリース・レベルのリンカーを使用していることを確認する必要があります。

明らかに参照されていたり、ソース・コードで設定されているシンボルは、IPA によって最適化される可能性があり、**debug** または **nm** 出力へと見失う可能性があります。通常、**-g** コンパイラーとともに IPA を使用すると、非ステップ可能出力となります。

-# を使用して **-qipa** を指定すると、コンパイラーは IPA リンク・ステップに続くリンカー情報を表示しません。

-qipa の使用に関する推奨手順については、「*XL Fortran* 最適化およびプログラミング・ガイド」の『アプリケーションの最適化』を参照してください。

例

以下の例で、ファイル・セットをプロシージャー間分析でコンパイルする方法を示します。

```
xlf -c *.f -qipa
xlf -o product *.o -qipa
```

同じファイルのセットをコンパイルして、2 番目のコンパイルの最適化と最初のコンパイル・ステップの速度を改善する方法を以下に示します。ルーチン・セット `user_trace1`、`user_trace2`、および `user_trace3` が存在すると想定します。これらはほとんど実行されることがなく、`user_abort` ルーチンがプログラムを終了します。

```
xlf95 -c *.f -qipa=noobject
xlf95 -c *.o -qipa=lowfreq=user_trace[123]:exit=user_abort
```

関連情報

- -qinline
- 202 ページの『-qlibmpi』
- 317 ページの『-S』

- 環境変数の正しい設定方法

-qkeepparm

カテゴリー

エラー・チェックおよびデバッグ

@PROCESS

なし。

目的

-O2 以上の最適化で使用した場合に、プロシージャー・パラメーターをスタックに保管するかどうかを指定します。

プロシージャーは通常、着信パラメーターを入り口点のスタックに格納します。ただし、最適化オプションを使用可能にしてコードをコンパイルすると、最適化プログラムは、最適化が必要であれば、これらのパラメーターをスタックから除去する可能性があります。

構文

デフォルト

-qnokeepparm

使用法

-qkeepparm が有効な場合、最適化が有効であってもパラメーターがスタックに保管されます。このオプションは、さらに、着信パラメーターの値をスタックにただ保存しておくことにより、デバッガーなどのツールに送られる着信パラメーターの値を使用できるようにします。ただし、これは実行のパフォーマンスにマイナスの影響を与える可能性があります。

-qnokeepparm が有効な場合、最適化に効果的であれば、パラメーターはスタックから除去されます。

-qlanglvl

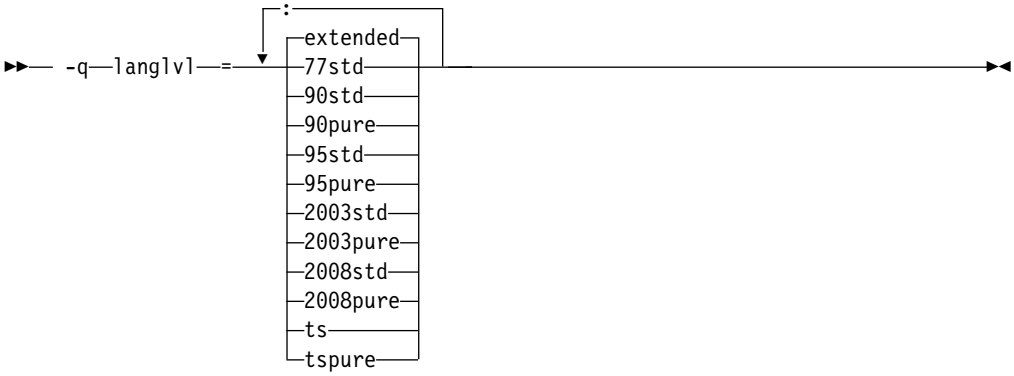
カテゴリー

言語エレメント制御

目的

非適合の検査を行う言語標準 (または標準のスーパーセットまたはサブセット) を決定する。これは、非準拠のソース・コードと、そのような非準拠を許容するオプションを識別します。

構文



@PROCESS:

@PROCESS LANGLVL({*suboption*})

デフォルト

-qlanglvl=extended

パラメーター

77std ANSI FORTRAN 77 標準で指定されている言語を受け入れて、他はすべて言語固有メッセージを使用して報告します。

90std ISO Fortran 90 標準で指定されている言語を受け入れて、他はすべて言語固有メッセージを使用して報告します。

90pure
廃止 Fortran 90 機能が使用されたことに対して言語レベル・メッセージを発行する以外は、**90std** と同じです。

95std ISO Fortran 95 標準で指定されている言語を受け入れて、他はすべて言語固有メッセージを使用して報告します。

95pure
廃止 Fortran 95 機能が使用されたことに対して言語レベル・メッセージを発行する以外は、**95std** と同じです。

2003std
ISO Fortran 2003 標準で指定されている言語を受け入れて、他はすべて言語固有メッセージを使用して報告します。

2003pure
廃止 Fortran 2003 機能が使用されたことに対して言語レベル・メッセージを発行する以外は、**2003std** と同じです。

2008std
ISO Fortran 2003 標準で指定されている言語、および XL Fortran がサポートするすべての Fortran 2008 機能を受け入れて、他はすべて言語レベル・メッセージを使用して報告します。

2008pure
廃止 Fortran 2008 機能が使用されたことに対して言語レベル・メッセージを発行する以外は、**2008std** と同じです。

ts これは、標準技術仕様で指定される言語を受け入れて、それ以外のものは言語レベル・メッセージにより報告します。技術仕様 29113 は、最新 Fortran 標準を補足するものです。このオプションを指定すると、ISO Fortran 2003 標準で指定されている言語、すべてのサポートされている ISO Fortran 2008 標準機能、およびすべてのサポートされている ISO TS 29113 機能がチェックされます。

tspure
これは、廃止された技術仕様 29113 機能が使用されていることに対しても言語レベル・メッセージを発行すること以外は、**ts** と同じです。

extended
言語レベルのチェックを効率的にオフにして、完全な Fortran 2003 言語標準、XL Fortran が提供するすべての Fortran 2008 機能、およびすべての拡張機能を受け入れます。

使用法

-qlanglvl 設定が指定されている場合、コンパイラーは、指定された言語レベルで許可されていない構文を検出すると、重大度コード **L** のメッセージを発行します。

-qflag オプションは、**-qlanglvl** オプションをオーバーライドすることができます。

45 ページの『実行時オプションの設定』に記載されている **langlvl** 実行時オプションは、コンパイル時にチェックできない実行時拡張機能を見つけるのに役立ちます。

例

次の例では、Fortran 標準の組み合わせに準拠するソース・コードが示されています。

```
!------------------------------------------------------------
! in free source form
program tt
    integer :: a(100,100), b(100), i
    real :: x, y
    ...
    goto (10, 20, 30), i
10 continue
    pause 'waiting for input'

20 continue
    y= gamma(x)

30 continue
    b = maxloc(a, dim=1, mask=a .lt 0)
```

```
end program
!------------------------------------------------------
```

下の表には、特定の **-qlanglvl** サブオプションがこのサンプル・プログラムに与える影響についての例が示されています。

指定した **-qlanglvl** サブオプション	結果	理由
95pure	**PAUSE** 文にフラグを付ける 計算型 **GOTO** にフラグを付ける ステートメント **GAMMA** 組み込み関数にフラグを付ける	Fortran 95 で削除された機能 Fortran 95 で廃止された機能 Fortran 95 への拡張機能
95std	**PAUSE** 文にフラグを付ける **GAMMA** 組み込み関数にフラグを付ける	Fortran 95 で削除された機能 Fortran 95 への拡張機能
extended	フラグを付けられるエラーはなし	

関連情報

- 151 ページの『-qflag』
- 167 ページの『-qhalt』
- 251 ページの『-qsaa』
- 45 ページの『実行時オプションの設定』の **langlvl** 実行時オプション

-qlibansi

カテゴリー

最適化およびチューニング

@PROCESS

なし。

目的

ANSI C ライブラリー関数の名前のすべての関数が、実際にライブラリー関数であり、異なるセマンティクスを持つユーザー関数でないことを想定する。

構文

デフォルト

-qnolibansi

使用法

このオプションを使用すると、最適化プログラムは副次作用があるかどうかなど特定の関数の振る舞いについて認識できるようになるので、より優れたコードを生成できます。

注**:** このオプションは、アプリケーションに標準のライブラリー関数と非互換のユーザー独自バージョンのライブラリー関数が含まれる場合には使用しないでください。

関連情報

192 ページの『-qipa』を参照してください。

-qlibmpi

カテゴリー

74 ページの『最適化およびチューニング』

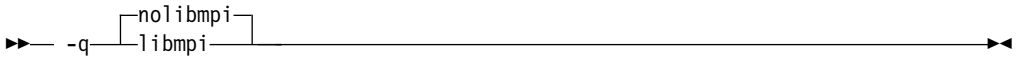
@PROCESS

なし

目的

Message Passing Interface (MPI) 名を持つすべての関数が事実上 MPI 関数であり、異なるセマンティクスを持つユーザー関数でないことを表明する。

構文



デフォルト

-qnolibmpi

使用法

MPI は、メッセージ引き渡しのためのライブラリー・インターフェース仕様です。この仕様は、あるプロセスのアドレス・スペースから別のプロセスに連携操作を介してデータが移動する、メッセージ引き渡し並列プログラミング・モデルに対応します。MPI の詳細については、Message Passing Interface Forum にアクセスしてください。

-qlibmpi により、コンパイラーは、所定の関数が何らかの副次作用を持っているかいないかなどの動作について知るようになるので、より良いコードを生成することができます。

-qlibmpi を使用すると、コンパイラーは MPI ライブラリー関数の名前を持つすべての関数が実際には MPI 関数であると想定します。**-qnolibmpi** ではそのような想定は行われません。

注: お客様のアプリケーションに、標準のライブラリー関数とは非互換のお客様の自身のバージョンのライブラリー関数が含まれている場合、このオプションを使用することはできません。

例

myprogram.f をコンパイルするには、以下のコマンドを入力します。

```
xlf -O5 myprogram.f -qlibmpi
```

関連情報

- Message Passing Interface Forum
- 192 ページの『-qipa』

-qlinedebug

カテゴリー

エラー・チェックおよびデバッグ

目的

デバッガー用に行番号およびソース・ファイル名の情報のみを生成する。

-qlinedebug が有効な場合、コンパイラーは最小限のデバッグ情報しか生成しないため、結果として得られるオブジェクト・サイズは、**-g** デバッグ・オプションを指定した場合に生成されるオブジェクトよりも小さくなります。 デバッガーを使用してソース・コードをステップスルーできますが、可変情報を見たり、照会することはできません。トレースバック・テーブルには、生成された場合、行番号が組み込まれます。

-qlinedebug は **-g1** と同等です。

構文

@PROCESS:

@PROCESS LINEDEBUG | **NOLINEDEBUG**

デフォルト

-qnolinedebug

使用法

-qlinedebug が有効な場合、関数のインライン化は使用不可になります。

すべてのデバッグ情報と同様に、**-qlinedebug** の出力は、コードが最適化されると、不完全になったり、誤った認識を招く場合があります。

-g オプションは、**-qlinedebug** オプションをオーバーライドします。コマンド行に **-g**with**-qnolinedebug** を指定した場合は、**-qnolinedebug** が無視され、警告が出されます。

関連情報

- 90 ページの『-g』
- 101 ページの『-O』

-qlist

カテゴリー

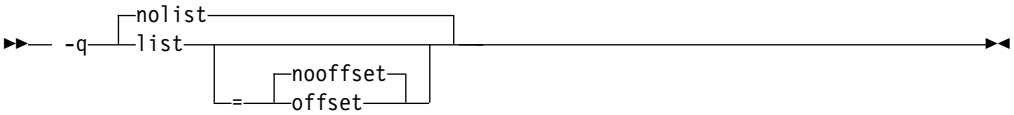
リスト、メッセージ、およびコンパイラー情報

目的

これは、オブジェクトおよび定数域セクションが含まれるコンパイラー・リスト・ファイルを生成します。

-qlist が有効であるとき、コマンド行で指定された各ソース・ファイルごとに、リスト・ファイルが **.lst** サフィックス付きで生成されます。

構文



@PROCESS:

@PROCESS LIST[([NO]OFFSET)] | **NOLIST**

デフォルト

-qnolist

パラメーター

offset | nooffset

-qlist=offset が有効なとき、リストはコード生成の開始からのオフセットよりもむしろプロシージャーの開始からのオフセットを示します。このサブオプションにより、**.lst** ファイルを読み込むすべてのプログラムは PDEF の値および問題の行を追加することができ、**offset** または **nooffset** が指定されたかを示す同一の値を生産します。

コンパイル単位に複数のプロシージャーがある場合にのみ **offset** サブオプションは関係があります。例えば、このようなことは、ネストされたプロシージャーがプログラムにおいて使用されている場合に発生する可能性があります。

サブオプションなしで **-qlist** を指定することは、**-qlist=nooffset** と同じです。

使用法

オブジェクト・リストを使用すると、生成コードのパフォーマンス特性の理解、および実行時の問題の診断に役立ちます。

-qipa を指定して、IPA リスト・ファイルを生成する場合は、**-qipa=list=*filename*** サブオプションを使用して、代替リストを取得します。

-qnoprint コンパイラー・オプションは、このオプションをオーバーライドします。

関連情報

- 72 ページの『リスト、メッセージ、およびコンパイラー情報』
- 357 ページの『オブジェクト・セクション』
- 219 ページの『-qnoprint』
- 317 ページの『-S』
- 「*XL Fortran* ランゲージ・リファレンス」の『プログラム単位およびプロシージャー』

-qlistfmt

カテゴリー

リスト、メッセージ、およびコンパイラー情報

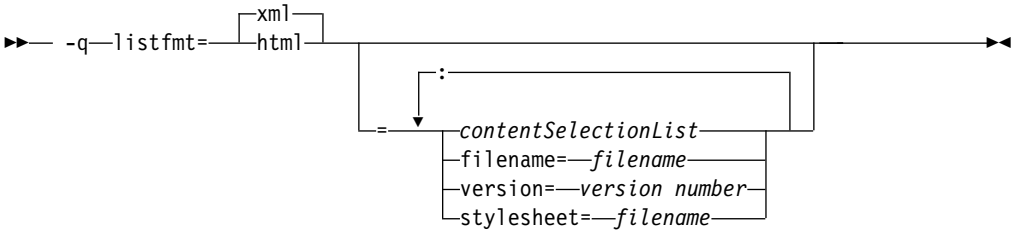
@PROCESS

なし。

目的

これは、最適化の機会を見いだすときに役立つレポートを XML 形式または HTML 形式で作成します。

構文



デフォルト

このオプションは、デフォルトではオフです。いずれの *contentSelectionList* サブオプションも指定されていない場合は、使用可能なすべてのレポート情報が生成されます。例えば、**-qlistfmt=xml** の指定は、**-qlistfmt=xml=all** と等価です。

パラメーター

以下のリストで **-qlistfmt** のパラメーターを説明します。

xml | html
コンパイラーは XML 形式または HTML 形式でレポートを生成するようになります。 XML レポートが前に生成されている場合は、**genhtml** コマンドを使用してそのレポートを HTML フォーマットに変換できます。このコマンドについては詳しくは、 328 ページの『genhtml』を参照してください。

contentSelectionList
以下のサブオプションは、レポート内の情報のタイプと量を制限するためのフィルターを提供します。

data | nodata
データ再編成情報を生成します。

inlines | noinlines
インライン化情報を生成します。

pdf | nopdf
Profile-Directed Feedback 情報が生成されます。

transforms | notransforms
ループ変換情報を生成します。

all
入手可能なすべてのレポート情報を生成します。

none
レポートを生成しません。

filename
レポート・ファイルの名前を指定します。コンパイル・フェーズで 1 つのファイルが生成され、IPA リンク・フェーズで 1 つのファイルが生成されます。ファイル名を指定しなかった場合は、そのプラットフォームの名前生成規則に整合する方法で、接尾部 `.xml` または `.html` が付いたファイルが生成されます。例えば、`foo.f` ファイルがコンパイルされる場合、コンパイル・ステップで `foo.xml` という XML ファイルが生成され、リンク・ステップで `a.xml` という XML ファイルが生成されます。

注: コンパイルとリンクを 1 つのステップで行い、このサブオプションを使用してレポートのファイル名を指定すると、IPA リンク・ステップからの情報は、コンパイル・ステップで生成された情報を上書きします。

`filename` サブオプションを使用して複数のファイルをコンパイルする場合にも同じことが当てはまります。コンパイラーは個々のファイルにレポートを作成するため、最後にコンパイルされたファイルのレポートが前のレポートを上書きします。例を以下に示します。

```
xlf -qlistfmt=xml=all:filename=abc.xml -O3 myfile1.f myfile2.f myfile3.f
```

これは、最後のファイル `myfile3.f` のコンパイルに基づいた `abc.xml` レポートのみを生成します。

stylesheet
結果のレポート内に埋め込まれる xml-stylesheet ディレクティブの対象となる、既存の XML スタイルシートの名前を指定します。デフォルトの振る舞いでは、スタイルシートは含まれません。XL Fortran に付属のスタイルシートは xlstyle.xsl です。このスタイルシートでは、XSLT をサポートするブラウザーで XML レポートが表示されるときに、その XML レポートが読みやすい形式に変換されます。

stylesheet サブオプションを使用して作成された XML レポートを表示するには、**stylesheet** サブオプションで指定されたパスに実際のスタイルシート (xlstyle.xsl)、および XML メッセージ・カタログ (XMLMessages-*locale*.xml: *locale* はコンパイル・マシン上で設定されているロケールを指します) を配置する必要があります。そのスタイルシートおよびメッセージ・カタログは /opt/ibm/xlf/16.1.0/listings/ ディレクトリーにインストールされます。

例えば、**stylesheet=xlstyle.xsl** を指定して a.xml を生成した場合に、ブラウザーで a.xml を正しく表示するには、xlstyle.xsl と XMLMessages-*locale*.xml の両方を a.xml と同じディレクトリーに配置しておく必要があります。

version
これは、生成されるコンテンツのメジャー・バージョンを指定します。当該レポートの特定バージョンを必要とするツールを作成した場合は、そのバージョンを指定する必要があります。

例えば、IBM XL Fortran for Linux, V16.1 は XML v1.1 のレポートを作成します。そのようなレポートを使用するツールを作成した場合は、version=v1 を指定してください。

使用法

-qlistfmt オプションによってレポートに生成される情報は、プログラムのコンパイルに使用される最適化オプションによって異なります。

- **-qlistfmt** と、インライン化を有効にするオプション (**-qinline** など) を両方指定した場合、レポートには、インライン化された関数が示されるほか、他の関数がインライン化されなかった理由も示されます。
- **-qlistfmt** と、ループのアンロールを有効にするオプションを両方指定した場合は、レポートには、プログラム・ループがどのように最適化されたのかについての概要が含まれます。また、レポートには、特定のループをベクトル化できない理由に関する診断情報も組み込まれます。 **-qlistfmt** でループ変換に関する情報を生成するには、さらに以下のオプションを少なくとも 1 つ指定する必要があります。
  - **-qhot**
  - **-qsmp**
  - **-O3** 以上
- **-qlistfmt** と、並列変換を有効にするオプションを両方指定した場合、レポートには、並列変換に関する情報が含まれます。また、**-qlistfmt** で、並列変換または並列パフォーマンス・メッセージに関する情報を生成するには、以下のオプションの中から少なくとも 1 つを指定する必要があります。
  - **-qsmp**

  - **-O5**
  - **-qipa=level=2**
- **-qlistfmt** と、プロファイル作成を有効にする **-qpdf** を両方指定した場合、レポートには、呼び出し数、ブロック数、およびキャッシュ・ミスに関する情報が含まれます。
- **-qlistfmt** と、データ再編成情報を生成するオプション (**-qipa=level=2** など) を両方指定した場合、レポートには、そのデータ再編成情報に関する情報が含まれます。

例

myprogram.f をコンパイルして、ループの最適化方法を示す XML レポートを生成するには、以下のように入力します。

```
xlf -qhot -O3 -qlistfmt=xml=transforms myprogram.f
```

myprogram.f をコンパイルして、インライン化された関数を示す XML レポートを生成するには、以下を入力します。

```
xlf -qinline -qlistfmt=xml=inlines myprogram.f
```

関連情報

- 248 ページの『-qreport』
- 328 ページの『genhtml』
- 「*XL Fortran* 最適化およびプログラミング・ガイド」の 『コンパイラー・レポートを使用した最適化の機会の診断』

-qlistopt

カテゴリー

リスト作成、メッセージ、およびコンパイラー情報

目的

コンパイラー呼び出し時に有効なすべてのオプションを含むコンパイラー・リスト・ファイルを作成する。

listopt が有効であるとき、コマンド行で指定された各ソース・ファイルごとに、リスト・ファイルが .lst サフィックス付きで生成されます。リストは、コンパイラーのデフォルト設定、構成ファイル、およびコマンド行設定で設定された、有効なオプションを示しています。

構文

@PROCESS:

@PROCESS LISTOPT | **NOLISTOPT**

デフォルト

-qnolistopt

使用法

このオプション・リストは、デバッグ中に使用すると、コンパイラー・オプションの特定の組み合わせにおいて問題が起きるかどうかをチェックすることができます。また、パフォーマンス・テスト中に使用すると、特定のコンパイルに対して有効な最適化オプションを記録することができます。

リストに常に表示されるオプションは次のとおりです。
- デフォルト時にオンであるすべての "on/off" オプション。例えば、**-qobject**。
- 構成ファイル、コマンド行オプション、**@PROCESS** ディレクティブなどで明示的にオフになるすべての "on/off" オプション。
- 任意の数値引数 (通常はサイズ) をとるすべてのオプション。
- 複数のサブオプションを持つすべてのオプション。

-qnoprint コンパイラー・オプションは、このオプションをオーバーライドします。

関連情報
- 72 ページの『リスト、メッセージ、およびコンパイラー情報』
- 352 ページの『オプション・セクション』

-qlog4

カテゴリー

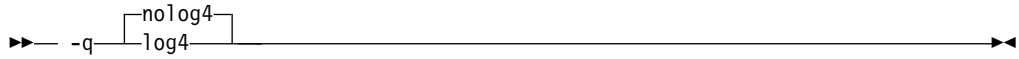
移植性とマイグレーション

目的

論理オペランドを使用する論理演算の結果が **LOGICAL(4)** であるか、または最大長のオペランドを使用する **LOGICAL** であるかを指定する。

このオプションを使用すると、元々 IBM VS FORTRAN コンパイラー用に書かれたコードを移植することができます。

構文



@PROCESS:

@PROCESS LOG4 | **NOLOG4**

デフォルト

-qnolog4 (オペランドの長さに応じて結果を作成)。

使用法

-qlog4 を指定すると、結果が **LOGICAL(4)** になります。

論理値のデフォルト・サイズを変更するのに **-qintsize** を使用する場合、**-qlog4** は無視されます。

-qmakedep

カテゴリー

1.

2.

出力制御

@PROCESS

なし。

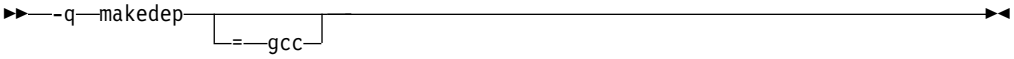
目的

make コマンドの記述ファイルの組み込みに適したターゲットを含む従属関係出力ファイルを生成する。

従属関係出力ファイルには .d サフィックス付きの名前が付けられます。

-qmakedep は **-MMD** の長い形式です。

構文



デフォルト

適用されません。

パラメーター

gcc
生成された **make** 規則のフォーマットで、GCC フォーマットに一致します。従属関係出力ファイルには、ソース・ファイルの従属関係をすべてリストした単一ターゲットが組み込まれます。

サブオプションなしで **-qmakedep** を指定すると、従属関係出力ファイルはソース・ファイルの従属関係それぞれに対して個別に規則を指定します。

使用法

make コマンドは、従属情報を使用してファイルのコンパイル順序を決定します。また、**make** は、ファイルが変更された場合に再コンパイルが必要な最小限のファイル・セットを決定する際にも従属情報を使用します。

XL Fortran は、以下のタイプのソース・ファイル従属関係を認識します。

- C プリプロセッサーの #include ディレクティブによって組み込まれるファイルに対する従属関係。
- Fortran プリプロセッサーの INCLUDE ディレクティブによって組み込まれるファイルに対する従属関係。
- 1 つ以上の Fortran モジュールを使用または拡張するファイル内のモジュール・シンボル・ファイルに対する従属関係。
- F2008 1 つ以上の Fortran サブモジュールを使用または拡張するファイル内のモジュール・シンボル・ファイルに対する従属関係。 F2008

コマンド・ラインで指定された を含むソース・ファイルごとに、オブジェクト・ファイルと同じ名前の従属関係出力ファイルが生成されます。ただし、その出力ファイルのサフィックスは .d になります。その他のタイプの入力ファイルに対しては、従属関係出力ファイルは作成されません。**-o** オプションを使用してオブジェクト・ファイルの名前を変更する場合、従属関係出力ファイルの名前は **-o** オプションで指定した名前に基づきます。詳しくは、『例』のセクションを参照してください。

これらのオプションを指定して生成された従属関係出力ファイルは、**make** 記述ファイルではありません。これらのファイルを **make** コマンドで使用するには、リンクさせる必要があります。このコマンドについて詳しくは、ご使用のオペレーティング・システムの資料を参照してください。

-qfullpath オプションも指定されている場合、ソースおよびインクルード・ファイルの絶対パス名は、従属関係出力ファイルに記録されます。

-qmakedep は、以下のオプションと共に使用することもできます。

-MF *file_path*
従属関係出力ファイルの名前を設定します。ここで *file_path* は従属関係出力ファイルの部分パスまたは完全なパス、あるいはファイル名です。詳しくは、 97 ページの『-MF』を参照してください。

-MT *target*
生成された従属関係ファイル内の **make** 規則のオブジェクト・ファイルのターゲット名を指定します。詳しくは、 99 ページの『-MT』を参照してください。

例

例 **1:** mysource.f をコンパイルして、mysource.d という名前の従属関係出力ファイルを作成するには、以下を入力します。

```
xlf -c -qmakedep mysource.f
```

例 **2:** source.f をコンパイルして、object.o という名前のオブジェクト・ファイル、および object という名前の従属関係出力ファイルを作成するには、以下を入力します。d の場合、以下を入力します。

```
xlf -c -qmakedep source.f -o object.o
```

例 **3:** 現行作業ディレクトリーに以下のファイルがあるとします。

- options.h

- constants.h
- n.F
- m.f

options.h ファイルには、次のコードが入っています。

```
@PROCESS free(f90)
```

constants.h ファイルには、次のコードが入っています。

```
real(4), parameter :: pi = 3.14
```

n.F ファイルには、次のコードが入っています。

```
#include "options.h"
module n
contains
  subroutine my_print(x)
    real, value :: x
    print *, x
  end subroutine
end module
```

m.f ファイルには、次のコードが入っています。

```
#include "options.h"
module m
  use n
contains
  subroutine sub
    implicit none
    include 'constants.h'
    call my_print(pi)
  end subroutine
end module
```

n.F をコンパイルして、n.d という名前の従属関係出力ファイルを ./dependencies ディレクトリー内に作成するには、以下を入力します。

```
xlf -c n.F -qmakedep -MF./dependencies -o n_obj.o
```

m.f をコンパイルして、m.d という名前の従属関係出力ファイルを ./dependencies ディレクトリー内に作成し、同時にパス情報 /home/user/sample/ をオブジェクト・ファイルのターゲット名として m.d ファイルに組み込むには、以下を入力します。

```
xlf -c m.F -qmakedep -MF./dependencies -MT '/home/user/sample/m.o'
```

その場合、以下のような n.d ファイルが生成されます。

```
n_obj.o n.mod: options.h
n_obj.o n.mod: n.F
```

その場合、以下のような m.d ファイルが生成されます。

```
/home/user/sample/m.o m.mod: n.mod
/home/user/sample/m.o m.mod: option.h
/home/user/sample/m.o m.mod: m.f
/home/user/sample/m.o m.mod: constants.h
```

関連情報

- 161 ページの『-qfullpath』
- 97 ページの『-MF』

- 99 ページの『-MT』
- 104 ページの『-o』

-qmaxerr

カテゴリー

エラー・チェックおよびデバッグ

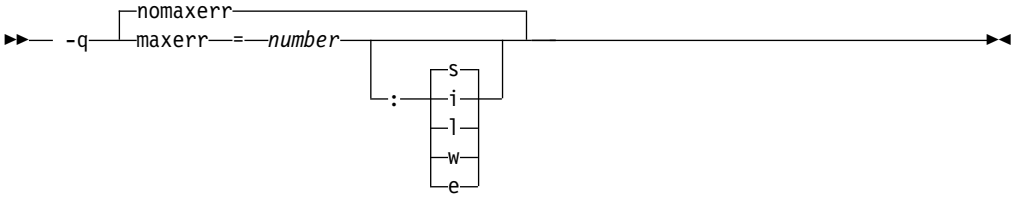
@PROCESS

```
@PROCESS MAXERR(number, severity) | NOMAXERR
```

目的

指定した重大度レベル以上のエラー・メッセージの数が指定した数に到達すると、コンパイルを停止する。

構文



デフォルト

-qnomaxerr

パラメーター

number
: コンパイラーが停止するまでに生成するメッセージの最大数を指定します。*number* は、1 以上の整数値でなければなりません。

重大度レベルは、最低から最高の順に次のとおりです。これらのレベルについて詳しくは、 341 ページの『エラーの重大度』を参照してください。

i 通知メッセージ。

l 言語レベル・メッセージ (**-qlanglvl** オプションの使用時に生成されるものなど)。

w 警告メッセージ。

e エラー・メッセージ。

s 重大エラー・メッセージ。

使用法

-qmaxerr オプションで重大度レベルを指定しない場合は、**-qhalt** オプションによって有効になる重大度が使用されます。それ以外の場合は、**-qmaxerr** と **-qhalt** のうち最後に記述されたオプションによって重大度レベルが指定されます。

-qflag を指定した場合、**-qmaxerr** は、**-qflag** オプションによって許容されたメッセージをカウントします。

-qsuppress を指定した場合、**-qmaxerr** は、**-qsuppress** オプションによって抑制されたメッセージをカウントしません。

例

5 件のエラー・メッセージが検出された場合に myprogram.f のコンパイルを停止するには、以下のコマンドを入力します。

```
xlf myprogram.f -qmaxerr=5:e
```

5 件の重大エラーが検出された場合に myprogram.f のコンパイルを停止するには、以下のコマンドを入力します。

```
xlf myprogram.f -qmaxerr=5
```

5 件の言語レベル・メッセージが検出された場合に myprogram.f のコンパイルを停止するには、以下のコマンドを入力します。

```
xlf myprogram.f -qmaxerr=5:l
```

または:

```
xlf myprogram.f -qmaxerr=5 -qhalt=l
```

関連情報

- 151 ページの『-qflag』
- 167 ページの『-qhalt』
- 282 ページの『-qsuppress』
- 341 ページの『エラーの重大度』

-qmaxmem

カテゴリー

最適化およびチューニング

目的

メモリーを消費する、指定したキロバイト数になるまでの特定の最適化の実行中に、コンパイラーによって割り振られるメモリーの量を制限する。

構文

@PROCESS:

```
@PROCESS MAXMEM(Kbytes)
```

デフォルト

- **maxmem=8192** (**-O2** が有効な場合)
- **maxmem=-1** (**-O3** 以上の最適化が有効な場合)

パラメーター

Kbytes
最適化によって使用されるメモリーの数値 (キロバイト)。その制限は、コンパイラー全体のものではなく、特定の最適化におけるメモリー量です。全コンパイル処理中に必要とされるテーブルは、この単位によって影響を受けることも組み込まれることもありません。

値 **-1** を指定すれば、制限チェックは行わず、必要なだけメモリーを使って最適化を実行します。

使用法

コンパイラーが特定の最適化を算出するには指定されたメモリーの容量が不十分な場合は、コンパイラーがメッセージを発行し最適化の度合いが減ります。

このオプションは、 **-O** オプションと組み合わせた場合以外は効果がありません。

-O2 を指定してコンパイルするときには、コンパイル時メッセージが限界を上げるように指示する場合にその指示に従うだけで十分です。**-O3** を指定してコンパイルするときには、マシンの実行によりストレージが不足するためにコンパイルが停止する場合に、限界の設定が必要な場合があります。この場合は 8192 以上の値で開始し、過大なストレージをコンパイルが要求し続けるときはこの値を減らします。

注**:**
1. 最適化が減じられるということは、その結果作成されたプログラムの速度が遅くなることを必ずしも意味しません。コンパイラーが、パフォーマンスを向上させる機会を際限なく探し続けてしまうということを意味するにすぎません。
2. 限界を高くするということは、その結果作成されたプログラムの速度が速くなることを必ずしも意味せず、パフォーマンスを向上させる機会があれば、その機会をコンパイラーが見つけやすくなるということを意味するにすぎません。
3. 大きな限界を設定しても、ソース・ファイルをコンパイルするときに、最適化の実行中にコンパイラーが大量のメモリーを使用する必要がない場合は、悪影響はありません。
4. メモリー限界を上げる別の方法として、最も複雑な計算を、その時点で完全に分析できるほど小さなプロシージャーに移動することができます。
5. すべてのメモリー集約的コンパイル・ステージを制限できるわけではありません。
6. **-O2** と **-O3** について行われる最適化のみを制限できます。 **-O4** および **-O5** 最適化は制限できません。
7. **-O4** および **-O5** 最適化では、/tmp ディレクトリー内のファイルが使用される可能性もあります。これは、**-qmaxmem** 設定によって制限されません。
8. いくつかの最適化は最大使用可能アドレス・スペースを超える前は、自動的にオフになりますが、そのときに使用可能なページング・スペース (マシンの作業負荷によって異なります) を超える前には、自動的にはオフになりません。

制約事項

コンパイルされるソース・ファイル、ソース・コード内のサブプログラムのサイズ、マシン構成、システム上の作業負荷によっては、限界を高く設定し過ぎると、ページング・スペースを使い果たしてしまう場合があります。特に、値 **-1** は、装

備の充実したマシンでも、ストレージを使い果たす場合があります。

関連情報

- 101 ページの『-O』
- 「*XL Fortran* 最適化およびプログラミング・ガイド」の『アプリケーションの最適化』

-qmbcs

カテゴリー

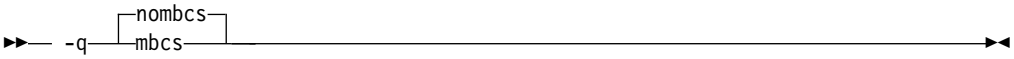
言語エレメント制御

目的

文字リテラル定数、ホレリス定数、H 編集記述子、および文字ストリング編集記述子にマルチバイト文字セット (MBCS) と Unicode 文字のどちらを含めることができるかをコンパイラーに指示する。

このオプションは、日本語のようなマルチバイト言語でデータを処理しなければならないアプリケーションのためのものです。

構文



@PROCESS:

@PROCESS MBCS | **NOMBCS**

デフォルト

-qnombcs

使用法

マルチバイト文字の個々のバイトは、1 桁としてカウントされます。

実行時にマルチバイト・データを正しく処理するには、コンパイル中と同じ値にロケールを設定してください (**LANG** 環境変数を使用するか、または **libc setlocale** ルーチンへの呼び出しを使用)。

Unicode データの読み書きを行うには、実行時にロケール値を **UNIVERSAL** に設定します。ロケールが設定されていないと、Unicode が使用可能なアプリケーションとデータを交換できない場合があります。

-qmixed

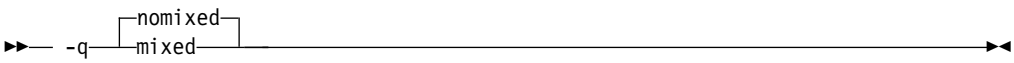
カテゴリー

入力制御

目的

コンパイラーが名前の大文字小文字を区別するようにする。

構文



@PROCESS:

@PROCESS MIXED | **NOMIXED**

デフォルト

-qnomixed

デフォルトでは、コンパイラーは、すべての名前を小文字であるかのように解釈します。例えば、Abc と ABC はどちらも abc であると解釈され、したがって、同じオブジェクトを参照します。

使用法

Fortran 名はデフォルト時にはすべて小文字であり、 C 言語およびその他の言語の名前は大文字と小文字が混在してもかまいません。混合言語プログラムを書く際にこのオプションを使用することができます。

-qmixed が指定される場合は、名前の大文字と小文字の区別が重要です。例えば、Abc という名前と ABC という名前は別々のオブジェクトを参照します。

このオプションは、コンパイル単位間の呼び出しを解決するのに使用されるリンク名を変更します。また、モジュールおよび F2008 サブモジュール F2008 の名前に影響を与え、したがって、それらの .mod および .smod ファイルの名前にも影響を与えます。

制約事項

-qmixed が有効な場合は、組み込み機能の名前はすべて小文字でなければなりません。小文字でない場合は、コンパイラーはエラーを送出しなくても名前を受け入れることはできますが、それらを組み込み関数ではなく外部プロシージャーの名前であると判断します。

-qmkshrobj

カテゴリー

出力制御

@PROCESS

なし。

目的

生成されたオブジェクト・ファイルから共有オブジェクトを作成する。

リンカーを直接呼び出す代わりに、このオプションをこのトピック内で説明する関連オプションと一緒に使用して、共有オブジェクトを作成します。 このオプションを使用する利点は、(**-O5** で実行されるような) **-qipa** リンク時最適化との互換性があることです。

構文

デフォルト

デフォルトで、出力オブジェクトをランタイム・ライブラリーおよび始動ルーチンとリンクすることで、実行可能ファイルが作成されます。

使用法

--version-script リンカー・オプションを使用してエクスポートするシンボルを指定しない限り、コンパイラーは自動的に共有オブジェクトからすべてのグローバル・シンボルをエクスポートします。 IBM hidden または internal の visibility 属性を持つシンボルは、エクスポートされません。 IBM

-qmkshrobj を指定すると、**-qpic** が暗黙的に指定されます。

-qmkshrobj と共に以下の関連オプションも使用できます。

-o *shared_file*
: 共有ファイルの情報を保持するファイルの名前です。デフォルトは a.outです。

-e *name*
: 共有実行可能ファイルの入り口名を *name* に設定します。

-qstaticlink=xllibs
: **-qstaticlink=xllibs** および **-qmkshrobj** を指定した場合は、両方のオプションが有効になります。コンパイラーは、XL ライブラリーへの参照がすべて静的にリンクされる共有オブジェクトを作成します。

-qmkshrobj を使用した共有ライブラリーの作成の詳細については、 36 ページの『ライブラリーのコンパイルとリンク』を参照してください。

例

3 つの小さなオブジェクト・ファイルから共有ライブラリー `big_lib.so` を構成するには、以下のコマンドを入力します。

```
xlf -qmkshrobj -o big_lib.so lib_a.o lib_b.o lib_c.o
```

関連情報

- 87 ページの『-e』
- 192 ページの『-qipa』
- 104 ページの『-o』

- 235 ページの『-qpic』

-qmoddir

カテゴリー

出力制御

@PROCESS

なし。

目的

コンパイラーが書き込むモジュール (.mod) または F2008 サブモジュール F2008 (.smod) ファイルの場所を指定する。

構文

デフォルト

適用されません。

使用法

-qmoddir を指定しない場合、.mod ファイルや .smod ファイルは現行ディレクトリーに配置されます。

モジュールや F2008 サブモジュール F2008 を参照するファイルをコンパイルするときに、このディレクトリーから .mod ファイルや .smod ファイルを読み取るには、**-I** オプションを使用します。

関連情報

- 25 ページの『XL Fortran 出力ファイル』
- 94 ページの『-I』
- 「*XL Fortran* ランゲージ・リファレンス」の『モジュール』
- F2008 「*XL Fortran* ランゲージ・リファレンス」の『サブモジュール』 F2008

-qnoprint

カテゴリー

リスト、メッセージ、およびコンパイラー情報

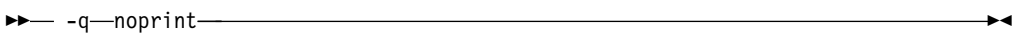
@PROCESS

なし。

目的

他のリスト・オプションの設定に関係なく、コンパイラーがリスト・ファイルを作成しないようにする。

構文



デフォルト

適用されません。

使用法

コマンド行に **-qnoprint** を指定すれば、構成ファイルまたは **@PROCESS** ディレクティブに他のリスト・オプションを入れることができ、リスト・ファイルが作成されるのを防止します。

通常、リスト・ファイルは、**-qattr**、**-qlist**、**-qlistopt**、**-qphsinfo**、**-qsource**、**-qreport** または **-qxref** のいずれかのオプションを指定すると作成されます。**-qnoprint** は、名前を **/dev/null** (書き込まれたあらゆるデータを廃棄するデバイス) に変更してリスト・ファイルが作成されるのを防止します。

関連情報

- 72 ページの『リスト、メッセージ、およびコンパイラー情報』

-qnullterm

カテゴリー

言語エレメント制御

目的

仮引数として渡される各文字定数式に NULL 文字を付加して、ストリングを C 関数に渡しやすくする。

このオプションを使用すると、個々のストリング引数に NULL 文字を追加しなくても C 関数へストリングを渡すことができます。

構文

@PROCESS:

@PROCESS NULLTERM | **NONULLTERM**

デフォルト

-qnonullterm

使用法

このオプションは、以下のオブジェクトで構成されている引数に影響を与えます。
- 基本文字定数
- 複数の文字定数の連結
- 型文字の指定された定数
- ホレリス定数
- インターフェース・ブロックが使用可能なとき、バイナリー、8 進、または 16 進数の型なし定数
- これらのオブジェクトで完全に構成される文字式。

CHAR および **ACHAR** 組み込み関数からの結果値にも、組み込み関数への引数が定数式である場合は NULL 文字が追加されます。

規則

このオプションは仮引数の長さ (XL Fortran 呼び出し規則の一部として渡された追加の長さの引数で定義されたもの) を変更しません。

制約事項

このオプションは **%REF** 組み込み関数を使用して渡された引数にもそうでない引数にも影響を及ぼしますが、値によって組み込み関数を使用して渡された引数には影響を及ぼしません。このオプションは、I/O ステートメントの中の文字式に影響を与えません。

例

このオプションを指定する場合と指定しない場合の 2 つの例を、同じ C 関数を使用して次に示します。

```
      @PROCESS NONULLTERM
      SUBROUTINE CALL_C_1
        CHARACTER*9, PARAMETER :: HOME = "/home/luc"
! Call the libc routine mkdir() to create some directories.
        CALL mkdir ("/home/luc/testfiles¥0", %val(448))
! Call the libc routine unlink() to remove a file in the home directory.
        CALL unlink (HOME // "/.hushlogin" // CHAR(0))
      END SUBROUTINE

      @PROCESS NULLTERM
      SUBROUTINE CALL_C_2
        CHARACTER*9, PARAMETER :: HOME = "/home/luc"
! With the option, there is no need to worry about the trailing null
! for each string argument.
        CALL mkdir ("/home/luc/testfiles", %val(448))
        CALL unlink (HOME // "/.hushlogin")
      END SUBROUTINE
!
```

関連情報

「*XL Fortran* 最適化およびプログラミング・ガイド」の『文字型の言語間受け渡し』を参照してください。

-qobject

カテゴリー

エラー・チェックおよびデバッグ

目的

ソース・ファイルの構文検査の後に、オブジェクト・ファイルを作成するか、即時に停止するかを指定する。

構文

@PROCESS:

@PROCESS **OBJect** | NOOBJect

デフォルト

-qobject

使用法

コンパイルに時間がかかる大きなプログラムをデバッグするときは、**-qnoobject** オプションを使用するようお勧めします。このオプションを使用すれば、コード作成のオーバーヘッドなしに、プログラムの構文を素早くチェックすることができます。 **.lst** ファイルは依然として作成されるので、デバッグを開始するための診断情報を得ることができます。

プログラム・エラーを修正した後に再びデフォルト (**-qobject**) に変更して、プログラムが正しく機能するかをテストし、正しく機能しない場合は、対話式デバッグのための **-g** オプションでコンパイルすることができます。

-qhalt オプションは **-qobject** オプションを、そして **-qnoobject** オプションは **-qhalt** オプションをオーバーライドできます。

関連情報

- 72 ページの『リスト、メッセージ、およびコンパイラー情報』
- 357 ページの『オブジェクト・セクション』
- 「*XL Fortran* はじめに」の『コンパイラー・フェーズ』
- 167 ページの『-qhalt』

-qoffload

カテゴリー

最適化およびチューニング

目的

OpenMP ターゲット領域の NVIDIA GPU へのオフロードのサポートを有効にする。

OpenMP ディレクティブ **TARGET** および **END TARGET** を使用して、ターゲット領域を定義できます。

-qoffload を有効にするには、**-qsmp** オプションを使用して、OpenMP ターゲット領域のサポートを有効にする必要があります。

構文

デフォルト

-qnooffload

使用法

-qoffload オプションを使用するには、CUDA Toolkit をインストールする必要があります。 CUDA Toolkit をインストールするには、Package Manager インストールを使用します。 Runfile インストールは現在、Power プロセッサーではサポートされていません。 Package Manager インストールの手順については、「NVIDIA CUDA Installation Guide for Linux」 (http://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html) を参照してください。

コンパイル・ステップとリンク・ステップの両方で **-qoffload** オプションを指定する必要があります。**-qoffload** を指定しないと、ターゲット領域はホストで実行され、コンパイラーは装置コードを生成しません。

コンパイラーはデフォルトで装置コードを最適化します。 **-qoffload** オプションは **-O2**、**-O3**、および **-Ofast** と併用できます。 **-qoffload** オプションは **-O4**、**-O5**、**-qipa**、**-qpdf1**、または **-qpdf2** のいずれとも併用できません。

例

`myopenmpprogram.f` をコンパイルするには、以下のコマンドを入力します。

```
xlf -qsmp -qoffload myopenmpprogram.f -o myopenmpprogram
```

`myopenmpprogram.f`をオブジェクト・ファイルにコンパイルし、後でリンクするには、次のコマンドを入力します。

```
xlf -c -qsmp -qoffload myopenmpprogram.f -o myopenmpprogram.o
xlf -qsmp -qoffload myopenmpprogram.o -o myopenmpprogram
```

関連情報

- -qsmp
- -qtgtarch
- 「*XL Fortran* 最適化およびプログラミング・ガイド」の TARGETセクション

- 「*XL Fortran* 最適化およびプログラミング・ガイド」の『計算を NVIDIA GPU にオフロードする』

-qonetrip

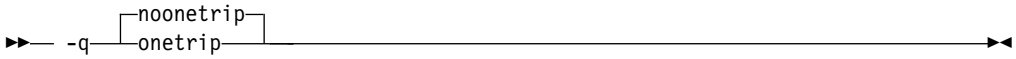
カテゴリー

言語エレメント制御

目的

これは、-1 オプションの長い形式です。

構文



@PROCESS:

@PROCESS ONETRIP | **NOONETRIP**

デフォルト

-qnoonetrip

-qoptfile

カテゴリー

コンパイラーのカスタマイズ

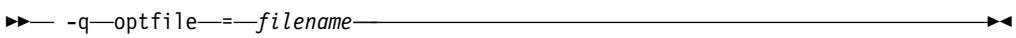
@PROCESS ディレクティブ

なし。

目的

コンパイルで使用する追加コマンド行オプションのリストが入った応答ファイルを指定する。応答ファイルは一般的に .rsp 接尾部を持ちます。

構文



デフォルト

なし。

パラメーター

filename
追加コマンド行オプションのリストが入った応答ファイルの名前を指定します。

filename には、相対パスまたは絶対パスを指定することができ、またパスを指定しないことも可能です。これは、1 行に 1 つ以上のコマンド行オプションが含まれたプレーン・テキスト・ファイルです。

使用法

応答ファイルのフォーマットは、以下の規則に従います。

- コマンド行の場合と同じ構文を使用して、ファイル内に組み込むオプションを指定します。応答ファイルは空白文字で区切られたオプションのリストです。特殊文字 ¥n、¥v、および ¥t は空白文字を示します。 (これらの文字の効果はすべて同じです。)
- 単一引用符または二重引用符のペアに囲まれた文字ストリングは、コンパイラーに 1 つのオプションとして渡されます。
- 応答ファイルにコメントを含めることができます。コメント行は # 文字から開始され、行末まで続きます。コンパイラーはコメントおよび空の行を無視します。

コンパイラーは処理時に **-qoptfile** オプションをコマンド行から削除し、指定した残りの後続オプションの前に、ファイルに含まれているオプションを連続して挿入します。

-qoptfile オプションは、応答ファイル内でも有効です。別の応答ファイルが含まれているファイルは、深さ優先方式で処理されます。コンパイラーは、応答ファイルの組み込みにおける循環を検出して無視することで、無限ループを回避します。

-qoptfile と **-qsaveopt** を同じコマンド行で指定すると、**-qsaveopt** に対して元のコマンド行が使用されます。各応答ファイルの内容を示すために、応答ファイルごとに改行が含まれます。ファイルに含まれているオプションは、コンパイルされたオブジェクト・ファイルに保存されます。

例 1

これは、応答ファイルの指定例です。

```
$ cat options.rsp
# To perform optimization at -O3 level, and high-order
# loop analysis and transformations during optimization
-O3 -qhot
# To indicate that the input source program is in fixed source form
-qfixed

$ xlf95 -qlist -qoptfile=options.rsp -qipa test.f
```

上記の例は、以下の呼び出しと同等です。

```
$ xlf95 -qlist -O3 -qhot -qfixed -qipa test.f
```

例 2

これは、循環のある **-qoptfile** が含まれた応答ファイルの指定例です。

```
$ cat options.file1
# To perform optimization at -O3 level, and high-order
# loop analysis and transformations during optimization
-O3 -qhot
# To include the -qoptfile option in the same response file
```

```
-qoptfile=options.file1
# To indicate that the input source program is in fixed source form
-qfixed
# To indicate that the source code is in free source form
-qfree

$ xlf95 -qlist -qoptfile=options.file1 -qipa test.f
```

上記の例は、以下の呼び出しと同等です。

```
$ xlf95 -qlist -O3 -qhot -qfixed -qfree -qipa test.f
```

例 3

これは、循環のない **-qoptfile** が含まれた応答ファイルの指定例です。

```
$ cat options.file1
-O3 -qhot
-qoptfile=options.file2

$ cat options.file2
-qfixed

$ xlf95 -qoptfile=options.file1 test.f
```

上記の例は、以下の呼び出しと同等です。

```
$ xlf95 -O3 -qhot -qfixed test.f
```

例 4

これは、同じコマンド行で **-qsaveopt** と **-qoptfile** を指定する例です。

```
$ cat options.file3
-O3
-qassert=contiguous

$ xlf95 -qsaveopt -qipa -qoptfile=options.file3 test.f -c

$ what test.o
test.o:
opt f xlf95 -qsaveopt -qipa -qoptfile=options.file3 test.f -c
optfile options.file3 -O3 -qassert=contiguous
```

関連情報

- 253 ページの『-qsaveopt』

-qoptimize

目的

これは、-O オプションの長い形式です。

構文

@PROCESS:

@PROCESS OPTimize[(*level*)] | **NOOPTimize**

デフォルト

-qnooptimize

-qpath

カテゴリー

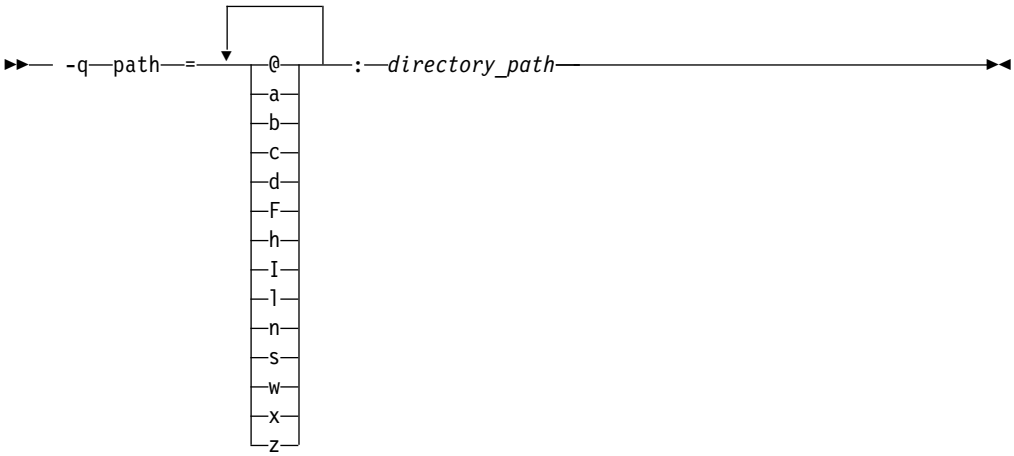
コンパイラーのカスタマイズ

目的

これは、XL Fortran コンポーネント (アセンブラー、C プリプロセッサー、リンカーなど) の代替パス名を指定します。

XL Fortran コンポーネントの一部またはすべてについて複数のレベルを保持し、どれを使用するかを指定できるようにする場合、このオプションを使用できます。

構文



デフォルト

デフォルトで、コンパイラーは構成ファイルで定義されたコンパイラー・コンポーネントのパスを使用します。

パラメーター

directory_path
コンパイラー・コンポーネントが配置されているディレクトリーへのパス。既存のディレクトリーでなければなりません。相対または絶対のいずれかにできます。

以下の表は、**-qpath** パラメーターとコンポーネント名との対応を示しています。

パラメーター	説明	コンポーネント名
GPU @	PTX アセンブラー	ptxas
a	アセンブラー	as

パラメーター	説明	コンポーネント名
b	低水準最適化プログラム	xlfcode
c	コンパイラーのフロントエンド	xlfentry
d	逆アセンブラー	dis
F	C プリプロセッサー	cpp
h	配列言語最適化プログラム	xlfhot
I (大文字の i)	高水準最適化プログラム、コンパイル・ステップ	ipa
l (小文字の L)	リンカー	ld
GPU n	NVIDIA C コンパイラー。装置リンカーとして使用	nvcc
GPU s	XL 中間言語 (W-Code) スプリッター	partitioner
GPU w	NVVM-IR 変換プログラムに対する XL 中間言語 (W-Code)	wc2llvm
GPU x	NVVM-IR から PTX への変換プログラム	llvm2ptx
z	バインダー	bolt

使用法

-qpath オプションは、 **-F**、**-t**、および **-B** オプションをオーバーライドします。

例

`/fix/FE/` から代替コンパイラー・フロントエンドとリンカーを使用し、デフォルトの場所から残りのコンパイラー・コンポーネントを使用して `myprogram.f` をコンパイルするには、以下のコマンドを入力します。

```
xlf myprogram.f -qpath=cl:/fix/FE
```

`/fix/FE` から代替コンパイラー・フロントエンドを、現行ディレクトリーから代替リンカーを、デフォルトの場所から残りのコンパイラー・コンポーネントを使用して `myprogram.f` をコンパイルするには、以下のコマンドを入力します。

```
xlf95 myprogram.f -qpath=c:/fix/FE -qpath=l:.
```

関連情報

- 83 ページの『-B』
- 318 ページの『-t』
- *XL Fortran* を使用した *CUDA Fortran* プログラミングの概要

-qpdf1、-qpdf2

カテゴリー

最適化およびチューニング

@PROCESS

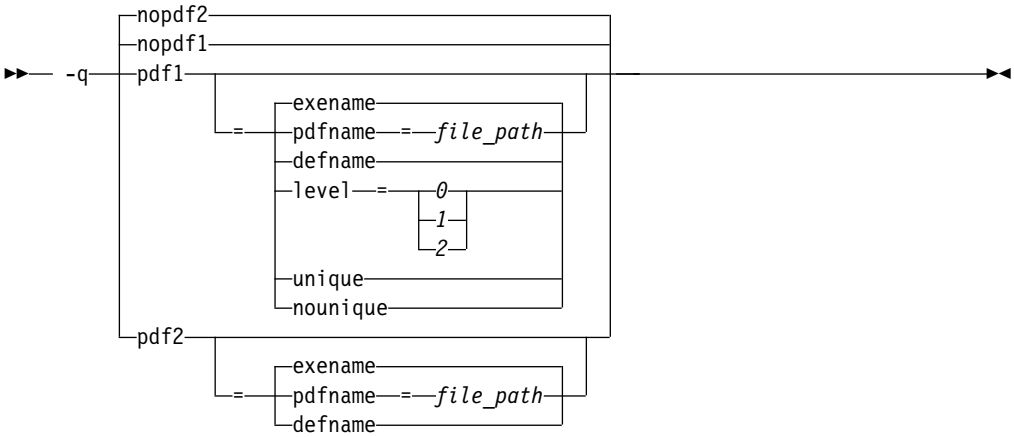
なし。

目的

profile-directed feedback (PDF) を介して最適化を調整する。サンプル・プログラムの実行から得られた結果を使用して、条件付き分岐の付近や頻繁に実行されるコード・セクションでの最適化を改善します。

分岐の使用頻度およびコード・ブロックの実行頻度の分析に基づいて、標準的な使用シナリオについてアプリケーションを最適化します。

構文



デフォルト

`-qpdf1` を指定しない場合は `-qnopdf1`。 `-qpdf2` を指定しない場合は `-qnopdf2`。

サブオプションなしで `-qpdf1` を指定した場合は `-qpdf1=exename`。サブオプションなしで `-qpdf2` を指定した場合は `-qpdf2=exename`。

パラメーター

exename
生成された PDF ファイルを *.<output_name>*_pdf と命名します。 *<output_name>* は、**-qpdf1** 指定でプログラムをコンパイルした場合に生成される出力ファイルの名前です。

pdfname= *file_path*
PDF ファイルおよび (存在する場合は) すべての既存の PDF マップ・ファイルのディレクトリーと名前を指定します。PDFDIR 環境変数が設定されている場合、PDF ファイルおよび PDF マップ・ファイルは、コンパイラーによって PDFDIR で指定されたディレクトリーに配置されます。 それ以外の場合、これらのファイルはコンパイラーによって現行作業ディレクトリーに配置されます。 PDFDIR 環境変数が設定されているが、指定されたディレクトリーが存在しない場合、コンパイラーから警告メッセージが出されます。 **-qpdf1=unique** オプションが指定されていない場合、PDF マップ・ファイルの名前は、PDF ファイ

ルの名前に従います。例えば、**-qpdf1=pdfname=/home/joe/func** オプションを指定した場合は、生成される PDF ファイルの名前は `func` になり、PDF マップ・ファイルの名前は `func_map` になります。どちらのファイルも `/home/joe` ディレクトリーに配置されます。 **pdfname** サブオプションを使用すると、同じディレクトリーを使用して複数の実行可能アプリケーションを同時に実行できます。この方法は、PDF で動的ライブラリーをチューニングする場合に特に便利です。

defname
生成された PDF ファイルを `._pdf` と命名します。

level=0 | 1 | 2
結果のアプリケーションによって生成される、各種レベルのプロファイル情報を指定します。以下の表に、各レベルでサポートされるプロファイル情報のタイプを示します。正符号 (+) は、プロファイル・タイプがサポートされることを示しています。

表 *19.* 各 *-qpdf1* レベルでサポートされるプロファイル・タイプ

	Level		
プロファイル・タイプ	**0**	**1**	**2**
ブロック・カウンター・プロファイル	+	+	+
呼び出しカウンター・プロファイル	+	+	+
値プロファイル		+	+
キャッシュ・ミス・プロファイル			+

-qpdf1=level=1 はデフォルト・レベルです。これは **-qpdf1** と同等です。 PDF レベルが高くなると、最適化が行われる機会も増えますが、オーバーヘッドが大きくなります。

注**:**
- 特定のプロセッサーでは、**-qpdf1=level=2** オプションを指定してコンパイルされたアプリケーションは一度に 1 つしか実行できません。
- キャッシュ・ミス・プロファイル情報にはいくつかのレベルがあります。各種レベルのキャッシュ・ミス・プロファイル情報を収集する場合は、PDF_PM_EVENT 環境変数を L1MISS、L2MISS、または L3MISS (使用可能な場合) に適宜設定します。一度に装備できるキャッシュ・ミス・プロファイル情報は 1 つのレベルのみです。L2 キャッシュ・ミス・プロファイルはデフォルト・レベルです。
- キャッシュ・ミス・プロファイル用に指定されたプロセッサーにアプリケーションをバインドする場合は、PDF_BIND_PROCESSOR 環境変数をプロセッサー番号と同じ値に設定します。

unique | nounique
PDF トレーニング・ステップで複数のプロセスが同じ PDF ファイルに書き込みを行う場合、**-qpdf1=unique** オプションを使用すると、単一の PDF ファイルがロッキングされるのを防ぐことができます。このオプションは、ランタイム

時に各プロセスに対して固有の PDF ファイルを作成するかどうかを指定します。 PDF ファイル名は *<pdf_file_name>.<pid>* です。*<pdf_file_name>* は以下のいずれかです。

- デフォルトで *.<output_name>*_pdf。
- サブオプション **pdfname** が有効な場合は、このサブオプションで指定された名前。
- **defname** サブオプションが有効な場合は ._pdf。

<pid> は、PDF トレーニング・ステップで実行中のプロセスの ID です。例えば、**-qpdf1=unique:pdfname=abc** オプションを指定し、12345678 と 87654321 の ID を持つ 2 つの PDF トレーニング用プロセスがある場合、2 つの PDF ファイル abc.12345678 および abc.87654321 が生成されます。

注**: -qpdf1=unique** が指定されている場合、プロセス ID がサフィックスとして付いた複数の PDF ファイルが生成されます。 PDF トレーニング・ステップの後で、**mergepdf** プログラムを使用して、これらすべての PDF ファイルを 1 つにマージする必要があります。

使用法

PDF プロセスは、以下の 3 つのステップから成ります。

1. **-qpdf1** オプションおよび最小の最適化レベル **-O2** を指定して、プログラムをコンパイルします。デフォルトでは、PDF マップ・ファイル *.<output_name>*_pdf_map と結果のアプリケーションが生成されます。
2. 標準的なデータ・セットを使用して、結果のアプリケーションを実行します。プロファイル情報が、デフォルトで *.<output_name>*_pdf という名前の PDF ファイルに書き込まれます。このステップは、PDF トレーニング・ステップと呼ばれています。
3. **-qpdf2** オプションおよび **-qpdf1** オプションとともに使用した最適化レベルを指定して、プログラムを再コンパイルおよびリンクまたは再リンクします。結果のアプリケーションの実行時に収集されるプロファイル情報に従って、**-qpdf2** プロセスが最適化を微調整します。

事前定義マクロ

なし。

例

例 **1**

例では、**-qpdf1=level=0** オプションを使用して、実行時のインスツルメンテーション・オーバーヘッドの可能性を低減します。

1. **-qpdf1=level=0** オプションを使用して、すべてのファイルをコンパイルします。

   ```
   xlf -qpdf1=level=0 -O3 file1.f file2.f file3.f
   ```

2. 1 セットの入力データを使用して実行します。

   ```
   ./a.out < sample.data
   ```

3. **-qpdf2** を使用してすべてのファイルを再コンパイルします。

```
xlf -qpdf2 -O3 file1.f file2.f file3.f
```

サンプル・データが標準的である場合、プログラムは PDF プロセスを使用しない場合より高速で実行できます。

例 **2**

以下の例では、**-qpdf1=level=1** オプションを使用します。

1. **-qpdf1** を使用してすべてのファイルをコンパイルします。

   ```
   xlf -qpdf1 -O3 file1.f file2.f file3.f
   ```

2. 1 セットの入力データを使用して実行します。

   ```
   ./a.out < sample.data
   ```

3. **-qpdf2** を使用してすべてのファイルを再コンパイルします。

   ```
   xlf -qpdf2 -O3 file1.f file2.f file3.f
   ```

サンプル・データが標準的である場合、プログラムは PDF プロセスを使用しない場合より高速で実行できます。

例 **3**

以下の例では、**-qpdf1=level=2** オプションを使用して、キャッシュ・ミス・プロファイル情報を収集します。

1. **-qpdf1=level=2** を使用して、すべてのファイルをコンパイルします。

   ```
   xlf -qpdf1=level=2 -O3 file1.f file2.f file3.f
   ```

2. PM_EVENT=L2MISS を設定して L2 キャッシュ・ミス・プロファイル情報を収集します。

   ```
   export PDF_PM_EVENT=L2MISS
   ```

3. 1 セットの入力データを使用して実行します。

   ```
   ./a.out < sample.data
   ```

4. **-qpdf2** を使用してすべてのファイルを再コンパイルします。

   ```
   xlf -qpdf2 -O3 file1.f file2.f file3.f
   ```

サンプル・データが標準的である場合、プログラムは PDF プロセスを使用しない場合より高速で実行できます。

例 **4**

この例では、**-qpdf[1|2]=exename** オプションの使用について示します。

1. **-qpdf1=exename** を使用してすべてのファイルをコンパイルします。

   ```
   xlf -qpdf1=exename -O3 -o final file1.f file2.f file3.f
   ```

2. サンプル入力データを使用して実行可能ファイルを実行します。

   ```
   ./final < typical.data
   ```

3. ディレクトリーの内容をリストします。

   ```
   >ls -lrta

   -rw-r--r-- 1 user staff 50 Dec 05 13:18 file1.f
    -rw-r--r-- 1 user staff 50 Dec 05 13:18 file2.f
    -rw-r--r-- 1 user staff 50 Dec 05 13:18 file3.f
    -rwxr-xr-x 1 user staff 12243 Dec 05 17:00 final
    -rwxr-Sr-- 1 user staff 762 Dec 05 17:03 .final_pdf
   ```

4. **-qpdf2=exename** を使用してすべてのファイルを再コンパイルします。

```
xlf -qpdf2=exename -O3 -o final file1.f file2.f file3.f
```

プログラムは、PDF 情報を使用して最適化されました。

例 **5**

以下の例では、**-qpdf[1|2]=pdfname** オプションの使用について示します。

1. **-qpdf1=pdfname** を使用してすべてのファイルをコンパイルします。静的プロファイル情報が final_map というファイルに記録されます。

```
xlf -qpdf1=pdfname=final -O3 file1.f file2.f file3.f
```

2. サンプル入力データを使用して実行可能ファイルを実行します。プロファイル情報が final というファイルに記録されます。

```
./a.out < typical.data
```

3. ディレクトリーの内容をリストします。

```
>ls -lrta

-rw-r--r-- 1 user staff 50 Dec 05 13:18 file1.f
 -rw-r--r-- 1 user staff 50 Dec 05 13:18 file2.f
 -rw-r--r-- 1 user staff 50 Dec 05 13:18 file3.f
 -rwxr-xr-x 1 user staff 12243 Dec 05 18:30 a.out
 -rwxr-Sr-- 1 user staff 762 Dec 05 18:32 final
```

4. **-qpdf2=pdfname** を使用してすべてのファイルを再コンパイルします。

```
xlf -qpdf2=pdfname=final -O3 file1.f file2.f file3.f
```

プログラムは、PDF 情報を使用して最適化されました。

例 **6**

以下の例では、PDF_BIND_PROCESSOR 環境変数の使用について示します。

1. **-qpdf1=level=1** を使用してすべてのファイルをコンパイルします。

```
xlf -qpdf1=level=1 -O3 file1.f file2.f file3.
```

2. この実行可能ファイルのすべてのプロセスがプロセッサー 1 で実行されるように、PDF_BIND_PROCESSOR 環境変数を設定します。

```
export PDF_BIND_PROCESSOR=1
```

3. サンプル入力データを使用して実行可能ファイルを実行します。

```
./a.out < sample.data
```

4. **-qpdf2** を使用してすべてのファイルを再コンパイルします。

```
xlf -qpdf2 -O3 file1.f file2.f file3.f
```

サンプル・データが標準的である場合、プログラムは PDF プロセスを使用しない場合より高速で実行できます。

関連情報

- 256 ページの『-qshowpdf』
- 192 ページの『-qipa』
- -qprefetch
- 248 ページの『-qreport』
- 327 ページの『cleanpdf』
- 328 ページの『mergepdf』

- 329 ページの『showpdf』
- 23 ページの『XL Fortran 入力ファイル』
- 25 ページの『XL Fortran 出力ファイル』
- 「*XL Fortran* 最適化およびプログラミング・ガイド」の『プロファイル指示フィードバック』
- 『環境変数の正しい設定方法』

-qphsinfo

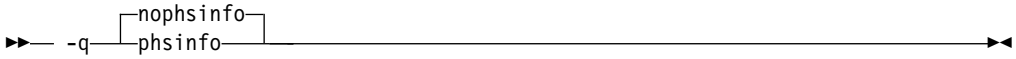
カテゴリー

リスト、メッセージ、およびコンパイラー情報

目的

各コンパイル・フェーズで標準出力にかかった時間を報告する。

構文



@PROCESS:

@PROCESS PHSINFO | **NOPHSINFO**

デフォルト

-qnophsinfo

使用法

出力は、各フェーズで *number1/number2* の形式をとります。ここで、*number1* はコンパイラーが使用する CPU 時間、*number2* はコンパイル時間と CPU がシステム・コールの処理に要する時間の合計を表します。

-qphsinfo によって報告される時間は秒単位です。

例

3 つのコンパイル単位からなる **app.f** をコンパイルして、コンパイルの各フェーズに要する時間をレポートするには、次のように入力します。

```
xlf90 app.f -qphsinfo
```

出力は以下のようなものになります。

```
FORTRAN phase 1 ftphas1      TIME =  0.000 / 0.000
** m_module   === End of Compilation 1 ===
FORTRAN phase 1 ftphas1      TIME =  0.000 / 0.000
** testassign   === End of Compilation 2 ===
FORTRAN phase 1 ftphas1      TIME =  0.000 / 0.010
** dataassign   === End of Compilation 3 ===
HOT       - Phase Ends;   0.000/  0.000
HOT       - Phase Ends;   0.000/  0.000
HOT       - Phase Ends;   0.000/  0.000
W-TRANS   - Phase Ends;   0.000/  0.010
```

```
OPTIMIZ   - Phase Ends;   0.000/  0.000
REGALLO   - Phase Ends;   0.000/  0.000
AS        - Phase Ends;   0.000/  0.000
W-TRANS   - Phase Ends;   0.000/  0.000
OPTIMIZ   - Phase Ends;   0.000/  0.000
REGALLO   - Phase Ends;   0.000/  0.000
AS        - Phase Ends;   0.000/  0.000
W-TRANS   - Phase Ends;   0.000/  0.000
OPTIMIZ   - Phase Ends;   0.000/  0.000
REGALLO   - Phase Ends;   0.000/  0.000
AS        - Phase Ends;   0.000/  0.000
1501-510  Compilation successful for file app.f.
```

各フェーズは、各コンパイル単位に対応して 3 回呼び出されます。FORTRAN はフロントエンド構文解析とセマンティック分析、 HOT はループ変換、W-TRANS は中間言語変換、OPTIMIZ は高水準最適化、REGALLO はレジスター割り振りと低水準最適化、および AS は最終アセンブリーを表します。

-qphsinfo を指定して、**app.f** を **-O4** 最適化レベルでコンパイルします。

```
xlf90 myprogram.f -qphsinfo -O4
```

出力結果は以下のようになります。

```
FORTRAN phase 1 ftphas1      TIME =  0.010 / 0.020
** m_module   === End of Compilation 1 ===
FORTRAN phase 1 ftphas1      TIME =  0.000 / 0.000
** testassign   === End of Compilation 2 ===
FORTRAN phase 1 ftphas1      TIME =  0.000 / 0.000
** dataassign   === End of Compilation 3 ===
HOT       - Phase Ends;   0.000/  0.000
HOT       - Phase Ends;   0.000/  0.000
HOT       - Phase Ends;   0.000/  0.000
IPA       - Phase Ends;   0.080/  0.100
1501-510  Compilation successful for file app.f.
IPA       - Phase Ends;   0.050/  0.070
W-TRANS   - Phase Ends;   0.010/  0.030
OPTIMIZ   - Phase Ends;   0.020/  0.020
REGALLO   - Phase Ends;   0.040/  0.040
AS        - Phase Ends;   0.000/  0.000
```

IPA (プロシージャー間分析) 最適化フェーズ中、プログラムの結果は 1 つのコンパイル単位になることに注意してください。つまり、すべてのプロシージャーがインライン化されます。

関連情報

「*XL Fortran* はじめに」の『コンパイラー・フェーズ』

-qpic

カテゴリー

オブジェクト・コード制御

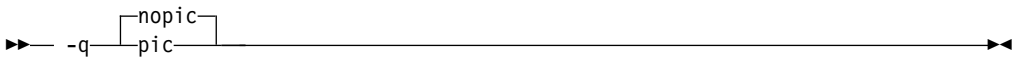
@PROCESS

なし。

目的

共有ライブラリーでの使用に必須の位置独立コードを生成する。

構文



デフォルト

- **-qnopic**

使用法

-qpic が有効になっている場合は、コンパイラーは TOC アクセスごとに 2 つの命令を生成してアクセス範囲を拡大します。これにより、目次 (TOC) のサイズが 64 Kb を超える場合でも TOC オーバーフロー条件が回避されます。

共有ライブラリーをビルドする際は、**-qpic** を指定する必要があります。

`-qpic -qtls` を指定すると、スレッド・ローカル・ストレージ (TLS) シンボルは **-qpic** によって影響されません。

アプリケーション内のコンパイル単位ごとに、異なる TOC アクセス・オプションを使用できます。

-qport

カテゴリー

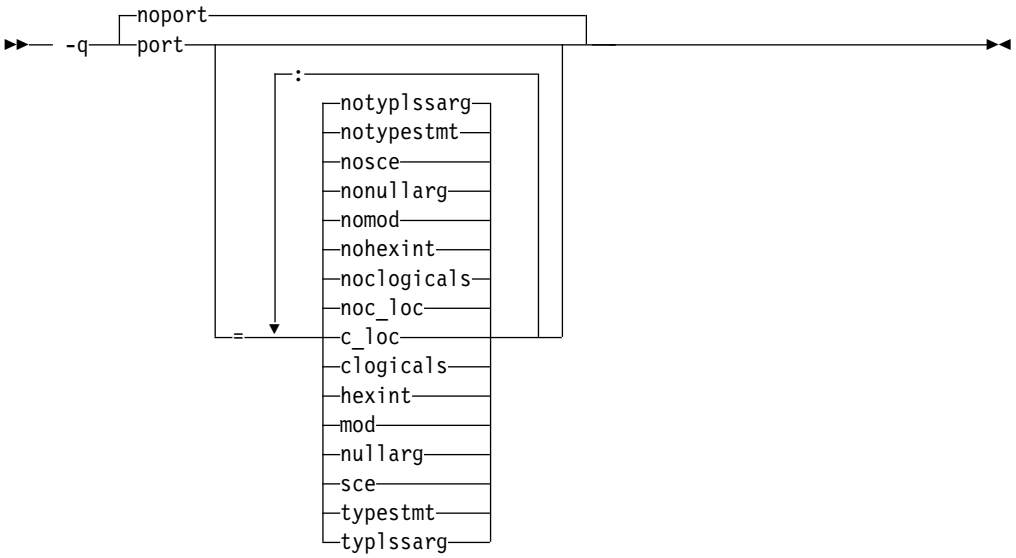
移植性とマイグレーション

目的

プログラムを XL Fortranに移植する際に、他の Fortran 言語の拡張機能に対応できるようにするためのオプションを提供する。

特定の **-qport** サブオプションは常に、他の **-qport** およびコンパイラー・オプションとは独立して機能します。

構文



@PROCESS:

@PROCESS PORT[(*suboptions*)]| **NOPORT**

デフォルト

-qnoport

パラメーター

c_loc | noc_loc
c_loc が有効であれば、ISO_C_BINDING モジュールからの C_LOC 関数は、**POINTER** 属性も **TARGET** 属性も持っていない引数を受け入れることができます。

C_LOC 関数のダミー引数に対応する実引数に **POINTER** 属性も **TARGET** 属性もない場合、**-qport=c_loc** は、C_LOC 関数への呼び出しが入っているコンパイル単位において、**TARGET** 属性を実引数に適用します。

注**:** ご使用のコードの移植性を確実にするには、このオプションを使用する代わりに、**TARGET** 属性を指定してください。

clogicals | noclogicals
clogicals が有効なとき、コンパイラーは論理式で使用されるすべての非ゼロ整数を TRUE として処理します。効果を生じさせるには、**-qport=clogicals** に対して、**-qintlog** を指定する必要があります。

この動作を期待する他の FORTRAN コンパイラーからこのアプリケーションの移植の際、**-qport=clogicals** オプションは便利です。ただし、プログラム間で論理データを共有するか受け渡す場合、非ゼロ整数に対して異なる設定値を使用するプログラムを混用するのは危険です。デフォルトの **-qintlog** 設定値を既に指定して書かれているデータ・ファイルは、**-qport=clogicals** オプション・アクティブを指定して読み込まれた場合、予期しない動作を引き起こします。

hexint | nohexint

hexint が有効なとき、型なし定数の 16 進定数ストリングは、**INT** 組み込み関数へ実引数として渡すとき、整数に変換されます。**INT** へ実引数として渡されない型のない 16 進定数ストリングは、影響を受けることはありません。

mod | nomod

mod を指定することにより、**MOD** 組み込み関数の既存の制約が緩和され、同じデータ型の 2 つの引数を別の kind 型パラメーターにすることができます。その結果は、その引数と同じ基本型になりますが、より大きい kind 型パラメーター値を持ちます。

nullarg | nonullarg

外部または内部プロシージャーで **nullarg** を指定すると、コンパイラーは左括弧とコンマ、2 つのコンマ、またはコンマと右括弧によって区切られた空の引数をヌル引数として扱います。このサブオプションは、引数リストが空の場合は効果がありません。

空の引数の例を次に示します。

```
call foo(,,z)

call foo(x,,z)

call foo(x,y,)
```

次のプログラムには、ヌル引数が含まれます。

Fortran プログラム:

```
program nularg
real(4) res/0.0/
integer(4) rc
integer(4), external :: add
rc = add(%val(2), res, 3.14, 2.18,) ! The last argument is a
                                    ! null argument.
if (rc == 0) then
print *, "res = ", res
else
print *, "number of arguments is invalid."
endif
end program
```

C プログラム:

```
int add(int a, float *res, float *b, float *c, float *d)
{
   int ret = 0;
   if (a == 2)
     *res = *b + *c;
   else if (a == 3)
     *res = (*b + *c + *d);
   else
     ret = 1;
   return (ret);
}
```

sce | nosce

デフォルトでは、コンパイラーは、選択した論理式で XL Fortran の規則を使用して短絡回路評価を実行します。**sce** を指定すると、コンパイラーは XL Fortran 以外の規則を使用できます。コンパイラーは、現行の規則で許可されている場合のみ短絡回路評価を実行します。

typestmt | **notypestmt**

PRINT 文と同様の方法で動作する TYPE 文は、**typestmt** を指定するときは常にサポートされます。

typlssarg | **notyplssarg**

定数が、関連する仮引数の型が整数である組み込みプロシージャーに対する実引数である場合に、すべての型なし定数をデフォルト整数に変換します。非整数型の仮引数に関連付けられている型なしの実引数は、このオプションの影響を受けません。

このオプションを使用した場合、いくつかの組み込みプロシージャーの種類が一致しないことがあります。その種類を最も長い引数の種類に変換するには、**-qxlf77=intarg** を指定してください。

関連情報

- 188 ページの『-qintlog』
- 303 ページの『-qxlf77』
- 詳しくは、「*XL Fortran* ランゲージ・リファレンス」の *INT* および *MOD* 組み込み関数に関するセクションを参照してください。

-qposition

カテゴリー

言語エレメント制御

目的

POSITION= 指定子および対応する **STATUS=** 値 (**OLD** または **UNKNOWN**) を持たない **OPEN** ステートメントが指定された後、データを書き込む際にファイル・ポインターをファイルの末尾に配置する。

最初の I/O 操作がファイル・ポインターを移動させ、その操作が **WRITE** 文または **PRINT** 文であると、位置は **APPEND** になります。 また、**BACKSPACE**、**ENDFILE**、**READ**、**REWIND** 文であると、位置は **REWIND** になります。

構文

@PROCESS:

```
@PROCESS POSITION({APPENDOLD | APPENDUNKNOWN} ...)
```

デフォルト

デフォルトの設定は、**OPEN** 文の I/O 指定子、およびコンパイラー呼び出しコマンドに依存します。

- **-qposition=appendold**(.f ファイル、.F ファイル、.f77 ファイル、または .F77 ファイルをコンパイルするために使用される **xlf** コマンドおよび **xlf_r** コマンドのデフォルト)

- **-qposition=appendold** (**f77** コマンドおよび **fort77** コマンドの場合)
- コマンド **xlf90**、**f90**、**xlf90_r**、**xlf95**、**f95**、**xlf95_r**、**xlf2003**、**f2003**、**xlf2003_r**、**xlf2008**、**f2008**、および **xlf2008_r** に対して定義されている Fortran 90、Fortran 95、Fortran 2003、および Fortran 2008 の動作
- CUDA Fortran **xlcuf** コマンド用に定義された Fortran 2008 の動作CUDA Fortran

例

次の例では、**POSITION=** 指定子を指定せずに **STATUS='old'** を指定する **OPEN** 文は、**POSITION='append'** が指定されているかのようにファイルをオープンします。

```
xlf95 -qposition=appendold opens_old_files.f
```

次の例では、**POSITION=** 指定子を指定せずに **STATUS='unknown'** を指定する **OPEN** 文は、**POSITION='append'** が指定されているかのようにファイルをオープンします。

```
xlf95 -qposition=appendunknown opens_unknown_files.f
```

次の例では、**POSITION=** 指定子を指定せずに **STATUS='old'** か **STATUS='unknown'** のいずれかを指定する **OPEN** 文は、**POSITION='append'** が指定されているかのようにファイルをオープンします。

```
xlf95 -qposition=appendold:appendunknown opens_many_files.f
```

関連情報

- 「*XL Fortran* 最適化およびプログラミング・ガイド」の『ファイルの位置決め』
- 「*XL Fortran* ランゲージ・リファレンス」の『**OPEN**』ステートメント

-qppsuborigarg

カテゴリー

入力制御

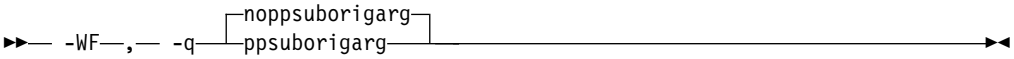
@PROCESS

なし。

目的

さらにマクロ展開を行う前に、C プリプロセッサーにオリジナルのマクロ引数を置換するよう指示します。

構文



デフォルト

- -WF, -qnoppsuborigarg

使用法

-qppsuborigarg C プリプロセッサー・オプションであり、**-WF** オプションを使用して指定する必要があります。

例

以下のサンプル・コード x.F を見てください。

```
#define PRINT_COMP(a) PRINT_4(SPLIT_COMP(a))
#define SPLIT_COMP(a) "Real:", real(a), "Imag:", imag(a)
#define PRINT_4(list) PRINT_LIST(list)
#define PRINT_LIST(list) print *, list

complex a
a = (3.5, -3.5)
PRINT_COMP(a)
end
```

このコードが **-qnoppsuborigarg** を指定してコンパイルされる場合、関数類似マクロ PRINT_4 内のパラメーター "list" がマクロ SPLIT_COMP(a) の拡張された置換テキストであるため、C プリプロセッサーはエラーを報告します。C プリプロセッサーのエラーの内容は、PRINT_LIST は、4 つの引数を指定して呼び出されますが、1 つの引数しか予測していないということです。

```
> xlf95 x.F -d
"x.F", line 8.1: 1506-215 (E) Too many arguments specified for macro PRINT_LIST.
** _main   === End of Compilation 1 ===
1501-510  Compilation successful for file x.F.
> cat Fx.f

complex a
a = (3.5, -3.5)
print *, "Real:"
end
```

コードが **-qppsuborigarg** を指定してコンパイルされる場合、 C プリプロセッサーは、関数類似マクロ PRINT_LIST の引数として、拡張された置換テキストの SPLIT_COMP(a) ではなくテキスト "SPLIT_COMP(a)" を使用します。C プリプロセッサーがマクロ "SPLIT_COMP(a)" を拡張するのは、マクロ PRINT_LIST が拡張された後のみです。結果的に、マクロ PRINT_LIST は、4 つの引数ではなく、予測した単一の引数 "SPLIT_COMP(a)" のみを受け取ります。

```
> xlf95 x.F -d -WF,-qppsuborigarg
** _main   === End of Compilation 1 ===
1501-510  Compilation successful for file x.F.
> cat Fx.f

complex a
a = (3.5, -3.5)
print *, "Real:", real(a), "Imag:", imag(a)
end
```

関連情報

- 323 ページの『-W、-X』
- 156 ページの『-qfpp』
- 39 ページの『C プリプロセッサーによる Fortran ファイルの引き渡し』

-qprefetch

カテゴリー

最適化およびチューニング

@PROCESS

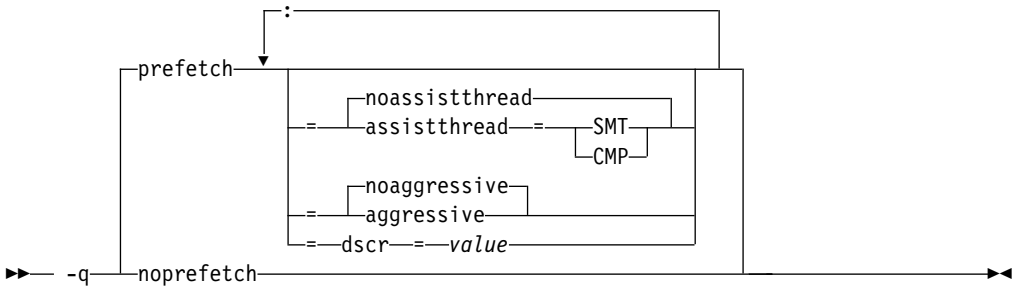
なし。

目的

コード・パフォーマンスを改善する機会がある場所に、プリフェッチ命令を自動的に挿入する。

-qprefetch が有効な場合、コンパイラーはコンパイルされたコードでプリフェッチ命令を挿入する可能性があります。**-qnoprefetch** が有効な場合、プリフェッチ命令はコンパイルされたコードに挿入されません。

構文



デフォルト

-qprefetch=noassistthread:noaggressive:dscr=0

パラメーター

assistthread | **noassistthread**
キャッシュ・ミス率の高いアプリケーションを使用している場合、**-qprefetch=assistthread** を使用して、データ・プリフェッチの支援スレッドを活用できます。このサブオプションは、支援スレッドを最適化レベル **-O3 -qhot** 以上で活用するようにコンパイラーを誘導します。**-qprefetch=assistthread** を指定しない場合は、**-qprefetch=noassistthread** が暗黙指定されます。

CMP
チップ・マルチプロセッサー (CMP) アーキテクチャーに基づいたシステムの場合は、**-qprefetch=assistthread=cmp** を使用できます。

SMT
同時マルチスレッディング (SMT) アーキテクチャーに基づいたシステムの場合は、**-qprefetch=assistthread=smt** を使用できます。

注: CMP と SMT のどちらも指定しなければ、コンパイラーはシステムのアーキテクチャーに基づいたデフォルト設定を使用します。

aggressive | noaggressive
このサブオプションは、積極的なデータ・プリフェッチを最適化レベル **-O3** 以上で生成するようにコンパイラーを誘導します。**aggressive** を指定しない場合は、暗黙的に **-qprefetch=noaggressive** が使用されます。

dscr
dscr サブオプションに値を指定して、アプリケーションのランタイム・パフォーマンスを向上できます。コンパイラーは、データ・ストリーム制御レジスター (DSCR) を指定の dscr の値に設定し、ハードウェア・プリフェッチ・エンジンを制御します。この値が有効なのは、**-qarch=pwr8** が有効であり、最適化レベルが -O2 以上である場合です。dscr のデフォルト値は 0 です。

value

dscr に指定する値は、0 以上で、64 ビット符号なし整数として表現可能でなければなりません。それ以外の場合、コンパイラーは警告メッセージを出して、dscr を 0 に設定します。コンパイラーには 10 進数および 16 進数の両方の数字を指定でき、16 進数の場合にはプレフィックス 0x が必要です。値の範囲は、システム・アーキテクチャーによって異なります。詳しくは、POWER Architecture の製品情報を参照してください。複数の dscr 値を指定した場合、最後の 1 つが有効になります。

使用法

-qnoprefetch オプションは、**__prefetch_by_stream** などの組み込み関数がプリフェッチ命令を生成するのを妨げません。

-qprefetch=assistthread を実行すると、コンパイラーは delinquent ロード (キャッシュ・ミス頻発ロード) 情報を使用して分析を行い、プリフェッチ支援スレッドを生成します。delinquent ロード (キャッシュ・ミス頻発ロード) 情報は、組み込み __mem_delay 関数 (const void *delinquent_load_address、const unsigned int delay_cycles) を通じて提供するか、**-qpdf1=level=2** を使用して動的プロファイルから収集することができます。

-qpdf を使用して **-qprefetch=assistthread** を呼び出す場合、以下のように従来の 2 ステップの PDF 呼び出しを使用する必要があります。

1. **-qpdf1=level=2** を実行する
2. **-qpdf2 -qprefetch=assistthread** を実行する

例

```
DO i = 1, 1000

!IBM* MEM_DELAY(x(i), 10)
x(i) = x(i) + 1

END DO
```

例

```
DO I = 1, 1000

!IBM* MEM_DELAY(X(I), 10)
```

```
X(I) = X(I) + 1

END DO
```

関連情報

- -qarch
- 170 ページの『-qhot』
- 228 ページの『-qpdf1、-qpdf2』
- 248 ページの『-qreport』
- 「*XL Fortran* ランゲージ・リファレンス」の MEM_DELAY セクション

-qpreprocess

カテゴリー

入力制御

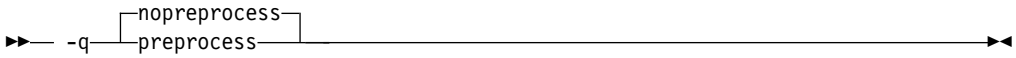
@PROCESS

なし。

目的

cpp を呼び出して、有効な Fortran ファイル接尾部 (.f や .f90 など) を持つファイルのプリプロセスを行います。

構文



デフォルト

-qnopreprocess

使用法

-qpreprocess オプションを使用すると、有効な Fortran ファイル・サフィックスを持つファイルを前処理するために、コンパイラーは **cpp** を呼び出します。

関連情報

- 281 ページの『-qsuffix』

-qqcount

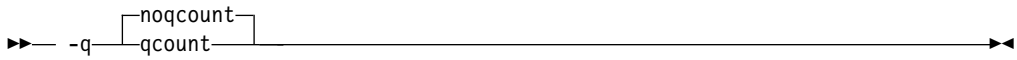
カテゴリー

言語エレメント制御

目的

Q 文字カウント編集記述子 (**Q**)、および拡張精度 **Q** 編集記述子 (**Q***w.d*) を受け入れる。

構文



@PROCESS:

@PROCESS QCOUNT | **NOQCOUNT**

デフォルト

-qnoqcount を使用すると、すべての **Q** 編集記述子が拡張精度 **Q** 編集記述子として解釈されます。

使用法

コンパイラーは、**Q** 編集記述子を構文によって拡張精度 Q 編集記述子または Q 文字カウント編集記述子であると解釈して、どちらの記述子が指定されるかを判別できない場合に、警告を出します。

関連情報

- 「*XL Fortran* ランゲージ・リファレンス」の 『*Q* (文字カウント) 編集』

-qrealsize

カテゴリー

浮動小数点および整数のコントロール

目的

REAL、**DOUBLE PRECISION**、**COMPLEX**、および **DOUBLE COMPLEX** の値のデフォルトのサイズを設定します。

このオプションは、他のシステム用に書かれたコードとの互換性を維持することを意図しています。ある状況においては、**-qautodbl** の代替オプションとして便利なことがわかります。

構文

@PROCESS:

@PROCESS REALSIZE(*bytes*)

デフォルト

-qrealsize=4

パラメーター

bytes に使用できる値は次のとおりです。
- 4
- 8

使用法

このオプションは、データのデフォルト・サイズが異なるシステムから、プログラムを変更せずに移植できるようにするためのものです。例えば、CRAY コンピューター用に書かれたプログラムには **-qrealsize=8** が必要です。

-qautodbl は、**-qrealsize** に関連していますが、これらのオプションは結合することはできません。**-qautodbl** オプションが自動精度倍増、埋め込み (またはその両方) をオンにすると、**-qrealsize** オプションは効果がなくなります。

-qrealsize を 8 に設定すると、 **-qdpc** オプションの設定がオーバーライドされます。

vector(real) 型には、**REAL** 型の他に **-qrealsize** 型も使用できます。

結果

このオプションは、定数と変数のサイズ[2]、および kind 型パラメーターが指定されていない派生型コンポーネントと関数 (組み込み関数を含む) に影響を与えます。**REAL(4)** や **COMPLEX*16** など、kind 型パラメーターまたは長さで宣言されたオブジェクトは影響を受けません。

このオプションは、影響を受けるオブジェクトのサイズを次のように判別します。

```
Data Object       REALSIZE(4) in Effect       REALSIZE(8) in Effect
-----------------------------------------------------------------
1.2               REAL(4)                      REAL(8)
1.2e0             REAL(4)                      REAL(8)
1.2d0             REAL(8)                      REAL(16)
1.2q0             REAL(16)                     REAL(16)

REAL              REAL(4)                      REAL(8)
DOUBLE PRECISION  REAL(8)                      REAL(16)
COMPLEX           COMPLEX(4)                   COMPLEX(8)
DOUBLE COMPLEX    COMPLEX(8)                   COMPLEX(16)
```

組み込み関数にも同様の規則が当てはまります。
- 組み込み関数に型宣言がない場合は、その引数と戻り値の型が **-qrealsize** 設定によって変更される場合があります。
- 組み込み関数に対する型宣言は、戻り値のデフォルトのサイズと一致しなければなりません。

2. Fortran 90/95 の用語では、これらの値は *kind* 型パラメーター と呼ばれます。

例

この例には、**-qrealsize** の設定を変更すると代表的なエンティティーがどのように変形するかが示されています。

```
@PROCESS REALSIZE(8)
         REAL R                  ! treated as a real(8)
         REAL(8) R8              ! treated as a real(8)
         VECTOR(REAL)            ! treated as a vector(real(8))
         VECTOR(REAL(4))         ! treated as a vector(real(4))
         DOUBLE PRECISION DP     ! treated as a real(16)
         DOUBLE COMPLEX DC       ! treated as a complex(16)
         COMPLEX(4) C            ! treated as a complex(4)
         PRINT *,DSIN(DP)        ! treated as qsin(real(16))
! Note: we cannot get dsin(r8) because dsin is being treated as qsin.
         END
```

-qrealsize=8 を指定すると、以下のように **DABS** などの組み込み関数に影響を与えます。

```
INTRINSIC DABS        ! Argument and return type become REAL(16).
DOUBLE PRECISION DABS ! OK, because DOUBLE PRECISION = REAL(16)
                      ! with -qrealsize=8 in effect.
REAL(16) DABS         ! OK, the declaration agrees with the option setting.
REAL(8) DABS          ! The declaration does not agree with the option
                      ! setting and is ignored.
```

関連情報

- 190 ページの『-qintsize』は、整数と論理オブジェクトに影響を与える同様のオプションです。
- 117 ページの『-qautodbl』
- 「*XL Fortran* ランゲージ・リファレンス」の『型宣言*:* 型パラメーターおよび指定子』

-qrecur

目的

外部サブプログラムを再帰的に呼び出せるかどうかを指定する。

このオプションの使用はお勧めできません。

構文

@PROCESS:

@PROCESS RECUR | **NORECUR**

デフォルト

-qnorecur

使用法

新規プログラムの場合、**RECURSIVE** キーワードを使用すると、標準適応した方法で再帰的プロシージャーを使用することができます。

-qrecur オプションを指定すると、コンパイラーはすべてのプロシージャーが再帰的であると見なしてしまいます。 再帰的プロシージャーのコード生成の効率が落ちる可能性があります。**RECURSIVE** キーワードを使用すれば、どのプロシージャーを再帰的にするかを正確に指定することができます。

次のコマンドを使用して再帰呼び出しが含まれたプログラムをコンパイルする場合は、**-qnosave** を指定して、デフォルトのストレージ・クラスを「自動」にします。

- .f ファイル、.F ファイル、.f77 ファイル、および .F77 ファイルの場合: **xlf** および **xlf_r**
- 任意のソース・ファイルの場合: **f77** および **fort77**

例

```
! The following RECUR recursive function:

        @process recur
        function factorial (n)
        integer factorial
        if (n .eq. 0) then
           factorial = 1
        else
           factorial = n * factorial (n-1)
        end if
        end function factorial

! can be rewritten to use F90/F95 RECURSIVE/RESULT features:

        recursive function factorial (n) result (res)
        integer res
        if (n .eq. 0) then
           res = 1
        else
           res = n * factorial (n-1)
        end if
        end function factorial
```

-qreport

カテゴリー

リスト、メッセージ、およびコンパイラー情報

目的

コードのセクションをどのように最適化したかを示すリスト・ファイルを作成する。

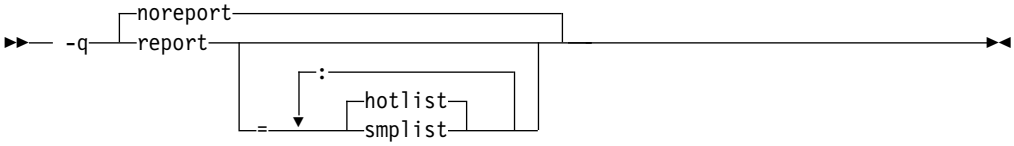
コマンド行にリストされたソース・ファイルごとに、リスト・ファイルが .lst 接尾部付きで生成されます。自動並列化またはベクトル化を有効にするオプションと一緒に **-qreport** を指定した場合、リスト・ファイルには、疑似 Fortran コード・リストと、プログラム・ループがどのように並列化または最適化されるのかについての要約が示されます。また、このレポートには、特定のループを並列化/ベクトル化

できない理由に関する診断情報も含まれます。例えば、**-qreport** が **-qsimd** と一緒に指定されると、ループのベクトル化を防ぐストライド 1 以外の参照を示すメッセージが出されます。

またコンパイラーは、ロード・ストリームとストア・ストリームの両方を含む、指定されたループ用に作成されたストリームの数も報告します。この情報は、リスト・ファイルの Loop Transformation セクションに組み込まれます。この情報を使用して、アプリケーション・コードを理解し、プログラム・コードのパフォーマンス・チューニングを行うことができます。例えば、基礎のアーキテクチャーがサポートする数より多いストリームを含むループを配布することができます。POWER8 以上のプロセッサー は、ロード・ストリームとストア・ストリームの両方のプリフェッチをサポートします。

構文

オプション:



@PROCESS:

@PROCESS REPORT[({SMPLIST |HOTLIST}...)] | **NOREPORT**

デフォルト

-qnoreport

パラメーター

smplist | **hotlist**

> **-qreport=smplist** が有効なとき、プログラムの並列化過程を示す疑似 Fortran リストを作成します。このリストが作成されるのは、ループや他の最適化が実行される前です。このリストの中には、修正すればより効率的にできるプログラムの箇所を示すメッセージも含められます。 **-qsmp** が有効な場合のみ、このレポートが作成されます。
>
> **-qreport=hotlist** が有効なとき、ループの変換過程を示す疑似 Fortran リストを作成します。このリストは、全ループのパフォーマンスを調整するのに役立ちます。**-qhot** が有効な場合のみ、このレポートが作成されます。
>
> また、**-qsmp** が有効なときに **-qreport=hotlist** オプションが指定された場合は、SMP ランタイム・ライブラリーへの呼び出し、および並列構文のために作成されたプロシージャーへの呼び出しを示す疑似 Fortran リストが作成されます。

サブオプションを指定しないで **-qreport** を指定することは、**-qreport=hotlist** と同じです。

使用法

ループ変換リストを生成するには、以下のいずれかのオプションとともに **-qreport** を指定する必要があります。

- **-qhot**
- **-qsmp**
- **-O3** 以上

-qreport と **-qpdf2** の両方を指定して、リスト・ファイル内にプログラムの調整に役立つ追加情報を生成することができます。この情報は、PDF Report セクションに書き込まれます。

並列変換リストまたは並列パフォーマンス・メッセージを生成するには、以下のいずれかのオプションとともに **-qreport** を指定する必要があります。

- **-qsmp**
- **-O5**
- **-qipa=level=2**

データ再編成情報を生成するには、最適化レベル **-qipa=level=2** または **-O5** を使用して **-qreport** を指定します。再編成には、共通ブロック分割、配列分割、配列転置、メモリー割り振りのマージ、配列インターリービング、および配列合体が含まれます。

データ・プリフェッチ挿入ロケーションに関する情報を生成するには、最適化レベル **-qhot** とともに、または **-qhot** を暗黙指定する他のいずれかのオプションをとともに、**-qreport** を指定します。この情報は、リスト・ファイルの LOOP TRANSFORMATION SECTION に表示されます。さらに、**-qprefetch=assistthread** を使用してプリフェッチ支援スレッドを生成する場合、「データ・プリフェッチの支援スレッドが生成されました。」というメッセージも、リスト・ファイルの LOOP TRANSFORMATION SECTION に表示されます。

リスト・ファイルの LOOP TRANSFORMATION SECTION に、ループ・ネストに対して行われた積極的なループ変換と並列化のリストを生成するには、最適化レベル **-qhot=level=2** および **-qsmp** を **-qreport** と一緒に使用します。

疑似 Fortran コード・リストは、コンパイル可能であるというわけではありません。プログラムに疑似 Fortran コードを読み込んだり、名前が疑似 Fortran コード・リストに出ている内部ルーチンを明示的に呼び出したりしないでください。

例

myprogram.f をコンパイルして、コンパイラー・リストにループがどのように最適化されるかを示すレポートが含まれるようにするには、以下のコマンドを入力します。

```
xlf -qhot -O3 -qreport myprogram.f
```

myprogram.f をコンパイルして、コンパイラー・リストに並列化されたループがどのように変換されるかを示すレポートも含まれるようにするには、以下のコマンドを入力します。

```
xlf_r -qhot -qsmp -qreport=smplist myprogram.f
```

関連情報
- 170 ページの『-qhot』
- 258 ページの『-qsimd』
- 192 ページの『-qipa』
- 228 ページの『-qpdf1、-qpdf2』
- 262 ページの『-qsmp』

-qsaa

カテゴリー

言語エレメント制御

目的

SAA FORTRAN 言語定義に準拠しているかどうかを検査する。これは、非準拠のソース・コードと、そのような非準拠を許容するオプションを識別します。

構文

@PROCESS:

@PROCESS SAA | **NOSAA**

デフォルト

-qnosaa

使用法

-qflag オプションは、このオプションをオーバーライドすることができます。

-qlanglvl オプションを使用して、コードが国際標準に従っているかどうかを調べます。

結果

警告には、言語レベル関係の問題を示すプレフィックス **(L)** が付きます。

関連情報
- 151 ページの『-qflag』
- 198 ページの『-qlanglvl』

-qsave

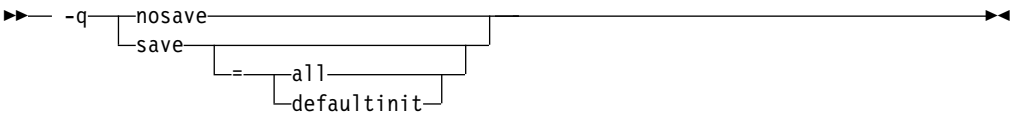
カテゴリー

言語エレメント制御

目的

ローカル変数のデフォルト・ストレージ・クラスを指定する。

構文



@PROCESS:

```
@PROCESS SAVE[({ALL | DEFAULTINIT})] | NOSAVE
```

デフォルト

このオプションのデフォルトは、使用される呼び出しコマンドによって異なります。

- **xlf** を使用して .f ファイル、.F ファイル、.f77 ファイル、または .F77 ファイルをコンパイルする場合、デフォルトは **-qsave=all** です。
- 呼び出しコマンド **f77** および **fort77** の場合、デフォルトは **-qsave=all** です。
- 他のすべての呼び出しコマンドでは、デフォルトは **-qnosave** です。

パラメーター

-qsave サブオプションは、次のとおりです。

all
デフォルトのストレージ・クラスは **STATIC** です。

defaultinit
デフォルト・ストレージ・クラスは、デフォルト初期化の指定がある派生型の変数においては **STATIC**、その他の場合は **AUTOMATIC** です。

all および **defaultinit** サブオプションは相互に排他的です。

使用法

-qnosave オプションは、デフォルト・ストレージ・クラスを **AUTOMATIC** に設定します。通常、マルチスレッド・アプリケーション、および **-qrecur** オプションを使用してコンパイルされたサブプログラムでは、この使用法に従って使用する必要があります。

-qsave オプションを指定して、FORTRAN 77 プログラムの動作を再現できます。**xlf**、**f77** および **fort77** コマンドでは、構成ファイルの中にデフォルト・オプションとして **-qsave** がリストされ、以前の動作が保持されます。デフォルトの構成ファイルのパスは、`/opt/ibm/xlf/16.1.0/etc/xlf.cfg` です。

例

以下には、派生データ型での **-qsave** オプションの影響を例示しています。

```
PROGRAM P
  CALL SUB
  CALL SUB
END PROGRAM P

SUBROUTINE SUB
  LOGICAL, SAVE :: FIRST_TIME = .TRUE.
  STRUCTURE /S/
    INTEGER I/17/
  END STRUCTURE
  RECORD /S/ LOCAL_STRUCT
  INTEGER LOCAL_VAR

  IF (FIRST_TIME) THEN
    LOCAL_STRUCT.I = 13
    LOCAL_VAR = 19
    FIRST_TIME = .FALSE.
  ELSE
    ! Prints " 13" if compiled with -qsave or -qsave=all
    ! Prints " 13" if compiled with -qsave=defaultinit
    ! Prints " 17" if compiled with -qnosave
    PRINT *, LOCAL_STRUCT
    ! Prints " 19" if compiled with -qsave or -qsave=all
    ! Value of LOCAL_VAR is undefined otherwise
    PRINT *, LOCAL_VAR
  END IF
END SUBROUTINE SUB
```

関連情報

- 247 ページの『-qrecur』
- このオプションが変数のストレージ・クラスにどのような影響を与えるかについては、「*XL Fortran* ランゲージ・リファレンス」の 『変数のストレージ・クラス』を参照してください。

-qsaveopt

カテゴリー

オブジェクト・コード制御

@PROCESS

なし。

目的

ソース・ファイルのコンパイルに使用するコマンド行オプション、構成ファイルで指定されたユーザーの構成ファイル名とオプション、コンパイル中に呼び出される各コンパイラー・コンポーネントのバージョンとレベル、およびその他の情報を対応するオブジェクト・ファイルに保管する。

構文

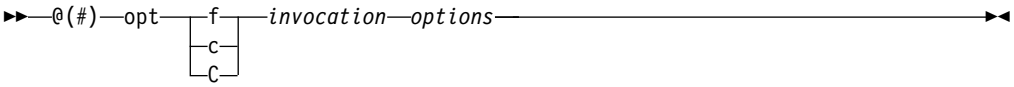
デフォルト

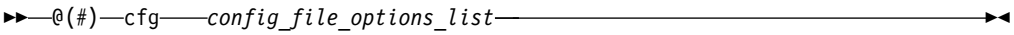
-qnosaveopt

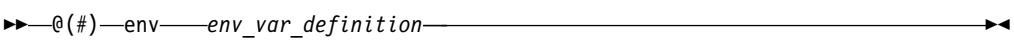
使用法

このオプションは、オブジェクト・ファイル (.o) へコンパイルするときのみ有効です (つまり、**-c** オプションを使用)。各オブジェクトには複数のコンパイル単位が含まれている場合がありますが、コマンド行オプションの 1 つのコピーのみが保存されます。@PROCESS ディレクティブで指定されたコンパイラー・オプションは、無視されます。

以下の形式を使用して、コマンド行のコンパイラー・オプション情報は、ストリングとしてオブジェクト・ファイルにコピーされます。







ここで、

f FORTRAN 言語のコンパイルを指定します。

c C 言語のコンパイルを指定します。

C C++ 言語のコンパイルを指定します。

invocation
コンパイルで使用されるコマンド (例えば、**xlf**) を表示します。

options
コマンド行に指定されたコマンド行オプションのリスト。個々のオプションはスペースで区切られています。

config_file_options_list
コンパイルで有効な、すべての構成ファイルにある **options** 属性で指定されたオプションのリスト。スペースで区切られています。

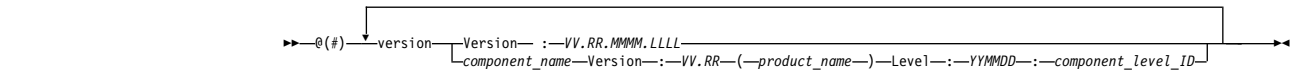
env_var_definition
コンパイラーによって使用される環境変数。現在、**XLF_USR_CONFIG** のみがリストされています。

注**:** このオプションは常時使用することができますが、環境変数 **XLF_USR_CONFIG**が設定されている場合にのみ、対応する情報が生成されます。

環境変数 **XLF_USR_CONFIG** について詳しくは、『XLF_USR_CONFIG』を参照してください。

注**:** コマンド行オプションのストリングは、64,000 バイトを超えると切り捨てられます。

コンパイル中に呼び出される各コンポーネントのバージョンとレベルと、コンパイラー・バージョンおよびリリース情報は、次の形式でオブジェクト・ファイルにも保存されます。



ここで、

V　　バージョンを表します。

R　　リリースを表します。

M　　変更を表します。

L　　レベルを表します。

component_name

このコンパイルで呼び出されたコンポーネントを指定します (低水準最適化プログラムなど)。

product_name

コンポーネントが属する製品を示します (例えば、C/C++ または Fortran)。

YYMMDD

インストールされた更新の年、月、日を表します。インストールされた更新が基本レベルである場合、そのレベルは BASE として表示されます。

component_level_ID

インストール済みコンポーネントのレベルに関連付けられている ID を表します。

この情報をオブジェクト・ファイルに書き込まずに、単に標準出力に出力する場合は、**-qversion** オプションを使用します。

例

t.f を次のコマンドでコンパイルします。

```
xlf t.f -c -qsaveopt -qhot
```

作成された t.o オブジェクト・ファイルで**strings -a** コマンドを発行すると、以下のような情報が作成されます。

```
IBM XL Fortran for Linux, Version 16.1.0
@(#)opt f /opt/ibm/xlf/16.1.0/bin/xlf t.f -c -qsaveopt -qhot
@(#)cfg -qnozerosize -qsave -qalias=intptr -qposition=appendold
 -qxlf90=noautodealloc:nosignedzero:oldpad
 -qxlf77=intarg:intxor:persistent:noleadzero:gedit77:noblankpad:oldboz:softeof
 -qxlf2003=nopolymorphic:nobozlitargs:nostopexcept:novolatile:noautorealloc:oldnaninf -bh:4
@(#)version IBM XL Fortran for Linux, V16.1
@(#)version Version: 16.01.0000.0000
@(#)version Driver Version: 16.1.0(Fortran) Level: 121020 ID: _acSDAheyEeK928eKYVtYGg
@(#)version Fortran Front End and Run Time Version: 16.1.0(Fortran) Level: 121020
ID: _0lpiYhmQEeK928eKYVtYGg
@(#)version Fortran Transformer Version: 16.1.0(Fortran) Level: 121021
ID: _gYSYgRpREeK928eKYVtYGg
@(#)version High-Level Optimizer Version: 16.1.0(C/C++) and 16.1.0(Fortran) Level: 151106
ID: _JfAAgYQ_EeWg_O7EssfHAg
@(#)version Low-Level Optimizer Version: 16.1.0(C/C++) and 16.1.0(Fortran) Level: 151030
ID: _sk208X8mEeWg_O7EssfHAg
```

1 行目では、t.f は、Fortran として使用されたソースを示し、bin/xlf は、使用された呼び出しコマンドを示し、-qhot -qsaveopt は、コンパイル・オプションを示しています。cfg で始まる 2 番目の行は、構成ファイルによって追加されたコンパイラー・オプションを示します。

残りの行は、コンパイル中に呼び出されるそれぞれのコンパイラー・コンポーネント、そのバージョンおよびレベルを一覧表示します。複数の製品で共有されるコンポーネントは、複数のバージョン番号を表示する可能性があります。表示されるレベル番号は、ご使用のシステムにインストールされた更新 によって異なる場合があります。

関連情報

- 298 ページの『-qversion』
- COMPILER_OPTIONS

-qsclk

カテゴリー

言語エレメント制御

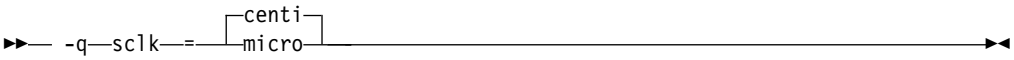
@PROCESS

なし。

目的

SYSTEM_CLOCK 組み込みプロシージャーがプログラム内で使用する解決を指定する。

構文



デフォルト

デフォルトは、センチ秒レゾリューション (**-qsclk=centi**) です。

関連情報

リアルタイム・クロックからの整数データの戻りについて詳しくは、「*XL Fortran* ランゲージ・リファレンス」の『**SYSTEM_CLOCK**』を参照してください。

-qshowpdf

カテゴリー

最適化およびチューニング

@PROCESS

なし。

目的

コンパイル・ステップとリンク・ステップで **-qpdf1** および最小最適化レベル **-O2** と共に使用すると、アプリケーション内のすべてのプロシージャーについて、追加プロファイル情報を含む PDF マップ・ファイルを作成する。

構文

デフォルト

-qshowpdf

使用法

標準的なデータを使用してアプリケーションを実行すると、プロファイル情報が Profile-Directed Feedback (PDF) ファイルに記録されます。デフォルトでは、この PDF ファイルは *.<output_name>*_pdf という名前です。 *<output_name>* は、**-qpdf1** 指定でのプログラムのコンパイル時に生成された出力ファイルの名前です。

PDF ファイルに加えて、コンパイラーは、PDF1 ステップで静的情報が入った PDF マップ・ファイルも生成します。デフォルトでは、この PDF マップ・ファイルは *.<output_name>*_pdf_map と名前が付けられます。 これらの 2 つのファイルを **showpdf** ユーティリティーとともに使用して、アプリケーションのプロファイル情報の一部をテキストまたは XML フォーマットで表示できます。**showpdf** ユーティリティーについて詳しくは、「*XL Fortran* 最適化およびプログラミング・ガイド」の『showpdf』を参照してください。

プロファイル情報を表示する必要がない場合は、PDF1 ステップで **-qnoshowpdf** オプションを指定して、 PDF マップ・ファイルが生成されないようにしてください。この方法により、コンパイル時間を短縮できます。

関連情報

- 228 ページの『-qpdf1、-qpdf2』
- 329 ページの『showpdf』
- 『環境変数の正しい設定方法』

-qsigtrap

カテゴリー

エラー・チェックおよびデバッグ

@PROCESS

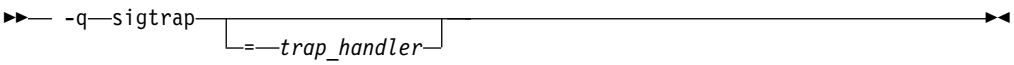
なし。

目的

メインプログラムを含むファイルのコンパイル時に、**SIGTRAP** および **SIGFPE** 例外をキャッチするために指定されたトラップ・ハンドラーをセットアップする。

このオプションを使用すれば、プログラム内の **SIGNAL** または **SIGFPE** サブプログラムを呼び出さなくても、**SIGTRAP** シグナル用にハンドラーをインストールすることができます。

構文



デフォルト

適用されません。

使用法

ハンドラー名を指定せずに **-qsigtrap** オプションを指定した場合、デフォルトで `xl__trce` トラップ・ハンドラーが使用されます。 別のトラップ・ハンドラーを使用可能にするには、 **-qsigtrap** オプションでそのハンドラー名を指定してください。

別のハンドラーを指定する場合は、それが入っているオブジェクト・モジュールがプログラムとリンクされていることを確認してください。 XL Fortran によって提供されるトラップ・ハンドラーより生成されるトレースバック内の詳細情報 (xl__trce など) を表示するには、**-qlinedebug** または **-g** オプションを指定します。

関連情報

- 58 ページの『XL Fortran 実行時例外』 は例外の考えられる原因を説明します。
- 「*XL Fortran* 最適化およびプログラミング・ガイド」の『浮動小数点例外の検出とトラッピング』では、浮動小数点計算の結果生じる例外を処理するためのいくつかの方法が説明されています。
- 「*XL Fortran* 最適化およびプログラミング・ガイド」の『例外ハンドラーのインストール』には、XL Fortran が提供する例外ハンドラーがリストされています。

-qsimd

カテゴリー

最適化およびチューニング

@PROCESS

なし。

目的

ベクトル命令をサポートするプロセッサーでコンパイラーがベクトル命令を自動的に利用できるかどうかを制御する。

これらの命令は、マルチメディア・アプリケーションなどの算法集中型のタスクと共に使用された場合、より高いハイパフォーマンスを提供することができます。

構文

デフォルト

-qsimd が指定されているかどうかに関係なく、**-O3** 以上の最適化レベルで **-qsimd=auto** が暗黙指定され、**-O2** 以下の最適化レベルで **-qsimd=noauto** が暗黙指定されます。

使用法

-qsimd=auto オプションは、ベクトル命令をサポートするプロセッサーでベクトル命令の自動生成を行えるようにします。 **-qsimd=auto** が有効な場合、コンパイラーは、連続する配列エレメントのループ内で実行される特定の演算を、ベクトル命令に変換します。これらの命令は一度にいくつかの結果を計算するため、それぞれの結果を順次計算するよりも高速です。これらのオプションは、重要なイメージ処理要求を持つアプリケーションに役立ちます。

-qsimd=noauto オプションは、ループ配列演算のベクトル命令への変換を使用不可にします。さらに細かく制御を行うには、**-qstrict=ieeefp**、**-qstrict=operationprecision**、および **-qstrict=vectorprecision** を使用します。詳しくは、 274 ページの『-qstrict』を参照してください。

-qsimd=auto オプションは、自動 SIMD 化を制御します。これは以前は非推奨の **-qhot=simd** オプションによって実行されていました。 **-qhot=simd** を指定すると、コンパイラーはそれを無視し、警告メッセージを発行しません。

注**:**

- サブオプションなしで **-qsimd** を指定することは、 **-qsimd=auto** を指定することと同等です。
- **-qsimd=auto** を指定しても、自動 SIMD 化が行われることは保証されません。
- ベクトル命令を使用して一度に複数の結果を計算すると、一部のアーキテクチャーで遅延が生じたり、さらには浮動小数点例外の検出ミスが生じたりすることがあります。例外の検出が重要である場合は、**-qsimd=auto** を使用しないでください。

規則

IPA を使用可能にして、IPA のコンパイル・ステップで **-qsimd=auto** で指定したにもかかわらず、IPA リンク・ステップで **-qsimd=noauto** を指定した場合、コン

パイラーは自動的に IPA リンク・ステップで **-qsimd=auto** を設定します。同様に、IPA を使用可能にして、IPA コンパイル・ステップでは **-qsimd=noauto** を指定し、IPA リンク・ステップでは **-qsimd=auto** を指定する場合、コンパイラーはコンパイル・ステップで **-qsimd=auto** を自動的に設定します。

関連情報

- 113 ページの『-qarch』
- 248 ページの『-qreport』
- 274 ページの『-qstrict』
- **NOSIMD**ディレクティブ (「*XL Fortran* ランゲージ・リファレンス」内)。
- 「*XL Fortran* 最適化およびプログラミング・ガイド」の『プロシージャー間分析 *(IPA)*』。

-qslmtags

カテゴリー

リスト、メッセージ、およびコンパイラー情報

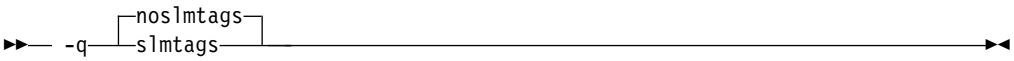
@PROCESS

なし。

目的

これは、SLM タグ・ロギングがコンパイラー・ライセンス使用状況をトラッキングするかどうかを制御します。

構文



デフォルト

-qnoslmtags

使用法

-qslmtags を指定して、ライセンス使用状況の追跡を有効にすることができます。**-qslmtags** が有効になっている場合、コンパイラーは SLM タグ形式でライセンス使用状況をログに記録します。記録先の場所は、構成ファイルの **slm_dir** 属性を指定することによって定義できます。デフォルトの場所は `/var/opt/ibm/xl-compiler/` (デフォルト・インストールの場合) または *$prefix*`/var/opt/ibm/xl-compiler/` (デフォルト以外のインストールの場合) です。 *$prefix* はデフォルト以外のインストール・パスです。 **slm_dir** のデフォルトを変更する場合は、ターゲット・ディレクトリーを作成し、その許可をすべてのコンパイラー・ユーザーによる読み取り可能、書き込み可能、および実行可能に設定する必要があります。例えば、次のコマンドを実行できます。

```
chmod 777 $slm_dir
```

コンパイラーは呼び出されるたびに、許可ユーザーがリストされているファイルに、ユーザーの uid が存在するかどうかに応じて、その呼び出し内容を同時ユーザーによる呼び出しとして記録したり許可ユーザーによる呼び出しとして記録したりします。

関連情報

- 337 ページの『第 9 章 コンパイラー・ライセンス使用状況の追跡』
- 17 ページの『構成ファイルの属性』

-qsmallstack

カテゴリー

最適化およびチューニング

@PROCESS

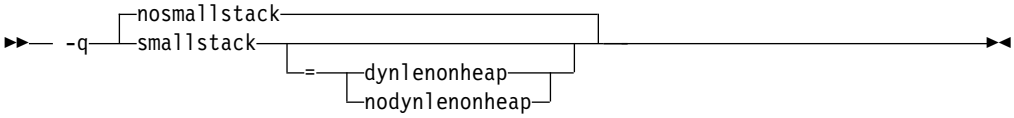
なし。

目的

可能な場合にスタック使用量を最小にする。

このコンパイラー・オプション 2 つの相異なる、しかし関連した変換のセット (一般小スタック変換および動的長さ変数割り振り変換) を制御します。これらの 2 種類の変換は互いに独立して制御できます。

構文



デフォルト

-qnosmallstack

パラメーター

dynlenonheap | nodynlenonheap
-qsmallstack=dynlenonheap サブオプションは、非定数の文字長または非定数の配列境界 (DYNamic LENgth ON HEAP) を持つ自動オブジェクトに影響を与えます。指定すると、これらの自動変数はヒープ上に割り振られます。このサブオプションを指定しないと、これらの自動変数はスタック上に割り振られます。

デフォルト

デフォルトの **-qnosmallstack** では、すべてのサブオプションがオフであることが暗黙指定されます。

使用法

このオプションの使用によりプログラム・パフォーマンスに逆に作用することがあるので、スタックに大量のデータを割り振るプログラム用にのみ使用する必要があります。

サブオプションを指定しない **-qsmallstack** では一般的な小規模なスタック変換のみが可能です。

-qnosmallstack では一般的な小規模なスタック変換のみが使用不可です。**dynlenonheap** 変換を使用不可にするには、**-qsmallstack=nodynlenonheap** を同様に指定します。

-qsmallstack=dynlenonheap では動的長さ変数割り振りおよび一般的な小規模スタック変換が使用可能です。

dynlenonheap 変換のみを使用可能にするには、**-qsmallstack=dynlenonheap -qnosmallstack** を指定します。

-qsmallstack および **-qstacktemp** オプションの両方が使用されるとき、一時的変数の値が非ゼロの場合、この設定値が **-qsmallstack** の設定値と競合するとしても、適用できる一時的変数を割り振るため **-qstacktemp** 設定値は使用されます。**-qsmallstack** 設定値は **-qstacktemp** により影響を与えられない変換を適用し続けます。

関連情報

- 270 ページの『-qstacktemp』

-qsmp

カテゴリー

最適化およびチューニング

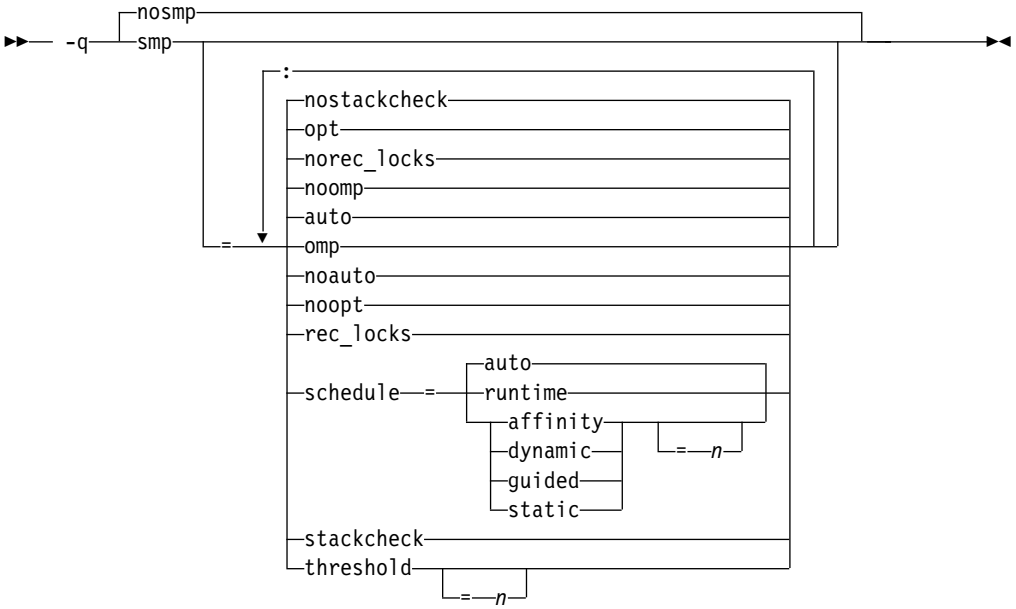
@PROCESS

なし。

目的

プログラム・コードの並列化を使用可能にする。

構文



デフォルト

-qnosmp。コードが、ユニプロセッサー・マシン用に作成されます。

パラメーター

auto | noauto
auto は、プログラム・コードの自動並列化と最適化を有効にします。これは、ユーザーとコンパイラーのどちらが生成したループも自動的に並列化する、コンパイラーによる試行です。**noauto** は、OpenMP ディレクティブで明示的に注釈が付けられたプログラム・コードのみを並列化します。 **-qsmp=omp** または **-qsmp=noopt** を指定すると、**noauto** が暗黙指定されます。

omp | noomp
omp は **noauto** を暗黙指定します。つまり、OpenMP ディレクティブで注釈が明示的に付けられたプログラム・コードのみを並列化します。**noomp** が有効な場合は、**auto** が暗黙指定されます。

また、**omp** を指定すると、以下のような効果があります。

- **-qcclines** コンパイラー・オプションが使用可能になります。
- C プリプロセッサーが起動すると、_OPENMP という C プリプロセッサー・マクロも、XL Fortran がサポートする最新の OpenMP API 仕様に基づいて定義されます。このマクロは、条件付きコンパイルをサポートする場合に役立ちます。詳細については、「*XL Fortran* ランゲージ・リファレンス」の『条件付きコンパイル』を参照してください。

opt | noopt
opt により、並列化プログラム・コードの最適化を有効にします。**noopt** は、コードを並列化するために必要な最小量の最適化を実行します。これは、デフォルトで **-qsmp** が **-O2** および **-qhot** オプションを使用可能にし、いくつかの変数をレジスターに移動してデバッガーでアクセス不能にする結果になるので、デバッグに役立ちます。しかし、**-qsmp=noopt** および **-g** オプションを指定すると、これらの変数はまだデバッガーからは使用できます。

rec_locks | norec_locks

CRITICAL 構造体と関連付けられている問題を避けるために、再帰的ロックを使用するかどうかを判別します。**rec_locks** が有効なとき、ネストされたクリティカル・セクションはデッドロックを起こしません。同じ名前の別の CRITICAL 構文の動的範囲から、CRITICAL 構文を実行することができます。**rec_locks** サブオプションは、OpenMP API との整合性のない CRITICAL 構文の動作を指定します。

schedule

ソース・コードで他のスケジューリング・アルゴリズムが明示的に割り当てられていないループに使用されている、スケジューリング・アルゴリズムの種類、および (**auto** の場合を除き) チャンク・サイズ (*n*) を指定します。 **schedule** サブオプションのサブオプションは、次のとおりです。

affinity[=*n*]

最初にループの繰り返しは、**ceiling**(*number_of_iterations*/*number_of_threads*) 回の繰り返しを含む *n* 個の区画に分割されます。各区画は、最初にスレッドに割り当てられてから、それぞれ *n* 回の繰り返しを含むチャンクにさらに分割されます。*n* が指定されていないと、チャンクは **ceiling**(*number_of_iterations_left_in_partition* / 2) 回のループの繰り返しを含みます。

スレッドが解放されると、このスレッドに最初に割り当てられていた区画から次のチャンクが取得されます。その区画の中にチャンクがなくなると、スレッドは、別のスレッドに最初に割り当てられた区画から使用可能な次のチャンクを取得します。

最初にスリープ・スレッドに割り当てられている区画での作業は、アクティブなスレッドによって完了します。

類縁性スケジューリング・タイプは、OpenMP API 仕様の一部ではありません。

注**:** これは推奨されないサブオプションです。類似の機能として、**OMP_SCHEDULE** 環境変数に **dynamic** 節を指定して使用できます。

auto

ループの繰り返しのスケジューリングがコンパイラーとランタイム・システムに委任されます。コンパイラーとランタイム・システムは、実行可能なスレッドへの繰り返しのマッピングを任意に選択することができ (考えられるすべての有効なスケジュール・タイプを含みます)、別のループではそれらが異なっていることがあります。チャンク・サイズ (*n*) を指定しないでください。

dynamic[=*n*]

ループの繰り返しは、それぞれ *n* 回の繰り返しを含むチャンクに分割されます。*n* が指定されていない場合は、各チャンクには 1 つの繰り返しが含まれます。

アクティブ・スレッドには、これらのチャンクが「先着順実行」の基準で割り当てられます。残りの作業のチャンクは、すべての作業の割り当てが完了するまで、使用可能なスレッドに割り当てられます。

guided[=*n*]
ループの繰り返しは、チャンクの最小サイズである *n* ループの繰り返しに達するまで、徐々により小さなチャンクに分割されます。 *n* が指定されなかった場合、*n* のデフォルト値は 1 回の繰り返しです。

アクティブ・スレッドには、チャンクが「先着順実行」の基準で割り当てられます。最初のチャンクには **ceiling**(*number_of_iterations*/*number_of_threads*) 繰り返しが含まれます。それ以降のチャンクは、**ceiling**(*number_of_iterations_left* / *number_of_threads*) 繰り返しを含みます。

runtime
チャンク入れのアルゴリズムを実行時に決定することを指定します。

static[=*n*]
ループの反復は、*n* 反復を含むチャンクに 1 つずつ分割されます。各スレッドには、「ラウンドロビン」方式でチャンクが割り当てられます。これをブロック巡回スケジューリング と言います。 *n* の値が 1 である場合は、*n* の値が 1 のスケジューリング・タイプは、特に巡回スケジューリングと呼ばれます。

n を指定しない場合、チャンクには **floor**(*number_of_iterations*/*number_of_threads*) 回の繰り返しが含まれます。最初の **remainder**(*number_of_iterations*/*number_of_threads*) チャンクには複数の繰り返しがあります。スレッドにはそれぞれ別個のチャンクが割り当てられます。 これをブロック・スケジューリング と言います。

スレッドは、スリープ状態で作業を割り当てられている場合、その作業を完了できるようスリープ状態から解除されます。

n 1 以上の整数の値でなければなりません。

サブオプションなしで **schedule** を指定すると、**schedule=auto** を指定した場合と同じになります。

チャンク入れのアルゴリズムと SCHEDULE について詳しくは、「*XL Fortran* ランゲージ・リファレンス」の『ディレクティブ』を参照してください。

stackcheck | nostackcheck
実行時にスレーブ・スレッドによってスタック・オーバーフローの有無を確認し、残りのスタック・サイズが XLSMPOPTS 環境変数の **stackcheck** オプションに指定されたバイト数に満たない場合は警告を出すようにコンパイラーに指示します。このサブオプションはデバッグの目的で設けられているもので、**XLSMPOPTS=stackcheck** も設定されている場合にのみ効果があります。詳しくは、XLSMPOPTS (「*XL Fortran* 最適化およびプログラミング・ガイド」)を参照してください。

threshold[=*n*]
-qsmp=auto が有効なとき、行われる自動ループ並列化の程度を制御します。*n* の値は、並列化されるためにループで必要な最小の作業量を表します。現在、「work」の計算の大部分は、ループ内の繰り返し回数で占められています。通常は、*n* に高い値を指定すればするほど、並列化されるループの数は少なくなります。値 0 を指定すると、それが有益であろうとなかろうと、コンパイラーがすべての自動並列可能なループを並列化するよう指示します。値 100 を指定

すると、コンパイラーに、有益であると思われる自動並列可能なループだけを並列化するよう指示します。100 より大きい値を指定すると、より多くのループがシリアライズされます。

n 0 以上の正整数になる必要があります。

サブオプションなしで **threshold** を指定すると、プログラムはデフォルト値の 100 を使用します。

サブオプションなしで **-qsmp** を指定すると、以下と等価になります。

```
-qsmp=auto:opt:noomp:norec_locks:schedule=auto:
nostackcheck:threshold=100
```

使用法

- **omp** サブオプションを指定すると、**noauto** が暗黙指定されます。**-qsmp=omp:auto** を指定して、OpenMP 準拠アプリケーションに自動並列化を同様に適用します。
- **-qsmp** オプションは **-qdirective=SMP¥$:¥$OMP:IBMP** を暗黙指定します。これにより、デフォルトのトリガー定数 **IBM*** に加えて、トリガー定数 **SMP$**、**$OMP**、および **IBMP** がオンになります。
- **f77** または **fort77** コマンドを **-qsmp** オプションと一緒に使用してプログラムをコンパイルする場合は、デフォルト・ストレージ・クラスを自動的に作成するために **-qnosave** を指定し、スレッド・セーフ・コードを生成するようコンパイラーに指示するために **-qthreaded** を指定します。
- **-qsmp=opt** オプションで生成されたオブジェクト・ファイルは、 **-qsmp=noopt** で生成されたオブジェクト・ファイルとリンクすることができます。各オブジェクト・ファイル内の変数のデバッガーにおける可視性は、リンクによって影響を受けることはありません。
- **-qsmp** を指定すると、**-O2** が暗黙的に設定されます。 **-qsmp** オプションは **-qnooptimize** をオーバーライドしますが、 **-O3**、**-O4** または **05** はオーバーライドしません。並列化されたプログラム・コードをデバッグするときには、**-qsmp=noopt** を指定して、並列化されたプログラム・コードの最適化を使用不可にすることができます。
- **-qsmp=noopt** サブオプションは、コマンド行上のどこにあっても、パフォーマンス最適化オプションをオーバーライドします (**-qsmp** の前に **-qsmp=noopt** が現れる場合以外は)。例えば、**-qsmp=noopt -O3** は **-qsmp=noopt** と、**-qsmp=noopt -O3 -qsmp** は **-qsmp -O3** と等しいです。

例

以下の例では、critical 構文が原因で生じるデッドロックを回避するために、**-qsmp=rec_locks** を指定してください。

```
   program t
     integer  i, a, b

     a = 0
     b = 0
!smp$ parallel do
     do i=1, 10
!smp$ critical
     a = a + 1
```

```
!smp$ critical
     b = b + 1
!smp$ end critical
!smp$ end critical
     enddo
   end
```

関連情報

- 138 ページの『-qdirective』
- 101 ページの『-O』
- 「*XL Fortran* 最適化およびプログラミング・ガイド」のXLSMPOPTS 環境変数、および 並列化ディレクティブ

-qsource

カテゴリー

リスト、メッセージ、およびコンパイラー情報

目的

リストのソース・セクションを含むコンパイラー・リスト・ファイルを作成し、エラー・メッセージの印刷時にソース情報を追加して提供する。

構文

@PROCESS:

@PROCESS SOURCE | **NOSOURCE**

デフォルト

-qnosource

使用法

コンパイラーが問題を検出すると、このオプションは端末に個々のソース行を表示します。これは、Fortran ソース・ファイルにおけるプログラム・エラーを診断するのに非常に役立ちます。

印刷したいプログラムのそれらのソース・コード部分を囲むソース・ファイル内の **@PROCESS** ディレクティブに **SOURCE** および **NOSOURCE** を使用することにより、ソース・コードの一部を選択的に印刷することができます。この場合に限り、**@PROCESS** ディレクティブはコンパイル単位の最初の文の前にある必要はありません。

例

次の例では、 **-qsource** オプションでプログラムがコンパイルされた場合に、誤った呼び出しが行われる時点がさらにはっきりと識別されます。

```
$ cat argument_mismatch.f
         subroutine mult(x,y)
         integer x,y
         print *,x*y
         end

         program wrong_args
         interface
                 subroutine mult(a,b)   ! Specify the interface for this
                         integer a,b    ! subroutine so that calls to it
                 end subroutine mult    ! can be checked.
         end interface
         real i,j
         i = 5.0
         j = 6.0
         call mult(i,j)
         end
```

```
$ xlf95 argument_mismatch.f
** mult   === End of Compilation 1 ===
"argument_mismatch.f", line 15.20: 1513-061 (S) Actual argument attributes
do not match those specified by an accessible explicit interface.
** wrong_args  === End of Compilation 2 ===
1501-511  Compilation failed for file argument_mismatch.f.
$ xlf95 -qsource argument_mismatch.f
** mult   === End of Compilation 1 ===
        15 |   call mult(i,j)
                .....a...
a - "argument_mismatch.f", line 15.20: 1513-061 (S) Actual argument attributes do not match those
specified by an accessible explicit interface.
** wrong_args   === End of Compilation 2 ===
1501-511  Compilation failed for file argument_mismatch.f.
```

関連情報

- 72 ページの『リスト、メッセージ、およびコンパイラー情報』
- 352 ページの『ソース・セクション』

-qspillsize

カテゴリー

コンパイラーのカスタマイズ

目的

-qspillsize は **-NS** の長い形式です。 101 ページの『-NS』を参照してください。

構文

@PROCESS:

@PROCESS SPILLSIZE(*bytes*)

デフォルト

適用されません。

-qstackprotect

カテゴリー

エラー・チェックおよびデバッグ

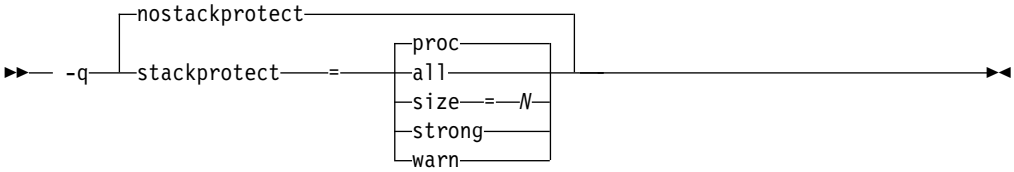
@PROCESS

なし。

目的

スタックを上書きまたは破壊する悪意のある入力データやプログラミング・エラーから保護する。

構文



デフォルト

-qstackprotect が指定されていない場合は -qnostackprotect。

-qstackprotect がサブオプションなしで指定されている場合は -qstackprotect=proc。

パラメーター

all
これは、脆弱なオブジェクトがプロシージャー に含まれているかどうかに関係なく、すべてのプロシージャーを保護します。

proc
バッファー・オーバーフローを防ぐためのコードを提供します。

size=*N*
これは、*N* バイト以上のサイズを持つ自動配列を含むプロシージャーを保護します。**-qstackprotect** オプションが有効になっているときのデフォルト・サイズは 8 バイトです。

strong
これは、ローカル配列定義を持つ、またはローカル・フレーム・アドレスに対する参照を持つ、追加プロシージャーを保護します。

warn
関数に含まれる配列のサイズが *N* バイトより小さい場合に、警告を出します。

使用法

-qstackprotect は、脆弱なオブジェクトを持つプロシージャーをスタック破壊から保護するための追加コードを生成します。実行時のパフォーマンスが低下する可能性があるため、 オプションはデフォルトでは無効になっています。

脆弱なオブジェクトを持つすべてのプロシージャーを保護するためのコードを生成するには、次のコマンドを入力します。

```
xlf myprogram.f -qstackprotect=all
```

特定サイズのオブジェクトを持つプロシージャーを保護するためのコードを生成するには、**size=** パラメーターにそのオブジェクト・サイズをバイト単位で設定して、次のコマンドを入力します。

```
xlf myprogram.f -qstackprotect=size=8
```

-qstacktemp

カテゴリー

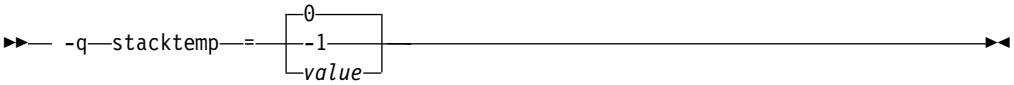
最適化およびチューニング

目的

実行時に特定の XL Fortran コンパイラー一時ファイルを割り振る場所を決定する。

適当なコンパイラーのテンポラリー (temporary) は、安全にこれらを適用できるとコンパイラーが判別するとき、コンパイラーが自身の使用のため、自身により作成された一連の臨時的な変数です。最も一般的には、アレイ言語セマンティクス用の XL Fortran アレイのコピーを保持するため、コンパイラーはこれらの種類のテンポラリー (temporary) を (組み込み関数またはユーザー・サブプログラムと連動して) 作成します。

構文



@PROCESS:

```
@PROCESS STACKTEMP={0 | -1 | value}
```

デフォルト

-qstacktemp=0

パラメーター

使用できるサブオプションは、次のとおりです。

0 ターゲットの環境に基づき、コンパイラーは、ヒープまたはスタック上で適当なテンポラリー (temporary) を割り当てるかどうかを判別します。この

設定によりご使用のプログラムがスタック・ストレージを使い尽くす場合、代わりに非ゼロ値の指定を試みるか、**-qsmallstack** オプションの使用を試みてください。

-1 スタックに適当なテンポラリー (temporary) を割り振ります。一般に、これが最良の実行設定ですが、スタック・ストレージの大部分を使用します。

value スタックの値より小さい適当なテンポラリー (temporary) を、ヒープの値より大か等しいものを割り当てます。 *value* 正の整数です。多くのプログラムの場合、スタック・ストレージおよびパフォーマンス間で、1 MB の値はよい妥協であると示されていますが、ご使用のアプリケーションの特性に基づきこの数値を調整する必要がある場合があります。

使用法

大規模アレイを利用するプログラムを所有している場合、それらを実行するときスタック・スペースのオーバーフローを防ぐためこのオプションを使用する必要があります。例えば、スタック・スペースにより拘束されるSMP または OpenMP アプリケーションの場合、これらのオプションを使用してコンパイラーのテンポラリー (temporary) をスタックからのヒープに移動できます。

コンパイラーは、アプリケーションが実行しているとき、スタックの限界が越えられているか否かを検出できません。ご使用のアプリケーション用に作動する設定値を見つける前に、いくつかの設定値を経験する必要があります。現在の設定値をオーバーライドするには、新規の設定値を指定する必要があります。

-qstacktemp オプションはある種のコンパイラー生成テンポラリー (temporary) の場合は、**-qsmallstack** オプションに対して優先します。

関連情報

- 261 ページの『-qsmallstack』

-qstaticlink

カテゴリー

リンク

@PROCESS ディレクティブ

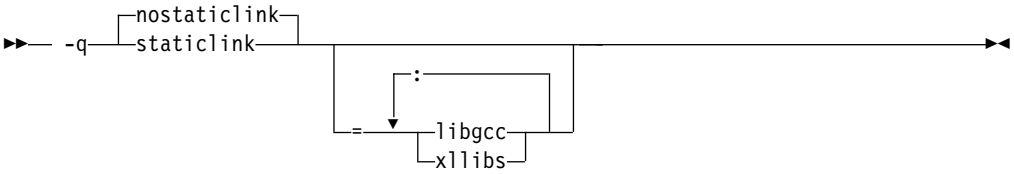
なし。

目的

静的または共有のランタイム・ライブラリーをアプリケーションにリンクするかどうかを制御する。

このオプションは、単独または組み合わせて使用される GNU オプション **-static**、**-static-libgcc**、**-shared**、および **-shared-libgcc** によって暗黙指定されるリンク規則と同等の規則を指定できるようにします。

構文



デフォルト

-qnostaticlink

パラメーター

libgcc

- **libgcc** と **nostaticlink** を同時に指定すると、コンパイラーは **libgcc** の共有バージョンをリンクします。
- **libgcc** と **staticlink** を同時に指定すると、コンパイラーは **libgcc** の静的バージョンをリンクします。

このサブオプションは、GNU オプションの **-static-libgcc** および **-shared-libgcc** によって有効にされるものと同等の機能を提供します。

xllibs

- **xllibs** を **-qnostaticlink** とともに指定すると、コンパイラーは、XL コンパイラー・ライブラリーの共有バージョンをリンクします。
- **xllibs** を **-qstaticlink** とともに指定すると、コンパイラーは、XL コンパイラー・ライブラリーの静的バージョンをリンクします。

使用法

-qstaticlink をサブオプションなしで指定すると、静的ライブラリーのみがオブジェクト・ファイルとリンクします。

-qnostaticlink をサブオプションなしで指定すると、共有ライブラリーがオブジェクト・ファイルとリンクします。

-qstaticlink=xllibs および **-qmkshrobj** を指定した場合は、両方のオプションが有効になります。コンパイラーは、XL ライブラリーへの参照がすべて静的にリンクされる共有オブジェクトを作成します。

競合するコンパイラー・オプションは、以下のように解決されます。

- 最初に **-qnostaticlink** をサブオプションなしで指定し、次に **-qstaticlink** をサブオプションありまたはなしで指定すると、**-qnostaticlink** はオーバーライドされます。例えば、**-qnostaticlink -qstaticlink=xllibs** は **-qstaticlink=xllibs** と同等です。
- **-qstaticlink** をサブオプションありまたはなしで指定し、それに続いて **-qnostaticlink** をサブオプションなしで指定すると、**-qnostaticlink** が有効になり、共有ライブラリーがリンクします。または、**-qstaticlink** をサブオプション

なしで指定すると、**-qstaticlink** が有効になり、静的ライブラリーのみがオブジェクト・ファイルとリンクします。次の例を参照してください。

表 *20.* 競合するコンパイラー・オプションの例と解決

オプションの組み合わせ	コンパイラーの動作
-qstaticlink=libgcc -qnostaticlink	共有ライブラリーがリンクされます。
-qstaticlink -qnostaticlink=libgcc	すべてのライブラリーが静的にリンクされます。コンパイラーは次の警告メッセージを発行します。 (W) The options -qnostaticlink=libgcc and -qstaticlink are incompatible. Option -qnostaticlink=libgcc is ignored.
-qstaticlink -qnostaticlink=libgcc:xllibs	すべてのライブラリーが静的にリンクされます。コンパイラーは次の警告メッセージを発行します。 (W) The options -qnostaticlink=libgcc and -qstaticlink are incompatible. Option -qnostaticlink=libgcc is ignored. (W) The options -qnostaticlink=xllibs and -qstaticlink are incompatible. Option -qnostaticlink=xllibs is ignored.
-qstaticlink -qstaticlink=libgcc	すべてのライブラリーが静的にリンクされます。
-qnostaticlink=libgcc -qstaticlink	すべてのライブラリーが静的にリンクされます。

注**:** メッセージ・カタログがシステムにインストールされていない状態でランタイム・ライブラリーが静的にリンクされると、メッセージ番号のみのメッセージが発行され、メッセージ・テキストは表示されません。

重要**:** サード・パーティーのライブラリーまたは製品を使用する場合は、それぞれ該当するライセンス条件の制約を受けます。**-qstaticlink** オプションを使用すると、コンパイルしたプログラムに重大な法律上の影響をもたらす可能性があります。このオプションを使用する前に、法律上のアドバイスを求めることを強く推奨します。

以下の表は、共有ライブラリーおよび非共有ライブラリーのリンケージを指定する同等の GNU オプションおよび XL Fortran オプションを示しています。

表 *21.* オプション・マッピング*: GNU* リンカーの制御

GNU オプション	意味	**XL Fortran** オプション
-shared	共有オブジェクトをビルドする。	-qmkshrobj
-static	静的オブジェクトをビルドして、共有ライブラリーとリンクしないようにする。リンクしているライブラリーはすべて静的ライブラリーである必要があります。	-qstaticlink
-shared-libgcc	共有バージョンの libgcc とのリンク。	-qnostaticlink=libgcc **1**
-static-libgcc	静的バージョンの libgcc とのリンク。ユーザーの共有ライブラリーはリンクすることができます。	-qstaticlink=libgcc
注: **1** これはデフォルトの設定です。		

関連情報

- 217 ページの『-qmkshrobj』

-qstrict

カテゴリー

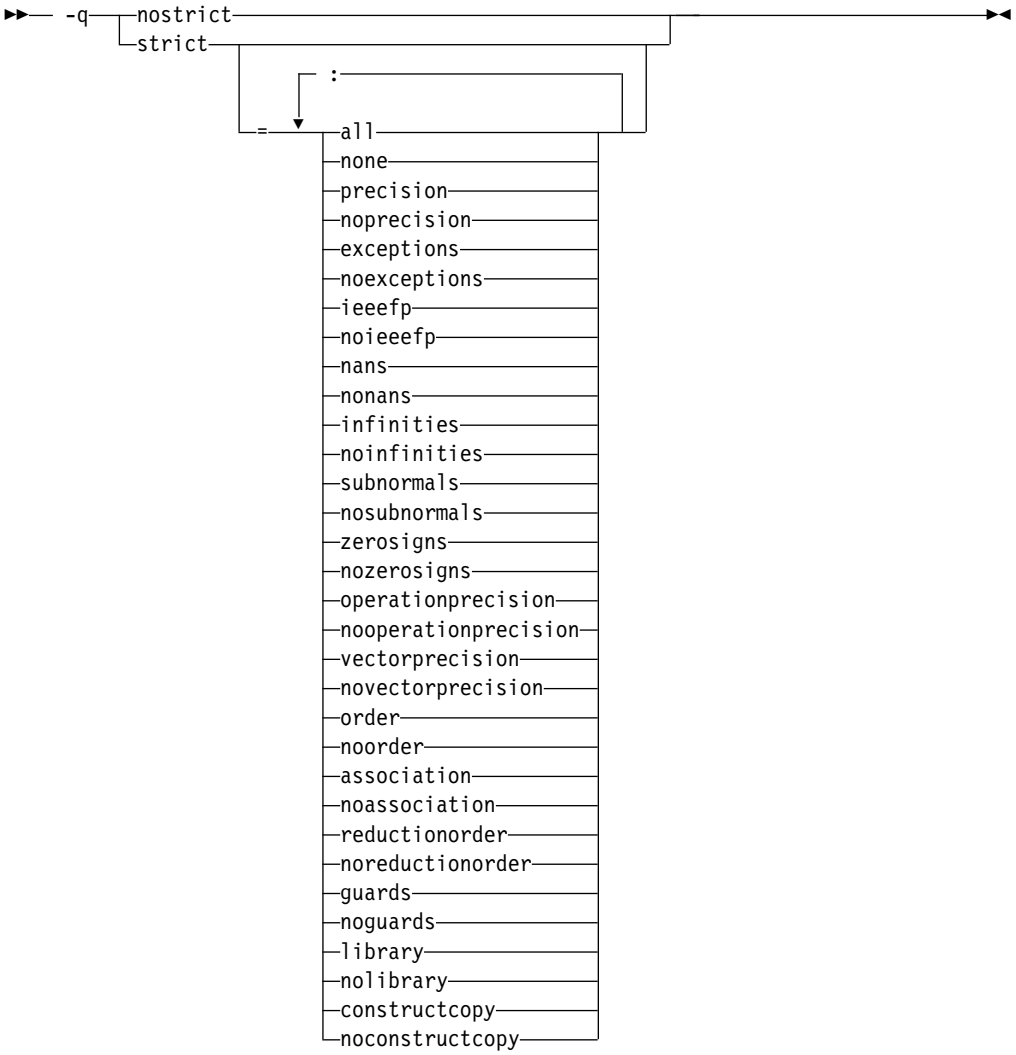
最適化およびチューニング

目的

これは、デフォルトでは最適化レベル **-O3** 以上で行われ、オプションでは **-O2** で行われる最適化によって、厳密な IEEE 浮動小数点準拠に大きく関連する特定のプログラムのセマンティクスが変更されないようにします。

このオプションは、最適化されたプログラムのプログラム実行における変更により、最適化されていないプログラムとは異なる結果が生成される状態の場合に使用します。

構文



@PROCESS:

```
@PROCESS STRICT[(suboptions)]  | NOSTRICT
```

デフォルト

- 最適化レベル **-qnoopt** または **-O0** が有効になっているときは **-qstrict** または **-qstrict=all** は常に有効です。
- **-O2** または **-O** 最適化レベルが有効な場合は、**-qstrict** または **-qstrict=all** がデフォルトです。
- **-O3** 以上の最適化レベルが有効な場合は、**-qnostrict** または **-qstrict=none** がデフォルトです。

パラメーター

-qstrict サブオプションには以下が含まれます。

all | none
all は、**ieeefp**、**order**、**library**、**constructcopy**、**precision**、および **exceptions**

サブオプションによって制御されるものを含め、すべてのセマンティクス変更トランスフォーメーションを無効にします。**none** では、これらのトランスフォーメーションが有効になります。

precision | noprecision
precision は、**subnormals**、**operationprecision**、**vectorprecision**、**association**、**reductionorder**、および **library** サブオプションによって制御されるものを含め、浮動小数点の精度に影響を与える可能性のあるすべてのトランスフォーメーションを無効にします。**noprecision** では、これらのトランスフォーメーションが有効になります。

exceptions | noexceptions
exceptions は、**nans**、**infinities**、**subnormals**、**guards library**、および **constructcopy** サブオプションによって制御されるものを含め、例外に影響を与える、または例外の影響を受ける可能性のあるすべてのトランスフォーメーションを無効にします。**noexceptions** では、これらのトランスフォーメーションが有効になります。

ieeefp | noieeefp
ieeefp は、**nans**、**infinities**、**subnormals**、**zerosigns**、**vectorprecision**、および **operationprecision** サブオプションによって制御されるものを含め、IEEE 浮動小数点への準拠に影響を与えるトランスフォーメーションを無効にします。**noieeefp** では、これらのトランスフォーメーションが有効になります。

nans | nonans
nans は、IEEE 浮動小数点 NaN (非数字) 値が存在する場合に誤った結果を生成する可能性があるか、またはこの値を不正確に生成する可能性があるトランスフォーメーションを無効にします。**nonans** では、これらのトランスフォーメーションが有効になります。

infinities | noinfinities
infinities は、浮動小数点の無限大が存在する場合に誤った結果を生成する可能性があるか、または浮動小数点の無限大を不正確に生成する可能性のあるトランスフォーメーションを無効にします。**noinfinities** では、これらのトランスフォーメーションが有効になります。

subnormals | nosubnormals
subnormals は、IEEE 浮動小数点のサブノーマル (非正規化数) (以前は「デノーマル」と呼ばれていました) が存在する場合に誤った結果を生成する可能性があるか、またはサブノーマルを不正確に生成する可能性があるトランスフォーメーションを無効にします。**nosubnormals** では、これらのトランスフォーメーションが有効になります。

zerosigns | nozerosigns
zerosigns は、浮動小数点ゼロの符号が正しいかどうかに影響を与える、またはその影響を受ける可能性のあるトランスフォーメーションを無効にします。**nozerosigns** では、これらのトランスフォーメーションが有効になります。

operationprecision | nooperationprecision
operationprecision は、各浮動小数点演算の概算結果を生成するトランスフォーメーションを無効にします。 **nooperationprecision** では、これらのトランスフォーメーションが有効になります。

vectorprecision | novectorprecision

vectorprecision は、ベクトル化した繰り返しによって、ベクトル化しない残りの繰り返しと異なる結果が生成される可能性がある場合、ループでのベクトル化を使用不可にします。**vectorprecision** を使用すると、同一のデータに対する同一の浮動小数点演算のすべてループの繰り返しで、同一の結果が生成されます。

novectorprecision は、さまざまな繰り返しによって、同じ入力から異なる結果が生成される可能性があっても、ベクトル化を使用可能にします。

order | noorder

order は、**association**、**reductionorder**、および **guards** サブオプションによって制御されるものを含め、結果または例外に影響を与える可能性のある、複数の演算間のすべてのコードの再配列を無効にします。**noorder** では、コードの再配列が有効になります。

association | noassociation

association は、1 つの式の中の再配列操作を無効にします。**noassociation** では、再配列操作が有効になります。

reductionorder | noreductionorder

reductionorder は、浮動小数点の縮約の並列化を無効にします。**noreductionorder** では、それらの縮約の並列化を有効にします。

guards | noguards

guards は、演算を実行すべきかどうかを制御する保護領域を越える演算の移動 (つまり、**IF** ステートメントを越える演算の移動、ループの外側への演算の移動、またはプログラムを終了させる可能性があるサブルーチン/関数呼び出しを越える演算の移動) を無効にします。 **noguards** は、保護領域の境界を越えた移動操作を有効にします。

library | nolibrary

library は、浮動小数点ライブラリー関数に影響を与えるトランスフォーメーション (例えば、浮動小数点ライブラリー関数を他のライブラリー関数または定数と置き換えるトランスフォーメーション) を無効にします。 **nolibrary** では、これらのトランスフォーメーションが有効になります。

constructcopy | noconstructcopy

constructcopy は、例外が起こる可能性のある一時コピーを使用する代わりに、同じ場所に配列を作成することを無効にします。 **noconstructcopy** では、その配列の作成が有効になります。

使用法

all、**precision**、**exceptions**、**ieeefp**、および **order** サブオプションとその負の形式は、複数の個別のサブオプションに影響を与えるグループ・サブオプションです。多くの状態において、グループ・サブオプションはトランスフォーメーションに対して十分かつ粒度の細かい制御を行います。グループ・サブオプションは、そのグループの正の形式または形式なしのすべてのサブオプションが指定されている場合と同様に機能します。必要な場合は、1 つのグループ内の個々のサブオプション (例えば、**precision** グループ内の **subnormals** または **operationprecision**) が、そのグループ内の特定のトランスフォーメーションの制御を行います。

-qnostrict または **-qstrict=none** が有効な場合、以下の最適化がオンになります。

- 例外の原因となる可能性のあるコードは、再配置される場合があります。実行時の別の時点でそれに対応する例外が起こる可能性がありますが、まったく起こらない場合もあります。 (コンパイラーは引き続きその状態を最小限にくい止めるように試みます。)
- 浮動小数点演算では、ゼロ値の符号が保存されない場合があります。(ゼロ値の符号が確実に保存されるようにするには、**-qfloat=rrm** または **-qfloat=nomaf** も指定してください。)
- 浮動小数点式の関連付けがやり直される場合があります。例えば、**(2.0*3.1)*4.2** は **2.0*(3.1*4.2)** になる可能性があります。これは、結果が同一にならない場合でも、その方が速ければ実施されます。
- この最適化関数は **-qfloat=rsqrt** により有効化されます。最適化関数は、**-qstrict** オプションまたは **-qfloat=norsqrt** を使用してオフにすることができます。低レベルの最適化を指定した場合や、最適化を指定しなかった場合、これらの最適化関数はデフォルトではオフになります。

-qstrict[=suboptions] または **-qnostrict** の組み合わせのさまざまなサブオプションを指定することにより、以下のサブオプションが設定されます。

- **-qstrict** または **-qstrict=all** では **-qfloat=norsqrt:rngchk** が設定されます。**-qnostrict** または **-qstrict=none** では **-qfloat=rsqrt:norngchk** が設定されます。
- **-qstrict=infinities**、**-qstrict=operationprecision**、または **-qstrict=exceptions** では **-qfloat=norsqrt** が設定されます。
- **-qstrict=noinfinities:nooperationprecision:noexceptions** では **-qfloat=rsqrt** が設定されます。
- **-qstrict=nans**、**-qstrict=infinities**、**-qstrict=zerosigns**、または **-qstrict=exceptions** では **-qfloat=rngchk** が設定されます。すべての **-qstrict=nonans:nozerosigns:noexceptions** または **-qstrict=noinfinities:nozerosigns:noexceptions** を指定するか、あるいはこれらすべてを暗黙指定する任意のグループ・サブオプションを指定すると、**-qfloat=norngchk** が設定されます。

注: **-qstrict** サブオプションとそれらに対応する **-qfloat** の関係について詳しくは、153 ページの『-qfloat』を参照してください。

これらのいずれかの設定をオーバーライドするには、コマンド行上の **-qstrict** オプションの後に適切な **-qfloat** サブオプションを指定します。

例

-O3 の積極的な最適化がオフにされ、平方根の結果による除算が逆数による乗算によって置換される (**-qfloat=rsqrt**) ように `myprogram.f` をコンパイルするには、以下のように入力します。

```
xlf myprogram.f -O3 -qstrict -qfloat=rsqrt
```

精度に影響を与えるもの以外のすべてのトランスフォーメーションを有効にするには、以下のように指定します。

```
xlf myprogram.f -qstrict=none:precision
```

NaNs および無限大に関係するものを除くすべてのトランスフォーメーションを無効にするには、以下のように指定します。

```
xlf myprogram.f -qstrict=all:nonans:noinfinities
```

関連情報

- 258 ページの『-qsimd』
- 145 ページの『-qessl』
- 153 ページの『-qfloat』
- 170 ページの『-qhot』
- 101 ページの『-O』
- 306 ページの『-qxlf90』

-qstrictieeemod

カテゴリー

浮動小数点および整数のコントロール

目的

コンパイラーが、**ieee_arithmetic** および **ieee_exceptions** 組み込みモジュールに関する Fortran 2003 IEEE 算術計算規則に従うかどうかを指定する。

構文

@PROCESS:

```
@PROCESS STRICTIEEEMOD | NOSTRICTIEEEMOD
```

デフォルト

-qstrictieeemod

使用法

-qstrictieeemod を指定すると、コンパイラーは、次の規則を順守します。

- IEEE 組み込みモジュールを使用するプロシージャーへの入り口で例外フラグがオンに設定されている場合は、そのフラグは出口でオンに設定されます。IEEE 組み込みモジュールを使用するプロシージャーへの入り口でフラグがオンをクリアする場合は、そのフラグは出口でオンに設定されます。
- IEEE 組み込みモジュールを使用するプロシージャーへの入り口で例外フラグがオンに設定される場合、プロシージャーへの入り口でオンをクリアし、そのプロシージャーから戻る時リセットします。
- IEEE 組み込みモジュールを使用するプロシージャーから戻るとき、停止モードおよび丸めモードの設定は、プロシージャーの入り口で持っていた値に戻ります。

- **ieee_arithmetic** あるいは **ieee_exceptions** 組み込みモジュールを使用するプロシージャーから、それらを使用しないプロシージャーへの呼び出しは、例外フラグを設定する場合を除いて、浮動小数点状況を変更しません。

上記の規則はパフォーマンスに影響を与えるため、**-qnostrictieeemod** を指定すると、浮動小数点状況を保存したり、復元したりする規則から解放されます。これは関連するパフォーマンスの影響を防ぎます。

-qstrict_induction

カテゴリー

最適化およびチューニング

@PROCESS

なし。

目的

コンパイラーが帰納 (ループ・カウンター) 変数の最適化を実行しないようにする。このような最適化は、帰納変数に関係する整数オーバーフロー操作が発生した場合に問題となる可能性があります。

構文

デフォルト

-qnostrict_induction

使用法

-qstrict_induction を指定すると性能低下の恐れがあるため、どうしても必要な場合を除き、指定しないようにする必要があります。

例

以下の 2 つの例を見てください。

例 1

```
integer(1) :: i, j                ! Variable i can hold a
j = 0                             ! maximum value of 127.

do i = 1, 200                     ! Integer overflow occurs when 128th
  j = j + 1                       ! iteration of loop is attempted.
enddo
```

例 2

```
integer(1) :: i
i = 1_1                           ! Variable i can hold a maximum
                                  ! value of 127.
```

```
100 continue
    if (i == -127) goto 200            ! Go to label 200 once decimal overflow
      i = i + 1_1                      ! occurs and i == -127.
      goto 100
200 continue
    print *, i
    end
```

-qstrict_induction オプションを指定してこれらの例をコンパイルすると、コンパイラーは帰納変数の最適化を実行しませんが、コードのパフォーマンスに影響する可能性があります。 **-qnostrict_induction** オプションを指定してこれらの例をコンパイルすると、コンパイラーは問題のある最適化を実行する可能性があります。

関連情報

- 101 ページの『-O』

-qsuffix

カテゴリー

入力制御

@PROCESS

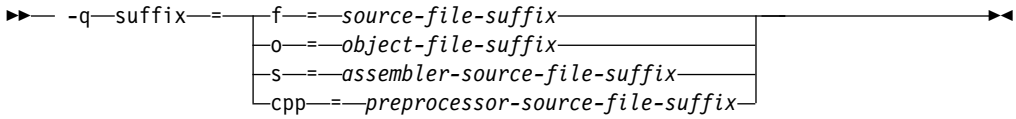
なし。

目的

コマンド行でソース・ファイルの接尾部を指定する。

このオプションを使用すれば、makefile の名前をほんの少し修正するだけでファイルを使用でき、無駄な時間を節約できます。どのファイル・タイプの場合にも、一度に 1 つの設定だけがサポートされます。

構文



デフォルト

適用されません。

パラメーター

f=*suffix*
ここで *suffix* は、新しいソース・ファイルのサフィックス です。

o=*suffix*
ここで *suffix* は、新しいオブジェクト・ファイルのサフィックス です。

s=*suffix*

ここで *suffix* は、新しいアセンブラー・ソース・ファイルのサフィックスです。

cpp=*suffix*

ここで *suffix* は、新しいプリプロセッサー・ソース・ファイルのサフィックス です。

規則

- 新しいサフィックスの設定には、大文字と小文字の区別があります。
- 新しいサフィックスの長さに制限はありません。

例

以下に例を示します。

```
xlf a1.f2k a2.F2K -qsuffix=f=f2k:cpp=F2K
```

これにより、以下の効果があります。

- コンパイラーが呼び出され、サフィックスが .f2k および .F2K のソース・ファイルを処理します。
- cpp は、サフィックスが .F2K のファイルで呼び出されます。

-qsuppress

カテゴリー

リスト、メッセージ、およびコンパイラー情報

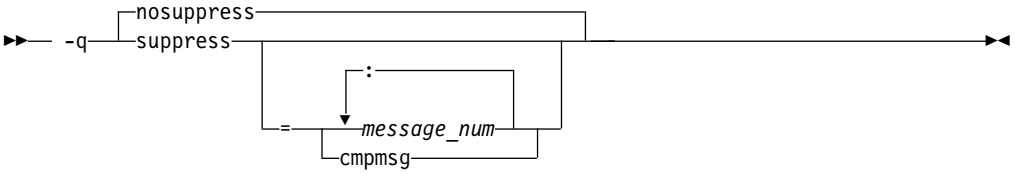
@PROCESS

なし。

目的

特定の通知メッセージ、または警告メッセージが生成された場合、表示されたりリスト・ファイルに追加されるのを防ぐ。

構文



デフォルト

適用されません。

パラメーター

message_num[:message_num ...]

特定のコンパイラー・メッセージ (*nnnn-mmm*) またはメッセージのリスト (*nnnn-mmm[:nnnn-mmm ...]*) の表示を抑止します。メッセージのリストを抑止するには、それぞれのメッセージ番号をコロンで区切ってください。

nnnn-mmm は、メッセージ番号です。ここで、

- *nnnn* は、1500 から 1585 の範囲にある 4 桁の整数でなければなりません。 XL Fortran メッセージ番号はこの範囲内にあります。
- *mmm* は、3 桁の任意の整数です (必要であればゼロを先行させます)。

cmpmsg

コンパイルの進行および正常終了を報告する情報メッセージを抑止します。

このサブオプションは、出力されるエラー・メッセージには影響しません。

使用法

状況によっては、非常に多くのコンパイラー・メッセージがユーザーに送られてくることがあります。多くの場合、これらのコンパイラー・メッセージには重要な情報が示されています。しかし、そのようなメッセージの中には冗長なものや、まったく無視して問題がないものもあります。コンパイル時に複数のエラーや警告メッセージが表示された場合は、どのメッセージに注意を払うべきか非常に判断の難しい場合があります。 **-qsuppress** を使用すれば、無関係なメッセージを除去できます。

注**:**

- コンパイラーは、**-qsuppress** に指定されたメッセージ番号をトラッキングします。その後、これらのメッセージのいずれかをコンパイラーが生成するような状況になっても、そのメッセージがリストに表示されたりすることはありません。
- 表示を抑止できるのは、コンパイラー・メッセージとドライバー・メッセージだけです。リンカーまたはオペレーティング・システムのメッセージ番号は、**-qsuppress** で指定された場合は無視されます。
- IPA メッセージを抑制するには、コマンド行で **-qipa** の前に **-qsuppress** を入力します。
- **-qhaltonmsg** オプションは、**-qsuppress** よりも優先されます。**-qhaltonmsg** と **-qsuppress** が両方とも指定された場合は、**-qsuppress** が抑止するメッセージも出力され、コンパイルが停止します。

例

```
@process nullterm
      i = 1; j = 2;
      call printf("i=%d¥n", %val(i));
      call printf("i=%d, j=%d¥n", %val(i), %val(j));
      end
```

このサンプル・プログラムをコンパイルすると、通常は次のような出力が得られます。

```
"t.f", line 4.36: 1513-029 (W) The number of arguments to "printf" differ
from the number of arguments in a previous reference.  You should use the
OPTIONAL attribute and an explicit interface to define a procedure with
```

```
optional arguments.
** _main   === End of Compilation 1 ===
1501-510  Compilation successful for file t.f.
```

-qsuppress=1513-029 を指定してプログラムをコンパイルした場合、出力は次のようになります。

```
** _main   === End of Compilation 1 ===
1501-510   Compilation successful for file t.f.
```

関連情報

- 151 ページの『-qflag』
- 168 ページの『-qhaltonmsg』
- 213 ページの『-qmaxerr』

-qswapomp

カテゴリー

移植性とマイグレーション

目的

XL Fortran プログラムでコンパイラーが OpenMP ルーチンを認識し、置換しなければならないことを指定する。

Fortran と C の OpenMP ルーチンには、別々のインターフェースがあります。OpenMP ルーチンを使用する複数言語アプリケーションをサポートするには、コンパイラーは OpenMP ルーチン名を認識し、そうしたルーチンの他のインプリメンテーションが存在しているかどうかにかかわらず、そのルーチンを XL Fortran バージョンのルーチンに置換する必要があります。

構文

@PROCESS:

@PROCESS **SWAPOMP** | NOSWAPOMP

デフォルト

-qswapomp

使用法

コンパイラーは、**-qnoswapomp** オプションを指定すると、 OpenMP ルーチンの置換は実行しません。

-qswapomp および **-qnoswapomp** オプションは、プログラムに存在する OpenMP ルーチンを参照する Fortran サブプログラムだけに影響を与えます。

規則

- OpenMP ルーチンへの呼び出しが解決されて、ダミー・プロシージャー、モジュール・プロシージャー、内部プロシージャー、プロシージャーそのものの直接呼び出し、またはステートメント関数になる場合、コンパイラーは置換を実行しません。
- OpenMP ルーチンを指定すると、コンパイラーは **-qintsize** オプションの設定に応じて、その呼び出しを別の特殊ルーチンに置換します。この方法では、OpenMP ルーチンは汎用組み込みプロシージャーとして扱われます。
- 汎用組み込みプロシージャーとは異なり、 OpenMP ルーチンを **EXTERNAL** 文に指定すると、コンパイラーはその名前をユーザー定義の外部プロシージャーとしては扱いません。その代わり、コンパイラーは引き続き **-qintsize** オプションの設定に応じて、呼び出しを特殊ルーチンに置換します。
- OpenMP ルーチンは、汎用組み込みプロシージャーとは異なり、拡張したり再定義したりすることはできません。

例

次の例では、OpenMP ルーチンが **INTERFACE** 文で宣言されます。

```
@PROCESS SWAPOMP

         INTERFACE
            FUNCTION OMP_GET_THREAD_NUM()
                INTEGER OMP_GET_THREAD_NUM
            END FUNCTION OMP_GET_THREAD_NUM

            FUNCTION OMP_GET_NUM_THREADS()
                INTEGER OMP_GET_NUM_THREADS
            END FUNCTION OMP_GET_NUM_THREADS
         END INTERFACE

         IAM = OMP_GET_THREAD_NUM()
         NP = OMP_GET_NUM_THREADS()
         PRINT *, IAM, NP
         END
```

関連情報

「*XL Fortran* 最適化およびプログラミング・ガイド」の『*OpenMP* 実行環境、ロックおよびタイミング・ルーチン』セクションを参照してください。

-qtbtable

カテゴリー

オブジェクト・コードの制御

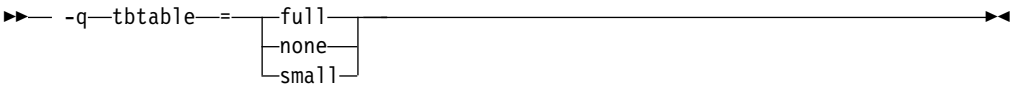
@PROCESS

なし。

目的

オブジェクト・ファイルに組み込まれるデバッグ・トレースバック情報の量を制御する。

構文



デフォルト

適用されません。

パラメーター

full オブジェクト・コードにはトレースバックの全情報が入ります。プログラムはデバッグ可能で、実行時例外のために停止する場合は、トレースバック・リストを作成します。これには、呼び出しチェーン内のプロシージャーすべての名前が入っています。

none オブジェクト・コードにはトレースバック情報がまったく入りません。デバッガーや他のコード検査ツールが実行時にプログラムのスタックをアンワインドできないので、プログラムをデバッグすることはできません。実行時例外のためにプログラムが停止する場合は、例外の発生場所を説明しません。

small オブジェクト・コードにはトレースバック情報が入りますが、プロシージャーの名前やプロシージャー・パラメーターの情報は入りません。プログラムのデバッグは可能ですが、必須でない情報の中にはデバッガーが利用不能なものがあります。実行時例外のためにプログラムが停止する場合は、例外の発生場所を説明しますが、プロシージャー名ではなくマシン・アドレスを報告します。

デフォルト

- **-g** を指定してコンパイルされたコードや **-O** を指定しないでコンパイルされたコードにはトレースバックの全情報が入ります (**-qtbtable=full**)。
- **-O** またはより高い最適化を指定してコンパイルされたコードには、それよりも小さなトレースバック情報が入ります (**-qtbtable=small**)。

使用法

多くの長いプロシージャー名 (モジュール・プロシージャー用に作成された内部名など) が入っているプログラムには、このオプションが非常に適しています。Fortran プログラムよりも C++ プログラムに対する方が適用度が高い場合があります。

このオプションを使用して、プログラムを小さくすることができます。その代わりデバッグは難しくなります。実動ステージに到達しているときにできるだけコンパクトなプログラムを作成したい場合は、 **-qtbtable=none** を指定することができます。そうでない場合は、通常のデフォルトが適用されます。

関連情報

- 90 ページの『-g』
- 128 ページの『-qcompact』
- 101 ページの『-O』

- 「*XL Fortran* 最適化およびプログラミング・ガイド」の『最適化されたコードをデバッグ』

-qtgtarch

カテゴリー

オブジェクト・コード制御

@PROCESS

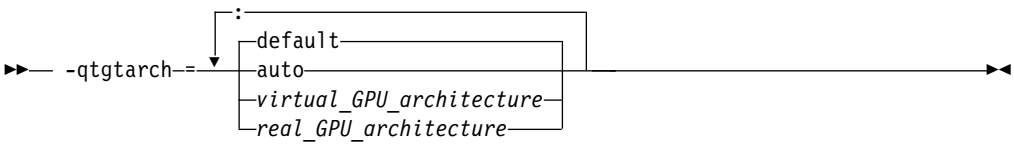
なし。

目的

コードが実行される可能性がある、実 GPU アーキテクチャーまたは仮想 GPU アーキテクチャーを指定します。これにより、GPU アーキテクチャーに固有であるか、または仮想アーキテクチャーに共通である機能またはマシン命令を、コンパイラーが最大限に活用できるようになります。

コンパイラーはコンパイラー構成時に GPU アーキテクチャーを自動的に検出します。この GPU アーキテクチャーはコンパイラー構成ファイルにエンコードされます。 **-qtgtarch** オプションを使用してデフォルトをオーバーライドできます。

構文



デフォルト

-qtgtarch=default

パラメーター

auto
コンパイラーが実行されているシステムの装置 0 のアーキテクチャー。

default
以下のように決定される、デフォルトのアーキテクチャー。

1. 構成ファイルに設定される cuda_cc_major プロパティーおよび cuda_cc_minor プロパティーで指定されたアーキテクチャー。
2. 指定されていない場合、コンパイラーが実行されるシステムの装置 0 のアーキテクチャー。
3. 装置 0 がない場合、sm_35。

real_GPU_architecture
CUDA ツールキットで定義されている sm_35、sm_60、または sm_70 などの、実 GPU アーキテクチャー。

virtual_GPU_architecture
CUDA ツールキットで定義されている compute_35、compute_60、または compute_70 などの、仮想 GPU アーキテクチャー。仮想 GPU アーキテクチャーは、ハイレベル PTX コード内でサポートされるフィーチャーを指定します。

規則

PTX 中間コードが、指定された仮想 GPU アーキテクチャーに基づいて生成され、その後、結果のオブジェクト・ファイルまたは実行可能ファイルに埋め込まれます。コンパイル済みコード・イメージを生成して埋め込むには、実 GPU アーキテクチャーを指定します。実 GPU アーキテクチャー用のコンパイル済みコード・イメージは、PTX コードから生成されます。

各 **-qtgtarch** オプションは、1 つのみの仮想 GPU アーキテクチャー用の PTX コードと、オプションで 1 つ以上の互換性のある実 GPU アーキテクチャー用のコンパイル済みコード・イメージを生成するために使用されます。複数の仮想 GPU アーキテクチャー用に PTX コードを生成する必要がある場合は、**-qtgtarch** オプションを複数回 (仮想 GPU アーキテクチャーごとに 1 回ずつ) 指定します。

例えば、仮想アーキテクチャーが指定されていない場合や、複数の仮想 GPU アーキテクチャーが指定されている場合など、必要であれば、コンパイラーは仮想 GPU アーキテクチャーと実 GPU アーキテクチャーとの間での変換を行います。

同じ仮想 GPU アーキテクチャーに対してでも、**-qtgtarch** オプションを複数回指定することができます。その結果として生じる効果は累積します。

-qtgtarch オプションの指定に関する詳しいルールは以下のリストのとおりです。

表 *22. 1* つの **-qtgtarch** オプションの指定に関する詳細ルール

指定される仮想 **GPU** アーキテクチャーの数	指定される実 **GPU** アーキテクチャーの数	**PTX** コードが生成される仮想 **GPU** アーキテクチャー	コンパイル済みコード・イメージが生成される実 **GPU** アーキテクチャー
0	1 つ以上	指定された最低レベルの実 GPU アーキテクチャーに対応する仮想 GPU アーキテクチャー	指定された実 GPU アーキテクチャー

表 22. 1 つの **-qtgtarch** オプションの指定に関する詳細ルール (続き)

指定される仮想 **GPU** アーキテクチャーの数	指定される実 **GPU** アーキテクチャーの数	**PTX** コードが生成される仮想 **GPU** アーキテクチャー	コンパイル済みコード・イメージが生成される実 **GPU** アーキテクチャー
1	0	指定された仮想 GPU アーキテクチャー	N/A **注:** 結果のオブジェクト・ファイルまたは実行可能ファイルに埋め込まれるコンパイル済みコード・イメージがない場合、必要に応じて、リンク時または実行時にジャストインタイム・コンパイルを使用して、1 つのコンパイル済みコード・イメージが PTX コードから生成されます。
1	1 つ以上	指定された仮想 GPU アーキテクチャー	指定された実 GPU アーキテクチャー
複数	0	指定された最低レベルの仮想 GPU アーキテクチャー	最低を除くすべての指定された仮想 GPU アーキテクチャーに対応する実 GPU アーキテクチャー
複数	1 つ以上	指定された最低レベルの仮想 GPU アーキテクチャー	指定された実 GPU アーキテクチャーと、最低を除くすべての指定された仮想 GPU アーキテクチャーに対応する実 GPU アーキテクチャー

例

-qtgtarch オプションの指定例を以下にリストします。

表 23. **-qtgtarch** オプションの指定例

コマンド例	**PTX** コードが生成される仮想 **GPU** アーキテクチャー	コンパイル済みコード・イメージが生成される実 **GPU** アーキテクチャー
`-qtgtarch=sm_60`	compute_60	sm_60 **注:** コンパイル済みコード・イメージは PTX コードから生成されます。

表 23. **-qtgtarch** オプションの指定例 (続き)

コマンド例	**PTX** コードが生成される仮想 **GPU** アーキテクチャー	コンパイル済みコード・イメージが生成される実 **GPU** アーキテクチャー
アーキテクチャー sm_37 のマシンでコンパイラーが実行されると想定します: `-qtgtarch=auto`	compute_37	sm_37 **注:** コンパイル済みコード・イメージは PTX コードから生成されます。
`-qtgtarch=compute_35:compute_37:sm_37:sm_60`	compute_35	sm_37 および sm_60 **注:** コンパイル済みコード・イメージは PTX コードから生成されます。
`-qtgtarch=sm_37:sm_60`	compute_37	sm_37 および sm_60 **注:** コンパイル済みコード・イメージは PTX コードから生成されます。
アーキテクチャー sm_37 のマシンでコンパイラーが実行されると想定します: `-qtgtarch=auto:sm_60`	compute_37	sm_37 および sm_60 **注:** コンパイル済みコード・イメージは PTX コードから生成されます。
`-qtgtarch=sm_35` `-qtgtarch=sm_60`	compute_35 および compute_60	sm_35 および sm_60 **注:** sm_35 および sm_60 のコンパイル済みコード・イメージは、それぞれ、compute_35 および compute_60 用の PTX コードから生成されます。

関連情報

- -qoffload
- http://docs.nvidia.com/cuda/cuda-compiler-driver-nvcc/index.html#options-for-steering-gpu-code-generation にある、*CUDA* ツールキット内の *GPU* アーキテクチャーについての資料

-qthreaded

カテゴリー

オブジェクト・コードの制御

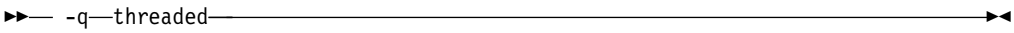
@PROCESS

なし。

目的

スレッド・セーフ・コードを生成する必要があるかどうかをコンパイラーに指示する。

構文



デフォルト

-qthreaded は、コマンド **xlf_r**、**xlf90_r**、**xlf95_r**、**xlf2003_r**、**xlf2008_r**、および CUDA Fortran **xlcuf** CUDA Fortran のデフォルトです。

使用法

-qthreaded オプションを指定すると **-qdirective=ibmt** が暗黙指定されますが、デフォルトでは *trigger_constant* **IBMT** が認識されます。

-qthreaded オプションを指定しても、 **-qnosave** オプションも暗黙的に指定されるということはありません。 **-qnosave** オプションは、ユーザー・ローカル変数のデフォルト時自動ストレージ・クラスを指定するものです。一般に、スレッド・セーフなコードを生成するためには、これらのオプションを両方とも使用する必要があります。これらのオプションを指定すると、コンパイラーによって作成される変数およびコードがスレッド・セーフであることを確実にしますが、ユーザー作成のコードのスレッド・セーフティーは保証しません。

-qthreaded オプションを指定しても、**-qxlf77=nopersistent** オプションも暗黙的に指定されるということはありません。**-qxlf77=nopersistent** オプションは、**ENTRY** ステートメントが含まれたサブプログラムに対する引数のアドレスをコンパイラーが静的ストレージに保存することを防止することで、スレッド・セーフティーを向上させます。

-qtimestamps

カテゴリー

61 ページの『出力制御』

@PROCESS

なし。

目的

暗黙的なタイム・スタンプがオブジェクト・ファイルに挿入されるようにするかどうかを制御する。

構文

デフォルト

-qtimestamps

使用法

デフォルトでは、コンパイラーはオブジェクト・ファイルの作成時にその中に暗黙のタイム・スタンプを挿入します。場合によっては、比較ツールがそのようなバイナリーの情報を正しく処理できないおそれがあります。タイム・スタンプの生成を制御することにより、この問題を回避することができます。タイム・スタンプを省略するには、**-qnotimestamps** オプションを使用します。

プラグマまたはその他の明示的な手段によって挿入されたタイム・スタンプはこのオプションの対象になりません。

-qtune

カテゴリー

最適化およびチューニング

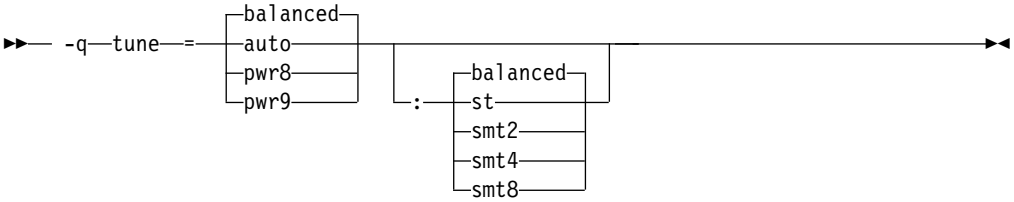
@PROCESS

なし。

目的

特定のハードウェア・アーキテクチャーで最も効率よく実行できるように、命令選択、スケジューリング、およびその他のアーキテクチャー依存のパフォーマンスの強化をチューニングする。ターゲット SMT モードの指定で、そのモードで最高のパフォーマンスが得られる最適化を指示できるようにする。

構文



デフォルト

有効な **-qarch** 設定が適用されていない場合は **-qtune=balanced:balanced**。それ以外の場合は、デフォルトは、有効な **-qarch** 値によって異なります。『-qarch と -qtune の使用可能な組み合わせ』に詳細があります。

CPU サブオプション向けのパラメーター

以下の CPU サブオプションを使用すると、特定のアーキテクチャーを指定して、コンパイラーが最高のパフォーマンスを発揮できるようにすることができます。

auto
アプリケーションがコンパイルされるプラットフォームで最適化がチューニングされます。

balanced
最適化が、最新ハードウェアの選択範囲全体で調整されます。

pwr8
POWER8 ハードウェア・プラットフォーム用に、最適化が調整されます。

pwr9
POWER9 テクノロジーを利用するように最適化が調整されます。

SMT サブオプション向けのパラメーター

以下の同時マルチスレッド化 (SMT) サブオプションを使用すると、オプションでコンパイラーの実行モードを指定して、最高のパフォーマンスを発揮できるようにすることができます。

balanced
選択された範囲の最新ハードウェア用のさまざまな SMT モード全体でパフォーマンスが得られるよう、最適化が調整されます。

st 単一スレッド実行用に、最適化が調整されます。

smt2
SMT2 実行モード (2 スレッド) 用に、最適化が調整されます。

smt4
SMT4 実行モード (4 スレッド) 用に、最適化が調整されます。

smt8
SMT8 実行モード (8 スレッド) 用に、最適化が調整されます。

使用法

-qtune では、キャッシュ・サイズおよびパイプラインなどのハードウェア・フィーチャーを最大限に活用するように、生成されたマシン命令を配置 (スケジューリング) することで、パフォーマンスを改善できます。このオプションは、最適化を使用可能にするオプションと組み合わせて使用した場合にのみ効果があります。

-qtune 設定を変更すると、その結果作成される実行可能ファイルのパフォーマンスに影響する場合がありますが、実行可能ファイルが特定のハードウェア・プラットフォーム上で正しく実行できるかどうかには、まったく影響を与えません。

受け入れられる **-qarch** と **-qtune** の組み合わせを、次の表に示します。

表 *24.* **-qarch** と **-qtune** の使用可能な組み合わせ

-qarch オプション	デフォルトの **-qtune** 設定	使用可能な **-qtune CPU** 設定	使用可能な **-qtune SMT** 設定
pwr8	pwr8:st	auto | pwr8 | pwr9 | balanced	balanced | st | smt2 | smt4 | smt8
pwr9	pwr9:st	auto | pwr9 | balanced	balanced | st | smt2 | smt4 | smt8

例

実行可能プログラム testing は myprogram.c からコンパイルされます。 testing が POWER8 ハードウェア・プラットフォーム向けに最適化されるように、また SMT4 モード向けに構成されるように指定するには、以下を入力します。

```
xlf -o testing myprogram.f -qtune=pwr8:smt4
```

関連情報

- 113 ページの『-qarch』
- 「*XL Fortran* 最適化およびプログラミング・ガイド」の 『アプリケーションの最適化』

-qufmt

カテゴリー

移植性とマイグレーション

目的

不定形式データ・ファイルで入出力操作のバイト・オーダーを設定する。

構文


```
@PROCESS:

@PROCESS UFMT[({BE | LE})]
```

デフォルト

-qufmt=le

パラメーター

be 不定形式データ・ファイルの入出力操作にビッグ・エンディアンのバイト・オーダーを使用することを指定します。このオプションにより、ビッグ・エンディアンのデータ・ファイルとの互換性が提供されますが、実行時のパフォーマンスが低下します。非文字データおよびレコード・マーカーのバイト・オーダーは、入出力操作時にリアルタイムに変換されます。

le 不定形式データ・ファイルの入出力操作にリトル・エンディアンのバイト・オーダーを使用することを指定します。このオプションは、データとレコード・マーカーのバイト・オーダーが変換されることなく不定形式データ・ファイルから読み取りおよび書き込みされるため、パフォーマンスを向上させます。

使用法

次の方法で入出力操作のバイト・オーダーを指定できます。同一装置に対して複数のバイト・オーダーが指定され、それらがお互いに競合する場合、リストの最初のバイト・オーダーが優先されます。

1. 実行時オプション **XLFRTEOPTS=ufmt_bigendian** を設定する。
2. **OPEN** ステートメントの **CONVERT=** *char_expr* を設定する。ここで *char_expr* は **NATIVE**、**BIG_ENDIAN**、または **LITTLE_ENDIAN**。
3. `@PROCESS UFMT(BE)` または `@PROCESS UFMT(LE)`
4. コンパイラー・オプション **-qufmt=be** または **-qufmt=le** を設定する。

関連情報

- 「*XL Fortran* ランゲージ・リファレンス」の『OPEN ステートメント』
- 45 ページの『XLFRTEOPTS 環境変数』

-qundef

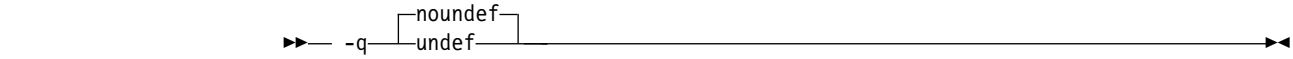
カテゴリー

言語エレメント制御

目的

-qundef は 320 ページの『-u』 オプションの長形式です。

構文



@PROCESS:

`@PROCESS UNDEF | NOUNDEF`

デフォルト

-qnoundef

-qunroll

カテゴリー

最適化およびチューニング

@PROCESS

なし。

目的

プログラム内で **DO** ループのアンロールが許可されるかどうかを指定する。アンロールは外部および内部の **DO** ループで許可されます。

構文

デフォルト

-qunroll=auto

パラメーター

auto コンパイラーは、基本ループ・アンロール (展開) を行います。

yes コンパイラーは、**-qunroll=auto** を指定して実行されるより多くの、ループ・アンロールを実行する機会を探します。一般にこのサブオプションは、**-qunroll=auto** 処理より、コンパイル時間あるいはプログラム・サイズが増える可能性がありますが、アプリケーションのパフォーマンスを向上させることもあります。

n 係数 *n* までループをアンロールするようコンパイラーに指示します。つまり、ループの本体が複製されて *n* 個のコピーが作成され、反復数は係数 *1/n* に削減されます。 *n* の値は正整数でなければなりません。

-qunroll=1 と指定すると、ループのアンロールが無効になり、**-qnounroll** と指定したのと同じことになります。*n* が指定されていない場合、および **-qhot**、**-qsmp**、**-O4**、または **-O5** が指定されている場合は、最適化プログラムが、ネストされたループごとに適切なアンロール係数を決定します。

コンパイラーは、*n* に指定した値より少ない回数にアンロールを制限する場合があります。これは、オプションが適用先のソース・ファイル内のすべてのループに作用し、アンロール係数が大きいと、必ずしもランタイム・パフォーマンスが向上することなくコンパイル時間が大幅に増加する可能性があるためです。特定のループに固有のアンロール係数を指定するには、それらのループで unroll ディレクティブを使用します。

ループをアンロールすることに決定した場合、上記のサブオプションの 1 つを指定することが自動的に、コンパイラーがその操作を実行することを保証するわけではありません。パフォーマンス上の利点を考慮して、コンパイラーはプログラムにとってアンロールが有利かどうかを判断します。熟練したコンパイラー・ユーザーは、前もって有利かどうかを判断できるようであるべきです。

使用法

サブオプションを指定しないで **-qunroll** を指定することは、**-qunroll=yes** と同じです。

STREAM_UNROLL、 **UNROLL**、または **UNROLL_AND_FUSE** ディレクティブを特定のループに指定していなければ、**-qnounroll** オプションはアンロールを禁止します。これらのディレクティブは常に、コマンド行オプションをオーバーライドします。

例

次の例では、**UNROLL(2)** ディレクティブを使用して、コンパイラーにループの本体が複製可能であることを示し、単一の反復で 2 度の反復作業を実行できるようにしています。コンパイラーがループをアンロールすれば、 1000 回反復を実行するところを、500 回だけ実行すれば済むようになります。

```
!IBM* UNROLL(2)
      DO I = 1, 1000
         A(I) = I
      END DO
```

コンパイラーが前のループのアンロールを選択すると、コンパイラーはそのループを変換して、次の例と本質的に同じになるようにします。

```
      DO I = 1, 1000, 2
         A(I) = I
         A(I+1) = I + 1
      END DO
```

関連情報

「*XL Fortran* ランゲージ・リファレンス」でループのアンロールの該当するディレクティブを参照してください。

- **STREAM_UNROLL**
- **UNROLL**
- **UNROLL_AND_FUSE**

「*XL Fortran* 最適化およびプログラミング・ガイド」の『高次変換 *(HOT)*』を参照してください。

-qunwind

カテゴリー

最適化およびチューニング

目的

プロシージャー呼び出し中に、コンパイラーが揮発性レジスターへの保存および復元のデフォルトの振る舞いを保持することを指定する。

構文

@PROCESS:

@PROCESS **UNWIND** | NOUNWIND

デフォルト

-qunwind

使用法

-qnounwind を指定すると、コンパイラーは、サブプログラムを再調整して、揮発性レジスターの保存と復元を最小化します。この再調整により、プログラムまたはデバッガーがサブプログラム・スタック・フレーム・チェーンをウォークスルーまたは「アンワインド」するのを不可能にする場合があります。

コードのセマンティクスが保持されている間は、保存と復元のデフォルト動作に依存する例外ハンドラーのようなアプリケーションは、未定義の結果を生成する可能性があります。 **-qnounwind** を **-g** コンパイラー・オプションと組み合わせて使用する場合、プログラム・スタックをアンワインドするときの例外処理操作に関するデバッグ情報は、不正確になる可能性があります。

-qversion

カテゴリー

リスト、メッセージ、およびコンパイラー情報

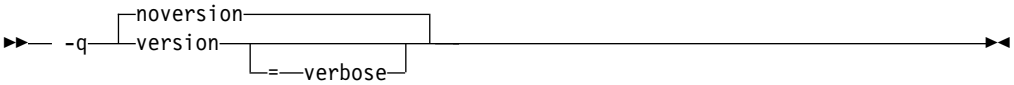
@PROCESS

なし。

目的

呼び出しているコンパイラーのバージョンおよびリリースを表示する。

構文



デフォルト

-qnoversion

パラメーター

verbose
インストール済みの各コンパイラー・コンポーネントのバージョン、リリース、およびレベルについての情報を表示します。

使用法

-qversion を指定すると、コンパイラーは、バージョン情報を表示し終了し、コンパイルは停止されます。出力オブジェクト・ファイルにこの情報を保存する場合は、**-qsaveopt -c** オプションを指定して保存することができます。

verbose サブオプションを指定しないで **-qversion** を指定すると、次の形式でコンパイラー情報が表示されます。

*product_name*Version: *VV.RR.MMMM.LLLL*

ここで、
V　　バージョンを表します。
R　　リリースを表します。
M　　変更を表します。
L　　レベルを表します。

詳しくは、『例 1』を参照してください。

-qversion=verbose は、以下の形式でコンポーネント情報を表示します。

```
component_name Version: VV.RR(product_name) Level: component_build_date ID:
component_level_ID
```

ここで、
component_name
　　低水準最適化プログラムなどのインストール済みコンポーネントを指定します。
component_build_date
　　インストール済みコンポーネントのビルド日付を表します。
component_level_ID
　　インストール済みコンポーネントのレベルに関連付けられている ID を表します。

詳しくは、『例 2』を参照してください。

例 1

-qversion オプションを指定した場合の出力:

```
IBM XL Fortran for Linux, V16.1 (5765-J10; 5725-C75)
Version: 16.01.0000.0000
```

例 2

-qversion=verbose オプションを指定した場合の出力:

```
IBM XL Fortran for Linux, V16.1 (5765-J10; 5725-C75)
Version: 16.01.0000.0000
Driver Version: 16.1.0(Fortran) Level: 150508
ID: _hnbfIvWfEeSjz7qEhQiYJQ
Fortran Front End and Run Time Version: 16.1.0(Fortran) Level: 150512
ID: _mQf28vkLEeSjz7qEhQiYJQ
Fortran Transformer Version: 16.1.0(Fortran) Level: 150506
ID: _Ax_9Eu1CEeSbzZ-i2Itj4A
High-Level Optimizer Version: 16.1.0(C/C++) and 16.1.0(Fortran)
Level: 150512 ID: _mSHAgvkLEeSjz7qEhQiYJQ
Low-Level Optimizer Version: 16.1.0(C/C++) and 16.1.0(Fortran)
Level: 150511 ID: _YY5AQvhCEeSjz7qEhQiYJQ
```

関連情報

- 253 ページの『-qsaveopt』
- COMPILER_VERSION

-qvisibility

カテゴリー

最適化およびチューニング

目的

オブジェクト・ファイル内の外部リンケージ・シンボルの可視性属性を指定します。

構文


```
@PROCESS:
@PROCESS VISIBILITY(suboption)
```

デフォルト

-qvisibility=default

パラメーター

default
これは、影響を受ける外部リンケージ・シンボルがデフォルト可視性属性を持つことを示します。そのようなシンボルは共有ライブラリーにエクスポートされて、優先使用できるようになります。

hidden
これは、影響を受ける外部リンケージ・シンボルが非表示可視性属性を持つことを示します。このようなシンボルは共有ライブラリーにはエクスポートされませんが、このようなシンボルのアドレスがポインターで間接的に参照できるようになります。

internal
これは、影響を受ける外部リンケージ・シンボルが内部可視性属性を持つことを示します。このようなシンボルは共有ライブラリーにはエクスポートされず、このようなシンボルのアドレスは、共有ライブラリーにある他のモジュールからは使用できません。

protected
これは、影響を受ける外部リンケージ・シンボルが保護可視性属性を持つことを示します。このようなシンボルは共有ライブラリーにエクスポートされますが、優先使用はできません。

使用法

-qvisibility オプションは、外部リンケージ・シンボルに対してグローバルに可視性属性を設定します。その目的は、あるモジュールで定義されているシンボルを他のモジュールで参照したり使用したりできるかどうかを記述することです。シンボルの可視性属性は、外部リンケージを持つシンボルにのみ影響します。この属性で他

のシンボルの可視性を増幅させることはできません。シンボルが優先使用されるのは、リンク時にシンボル定義が解決されたにもかかわらず実行時に別のシンボル定義に置き換えられる場合です。

例

コンパイル単位 myprogram.f に含まれる外部リンケージ・シンボルに対して保護可視性属性を設定するには、次のコマンドを実行します。

```
xlf myprogram.f -qvisibility=protected -c
```

この例では、コンパイル単位 myprogram.f に含まれるすべての外部リンケージ・シンボルが保護可視性を持ちます。

関連情報

36 ページの『アプリケーションへのライブラリーのリンク』
42 ページの『動的および静的リンク』
25 ページの『XL Fortran 出力ファイル』

-qwarn64

カテゴリー

エラー・チェックおよびデバッグ

@PROCESS

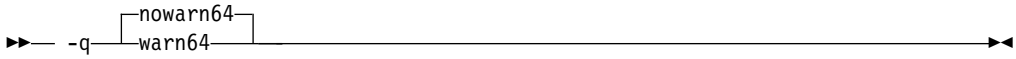
なし。

目的

32 ビットから 64 ビットへのマイグレーションで問題を起こす可能性のあるステートメントを識別する通知メッセージを表示する。

このオプションは、32 ビットから 64 ビット環境へのコードの移植の際に役立ちます。つまり、8 バイト整数ポインターの 4 バイトへの切り捨てが検出されます。

構文



デフォルト

-qnowarn64

使用法

コンパイラーは、次のような状態には通知メッセージのフラグを付けます。

- **INTEGER(4)** 変数に対する **LOC** 組み込みの参照割り当て。
- **INTEGER(4)** 変数または **INTEGER(4)** 定数と、整数ポインターとの間の割り当て。
- 共通ブロック内の整数ポインターの指定。

- 等価文内の整数ポインターの指定。

引数の検査には、インターフェース・ブロックをお勧めします。

関連情報

- 335 ページの『第 8 章 64 ビット環境での XL Fortran の使用』

-qxflag=dvz

カテゴリー

エラー・チェックおよびデバッグ

@PROCESS

なし。

目的

コンパイラーが浮動小数点のゼロ除算演算を検出するためのコードを生成できるようにする。

構文

デフォルト

適用されません。

使用法

このオプションは、最適化レベル **-O** 以上で有効になります。

このオプションをオンにした場合、除数がゼロのとき、追加のコードは外部ハンドラー関数 **__xl_dzx** を呼び出します。この関数の戻り値は、除算の結果として使用されます。ユーザーは、ゼロ除算演算を処理するための関数を指定する必要があります。 **-qxflag=dvz** を指定すると、単精度 (REAL*4) および倍精度 (REAL*8) 除算のみが処理されます。

関数のインターフェースは次のとおりです。

```
real(8) function __xl_dzx(x, y, kind_type)
  real(8), value :: x, y
  integer, value :: kind_type
end function
```

用語の説明

x これは被除数です。

y これは除数値です。

kind_type
x および **y** に関連付けられた実引数のサイズを指定します。

ゼロと等しい **kind_type** 値は、x および y に関連付けられた実引数が REAL(8) 型であることを示します。 1 と等しい **kind_type** 値は、x および y に関連付けられた実引数が REAL(4) 型であることを示します。

除算は常に、ハンドラー・ルーチンが呼び出される前に実行されます。すなわち、例外はハンドラー関数が呼び出される前に通知および処理されます。

関連情報

- 「*XL Fortran* 最適化およびプログラミング・ガイド」の『*XL Fortran* 浮動小数点処理のインプリメンテーションの詳細』
- 157 ページの『-qflttrap』
- 341 ページの『XL Fortran エラー・メッセージに関する情報』

-qxflag=oldtab

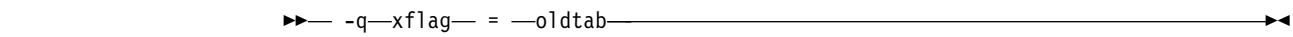
カテゴリー

移植性とマイグレーション

目的

列 1 から 5 のタブを 1 文字として解釈する (固定ソース形式プログラムの場合)。

構文



@PROCESS:

```
@PROCESS XFLAG(OLDTAB)
```

デフォルト

デフォルトでは、コンパイラーはソース行の桁 6 の後に 66 文字の有効文字を許可します。桁 1 から 5 のタブは、桁カウンターを桁 6 の後に移動する適切な数のブランクであると解釈されます。行番号またはその他のデータを桁 73 から 80 に含んでいる従来の Fortran の慣例に従っている方には、このデフォルトは便利です。

使用法

-qxflag=oldtab オプションを指定しても、ソース・ステートメントは依然としてタブの直後に始まりますが、タブ文字は桁をカウントするための単一の文字として処理されます。 この設定を使用すれば、最大 71 文字の入力を行うことができます。文字数はタブ文字が発生する場所によって異なります。

-qxlf77

カテゴリー

言語エレメント制御

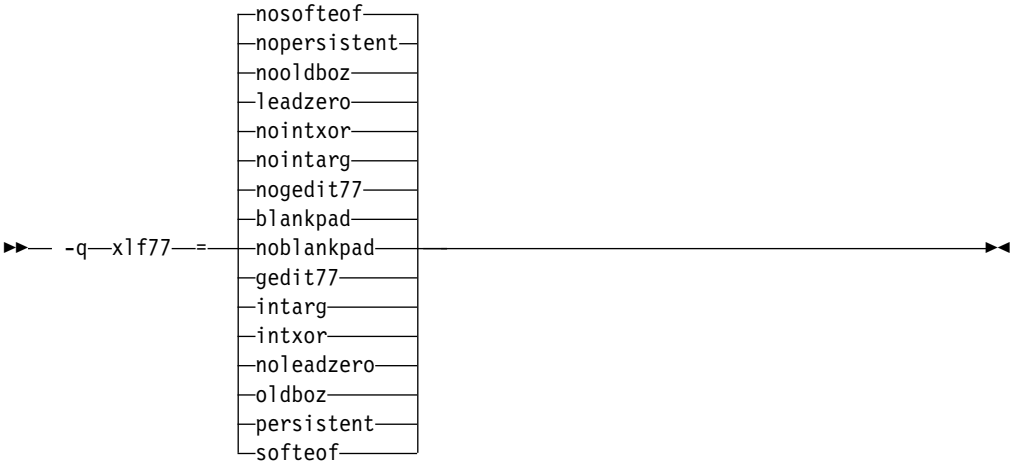
目的

変更された言語セマンティクスと I/O データ形式の FORTRAN 77 の特質との互換性を確保する。

これらの変更のほとんどは、Fortran 90 標準で必要です。

構文

オプション:



@PROCESS:

@PROCESS XLF77(*settings*)

デフォルト

デフォルトでは、コンパイラーはあらゆる場合に Fortran 90、Fortran 95、Fortran 2003、Fortran 2008、および最新バージョンのコンパイラーに適用される設定を使用します。

デフォルトのサブオプションは、**blankpad**、**nogedit77**、**nointarg**、**nointxor**、**leadzero**、**nooldboz**、**nopersistent**、および **nosofteof** です。

これらのデフォルトは、 **xlf90**、 **f90**、**xlf90_r**、 **xlf95**、**f95**、**xlf95_r**、**xlf2003**、**f2003**、**xlf2003_r**、**xlf2008**、**f2008**、**xlf2008_r**、および CUDA Fortran **xlcuf** CUDA Fortran コマンドによってのみ使用されます。これらのコマンドは、新規プログラムをコンパイルするのに使用する必要があります。

パラメーター

XL Fortran バージョン 2 の動作におけるさまざまな面を理解するために、以下のサブオプションの 1 つ以上に対してデフォルト以外の選択項目を選んでください。説明には、デフォルト以外の選択項目を指定した場合に生じる事柄が記載されています。

blankpad | **noblankpad**
内部ファイル、直接アクセス・ファイル、およびストリーム・アクセス・ファイルには、 **pad='no'** と同等のデフォルト設定を使用します。この設定で

は、レコードが持っているよりも多くの文字を形式が必要とする場合にこのようなファイルからの読み取りを行うと、変換エラーが発生します。このサブオプションは、**pad=** 指定子を指定してオープンされた直接アクセス・ファイルまたはストリーム・アクセス・ファイルには影響を与えません。

gedit77 | **nogedit77**
G 編集記述子を持つ **REAL** オブジェクトの出力に FORTRAN 77 のセマンティクスを適用します。フォーマット済み出力文内のリスト項目について、0 の表現が FORTRAN 77 と Fortran 90 では異なります (丸め方式も異なる)。したがって、値と **G** 編集記述子の組み合わせによっては出力内容が異なります。

intarg | **nointarg**
組み込みプロシージャーのすべての整数引数を最も長い引数の種類に変換します (種類が異なる場合)。 Fortran 90 または 95 の規則の下では、最初の引数の種類に基づいて結果タイプを判別する組み込み機能もあります (例えば、**IBSET**)。また、すべての引数が同じ種類でなければならない組み込み関数もあります (例えば、**MIN** および **MAX**)。

intxor | **nointxor**
.XOR. を論理バイナリー組み込み演算子として扱います。これは、**.EQV.** および **.NEQV.** 演算子と同じ優先順位を持っていて、オペレーター・インターフェースで拡張することができます。 (**.XOR.** のセマンティクスは **.NEQV.** のセマンティクスと同じであるため、 **.XOR.** は Fortran 90 または Fortran 95 言語標準では使用されません。)

それ以外の場合、**.XOR.** 演算子は定義された演算子としてのみ認識されます。組み込み演算はアクセス不能で、優先順位は、演算子が単項コンテキストで使用されているか、それともバイナリー・コンテキストで使用されているかによって異なります。

leadzero | **noleadzero**
D、**E**、**L**、**F**、**Q** などの編集記述子を使用して、実際の出力で先行ゼロを発生させます。

oldboz | **nooldboz**
BLANK= 指定子や **BN** または **BZ** 編集制御記述子とは無関係に、**B**、**O**、**Z** などの編集記述子によって読み取られたデータに対して、ブランクをゼロにします。また、先行ゼロ、長すぎる出力の切り捨てを維持します。これは、Fortran 90 または Fortran 95 標準の一部ではありません。

persistent | **nopersistent**
ステートメント**ENTRY** ステートメントが含まれているサブプログラムへの引数のアドレスを静的ストレージに保存します。これは、パフォーマンスを向上させるために変更された実装上の選択項目です。

softeof | **nosofteof**
ユニットが endfile レコードの後に位置付けられているときに、 **READ** 操作と **WRITE** 操作を実行できるようにします。ただし、その位置が **ENDFILE** 文を実行した結果である場合は除きます。このサブオプションは、一部の既存プログラムが依存している旧バージョンの XL Fortran の FORTRAN 77 拡張機能を再現します。

使用法

前のプログラムを変更しないでコンパイルして実行する場合のみ、適切な呼び出しコマンドを引き続き使用しても、このオプションを意識する必要はありません。

このオプションを使用できるのは、既存のソース・ファイルまたはデータ・ファイルを Fortran 90、Fortran 95、Fortran 2003、および Fortran 2008 で **xlf90**、**f90**、**xlf90_r**、**xlf95**、**f95**、**xlf95_r**、**xlf2003**、**f2003**、**xlf2003_r**、**xlf2008**、**f2008**、**xlf2008_r**、または CUDA Fortran **xlcuf** CUDA Fortran コマンドとともに使用していて、変更された動作またはデータ・フォーマットが原因で非互換性が検出された場合に限られます。

最終的に、データ・ファイルを再作成する、またはソース・ファイルを変更して、古い動作の依存性を除去することができます。

-qxlf90

カテゴリー

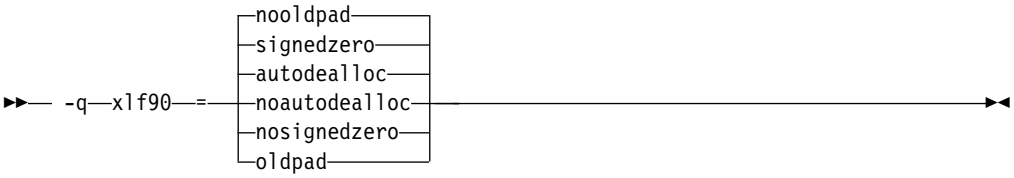
言語エレメント制御

目的

Fortran 言語の特質に関する Fortran 90 標準との互換性を確保する。

構文

オプション:



@PROCESS:

@PROCESS XLF90(*settings*)

デフォルト

-qxlf90 のデフォルトのサブオプションは、指定する呼び出しコマンドによって異なります。

xlf95、**f95**、**xlf95_r**、**xlf2003**、**f2003**、**xlf2003_r**、**xlf2008**、**f2008**、**xlf2008_r**、または CUDA Fortran **xlcuf** CUDA Fortran コマンドの場合は、デフォルトのサブオプションは **signedzero**、**autodealloc**、および **nooldpad** です。

他のすべての呼び出しコマンドでは、デフォルトは **nosignedzero**、**noautodealloc** および **oldpad** です。

パラメーター

signedzero | nosignedzero

SIGN(A,B) 関数が符号付きの実数 0.0 を処理する方法を決定します。**-qxlf90=signedzero** コンパイラー・オプションを指定した場合、B=-0.0 のときに、**SIGN(A,B)** は -|A| を戻します。この動作は Fortran 95 標準に準拠するものであり、バイナリー浮動小数点演算のための IEEE 標準と整合しています。 **REAL(16)** データ型では、 XL Fortran はゼロを負のゼロとしては扱わないことに注意してください。

このサブオプションでは、以下の場合に負符号 (-) が印刷されるかどうかも決定します。

- 形式化された出力に負のゼロが含まれる場合。この場合も、**REAL(16)** データ型では、 XL Fortran はゼロを負のゼロとしては扱わないことに注意してください。
- 出力形式のゼロ (つまり、結果出力がゼロであるように見せるために、ゼロ以外の末尾の桁は出力から切り捨てられる) を含む負の値の場合。この場合、**signedzero** サブオプションは **REAL(16)** データ型に影響します。出力形式がゼロであるゼロ以外の負の値は、マイナス記号付きで表示されます。

-qxlf90=nosignedzero を使用する場合、パフォーマンスを向上させるために、**-qstrict=nozerosigns** オプションを設定することを考慮してください。

autodealloc | noautodealloc

SAVE または **STATIC** 属性のいずれかを指定せずにローカルに宣言された割り振り可能で、サブプログラムを終了するときに現在割り振り済みの状況にある割り振り可能なオブジェクトを、コンパイラーが割り振り解除するかどうかを決定します。この動作は、Fortran 95 標準に準拠しています。ローカルに割り振り可能なオブジェクトすべてを明示的に割り振り解除していることが明白な場合は、このサブオプションをオフにして、パフォーマンスの低下を回避することをお勧めします。

oldpad | nooldpad

PAD=specifier が **INQUIRE -qxlf90=nooldpad** を指定すると、接続がない場合や接続が不定形式 I/O 用の場合に UNDEFINED を戻します。この動作は、Fortran 95 標準以上で準拠されます。**-qxlf90=oldpad** を指定すると、Fortran 90 動作が保持されます。

例

次のプログラムを見てください。

```
PROGRAM TESTSIGN
REAL X、Y、Z
X=1.0
Y=-0.0
Z=SIGN(X,Y)
PRINT *,Z
END PROGRAM TESTSIGN
```

この例の出力は、呼び出しコマンドと、指定する **-qxlf90** サブオプションによって異なってきます。 例を以下に示します。

呼び出しコマンド**/xlf2008** サブオプション	出力
xlf2008	-1.0
xlf2008 -qxlf90=signedzero	-1.0
xlf2008 -qxlf90=nosignedzero	1.0
xlf2003	-1.0
xlf2003 -qxlf90=signedzero	-1.0
xlf2003 -qxlf90=nosignedzero	1.0
xlf95	-1.0
xlf95 -qxlf90=signedzero	-1.0
xlf95 -qxlf90=nosignedzero	1.0
xlf90	1.0
xlf	1.0

関連情報

- 「*XL Fortran* ランゲージ・リファレンス」の『組み込みプロシージャー』セクションおよび『配列の概念』セクションに記載されている **SIGN** に関する情報を参照してください。
- 274 ページの『-qstrict』

-qxlf2003

カテゴリー

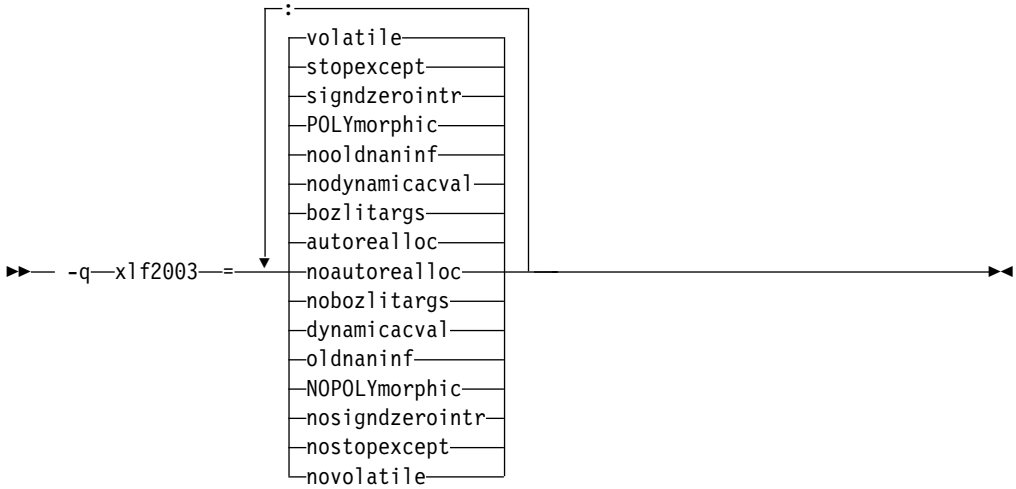
言語エレメント制御

目的

前の Fortran 標準に準拠するコンパイラー呼び出しによってコンパイルする際に、Fortran 2003 標準固有の言語機能を使用できるようにする機能、および Fortran 2003 標準に準拠するコンパイラー呼び出しによってコンパイルする際に、それらの機能を使用不可にする機能を提供する。

構文

オプション:



@PROCESS:

@PROCESS XLF2003(*suboption*,*suboption*,...)

デフォルト

デフォルトのサブオプションは、指定する呼び出しコマンドによって異なります。

f2003、**xlf2003**、または **xlf2003_r** コマンドのデフォルトは以下のとおりです。

autorealloc:bozlitargs:nodynamicacval:nooldnaninf:polymorphic:signdzerointr:

stopexcept:volatile

他のすべての呼び出しコマンドのデフォルトは以下のとおりです。

noautorealloc:nobozlitargs:nodynamicacval:oldnaninf:nopolymorphic:

nosigndzerointr:nostopexcept:novolatile

パラメーター

autorealloc | **noautorealloc**
割り振り可能な変数への割り当て時に、コンパイラーが右側 (RHS) の形状と一緒に左側 (LHS) を自動的に再割り振りするかどうかを制御します。割り当ての前に LHS 変数が割り振られなかったならば、LHS 変数が自動的に割り振られます。 デフォルトは、**f2003**、**xlf2003**、および **xlf2003_r** コマンドでは **autorealloc**、他のすべてのコマンドでは、**noautorealloc** です。このサブオプションは、LHS および RHS の length 型パラメーターの値が異なる場合、再割り当てにおいて効力がありません。

bozlitargs | **nobozlitargs**
bozlitargs サブオプションにより、**INT**、**REAL**、**CMPLX**、または **DBLE** 組み込み関数への引数としての boz-literal 定数の引き渡しが、Fortran 2003 標準に準拠するようにします。**f2003** コマンド、**xlf2003** コマンド、および **xlf2003_r** コマンドの場合、デフォルトは **bozlitargs** です。**-qlanglvl=2003pure** または **-qlanglvl=2003std** オプションも指定する

必要があります。**-qport=typlssarg** および **-qxlf2003=bozlitargs** が指定されると、boz-literal 定数の **CMPLX** 組み込みに引き渡され、標準ではない結果が発生します。

dynamicacval | **nodynamicacval**

dynamicacval が有効な場合、動的な型の配列コンストラクター値を使用して、配列コンストラクターの型を判別するため、配列コンストラクター内で無制限ポリモアフィック・エンティティーを使用できます。

nodynamicacval が有効な場合、宣言された型の配列コンストラクターを使用して、配列コンストラクターの型を判別するため、配列コンストラクター内で無制限ポリモアフィック・エンティティーを使用できません。

注**: -qxlf2003=dynamicacval** オプションを有効にするには、**-qxlf2003=polymorphic** も指定する必要があります。

oldnaninf | **nooldnaninf**

oldnaninf サブオプションは、IEEE NaN および無限大例外値の出力の書式設定を制御します。このサブオプションは入力には影響しません。

oldnaninf が有効なとき、コンパイラーは、XL Fortran V10.1 (および旧バージョン) の出力の動作を使用します。つまり、無限大の場合は INF、静止またはシグナル通知 NaN の場合は NAN となります。

nooldnaninf が有効なとき、IEEE 例外値のコンパイラー出力が Fortran 2003 標準に準拠します。つまり、無限大は Inf、静止 NaN は NaN(Q)、通知 NaN では NaN(S) です。

polymorphic | **nopolymorphic**

polymorphic が有効なとき、コンパイラーは、Fortran ソース・ファイルでポリモアフィック項目を許可します。**CLASS** 型指定子、**SELECT TYPE** 構成体を指定し、他の Fortran 文ではポリモアフィック項目を使用することができます。ポリモアフィック引数を使用すると、コンパイラーがそれぞれの派生型定義のランタイム型情報を作成することもできます。

nopolymorphic が有効なとき、ポリモアフィック項目は、Fortran ソース・ファイルで指定されず、ランタイム型情報も生成されません。

signdzerointr | **nosigndzerointr**

signdzerointr が有効なとき、符号付きゼロを、 **SQRT**、**LOG**、および **ATAN2** 組み込み関数に引き渡すと、Fortran 2003 標準に整合した結果を戻します。 **-qxlf90=signedzero** オプションも有効にする必要があります。**xlf**、**xlf_r**、**f77**、**fort77**、**xlf90**、**xlf90_r**、および **f90** 呼び出しの場合は、両方のオプションを指定して Fortran 2003 の動作を実現します。

次の例は、このサブオプションの使用法を示しています。

```
! If the Test program is compiled with -qxlf2003=signdzerointr
! and -qxlf90=signedzero, then Fortran 2003 behavior is seen.
! Otherwise, this program will demonstrate Fortran 95 behavior.

Program Test

real a, b
complex j, l
a = -0.0
j = sqrt(cmplx(-1.0,a))
b = atan2(a,-1.0)
l = log(cmplx(-1.0,a))
```

```
print *, 'j=', j
print *, 'b=', b
print *, 'l=', l
end

! Fortran 95 output:

 j= (-0.0000000000E+00,1.000000000)
 b= 3.141592741
 l= (0.0000000000E+00,3.141592741)

! Fortran 2003 output:

 j= (0.0000000000E+00,-1.000000000)
 b= -3.141592741
 l= (0.0000000000E+00,-3.141592741)
```

stopexcept | nostopexcept

stopexcept が有効なとき、IEEE 浮動小数点例外が **STOP** 文によってシグナル通知されると、情報メッセージが表示されます。メッセージは、次の形式をとります。

```
STOP [stop-code]
(OVERFLOW, DIV-BY-ZERO, INVALID, UNDERFLOW, INEXACT)
```

ここで、*stop-code* は、**STOP** 文で指定されたオプションの桁ストリングまたは文字定数に相当します。対応するフラグが設定されている場合のみ、OVERFLOW、DIV-BY-ZERO、INVALID、UNDERFLOW および INEXACT が表示されます。

次の例では、生成された対応するメッセージが表示されます。

```
real :: rl1, rl2, rl3, rl4
logical :: l

rl1 = 1.3
rl2 = 0.0

rl3 = rl1 / rl2 ! divide by zero

rl4 = rl3 ! to make sure rl3 is actually used

rl4 = log(-rl1) ! invalid input for log

stop "The End"

end
```

Output:

```
STOP The End
(DIV-BY-ZERO, INVALID)
```

nostopexcept が有効なとき、情報メッセージは抑制されます。

volatile | novolatile

volatile を有効にした場合は、参照結合または親子結合されている不揮発性エンティティーを、内部スコープまたはローカル・スコープ内では VOLATILE として指定できます。

使用法

アプリケーションが F2003 ポリモアフィズムを使用する場合、**polymorphic** を指定して各ユニットをコンパイルする必要があります。アプリケーションが、ポリモアフィズムを使用しない場合、**nopolymorphic** サブオプションを指定します。そのようにすると、コンパイル時間を節約でき、実行時のパフォーマンスも向上することも可能です。

関連情報

「*XL Fortran* ランゲージ・リファレンス」で、以下の情報を参照してください。

- ポリモアフィック・エンティティー
- 配列コンストラクター

-qxlf2008

カテゴリー

言語エレメント制御

目的

前の Fortran 標準に準拠するコンパイラー呼び出しによってコンパイルする際に、Fortran 2008 標準固有の言語機能を使用できるようにする機能、および Fortran 2008 標準に準拠するコンパイラー呼び出しによってコンパイルする際に、それらの機能を使用不可にする機能を提供する。

構文

オプション:

@PROCESS:

@PROCESS XLF2008(*suboption*,*suboption*,...)

デフォルト

デフォルトのサブオプションは、指定する呼び出しコマンドによって異なります。

コマンド **f2008**、 **xlf2008**、**xlf2008_r**、および CUDA Fortran **xlcuf** CUDA Fortran のデフォルトは次のとおりです。

checkpresence

他のすべての呼び出しコマンドのデフォルトは以下のとおりです。

nocheckpresence

パラメーター

checkpresence | **nocheckpresence**

checkpresence が有効なとき、Fortran 2008 標準に従って仮引数の存在がチェックされます。**nocheckpresence** が有効なとき、前の Fortran 標準に従って仮引数の存在がチェックされます。仮引数の存在について詳しくは、『指定されていないオプションの仮引数に関する制約事項』を参照してください。

注**: -qxlf2008=checkpresence** が有効なとき、実引数の割り振りおよび関連付け状況の実行時検査が原因で、プログラムのパフォーマンスが抑制されます。パフォーマンスへのこれらの影響を回避するには、**-qxlf2008=nocheckpresence** の使用を検討してください。

-qxlines

カテゴリー

入力制御

目的

列 1 に X がある固定ソース形式の行をコンパイルするか、またはコメントとして扱うかを指定する。

このオプションは、条件付きコンパイル (デバッグ) 文字として、桁 1 に文字「d」を認識するのに似ています。 **-qxlines** オプションは、このコンパイラー・オプションが使用可能なとき、条件付きコンパイル文字として桁 1 に文字「x」を認識します。桁 1 の「x」は、ブランクとして解釈され、その行はソース・コードとして処理されます。

構文

オプション:

@PROCESS:

@PROCESS XLINES | **NOXLINES**

デフォルト

-qnoxlines

このオプションは、デフォルトで **-qnoxlines** に設定され、固定ソース形式で桁 1 に文字「x」がある行はコメント行として扱われます。

条件付きコンパイル文字として「d」を使用するのに適用するデバッグ行の規則は、条件付きコンパイル文字「x」にも適用します。

-qxlines コンパイラー・オプションは、固定ソース形式にのみ適用可能です。

使用法

条件付きコンパイル文字「x」および「d」は、固定ソース形式プログラムと継続されるソース行内で混用することが可能です。条件付きコンパイル行が、次の行に継続する場合は、すべての継続行には、桁 1 に「x」または「d」が存在している必要があります。継続されるコンパイル・ステートメントの最初の行が、桁 1 の「x」または「d」のいずれで始まるデバッグ行ではない場合は、後続の継続行は、そのステートメントが構文的に正しい限り、デバッグ行として指定されます。

OMP 条件付きコンパイル文字「!$」、「C$」、および「*$」は、固定ソース形式および継続されるソース行内の両方で、条件付き文字「x」および「d」と混用することができます。 OMP 条件付き文字の規則は、このインスタンスでまだ適用されます。

例

-qxlines の基本ケースの例は以下のとおりです。

```
 C2345678901234567890
        program p
        i=3 ; j=4 ; k=5
 X      print *,i,j
 X    +        ,k
        end program p

<output>: 3 4 5     (if -qxlines is on)
          no output (if -qxlines is off)
```

この例では、条件付きコンパイル文字「x」および「d」は、最初の行の「x」と混用されています。

```
C2345678901234567890
       program p
       i=3 ; j=4 ; k=5
X      print *,i,
D    +         j,
X    +         k
       end program p

<output>: 3 4 5 (if both -qxlines and -qdlines are on)
          3 5   (if only -qxlines is turned on)
```

ここでは、条件付きコンパイル文字「x」および「d」は、最初の行の「d」と混用されています。

```
C2345678901234567890
       program p
       i=3 ; j=4 ; k=5
D      print *,i,
X    +         j,
D    +         k
       end program p

<output>: 3 4 5 (if both -qxlines and -qdlines are on)
          3 5   (if only -qdlines is turned on)
```

この例では、最初の行はデバッグ行ではありませんが、継続行は、桁 1 に「x」があるので、デバッグ行として解釈されます。

```
C2345678901234567890
       program p
       i=3 ; j=4 ; k=5
       print *,i
X    +        ,j
X    +        ,k
       end program p

<output>: 3 4 5 (if -qxlines is on)
          3     (if -qxlines is off)
```

関連情報

- 「*XL Fortran* ランゲージ・リファレンス」の『条件付きコンパイル』

-qxref

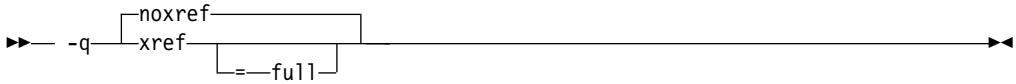
カテゴリー

リスト作成、メッセージ、およびコンパイラー情報

目的

リストの属性および相互参照の相互参照コンポーネントを含むコンパイラー・リストを作成する。

構文



@PROCESS:

@PROCESS XREF[(FULL)] | **NOXREF**

デフォルト

-qnoxref

使用法

-qxref だけを指定すると、使用される識別子だけが報告されます。 **-qxref=full** を指定すると、使用されてもされなくても、プログラム内にあるすべての識別子に関する情報がリストに含まれます。

-qxref=full の後に **-qxref** が指定されても、完全な相互参照リストが依然として作成されます。

デバッグ中に相互参照リストを使用して、問題 (変数の定義前使用や変数名の誤入力など) を見つけることができます。

関連情報

- 72 ページの『リスト、メッセージ、およびコンパイラー情報』
- 356 ページの『属性および相互参照セクション』

-qzerosize

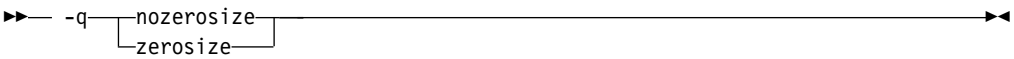
カテゴリー

64 ページの『言語エレメント制御』

目的

サイズがゼロの文字ストリングおよび配列を処理する可能性のあるプログラム内で、このようなオブジェクトの有無についての検査を行うかどうかを決定する。

構文



@PROCESS:

```
@PROCESS ZEROSIZE | NOZEROSIZE
```

デフォルト

デフォルト設定は、どのコマンドでコンパイラーを呼び出すかによって異なります。

- **-qzerosize**: コマンド **f90**、**xlf90**、**xlf90_r**、**f95**、**xlf95**、**xlf95_r**、**f2003**、**xlf2003**、**xlf2003_r**、**f2008**、**xlf2008**、**xlf2008_r**、および CUDA Fortran **xlcuf** CUDA Fortran のデフォルトです。
- **-qnozerosize**: **xlf** コマンドおよび **xlf_r** コマンドのデフォルト (これらのコマンドを使用して .f ファイル、.F ファイル、.f77 ファイル、または .F77 ファイルがコンパイルされる場合)
- **-qnozerosize**: **f77** コマンドおよび **fort77** コマンドのデフォルト

使用法

サイズがゼロの文字ストリングや配列を処理する可能性がある Fortran 90、Fortran 95、Fortran 2003、および Fortran 2008 プログラムの場合は、**-qzerosize** を使用します。

サイズがゼロのオブジェクトを使用できない FORTRAN 77 プログラムや、それらを使用しない Fortran 90 および Fortran 95 プログラムの場合は、**-qnozerosize** でコンパイルすると、いくつかの配列演算または文字ストリング演算のパフォーマンスを改善することができます。

-C オプションが実行する実行時検査は、 **-qzerosize** が有効であると、時間が多少長くかかります。

-r

カテゴリー

オブジェクト・コード制御

@PROCESS

なし。

目的

別の ID コマンド呼び出しでの入力ファイルとして使用する実行不能出力ファイルを生成する。このファイルには未解決のシンボルも含むことができる。

構文

デフォルト

適用されません。

使用法

このフラグを使用して作成されたファイルは、別のコンパイラー呼び出しまたは ID コマンド呼び出しの入力ファイルとして使用されることが予想されます。

事前定義マクロ

なし。

例

`myprogram.f` および `myprog2.f` を単一のオブジェクト・ファイル `mytest.o` にコンパイルするには、以下のように入力します。

```
xlf myprogram.f myprog2.f -r -o mytest.o
```

-S

カテゴリー

出力制御

@PROCESS

なし。

目的

ソース・ファイルごとにアセンブラー言語ファイルを生成する。

構文

規則

このオプションが指定されると、コンパイラーは、オブジェクト・ファイルまたは実行可能ファイルの代わりに、出力ファイルとしてアセンブラー・ソース・ファイルを作成します。

制約事項

作成されたアセンブラー・ファイルには、 **-qipa** オプションまたは **-g** オプションによって **.o** ファイルに入れられたすべてのデータが入っているわけではありません。

例

```
xlf95 -O3 -qhot -S test.f               # Produces test.s
```

関連情報

-o オプションを使用すれば、その結果作成されるアセンブラー・ソース・ファイルの名前を指定できます。

-t

カテゴリー

コンパイラーのカスタマイズ

@PROCESS

なし。

目的

-B オプションで指定したプレフィックスを、指定したコンポーネントに適用する。

構文

デフォルト

すべてのコンパイラー・コンポーネントのデフォルト・パスは、コンパイラー構成ファイルで定義されます。

パラメーター

以下の表は、**-t** パラメーターとコンポーネント名との対応を示しています。

パラメーター	説明	コンポーネント名
a	アセンブラー	as
b	低水準最適化プログラム	xlfcode
c	コンパイラーのフロントエンド	xlfentry
d	逆アセンブラー	dis
F	C プリプロセッサー	cpp
h	配列言語最適化プログラム	xlfhot
I (大文字の i)	高水準最適化プログラム、コンパイル・ステップ	ipa
l (小文字の L)	リンカー	ld
z	バインダー	bolt

使用法

このオプションは、**-B***prefix* オプションと一緒に使用します。

例

名前 `/u/newones/compilers/` が、コンパイラーおよびアセンブラー・プログラム名の接頭部として付くように `myprogram.f` をコンパイルするには、以下を入力してください。

```
xlf myprogram.f -B/u/newones/compilers/ -tca
```

関連情報

- 83 ページの『-B』

-U

カテゴリー

言語エレメント制御

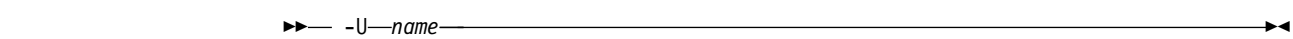
@PROCESS

なし。

目的

コンパイラーまたは **-D** コンパイラー・オプションによって定義されたプリプロセッサー・マクロ名の定義を解除する。

構文



デフォルト

適用されません。

パラメーター

name
　定義解除するマクロ。

使用法

-U オプションは、`#undef` プリプロセッサー・ディレクティブと同等ではありません。`#define` プリプロセッサー・ディレクティブによってソースに定義された名前を定義解除することはできません。定義解除できるのは、コンパイラーまたは **-D** オプションによって定義された名前のみです。

-U*name* オプションは、**-D***name* オプションよりも高い優先順位を持っています。

15.1.6 より前のバージョンの IBM XL Fortran for Linux では、**-U** オプションは **-qmixed** オプションの短縮形でした。

例

ご使用のオペレーティング・システムが名前 `__unix` を定義していても、その名前 `__unix` の定義が以下を入力することによって無効になるように、その名前が定義される条件でコード・セグメントをコンパイル `myprogram.f` で入力しない場合を想定します。

```
xlf myprogram.f  -U__unix
```

関連情報

- 86 ページの『-D』
- 216 ページの『-qmixed』
- 244 ページの『-qpreprocess』

-u

カテゴリー

言語エレメント制御

目的

変数名の暗黙型定義が許可されないことを指定する。

これには、暗黙の文を許可する個々の有効範囲に含まれる **IMPLICIT NONE** 文を使用するときと同じ効果があります。

構文

@PROCESS:

@PROCESS UNDEF | **NOUNDEF**

デフォルト

暗黙的な入力を許可する **-qnoundef**。

関連情報

「*XL Fortran* ランゲージ・リファレンス」の 『**IMPLICIT**』を参照してください。

これは、**-qundef** の短い形式です。 295 ページの『-qundef』を参照してください。

-v

カテゴリー

リスト、メッセージ、およびコンパイラー情報

@PROCESS

なし。

目的

呼び出されているプログラムと各プログラムに指定されているオプションの名前を戻すことによって、コンパイルの進行を報告する。

構文

デフォルト

適用されません。

使用法

特定のコンパイルに関して、このオプションが生成する出力を調べると、次の事項を判別するのに役立ちます。

- どのファイルが関係があるか
- 個々のステップに、どのオプションが有効であるか
- 障害発生時のコンパイルの進み具合

関連情報

- 82 ページの『-#』 は **-v** と似ていますが、実際にはどのコンパイル・ステップも実行しません。
- 322 ページの『-V』

-V

カテゴリー

リスト、メッセージ、およびコンパイラー情報

@PROCESS

なし。

目的

コマンドを作成するためにディスプレイから直接カット・アンド・ペーストできる点を除き、**-v** と同じ。

デフォルト

適用されません。

構文

関連情報

- 321 ページの『-v』
- 82 ページの『-#』

-w

カテゴリー

リスト、メッセージ、およびコンパイラー情報

@PROCESS

なし。

目的

警告メッセージを抑制する (-qflag=e:e と同等)。

構文

デフォルト

適用されません。

関連情報

- 151 ページの『-qflag』

-W、-X

カテゴリー

コンパイラー・カスタマイズ

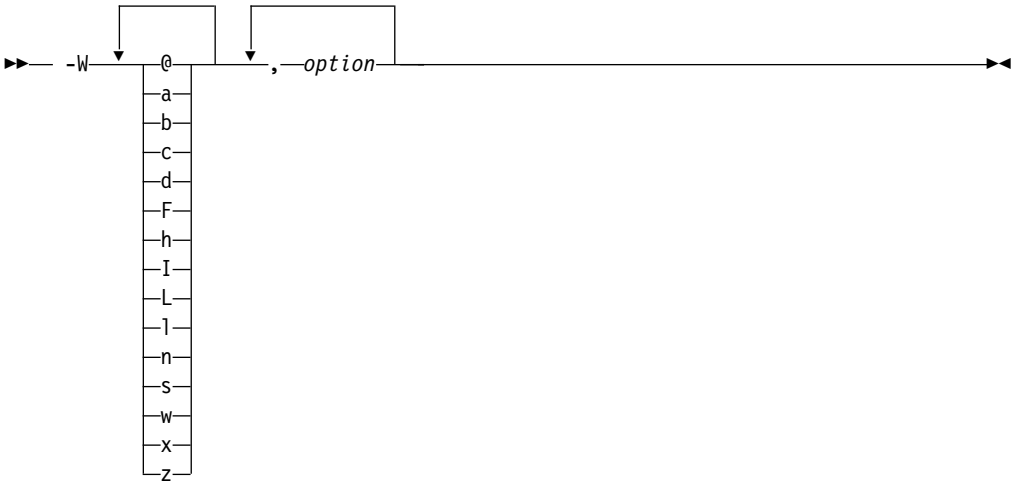
@PROCESS

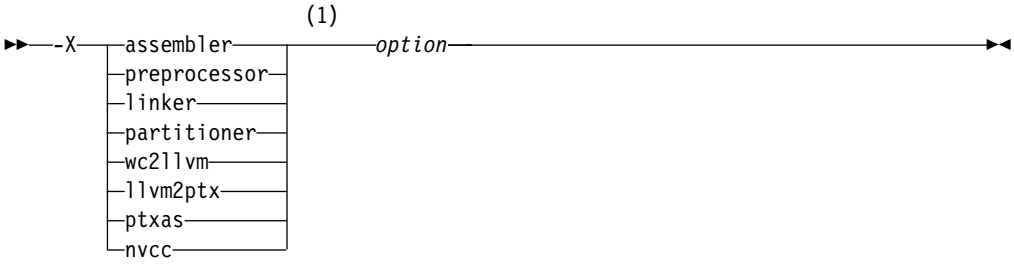
なし。

目的

1 つ以上のオプションをコンパイル中に実行されるコンポーネントに渡す。

構文





注:

1 *option* の前にスペースを少なくとも 1 つ挿入する必要があります。

パラメーター

以下の表は、**-X** パラメーターや**-W** パラメーターとコンポーネント名との間の対応を示しています。

-W のパラメーター	**-X** のパラメーター	説明	コンポーネント名
GPU @	ptxas	PTX アセンブラー	ptxas

-W のパラメーター	**-X** のパラメーター	説明	コンポーネント名
a	assembler	アセンブラー	as
b		低水準最適化プログラム	xlfcode
c		コンパイラーのフロントエンド	xlfentry
d		逆アセンブラー	dis
F	プリプロセッサー	C プリプロセッサー	cpp
h		配列言語最適化プログラム	xlfhot
I (大文字の i)		高水準最適化プログラム、コンパイル・ステップ	ipa
L		高水準最適化プログラム、リンク・ステップ	ipa
l (小文字の L)	リンカー (linker)	リンカー	ld
GPU n	nvcc	NVIDIA C コンパイラー。装置リンカーとして使用	nvcc
GPU s	partitioner	XL 中間言語 (W-Code) スプリッター	partitioner
GPU w	wc2llvm	NVVM-IR 変換プログラムに対する XL 中間言語 (W-Code)	wc2llvm
GPU x	llvm2ptx	NVVM-IR から PTX への変換プログラム	llvm2ptx
z		バインダー	bolt

オプション
受け渡し先のコンポーネントに有効なオプション。

注: GPU

- NVVM-IR から PTX への変換オプションは、CUDA Toolkit 資料 (http://docs.nvidia.com/cuda/libnvvm-api/group__compilation.html) の nvvmCompileProgram にある libNVVM API セクションにあります。
- PTX アセンブラー・オプションのリストは、CUDA Toolkit から **-h** を指定して **ptxas** を実行することによって入手できます。

GPU

使用法

オプション・ストリング内のシェルに特有の文字を入れる必要がある場合は、その文字の前にバックスラッシュを置いてください。例えば、構成ファイルに **-W** オプションや **-X** オプションを使用する場合は、エスケープ・シーケンスの円記号とコンマ (¥,) を使用して、パラメーター・ストリング内にコンマを表示できます。**-W**

オプションの後に続いているストリングでは、各オプションに対して分離文字としてコンマを使用し、スペースは入れないでください。

たいていのオプションは、リンカー **ld** に渡すときに **-W** オプションや **-X** オプションを使用する必要はありません。**-q** オプション以外の認識されないコマンド行オプションは自動的にリンカーに渡されます。厳密に **-W** や **-X** を必要とするのは、**-v** や **-S** などのコンパイラー・オプションと同じ文字を持つリンカー・オプションのみです。

例

CUDA Fortran

デバイス・コードを最適化できないようにするには、以下のいずれかのコマンドを使用して、NVVM-IR から PTX への変換プログラムや、PTX アセンブラーに、オプションを渡します。

```
xlcuf mycudaprogram.cuf -Xllvm2ptx -nvvm-compile-options=opt=0 -Xptxas -O0
xlcuf mycudaprogram.cuf -Wx,-nvvm-compile-options=opt=0 -W@,-O0
```

-nvvm-compile-options=opt=0 は NVVM-IR から PTX への変換プログラムのオプションです。 **-O0** は PTX アセンブラー・オプションです。

CUDA Fortran

関連情報

- *XL Fortran* を使用した *CUDA Fortran* プログラミングの概要

-y

カテゴリー

浮動小数点および整数のコントロール

目的

コンパイル時に定数浮動小数点式を評価する場合にコンパイラーが使用する丸めモードを指定する。

これは **-qieee** オプションと同等です。

構文

@PROCESS:

```
@PROCESS IEEE(Near | Minus | Plus | Zero)
```

デフォルト

-yn

パラメーター

n 最も近い値に丸めます。

m 負の無限大の方向に丸める。

p 正の無限大の方向に丸める。

z ゼロの方向に丸める。

使用法

プログラムが、実数 (16) 値を含む演算を含む場合、丸めモードは、最も近い値の **-yn** に設定される必要があります。

関連情報

- 101 ページの『-O』
- 153 ページの『-qfloat』
- 173 ページの『-qieee』

第 7 章 コンパイラー・コマンド参照

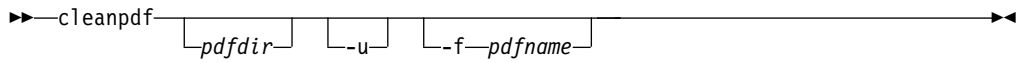
このセクションでは、XL Fortran に組み込まれているコマンドについて紹介します。

cleanpdf

目的

プロセス ID サフィックスを持つ PDF ファイルを含む、すべての PDF ファイルまたは指定された PDF ファイルを削除します。プログラムを変更して PDF プロセスをもう一度実行する場合、プロファイル情報を除去するとランタイム・オーバーヘッドが減ります。

構文



パラメーター

pdfdir 削除される PDF ファイルが含まれているディレクトリーを指定します。*pdfdir* が指定されていない場合、ディレクトリーは PDFDIR 環境変数により設定されます。PDFDIR が設定されていない場合、ディレクトリーは現行ディレクトリーです。

-f *pdfname*
削除する PDF ファイルの名前を指定します。 `-f` *pdfname* を指定しない場合、`.`*<output_name>*`_pdf` はデフォルトで削除されます。 *<output_name>* は、**-qpdf1** 指定でのプログラムのコンパイル時に生成された出力ファイルの名前です。

-u `-f` *pdfname* が指定されている場合、**-f** で削除されるファイルに加えて、命名規則 *pdfname*`.`*<pid>* に従っているファイル (該当する場合) も削除されます。 *<pid>* は、PDF トレーニング・ステップで実行中のプロセスの ID です。

`-f` *pdfname* が指定されていない場合、デフォルトの PDF ファイル `.`*<output_name>*`_pdf` が削除されます。該当する場合は、デフォルト命名規則 `.`*<output_name>*`_pdf.`*<pid>* を持つファイルも削除されます。

使用法

cleanpdf を実行するのは、特定のアプリケーションの PDF プロセスが終了した場合のみにしてください。これに従わないと、そのアプリケーションで PDF プロセスを使用して再開する場合に、**-qpdf1** を指定してすべてのファイルをもう一度コンパイルする必要が生じます。

cleanpdf は `/opt/ibm/xlf/16.1.0/bin/` にあります。

関連情報

- 228 ページの『-qpdf1、-qpdf2』
- 「*XL Fortran* 最適化およびプログラミング・ガイド」の『プロファイル指示フィードバック』
- 『環境変数の正しい設定方法』

genhtml

目的

既に生成済みの XML レポートの HTML バージョンを表示します。

使用法

以下のコマンドを使用して、既存の XML レポートを HTML フォーマットで表示します。以下のコマンドは、HTML コンテンツを標準出力に生成します。

```
genhtml xml_file
```

以下のコマンドを使用して、HTML コンテンツを、定義した HTML ファイルに生成します。 Web ブラウザーを使用して、生成された HTML ファイルを表示できます。

```
genhtml xml_file > target_html_file
```

注**:** HTML ファイル名のサフィックスは、静的 HTML ページの標準に準拠している必要があります (例えば、`.html` や `.htm`)。そうしないと、Web ブラウザーでファイルを開くことができない場合があります。

関連情報

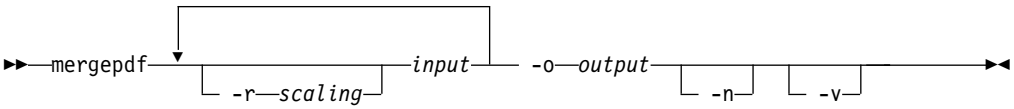
- 205 ページの『-qlistfmt』

mergepdf

目的

複数の PDF ファイルを単一の PDF ファイルにマージします。

構文



パラメーター

-r *scaling*
PDF ファイルの位取りの比率を指定します。この値はゼロより大でなければならず、整数または浮動小数点のいずれの値にすることもできます。指定されない場合は、1.0 の率が想定されます。

input PDF 入力ファイルの名前、または PDF ファイルが入っているディレクトリーの名前を指定します。

-o *output*
PDF 出力ファイルの名前、またはマージされた出力が書き込まれるディレクトリーの名前を指定します。

-n これは、PDF ファイルが正規化されないことを指定します。

-v 冗長モードを指定し、内部およびユーザー指定のスケーリング比率が標準出力に表示されるようにします。

使用法

デフォルトでは、**mergepdf** によってファイルが正規化されます。その際は、各プロファイルが同じ全加重を持ち、それに応じて個々のカウンターが拡大/縮小されるというような方法が使用されます。このプロシージャーは、ユーザー指定の比率が (**-r** を使用して) 適用される前に行われます。**-n** が指定された場合、正規化は行われません。 **-n** も **-r** も指定されない場合、PDF ファイルは拡大/縮小されません。

mergepdf は `/opt/ibm/xlf/16.1.0/bin/` にあります。

例

複数の PDF ファイルが存在する場合は、 **mergepdf** コマンドを使用して、それらのファイルを 1 つの PDF ファイルに結合します。例えば、時間の 53%、32%、15% にそれぞれ発生する使用パターンを表す 3 つの PDF ファイルを作成する場合は、次のコマンドが使用してください。

```
mergepdf -r 53 file_path1  -r 32 file_path2  -r 15 file_path3 -o file_path4
```

file_path1、*file_path2*、および *file_path3* はマージする PDF ファイルのディレクトリーおよび名前を指定し、*file_path4* は出力 PDF ファイルのディレクトリーおよび名前を指定します。

関連情報

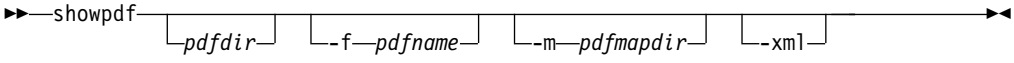
- 228 ページの『-qpdf1、-qpdf2』
- 「*XL Fortran* 最適化およびプログラミング・ガイド」の『プロファイル指示フィードバック』

showpdf

目的

PDF ファイルおよび PDF マップ・ファイルに書き込まれるプロファイル情報の一部を表示します。このコマンドを使用するには、最初に **-qpdf1** オプションを使用してプログラムをコンパイルする必要があります。

構文



パラメーター

pdfdir
プロファイル指示フィードバック (PDF) ファイルを格納するディレクトリー。PDF1 ステップの後に PDFDIR 環境変数を変更していない場合、PDF マップ・ファイルもこのディレクトリーに格納されます。このパラメーターを指定しない場合、コンパイラーは PDFDIR 環境変数の値をディレクトリーの名前として使用します。

pdfname
PDF ファイルの名前。このパラメーターを指定しない場合、コンパイラーはデフォルトで、PDF ファイルの名前として *.<output_name>*_pdf を使用します。*<output_name>* は、**-qpdf1** 指定でのプログラムのコンパイル時に生成された出力ファイルの名前です。

pdfmapdir
PDF マップ・ファイルを含むディレクトリー。このパラメーターを指定しない場合、コンパイラーは PDFDIR 環境変数の値をディレクトリーの名前として使用します。

-xml
PDF 情報の表示フォーマットを決定します。このパラメーターを指定した場合、PDF 情報は XML フォーマットで表示されます。指定しない場合は、テキスト・フォーマットで表示されます。値のプロファイルおよびキャッシュ・ミスのプロファイル情報は、XML フォーマットでしか表示できないため、XML フォーマットの PDF レポートは、テキスト・フォーマットのレポートよりも情報量が多くなります。

使用法

静的情報を格納する PDF マップ・ファイルは、PDF1 ステップで生成されます。これに対して PDF ファイルは、結果として生成されたアプリケーションの実行中に生成されます。**showpdf** コマンドを使用すると、アプリケーションから収集された以下のタイプのプロファイル情報を表示することができます。

- ブロック・カウンター・プロファイル
- 呼び出しカウンター・プロファイル
- 値プロファイル
- キャッシュ・ミスのプロファイル (PDF1 ステップで **-qpdf1=level=2** オプションを指定した場合)。

showpdf コマンドは、バイナリー・フォーマットの PDF ファイルのみを受け入れます。PDF 情報を表示するには、PDF ファイルと PDF マップ・ファイルの両方が必要です。最初の 2 つのタイプのプロファイル情報は、テキスト・フォーマットまたは XML フォーマットのいずれかで表示できます。しかし、値のプロファイルとキャッシュ・ミスのプロファイル情報は、XML フォーマットでしか表示できません。

PDF1 ステップの後、結果のアプリケーションの実行前に PDFDIR 環境変数を変更した場合、PDF ファイルと PDF マップ・ファイルが別々のディレクトリー内に生成されます。この場合は、この両方のファイルのディレクトリーを **showpdf** コマンドに指定する必要があります。

showpdf は /opt/ibm/xlf/16.1.0/bin/ にあります。

例

以下の例では、**showpdf** コマンドを使用して、「Hello World」アプリケーションのプロファイル情報を表示する方法を示しています。

プログラム・ファイル hello.f のソースは次のとおりです。

```
PROGRAM P
  CALL HelloWorld()

  CONTAINS

    SUBROUTINE HelloWorld()
    PRINT *, "Hello World"
    END SUBROUTINE HelloWorld

END PROGRAM P
END
```

1. ソース・ファイルをコンパイルします。

   ```
   xlf2008 -qpdf1 -O hello.f
   ```

2. 標準的なデータ・セットを 1 つ以上使用して、結果の実行可能プログラムを実行します。
3. 実行可能ファイルのプロファイル情報をテキスト・フォーマットで表示する場合は、**showpdf** コマンドをパラメーターを指定せずに実行します。

   ```
   showpdf
   ```

   結果は、次のようになります。

   ```
   ...
   ----------------------------------
   p(63): 1 (hello.f)

   Call Counters:
   2 | 1 @2@helloworld(64)
   2 | 1 _xlfExit(65)

   Call coverage = 100% ( 2/2 )

   Block Counters:
   1-10 | 1
   10 |

   Block coverage = 100% ( 1/1 )


   ----------------------------------
   @2@helloworld(64): 1 (hello.f)

   Call Counters:
   7 | 1 _xlfBeginIO(66)
   7 | 1 _xlfWriteLDChar(67)
   7 | 1 _xlfEndIO(68)

   Call coverage = 100% ( 3/3 )

   Block Counters:
   6-7 | 1
   8 |
   8 | 1
   ```

```
Block coverage = 100% ( 2/2 )


-----------------------------------
_xlfExit(65): 1 undefined node

-----------------------------------
_xlfBeginIO(66): 1 undefined node

-----------------------------------
_xlfWriteLDChar(67): 1 undefined node

-----------------------------------
_xlfEndIO(68): 1 undefined node

Total Call coverage = 100% ( 5/5 )


Total Block coverage = 100% ( 3/3 )
```

プロファイル情報を XML フォーマットで表示する場合は、**showpdf** コマンドを **-xml** パラメーターを指定して実行します。

```
showpdf -xml
```

結果は、次のようになります。

```
<?xml version="1.0" encoding="UTF-8" ?>
- <XLTransformationReport xmlns="http://www.ibm.com/2010/04/CompilerTransformation" version="1.0">
  - <CompilationStep name="showpdf">
      <StepDetails>
      ...
        <Detail>
          <FieldTitle>Total Call coverage</FieldTitle>
          <FieldValue>100% ( 5/5 )</FieldValue>
        </Detail>
        <Detail>
          <FieldTitle>Total Block coverage</FieldTitle>
          <FieldValue>100% ( 3/3 )</FieldValue>
        </Detail>
      </StepDetails>
     <ProgramHierarchy>
       <FileList>
         <File id="1" name="hello.f">
       <RegionList>
         <Region id="63" name="p" startLineNumber="1"/>
         <Region id="64" name="@2@helloworld" startLineNumber="6"/>
       </RegionList>
         </File>
       </FileList>
     </ProgramHierarchy>
     <TransformationHierarchy/>
       <ProfilingReports>
         <BlockCounterList>
           <BlockCounter regionId="63" execCount="1" coveredBlock="1" totalBlock="1">
             <BlockList>
               <Block index="3" execCount="1" startLineNumber="1" endLineNumber="10"/>
             </BlockList>
           </BlockCounter>
           <BlockCounter regionId="64" execCount="1" coveredBlock="2" totalBlock="2">
             <BlockList>
               <Block index="3" execCount="1" startLineNumber="6" endLineNumber="7"/>
               <Block index="4" execCount="1" startLineNumber="8" endLineNumber="8"/>
          </BlockList>
         </BlockCounter>
       </BlockCounterList>
       <CallCounterList>
         <CallCounter regionId="63" execCount="1" coveredCall="2" totalCall="2">
           <CallList>
             <Call name="@2@helloworld" execCount="1" lineNumber="2"/>
             <Call name="_xlfExit" execCount="1" lineNumber="2"/>
           </CallList>
         </CallCounter>
         <CallCounter regionId="64" execCount="1" coveredCall="3" totalCall="3">
           <CallList>
             <Call name="_xlfBeginIO" execCount="1" lineNumber="7"/>
             <Call name="_xlfWriteLDChar" execCount="1" lineNumber="7"/>
```

```
            <Call name="_xlfEndIO" execCount="1" lineNumber="7"/>
          </CallList>
        </CallCounter>
      </CallCounterList>
    </ProfilingReports>
  </CompilationStep>
</XLTransformationReport>
```

関連情報

- 228 ページの『-qpdf1、-qpdf2』
- 256 ページの『-qshowpdf』
- 「*XL Fortran* 最適化およびプログラミング・ガイド」の『プロファイル指示フィードバック』

第 8 章 64 ビット環境での XL Fortran の使用

64 ビット環境に関しては、さらに大きなストレージ要件と処理能力を求める需要がますます高まりつつあります。 Linux オペレーティング・システムでは、64 ビット・アドレス・スペースを使用することで、64 ビット・プロセッサーを活用するプログラムを開発かつ実行できる環境を提供しています。

第 9 章 コンパイラー・ライセンス使用状況の追跡

IBM Software License Metric (SLM) タグ・ロギングを有効にすれば、コンパイラー・ライセンス使用状況をトラッキングできます。この情報は、組織のコンパイラーの使用がコンパイラー・ライセンス資格の数を超えているかどうかを判別するのに役立ちます。

コンパイラー・ライセンス追跡の理解

IBM License Metric Tool (ILMT) がコンパイラー・ライセンス使用状況をトラッキングできるように、コンパイラーで IBM Software License Metric (SLM) タグ・ロギングを有効にすることができます。

記録されるコンパイラー・ライセンスのタイプ

コンパイラーは、以下の 2 つのタイプのコンパイラー・ライセンスの使用状況を記録します。

- 許可ユーザー・ライセンス: 各コンパイラー・ライセンスは特定のユーザー ID (そのユーザーの uid によって指定される) に結び付けられます。
- 同時ユーザー・ライセンス: 特定の数の同時ユーザーがいつでもコンパイラー・ライセンスの使用を許可されます。

コンパイラーは呼び出されるたびに、許可ユーザーがリストされているファイルに、コンパイラーを呼び出したユーザーの uid が存在するかどうかに応じて、その呼び出し内容を同時ユーザーによる呼び出しとして記録したり許可ユーザーによる呼び出しとして記録したりします。

SLM デーモン・プロセス

SLM タグ・ファイルは SLM デーモンによって作成されます。 SLM デーモンは、コンパイラーにおける正規の実行可能プログラムです。すべてのユーザー定義期間についてタグ・ファイルを出力します。ユーザー定義期間内にユーザーがコンパイラーを呼び出さないとデーモンは終了し、コンパイラーを新たに呼び出すと再始動します。両方の期間を構成ファイルで定義できます。

SLM タグ・ファイル

コンパイラーは、コンパイラー・ライセンス使用状況を SLM タグ・ファイル *slm_dir*/*hash*.slmtags に記録します。 *slm_dir* は構成可能ディレクトリー、*hash* は現行製品の SWID およびルート・ディレクトリーの MD5 ハッシュです。 *slm_dir* のデフォルト値は /var/opt/ibm/xl-compiler/ (デフォルト・インストールの場合) または *$prefix*/var/opt/ibm/xl-compiler/ (デフォルト以外のインストールの場合) です。 *$prefix* はデフォルト以外のインストール・パスです。 *slm_dir* ディレクトリーは、コンパイラーを呼び出すすべてのユーザーが読み取り可能および書き込み可能でなければなりません。

タグ・ファイルのメイン・エレメントは以下のとおりです。

<StartTime> および **<EndTime>**
呼び出しの開始時刻と終了時刻。

<Value>
<StartTime> エレメントと **<EndTime>** エレメントで示された期間内に同時にコンパイラーを呼び出すユーザーの最大数。

<Type>
記録されるコンパイラー・ライセンスのタイプ。

<Metric>
先行エレメントはすべて **<Metric>** エレメントに含まれます。 SLM デーモンは、すべての期間ごとに最大 2 つの **<Metric>** エレメントを出力します。 1 つは許可ユーザー用、もう 1 つは同時ユーザー用です。

次の例は、コンパイラーによって生成されたタグ・ファイルを示しています。これには、3 つの同時ユーザー呼び出しに続いて 1 つの許可ユーザー呼び出しが記録されています。

```
<SchemaVersion>2.1.1</SchemaVersion>
 <SoftwareIdentity>
         <PersistentId>0bce7313f2a24da7b1b27f33294ffe70</PersistentId>
         <Name>IBM XL
Fortran for Linux</Name>
         <InstanceId>/opt/ibm/xlC/16.1.0</InstanceId>
 </SoftwareIdentity>
<Metric logTime="2016-04-01T18:39:51Z">
         <Type>AUTHORIZED_USER</Type>
         <Value>1</Value>
         <Period>
                 <StartTime>2016-04-01T18:34:51Z</StartTime>
                 <EndTime>2016-04-01T18:39:51Z</EndTime>
         </Period>
</Metric>
<Metric logTime="2016-04-01T18:39:51Z">
         <Type>CONCURRENT_USER</Type>
         <Value>3</Value>
         <Period>
                 <StartTime>2016-04-01T18:14:51Z</StartTime>
                 <EndTime>2016-04-01T18:19:51Z</EndTime>
         </Period>
</Metric>
```

タグ・ファイルには構成可能なサイズ制限があり、デフォルトは 5,000,000 バイトです。現行ファイルのサイズが最大サイズを超えると、ファイル内の古い **<Metric>** エレメントが削除されます。

エラー・ログ

デーモンが fork する前にエラーが発生した場合は、戻りコード -1 が返されます。それ以外の場合、障害は報告されません。どちらの場合も、エラーは *slm_dir*/.hash/log に記録されます。

関連資料:

-qslmtags

関連情報:

ファイル属性の構成

SLM タグ・ロギングのセットアップ

ILMT を有効にしてコンパイラー使用状況を追跡するには、SLM タグ・ロギングをセットアップする必要があります。

このタスクについて

自分のコンパイラー・ライセンスが許可ユーザー・ライセンスの場合は、以下の手順を使用して XL コンパイラー SLM タグ・ロギングをセットアップします。

手順

1. 許可ユーザーのユーザー ID を判別します。
2. `XLAuthorizedUsers` という名前でファイルを `/etc` ディレクトリーに作成します。 `XLAuthorizedUsers` ファイルの場所は、構成ファイルの **slm_auth** 属性を指定することによって変更できます。このファイルには、許可ユーザーに関する情報が、ユーザーごとに 1 行ずつ含まれます。 各行には、許可ユーザーの数値 uid、コンマ、許可製品のソフトウェア ID (SWID) がその順序で含まれます。

   ユーザー ID の uid を取得するには、**id -u username** コマンドを使用します (*username* の部分を検索対象のユーザー ID で置き換えてください)。

   製品の SWID は、以下のコマンドを実行することによって確認できます。

   ```
   grep persistentId /opt/ibm/xlf/V.R.M/swidtag/*.swidtag
   ```

   V.R.M は、システムにインストールされているコンパイラーの Version.Release.Modification レベルです。
3. コンパイラーを呼び出すすべてのユーザーが `/etc/XLAuthorizedUsers` を読み取ることができるように設定します。

   ```
   chmod a+r /etc/XLAuthorizedUsers
   ```

タスクの結果

ユーザーの uid が `/etc/XLAuthorizedUsers` にリストされている場合は、コンパイラーは、許可ユーザーによる呼び出しと、使用されたコンパイラーの SWID を記録します。それ以外の場合、コンパイラーは同時ユーザーによる呼び出しを記録します。

XL コンパイラー SLM タグ・ロギングではライセンスの準拠は行われないことに注意してください。収集されたデータと IBM License Metric Tool を使用して、コンパイラーがコンパイラー・ライセンス条件内で使用されているかどうかを判別できるように、単にコンパイラーの呼び出しが記録されます。

例

3 人の許可ユーザーが存在し、それぞれの ID が bsmith、rsingh、および jchen であるとします。これらのユーザー ID について、コマンド・シェルで以下のコマンドを入力して、対応する出力を確認します。

```
$id -u bsmith
24461
$id -u rsingh
9204
$id -u jchen
7531
```

次のコマンドを実行して、SWID を入手します。

```
$ grep persistentId /opt/ibm/xlf/16.1.0/swidtag/*.swidtag
<Meta persistentId="0bce7313f2a24da7b1b27f33294ffe70"/>
```

次に、これらのユーザーにコンパイラーの使用を許可するために、以下の行を含む /etc/XLAuthorizedUsers を作成します。

```
24461,0bce7313f2a24da7b1b27f33294ffe70
9204,0bce7313f2a24da7b1b27f33294ffe70
7531,0bce7313f2a24da7b1b27f33294ffe70
```

関連資料:

-qslmtags

関連情報:

 IBM License Metric Tool

構成ファイルの属性

第 10 章 問題判別とデバッグ

このセクションでは、プログラムのコンパイルや実行で生じる問題を見つけて修正するために使用できる方法をいくつか説明します。

XL Fortran エラー・メッセージに関する情報

潜在的な問題や実際に発生する問題に関するほとんどの情報は、コンパイラーまたはアプリケーション・プログラムからのメッセージによって提示されます。これらのメッセージは、標準エラー・ストリームに書き込まれます。

エラーの重大度

コンパイル・エラーには、次のような重大度レベルがあります (重大度の高い方から低い方に示されています)。

U 回復不能エラー。内部のコンパイル時エラーが原因でコンパイルが失敗しました。

S 重大エラー。コンパイルが以下のいずれかの理由で失敗しました。
- 回復不能プログラム・エラーが検出されました。ソース・ファイルの処理は停止して、XL Fortran はオブジェクト・ファイルを生成しません。通常このエラーは、コンパイル中に報告されたプログラム・エラーを修正することによって、解決することができます。
- コンパイラーが修正できなかった条件が存在します。オブジェクト・ファイルは作成されますが、プログラムの実行を試行しないでください。
- 内部コンパイラー・テーブルがオーバーフローしました。プログラムの処理は停止して、 XL Fortran はオブジェクト・ファイルを生成しません。
- インクルード・ファイルが存在しません。プログラムの処理は停止して、 XL Fortran はオブジェクト・ファイルを生成しません。

E コンパイラーが修正できるエラー。プログラムは正しく動作します。

W 警告メッセージ。エラーを意味しているのではありませんが、何らかの予期しない状況を示している場合があります。

L さまざまな言語レベルに従っているかどうかをチェックするコンパイラー・オプションの 1 つによって生成される警告メッセージです。移植性を保持したい場合に回避しなければならない言語機能を示している場合があります。

I 通知メッセージ。エラーではなく、予期しない動作を回避するため、またはパフォーマンスを改善するために気を付けなければならないことを示しています。

注:
- メッセージ・レベル **S** と **U** は、コンパイルの失敗を示しています。
- メッセージ・レベル **I**、**L**、**W**、および **E** は、コンパイルの失敗を示していません。

デフォルト時には、重大エラー (重大度 S) を検出すると、コンパイラーは出力ファイルを作成しないで停止します。しかし、以下のオプションを使用して、メッセージの生成方法を制御できます。

- **-qhalt** オプションを使用して別の重大度を指定すれば、重大度のより低いエラーに対して、コンパイラーを停止させることができます。例えば、**-qhalt=e** を使用した場合、重大度 E またはそれ以上の重大度のエラーを検出するとコンパイラーが停止します。この手法を使用すると、プログラムの構文およびセマンティクスの妥当性をチェックするのに必要なコンパイル時間を短縮することができます。
- **-qflag** オプションを使用すると、コンパイラーを停止させることなく重大度が低いメッセージを制限することができます。
- **-qhaltonmsg** オプションを使用して、指定したエラー・メッセージが生成されたときにコンパイルを停止できます。
- **-qmaxerr** オプションを使用して、指定した最小重大度レベル以上のエラーが指定した数に達した場合に、コンパイルを停止できます。
- 特定のメッセージが出力ストリームに出力されないようにしたいだけであれば、 **-qsuppress** を参照してください。

コンパイラー戻りコード

コンパイラーの戻りコード、および対応する意味は以下のとおりです。

0 コンパイラーは、コンパイル単位の処理を停止させなければならないような重大なエラーを検出しませんでした。

1 コンパイラーがコンパイルを停止するのに十分な重大度のエラーを検出しました。*halt_severity* のレベルに従って、コンパイラーはエラーを出してコンパイル単位の処理を続行させることができます。

40 オプション・エラー。

41 構成ファイル・エラー。

250 メモリー不足エラー。コンパイラーは、使用するメモリーをこれ以上割り振ることができません。

251 シグナル受信エラー。回復不能エラーまたは割り込みシグナルが受信されました。

252 ファイルが存在しないエラー。

253 入出力エラー。ファイルの読み取りまたは書き込みができません。

254 fork エラー。新しいプロセスを作成できません。

255 プロセス実行中のエラー。

実行時戻りコード

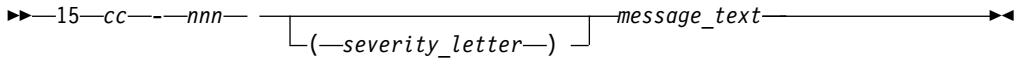
XLF コンパイル・プログラムが異常終了した場合は、オペレーティング・システムへの戻りコードは 1 です。

プログラムが正常終了した場合は、戻りコードは 0 (デフォルト) で、**STOP** *digit_string* 文の原因によってプログラムが終了した場合は、**MOD**(*digit_string*,256) です。

XL Fortran 診断メッセージの形式

-qsource コンパイラー・オプションを指定すると、診断メッセージが表示されるだけでなく、ソース行と、エラーが検出されたソース行内の位置を指し示すポインターが印刷または表示されます。 **-qnosource** が有効な場合は、メッセージとともに、エラーのファイル名、行番号、桁位置が表示されます。

XL Fortran 診断メッセージの形式は次のとおりです。



各項目の説明:

15 XL Fortran メッセージを示します。

cc 次のようなコンポーネント番号です。

00 コード作成または最適化メッセージを示します。

01 XL Fortran 共通メッセージを示します。

11-20 Fortran 特定のメッセージを示します。

25 XL Fortran アプリケーション・プログラムからの実行時メッセージを示します。

85 ループ変換メッセージを示します。

86 プロシージャー間分析 (IPA) メッセージを示します。

87 SMP ライブラリーからの実行時メッセージを示します。

nnn メッセージ番号です。

severity_letter
前の項で説明したように、問題の重大度を示します。

'*message text*'
エラーを説明するテキストです。

コンパイル時メッセージ数の制限

ユーザーが既に気付いていたり、気にかけていない問題に関する多数の低レベルのメッセージ (**I** または **W**) をコンパイラーが出す場合は、**-qflag** オプションまたは、その短形式の **-w** を使用して、メッセージを高レベルのものに限定してください。

```
# E, S, and U messages go in listing; U messages are displayed on screen.
xlf95 -qflag=e:u program.f

# E, S, and U messages go in listing and are displayed on screen.

xlf95 -w program.f
```

インストールまたはシステム環境の問題の修正

特定マシンの各ユーザーまたは全ユーザーがコンパイラーの実行時に困難を経験する場合は、システム環境に問題があると思われます。よく発生する問題と解決策を以下にいくつか示します。

I/O エラーでコンパイルが失敗します。

システム処置:
入出力エラーでコンパイルが失敗します。

ユーザー応答:
/tmp ファイル・システムのサイズを小さくするか、あるいは、環境変数 **TMPDIR** をフリー・スペースがよりたくさんあるファイル・システムのパスに設定してください。

説明: オブジェクト・ファイルが、ファイル・システムを保持できないほど大きくなりすぎた可能性があります。原因は、コンパイル単位が非常に大きいか、または宣言内の大きな配列の全部または一部の初期化にある可能性があります。

Could not load program *program*

メッセージ:

```
Could not load program program
Error was: not enough space
```

システム処置:
システムがコンパイラーまたはアプリケーション・プログラムをまったく実行できません。

ユーザー応答:

この問題を経験したユーザーは、スタックおよびデータ用のストレージ限界を、『unlimited』 に設定してください。例えば、ハード・リミットもソフト・リミットもこれらの **bash** コマンドで設定することができます。

```
ulimit -s unlimited
ulimit -d unlimited
```

または、ファイル /etc/security/limits.conf を編集して、すべてのユーザーに無制限のスタック・セグメントとデータ・セグメントを (これらのフィールドに -1 を入力することによって) 与えると便利です。

ストレージの問題が XLF コンパイル済みプログラムにある場合は、**-qsave** または **-qsmallstack** オプションを使用すれば、プログラムがスタック限界を超えないようにすることができます。

説明: コンパイラーは、ストレージの限界を超える場合のある大きな内部データ域をユーザー用に割り振ります。 XLF コンパイル済みプログラムは、デフォルト時には旧バージョンよりも多くのデータをスタック上に置き、ストレージの限界を超えることもあります。必要な限界の正確な値を判別することはむずかしいので、無制限にすることをお勧めします。

Could not load library *library_name*

メッセージ:

```
Could not load library library_name.so
Error was: no such file or directory
```

ユーザー応答:

XL Fortran ライブラリーが /opt/ibm/xlf/16.1.0/libにインストールされていることを確認してください。また、別のディレクトリーにインストールされている場合、**libxlf90.so** がインストールされているディレクトリーを組み込むよう **LD_RUN_PATH** 環境変数を設定してください。この環境変数の詳細は、10 ページの『ライブラリー検索パスの設定』を参照してください。

内部コンパイラー・エラー

システム処置:

内部コンパイラー・エラーでコンパイルが失敗します。

ユーザー応答:

スタック・サイズを増やしてみてください。例えば、**ulimit bash** コマンドを使用してハード制限とソフト制限の両方を設定したり、ファイル /etc/security/limits.conf を編集してスタック・セグメントを変更したりします。スタック・サイズを増やした後でも依然としてエラーが発生する場合は、別の手段を講じる必要があります。

注: スタック・サイズの制限が原因で発生する内部コンパイラー・エラーはほんの一部に過ぎません。そのような内部コンパイラー・エラーが発生する可能性があるシナリオ例としては、大きな派生型が定義されているモジュール・シンボル・ファイルを読み取るプログラムをコンパイルするときがあります。

invocation_command: not found

システム処置:

シェルは、コンパイラーを実行するコマンドを見つけることができません。

ユーザー応答:

PATH 環境変数にディレクトリー /opt/ibm/xlf/16.1.0/bin が登録されていることを確認してください。コンパイラーが正しくインストールされていれば、そのコンパイラーを実行するのに必要なコマンドは、このディレクトリーに入っています。

個々の makefile とコンパイル・スクリプトの数が多すぎる場合

システム処置:

個別の makefiles およびコンパイル・スクリプトの数が多過ぎて、簡単に保持または追跡ができません。

ユーザー応答:

構成ファイルにスタンザを追加して、これらのスタンザの名前を使用してコンパイラーとのリンクを作成してください。別のコマンド名でコンパイラーを実行すれば、一貫性のある一連のコンパイラー・オプションやその他の構成の設定を多数のユーザーに提供することができます。

コンパイル時の問題の修正

以降の項では、コンパイル時に生じる可能性のある共通問題と、そのような問題を回避する方法を説明しています。

他のシステムからの拡張機能の再現

移植されたプログラムの中には、他のシステムにある拡張機能に依存しているために、コンパイルで問題が起きるものもあります。 XL Fortran はそのような拡張機能を多数サポートしていますが、その中の一部はコンパイラー・オプションでオンにする必要があります。対象オプションのリストについては 78 ページの『移植性とマイグレーション』を参照してください。移植に関する一般的な説明は、「*XL Fortran* 移行ガイド」の『*XL Fortran* へのプログラムの移植』を参照してください。

個々のコンパイル単位の問題の分離

コンパイルを正しく実行するために、特定のコンパイル単位が特定のオプションを設定する必要のある場合は、**@PROCESS** ディレクティブを利用してソース・ファイル内の設定を適用した方が便利なことがわかります。ファイルの配置によっては、この方法を使用した方が、さまざまなコマンド行オプションを使用してさまざまなファイルを再コンパイルするよりも簡単な場合があります。

スレッド・セーフ・コマンドによるコンパイル

例えば **xlf_r** や **xlf90_r** のようなスレッド・セーフ呼び出しコマンドは、スレッド・セーフ以外の呼び出しとは異なる検索パスを使用し、異なるモジュールを呼び出します。プログラムの異なる使用法については考慮が必要です。ある環境で正常にコンパイルおよび実行されるプログラムは、異なる使用環境のためにコンパイルおよび実行されると、予期しない結果をもたらす場合があります。構成ファイル **xlf.cfg** には、呼び出しコマンドのそれぞれについて、パス、ライブラリーなどが示されます。 (内容の説明については、 17 ページの『デフォルト構成ファイルの編集』を参照してください。)

マシン・リソースのこ渇

いずれかのコンパイラー・コンポーネントが動作している間に、オペレーティング・システムのリソース (ページ・スペースまたはディスク・スペース) 上での動作が低下すると、以下のメッセージのいずれかが表示されます。

```
1501-229 Compilation ended because of lack of space.

1517-011 Compilation ended.  No more system resources available.

1501-053 (S) Too much initialized data.
1501-511. Compilation failed for file [filename].
```

システムのページ・スペースを増やしてプログラムを再コンパイルする必要がある場合があります。 ページ・スペースについて詳しくは、man ページ情報 **man 8 mkswap swapon** を参照してください。

例えば、大きな配列の全部または一部を初期化することによって、プログラムが大きなオブジェクト・ファイルを作成すると、以下のいずれかを行う必要がある場合があります。

- `/tmp` ディレクトリーを保持しているファイル・システムのサイズを大きくする。
- **TMPDIR** 環境変数を多数のフリー・スペースを持つファイル・システムに設定する。
- 非常に大きな配列では、静的に (コンパイル時に) ではなく実行時に配列を初期化します。

内部別名テーブル・オーバーフローを原因とするコンパイル失敗

最適化レベル **-O2** 以上では、XL Fortran コンパイラーは、別名情報を計算して、最適化プログラムがアプリケーションのセマンティクスを保持していることを確認します。アプリケーション内で使用されるオブジェクト定義および言語構文によっては、別名計算プロセスでは、多くのスペースが使用されることがあります。特殊な事例として、内部別名テーブルがオーバーフローして、コンパイルが失敗することがあります。このようなオーバーフローは、モジュールが定義または使用する派生型オブジェクトに多数の最終コンポーネントが含まれていて、同じモジュール内で定義されている多数のプロシージャーにそれらのオブジェクトを仮引数として渡す場合に発生することがあります。モジュール・プロシージャーのセットを分割し、各区画をそれぞれ別個のモジュール内に配置すると、オーバーフロー問題が解決することがあります。

リンク時の問題の修正

XL Fortran コンパイラーがソース・ファイルを処理した後、リンカーはその結果作成されたオブジェクト・ファイルをリンクします。この段階で出されるメッセージは、 **ld** コマンドからのものです。読者の便宜のために、頻繁に検出されるメッセージ、およびその解決方法を以下に示します。

未定義または未解決のシンボルが検出された場合

メッセージ**:**

filename.o(.text+0x14): "p"への未定義の参照
filename.o(.text+0x14): R_PPC_REL24 p に適合させるため、
再配置は切り捨てられました

システム処置**:**

未解決参照が原因で、プログラムをリンクできません。

説明**:** 必要なオブジェクト・ファイルまたはライブラリーがリンク中に使用されていないか、あるいは、1 つ以上の外部名の指定にエラーがあるか、1 つ以上のプロシージャー・インターフェースの指定にエラーがあります。

ユーザー応答**:**

以下の処置のうちの 1 つ以上を実行する必要がある場合があります。

- **-Wl** オプションまたは **-M** オプションを指定して再コンパイルを行い、未定義のシンボルに関する情報が含まれるファイルを作成します。
- **-U** オプションを使用する場合は、組み込み名がすべて小文字になっているかどうかを確認してください。

実行時の問題の修正

以下のいずれかの場合に、XL Fortran はプログラムの実行中にエラー・メッセージを出します。

- XL Fortran が入出力エラーを検出した場合。この種のメッセージの制御方法は、 45 ページの『実行時オプションの設定』で説明されています。
- XL Fortran が例外エラーを検出して、デフォルトの例外ハンドラーがインストールされている (**-qsigtrap** オプションまたは **SIGNAL** への呼び出しによる) 場合。 `Core dumped` よりも説明的なメッセージを表示するには、**gdb** 内部からプログラムを実行しなければならない場合があります。

  実行時例外の原因は、 58 ページの『XL Fortran 実行時例外』に列挙されています。

プログラムの実行中に起きるエラーは、**gdb** などのシンボリック・デバッガーを使用して調べることができます。

他のシステムからの拡張機能の再現

移植されたプログラムが他のシステムにある拡張機能に依存している場合、これらの中には正しく実行されないものがあります。 XL Fortran はそのような拡張機能を多数サポートしていますが、それらのいくつかを使用するにはコンパイラー・オプションをオンにしなければなりません。対象オプションのリストについては 78 ページの『移植性とマイグレーション』を参照してください。移植に関する一般的な説明は、「*XL Fortran* 移行ガイド」の『*XL Fortran* へのプログラムの移植』を参照してください。

引数のサイズまたは型の不一致

サイズまたは型が異なる引数によって、不正な実行および結果が発生する可能性があります。コンパイルの初期段階で型のチェックを行うには、プログラム内で呼び出されるプロシージャーに対してインターフェース・ブロックを指定してください。

最適化するときの問題の回避策

最適化すると、プログラムが誤った結果を発生させることがわかっていて、問題を特定の変数に限定できる場合は、その変数を **VOLATILE** と宣言することによって、問題を一時的に回避することができます。このようにすると、最適化のうち、その変数に影響を与えるものは行われなくなります。 (「*XL Fortran* ランゲージ・リファレンス」の『**VOLATILE**』 を参照してください。) これは一時的な解決策に過ぎないので、問題を解決するまでコードのデバッグを続行して、その後に **VOLATILE** キーワードを除去してください。ソース・コードとプログラム設計が正しいと確信していて、問題が続行する場合は、問題を解決するためにユーザーのサポート部門に連絡を取ってください。

入出力エラー

検出されたエラーが入出力エラーの場合に、エラーになっている入出力ステートメントで **IOSTAT** を指定したのであれば、「*XL Fortran* ランゲージ・リファレンス」の『条件および *IOSTAT* 値』の説明どおりに値が **IOSTAT** 変数に割り当てられます。

プログラムが実行されているシステム上に XL Fortran 実行時メッセージ・カタログをインストールした場合は、ある一定の I/O エラーに対して、メッセージ番号とメッセージ・テキストが端末 (標準エラー) に送出されます。入出力ステートメントで **IOMSG** を指定した場合は、エラーが検出されると、エラー・メッセージ・テキストが **IOMSG** 変数に割り当てられます。エラーが検出されない場合、**IOMSG** 変数の内容は変更されません。このカタログがシステムにインストールされていないと、メッセージ番号だけが表示されます。 45 ページの『実行時オプションの設定』の記載されているいくつかの設定を使用すれば、これらのエラー・メッセージをオンまたはオフにすることができます。

大きなデータ・ファイルの書き込み中にプログラムで障害が発生する場合は、ユーザー ID の最大ファイル・サイズ限界を大きくする必要がある場合もあります。これは、**bash**を使用して行うことができます。

トレースバックとメモリー・ダンプ

実行時例外の発生前に適切な例外ハンドラーをインストールしている場合、発生時にメッセージとトレースバック・リストが表示されます。ハンドラーによっては、コア・ファイルが作成されることがあります。その後、デバッガーを使用して例外の位置を調べることができます。

プログラムを終了させずにトレースバック・リストを作成する場合は、**xl__trbk** プロシージャーを呼び出してください。

```
IF (X .GT. Y) THEN    ! X > Y indicates that something is wrong.
  PRINT *, 'Error - X should not be greater than Y'
  CALL XL__TRBK       ! Generate a traceback listing.
END IF
```

例外ハンドラーについては、「*XL Fortran* 最適化およびプログラミング・ガイド」の『例外ハンドラーのインストール』 を参照してください。実行時例外の原因については、 58 ページの『XL Fortran 実行時例外』を参照してください。

Fortran プログラムのデバッグ

ご使用のプログラムをデバッグするには、**gdb** およびその他のシンボリック・デバッガーを使用できます。選択したデバッガーを使用するために指示については、そのデバッガーのオンライン・ヘルプまたはその資料を参照してください。

デバッグ用にプログラムをコンパイルする場合は、常に **-g** オプションを指定してください。

注**:** このリリースの XL Fortran では、Fortran 2003 のポリモアフィック・オブジェクトおよびパラメーター化された派生型のデバッグがサポートされていません。

関連情報**:**

- 69 ページの『エラー・チェックおよびデバッグ』

第 11 章 XL Fortran コンパイラー・リストについて

診断情報は、コンパイラー・オプション **-qlist**、**-qsource**、**-qxref**、**-qattr**、**-qreport**、**-qlistopt** によって作成される出力リストに置かれます。 **-S** オプションは、別個のファイルにアセンブラー・リストを作成します。

リストを利用して問題の原因を特定するためには、以下の部分を参照できます。

- ソース・セクション (ソース・プログラムのコンテキスト内のコンパイル・エラーを見つけるため)
- 属性および相互参照セクション (名前の誤ったデータ・オブジェクト、宣言なしで使用されているデータ・オブジェクト、または一致していないパラメーターを見つけるため)
- 変換およびオブジェクト・セクション (生成されたコードが予期したとおりのものかどうかを見るため)

リストの主要なセクションは、見出しによって識別されます。不等号 (より大記号) のストリングがセクション見出しの前に付き、見出しの先頭を簡単に見つけることができます。

```
>>>>> SECTION NAME <<<<<<
```

コンパイラー・オプションを指定して、リストに現れるセクションを選択できます。

関連情報

- 72 ページの『リスト、メッセージ、およびコンパイラー情報』

ヘッダー・セクション

リスト・ファイルには、以下の項目を含んでいるヘッダー・セクションがあります。

- 以下の要素で構成されるコンパイラー識別子
  - コンパイラー名
  - バージョン番号
  - リリース番号
  - 変更番号
  - 修正番号
- ソース・ファイル名
- コンパイル日付
- コンパイル時刻

ヘッダー・セクションは、リストに必ず存在します。それは最初の行であり、一度だけ出現します。複数のコンパイル単位が存在する場合、次のセクションは、個々のコンパイル単位に対して繰り返されます。

オプション・セクション

オプション・セクションは、リストに必ず存在します。コンパイル単位ごとに別個のセクションがあります。このセクションは、コンパイル単位に対して指定されている有効なオプションを示します。この情報は、矛盾するオプションが指定されている場合に役立ちます。 **-qlistopt** コンパイラー・オプションを指定すると、このセクションは、すべてのオプションの設定をリストします。

ソース・セクション

ソース・セクションには、行番号とファイル番号 (任意) の付いた入力ソース行が含まれています。ファイル番号は、ソース行が取り出されたソース・ファイル (またはインクルード・ファイル) を示します。 メイン・ファイルのすべてのソース行 (インクルード・ファイルからのソース行ではない) には、ファイル番号は印刷されません。個々のインクルード・ファイルにはファイル番号が関連づけられており、インクルード・ファイルからのソース行には、そのファイル番号が印刷されます。左から、ファイル番号、行番号、ソース行のテキストの順で印刷されます。 XL Fortran は、個々のファイルに相対的な行の番号を付けます。それらと関連づけられているソース行とソース番号は、 **-qsource** コンパイラー・オプションが有効な場合だけ印刷されます。プログラムを使用して、**@PROCESS** ディレクティブの **SOURCE** と **NOSOURCE** を使用することにより、ソースの一部を選択的に印刷することができます。

エラー・メッセージ

-qsource が有効な場合は、ソース・リスト中にエラー・メッセージが混在します。コンパイル・プロセスで生成されるエラー・メッセージには、次のエレメントが含まれています。

- ソース行
- エラーの桁を指し示す標識の行
- エラー・メッセージ。これは、以下のエレメントで構成されます。
  - 4 桁のコンポーネント番号
  - エラー・メッセージ番号
  - メッセージの重大度レベル
  - エラーを説明するテキスト

例を以下に示します。

```
        2 |         equivalence (i,j,i,j)
     ..............................a.b.
a - "t.f", line 2.24: 1514-117 (E) Same name appears more than once in an equivalence group.
b - "t.f", line 2.26: 1514-117 (E) Same name appears more than once in an equivalence group.
```

-qnosource オプションが有効な場合は、ソース・セクションに表示されるものすべてはエラー・メッセージで、エラー・メッセージには次のものが含まれます。

- 引用符で囲まれたファイル名
- エラーの行番号と桁位置
- エラー・メッセージ。これは、以下のエレメントで構成されます。
  - 4 桁のコンポーネント番号
  - エラー・メッセージ番号
  - メッセージの重大度レベル
  - エラーを説明するテキスト

例えば、次のように指定します。

```
"doc.f", line 6.11: 1513-039 (S) Number of arguments is not
permitted for INTRINSIC function abs.
```

PDF レポート・セクション

リスト・レポートに、プログラムの一部の局面を分析するのに役立つ以下のセクションが追加されました。**-qreport** オプションを **-qpdf2** オプションとともに使用すると、リスト・レポートの PDF Report というタイトルのセクションに以下のセクションが追加されます。

ループの繰り返しカウント
: 特定の入力データ・セットで最も多発したループ反復カウントと平均反復カウントが、プログラム内のほとんどのループについて計算されます。この情報は、プログラムが最適化レベル -O5 でコンパイルされる場合にのみ得られます。

ブロックおよび呼び出しカウント
: このセクションは、プログラムの呼び出し構造 と、呼び出された各関数の実行カウントを対象としています。また、関数ごとのブロック情報 も示されます。非ユーザー定義関数については、実行カウントのみが示されます。このレポート・セクションの最後に、Total Block and Call Coverage、および実行カウントの降順に並べられたユーザー関数のリストが印刷されます。さらに、リスト・ファイル内の疑似コードの各ブロックでは、冒頭にブロック・カウント情報が出力されます。

キャッシュ・ミス
: このセクションは、単一テーブルに出力されます。ここでは、特定の関数のキャッシュ・ミス の数が、関数に関する追加情報 (Cache Level、Cache Miss Ratio、Line Number、File Name、および Memory Reference など) とともに報告されます。

  注**:** このレポートを作成するには、**-qpdf1=level=2** オプションを使用する必要があります。
  実行時に PDF_PM_EVENT 環境変数を使用して、プロファイルを作成するキャッシュのレベルを選択することもできます。

プロファイル・データの関連度 **(Relevance of profiling data)**
: このセクションには、PDF1 ステップ中のソース・コードとプロファイル・データの間の関連度が含まれています。関連度は、0 から 100 の範囲の数値で示されます。値が大きいほどプロファイル・データとソース・コードの関連性が高く、プロファイル・データの使用によるパフォーマンスの向上幅も大きくなります。

欠落しているプロファイル・データ **(Missing profiling data)**
: このセクションには、欠落しているプロファイル・データについての警告メッセージが出力されます。この警告メッセージは、コンパイラーがプロファイル・データを検出しなかった関数それぞれについて出されます。

古いプロファイル・データ **(Outdated profiling data)**
: このセクションに、古いプロファイル・データについての警告メッセージが出力される場合があります。コンパイラーは、PDF1 ステップの後に変更さ

れた関数それぞれについてこの警告メッセージを出します。警告メッセージは、PDF1 ステップと PDF2 ステップとの間で最適化レベルが異なる場合にも出されます。

Profile-Directed Feedback について詳しくは、*XL Fortran* 最適化およびプログラミング・ガイドの『Profile-Directed Feedback』を参照してください。

リスト・ファイルに関する追加情報については、*XL Fortran* コンパイラー・リファレンスの『XL Fortran コンパイラー・リストについて』を参照してください。

変換レポート・セクション

-qreport オプションが有効である場合は、変換報告書リストには IBM XL Fortran for Linux, V16.1 がプログラムを最適化した方法が示されます。この `LOOP TRANSFORMATION` のセクションでは、元のソース・コードに対応する疑似 Fortran コードを表示し、**-qhot** または **-qsmp** オプションが生成した並列化およびループ変換がわかるようにします。レポートのこのセクションには、**-qsmp** および **-qhot=level=2** でコンパイルした場合に、ループ・ネストで実行された追加の変換と並列化に関する情報も示されます。

またコンパイラーは、指定されたループ用に作成されたストリームの数も報告します。この情報を使用して、アプリケーション・コードを理解し、プログラム・コードのパフォーマンス・チューニングを行うことができます。例えば、基礎のアーキテクチャーがサポートする数より多いストリームを含むループを配布することができます。

コンパイラーがデータ・プリフェッチ命令を挿入した位置に関する情報を生成するには、最適化レベル **-qhot**、**-O3**、**-O4**、または **-O5** を、**-qreport** とともに使用します。

サンプル報告書

次の報告書は、プログラム **t.f** について

```
xlf -qhot -qreport t.f
```

コマンドで作成されたものです。

プログラム **t.f**:

```
        integer a(100, 100)
        integer i,j

        do i = 1 , 100
          do j = 1, 100
            a(i,j) = j
          end do
        end do
        end
```

変換報告書**:**

```
>>>>> SOURCE SECTION <<<<<
** _main   === End of Compilation 1 ===

>>>>> LOOP TRANSFORMATION SECTION <<<<<
```

```
                PROGRAM _main ()
     4|           IF (.FALSE.) GOTO lab_9
                  @CIV2 = 0
       Id=1       DO @CIV2 = @CIV2, 24
     5|             IF (.FALSE.) GOTO lab_11
                    @LoopIV1 = 0
     6|             @CSE0 = @CIV2 * 4
                    @ICM0 = @CSE0 + 1
                    @ICM1 = @CSE0 + 2
                    @ICM2 = @CSE0 + 3
                    @ICM3 = @CSE0 + 4
     5|Id=2         DO @LoopIV1 = @LoopIV1, 99
                      ! DIR_INDEPENDENT loopId = 0
                      ! DIR_INDEPENDENT loopId = 0
     6|               @CSE1 = @LoopIV1 + 1
                      SHADV_M003_a(@CSE1,@ICM0) = @ICM0
                      SHADV_M002_a(@CSE1,@ICM1) = @ICM1
                      SHADV_M001_a(@CSE1,@ICM2) = @ICM2
                      SHADV_M000_a(@CSE1,@ICM3) = @ICM3
     7|               ENDDO
                      lab_11
     8|             ENDDO
                    lab_9
     4|             IF (.FALSE.) THEN
                      @LoopIV0 = int((100 - MOD(100, int(4))))
       Id=5           DO @LoopIV0 = @LoopIV0, 100
     5|                 IF (.FALSE.) GOTO lab_19
                        @LoopIV1 = 0
     6|                 @ICM4 = @LoopIV0 + 1
     5|Id=6             DO @LoopIV1 = @LoopIV1, 99
                        ! DIR_INDEPENDENT loopId = 0
                        ! DIR_INDEPENDENT loopId = 0
     6|                 a((@LoopIV1 + 1),@ICM4) = @ICM4
     7|                 ENDDO
                        lab_19
     8|               ENDDO
                    ENDIF
     9|           END PROGRAM _main


Source        Source        Loop Id       Action / Information
File          Line
----------    ----------    ----------    ---------------------------------------
         0             4             1    Loop interchanging applied to loop nest.
         0             4             1    Outer loop has been unrolled 4 time(s).
```

データ再編成レポート・セクション

コンパイラー *data reorganizations* によるプログラム変数データの再編成方法に関する、便利な情報の要約です。

データ再編成情報を生成するには、**-qreport** とともに最適化レベル **-qipa=level=2** または **-O5** を指定します。IPA リンク・パス時に、プログラム変数データのデータ再編成メッセージが、リスト・ファイルのデータ再編成セクションに作成されます。再編成には以下のものが含まれます。

- 共通ブロック分割
- 配列分割
- 配列転置
- メモリー割り振りのマージ
- 配列インターリービング

- 配列合体

属性および相互参照セクション

このセクションは、コンパイル単位で使用されるエンティティーに関する情報を提供します。このセクションは、**-qxref** または **-qattr** コンパイラー・オプションが有効な場合のみ存在します。有効なオプションに応じて、このセクションにはコンパイル単位で使用されるエンティティーに関する以下の情報の全部または一部が含まれます。

- エンティティー名
- エンティティーの属性 (**-qattr** が有効な場合)。属性情報には、以下の詳細のいずれか、またはすべてが含まれることがあります。
  - 名前のクラス
  - タイプ
  - 名前の相対アドレス
  - 境界合わせ
  - 次元
  - 配列の場合は、それが整合かどうか
  - それがポインター、ターゲット、整数ポインターのいずれであるか
  - それがパラメーターであるかどうか
  - それが揮発性であるかどうか
  - 仮引数について、その意図、それが値であるかどうか、それがオプショナルかどうか
  - モジュール・エンティティーについて、それが private か、public か、または protected かどうか
- エンティティーを定義、参照、あるいは変更した場所を示すための座標。エンティティーを宣言すると、座標に `$` マークが付きます。エンティティーを初期化すると、座標に `*` マークが付きます。 同じ場所でエンティティーの宣言および初期化の両方を行うと、座標に `&` マークが付きます。 エンティティーが設定されると、座標に `@` マークが付きます。エンティティーが参照されても、座標には何もマークが付きません。

クラスは以下の中のいずれかです。

- 自動
- BSS (初期化されていない静的内部クラス)
- 共通
- 共通ブロック
- 構文名
- 制御済み (割り当て可能オブジェクト)
- 制御済み自動 (自動オブジェクトの場合)
- 定義済み割り当て
- 定義済み演算子
- 派生型定義
- 入り口
- 外部サブプログラム
- 関数
- 総称名
- 内部サブプログラム

- 組み込み
- モジュール
- モジュール関数
- モジュール・サブルーチン
- 名前リスト
- ポインティング先
- プライベート・コンポーネント
- プログラム
- 参照引数
- 名前変更
- 静的
- F2008 サブモジュール F2008
- サブルーチン
- 使用関連付け
- 値パラメーター

-qxref または **-qattr** によって完全なサブオプションを指定すると、XL Fortran はコンパイル単位内のすべてのエンティティーについて報告します。 このサブオプションを指定しないと、実際に使用しているエンティティーだけが表示されます。

オブジェクト・セクション

XL Fortran は、**-qlist** コンパイラー・オプションが有効な場合のみ、このセクションを作成します。このセクションにはオブジェクト・コード・リストが入っていて、このリストは、ソース行番号、命令オフセット (16 進表記)、命令のアセンブラー・ニーモニック、命令の 16 進値を示します。右側には、命令のサイクル・タイムとコンパイラーの中間言語も示されます。最後に、作成されたマシン命令の合計数と合計サイクル・タイム (直線的実行時) が表示されます。コンパイル単位ごとに別個のセクションがあります。

ファイル・テーブル・セクション

このセクションには、使用されている個々のメイン・ソース・ファイルとインクルード・ファイルのファイル番号とファイル名を示すテーブルが含まれています。また、インクルード・ファイルが参照されるメイン・ソース・ファイルの行番号もリストします。このセクションは常に存在します。また、テーブルは、ファイルの作成日時も含んでいます。

コンパイル単位エピローグ・セクション

これは、各コンパイル単位のリストの最後のセクションです。これには診断の詳細が含まれていて、その単位が正常にコンパイルされたかどうかを示します。ファイルにコンパイル単位が 1 つしか含まれていない場合は、このセクションはリストに存在しません。

コンパイル・エピローグ・セクション

コンパイル・エピローグ・セクションは、リストの終わりに 1 回印刷されるだけです。コンパイルの完了時に、XL Fortran はコンパイルの概要を提示します。概要とは、読み取られたソース・レコードの数、コンパイル開始時刻、コンパイルの終了時刻、合計コンパイル時間、合計 CPU 時間、仮想 CPU 時間、および診断状態の要約です。このセクションは、リストに必ず存在します。

第 12 章 XL Fortran 技術情報

このセクションでは、一般プログラマーにはあまり関係のない、通常では発生しない問題の診断、特殊な環境でのコンパイラーの実行、その他の処理操作を行う場合に、上級プログラマーが必要とする XL Fortran の技術情報について詳述します。

XL Fortranライブラリー内の外部名

ユーザー定義の名前とランタイム・ライブラリーで定義されている名前との競合を最小限に抑えるために、ランタイム・ライブラリー内にある入出力ルーチンの名前の最初に下線 (_)、または _xl が付けられます。

XL Fortran ランタイム環境

XL Fortran コンパイラーが作成するオブジェクト・コードは、特定の複合タスクを処理するために、コンパイラーが提供するサブプログラムを実行時に呼び出すことがあります。 このようなサブプログラムは各種ライブラリーに入っています。

XL Fortran ランタイム環境の機能は、次のように分類されます。
- Fortran 入出力操作のサポート
- 数値計算
- オペレーティング・システム・サービス
- SMP 並列化のサポート

静的にバインドを行わない限り、XL Fortran ランタイム環境を使用しないで XL Fortran コンパイラー作成のオブジェクト・コードを実行することはできません。

XL Fortran ランタイム環境には上位互換性があります。あるレベルのランタイム環境とオペレーティング・システムでコンパイルおよびリンクしたプログラムを実行するには、コンパイル時と同じかそれ以上のレベルのランタイム環境とオペレーティング・システムが必要となります。

ランタイム環境の外部名

実行時サブプログラムはライブラリーに入れられます。デフォルトでは、コンパイラー呼び出しコマンドがリンカーを呼び出し、これにライブラリーの名前を付けます。このライブラリーには、Fortran オブジェクト・コードによって呼び出される実行時サブプログラムが含まれています。

このような実行時サブプログラムの名前が外部シンボルです。 XL Fortran コンパイラーによって作成されたオブジェクト・コードが実行時サブプログラムを呼び出す時点では、 **.o** オブジェクト・コード・ファイルにそのサブプログラムの名前の外部シンボル参照が含まれています。ライブラリーには、そのサブプログラムの外部シンボル定義が含まれています。リンカーはこのサブプログラム定義で実行時サブプログラム呼び出しを解決します。

XL Fortran プログラムでは、実行時サブプログラムの名前と競合する名前を使用しないでください。名前の競合は、次の 2 つの条件下で起こる可能性があります。

- Fortran プログラムで定義されたサブルーチン、関数、共通ブロックの名前がライブラリー・サブプログラムの名前と同じ場合。
- Fortran プログラムがライブラリー・サブプログラムと同名のサブルーチンや関数を呼び出したが、呼び出されたサブルーチンや関数の定義が行われていなかった場合。

-qfloat=hsflt オプションの技術情報

単精度表現の範囲外にある (結果型の範囲外にあるだけではない) 浮動小数点値の計算を行う最適化済みプログラムの場合、**-qfloat=hsflt** オプションはアンセーフです。この表現範囲には精度と指数範囲が含まれます。

前の段落および 153 ページの『-qfloat』で述べた規則に従っていても、精度の違いに依存しているプログラムでは、予想した結果を生成しない場合があります。**-qfloat=hsflt** は IEEE に準拠していないため、プログラムが常に予想どおりに稼働するとは限りません。

例えば以下のプログラムでは、X.EQ.Y は真である場合と偽である場合があります。

```
REAL X, Y, A(2)
DOUBLE PRECISION Z
LOGICAL SAME

READ *, Z
X = Z
Y = Z
IF (X.EQ.Y) SAME = .TRUE.
! ...
! ... Calculations that do not change X or Y
! ...
CALL SUB(X)         ! X is stored in memory with truncated fraction.
IF (X.EQ.Y) THEN    ! Result might be different than before.
...

A(1) = Z
X = Z
A(2) = 1.           ! A(1) is stored in memory with truncated fraction.
IF (A(1).EQ.X) THEN ! Result might be different than expected.
...
```

Z の値に単精度変数の精度外にある小数ビットがある場合、そのビットは保存される場合と、失われる場合があります。このため、倍精度値の Z が単精度変数に割り当てられる場合に正確な結果は予期不能となります。例えば、変数を仮引数として渡す場合、メモリーに記憶される値の小数部は丸められるのではなく切り捨てられます。

-qautodbl のプロモーションと埋め込みの実装の詳細

以下の項では、**-qautodbl** オプションの動作の詳細について説明し、プロモーションと埋め込みが行われているときの動作をユーザーが予測できるようにします。

用語

2 つのデータ・オブジェクト間のストレージの関係 (storage relationship) によって、これらのオブジェクトの相対開始アドレスと相対サイズを判別します。**-qautodbl** オプションは、可能な限りこの関係を保持するように動作します。

データ・オブジェクトは、1 つのオブジェクトの変更内容がもう 1 つのオブジェクトに反映されるように値の関係 (value relationship) を持つこともできます。例えば、あるプログラムで値をある変数に保管し、次にこの値を別のストレージ関連の変数を介して読み取るという場合です。 **-qautodbl** を指定して有効な状態にすると、1 つまたは両方の値の表現が異なるものになるため、値の関係が常に保持されるということがなくなります。

このオプションがオブジェクトに作用すると、オブジェクトは次のように処理されます。

- プロモートされる。 つまり、オブジェクトはより高い精度のデータ型に変換されます。通常、プロモートされたオブジェクトのサイズはデフォルトによりオブジェクト・サイズの 2 倍です。プロモーションは、該当する型の定数、変数、派生型コンポーネント、配列、関数 (組み込み関数を含む) に適用されます。

  注**: BYTE**、**INTEGER**、**LOGICAL**、**CHARACTER** オブジェクトはプロモートされません。
- 埋め込まれる。 つまり、オブジェクトは元の型を保持しますが、その後ろに未定義の記憶スペースが続きます。埋め込みは **BYTE**、**INTEGER**、**LOGICAL** およびプロモートされていない **REAL** と **COMPLEX** オブジェクトに適用されます。これら 2 つのプロモートされていないオブジェクトは、プロモートされた項目と記憶スペースを共用することがあります。安全のため、**POINTER**、**TARGET**、実引数と仮引数、**COMMON** ブロックのメンバー、構造体、ポインティング先配列、ポインティング先 **COMPLEX** オブジェクトは、**-qautodbl** サブオプションの設定に応じて常に適切に埋め込みが行われます。それらがプロモートされたオブジェクトとストレージを共用しているかどうかに関係なく、そのようになります。

  埋め込み用に追加されたスペースにより、変換前から存在しているストレージ共用関係が確実に維持されます。例えば、配列エレメント **I(20)** と **R(10)** がデフォルトにより同一アドレスから開始したときに、**R** のエレメントがプロモートされてサイズが 2 倍になった場合、**I(20)** と **R(10)** が同一アドレスから開始するようにするため、**I** のエレメントが埋め込まれます。

  I/O ステートメントは埋め込みを処理しません。ただし、不定形式の I/O ステートメントは除きます。これらのステートメントは、構造体内の埋め込みを読み書きします。

  注**:** コンパイラーは、**CHARACTER** オブジェクトに埋め込みを行いません。

-qautodbl サブオプションのストレージの関係の例

このセクションに示す例は、以下のエンティティー相互間でストレージが共用される関係を示しています。

- **REAL(4)**
- **REAL(8)**
- **REAL(16)**
- **COMPLEX(4)**
- **COMPLEX(8)**

- **COMPLEX(16)**
- **INTEGER(8)**
- **INTEGER(4)**
- **CHARACTER(16)**

注: 図中の実線は実データを、破線は埋め込みを表します。

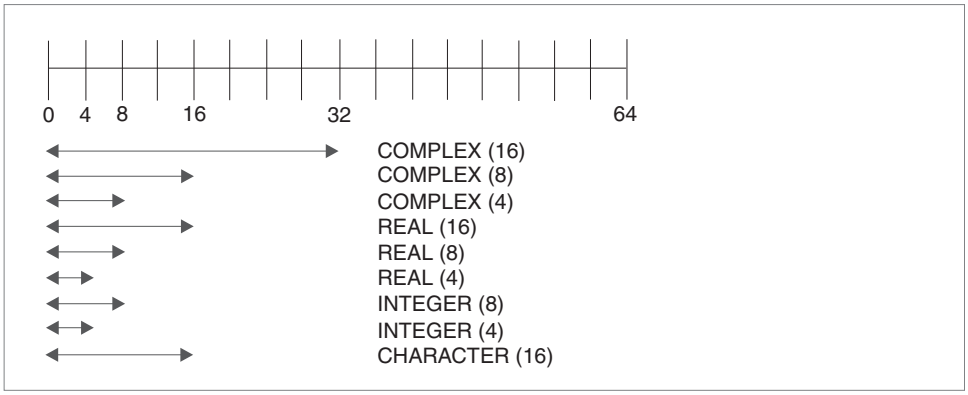


図 *5. -qautodbl* オプションを指定しなかった場合のストレージの関係

上の図は、コンパイラーのデフォルトのストレージの共用関係を示しています。

```
@process autodbl(none)
      block data
      complex(4) x8      /(1.123456789e0,2.123456789e0)/
      real(16) r16(2)    /1.123q0,2.123q0/
      integer(8) i8(2)   /1000,2000/
      character*5 c(2)  /"abcde","12345"/
      common /named/ x8,r16,i8,c
      end

      subroutine s()
         complex(4) x8
         real(16) r16(2)
         integer(8) i8(2)
         character*5 c(2)
         common /named/ x8,r16,i8,c
!          x8    = (1.123456e0,2.123456e0)      ! promotion did not occur
!         r16(1) = 1.123q0                      ! no padding
!         r16(2) = 2.123q0                      ! no padding
!          i8(1) = 1000                         ! no padding
!          i8(2) = 2000                         ! no padding
!           c(1) = "abcde"                      ! no padding
!           c(2) = "12345"                      ! no padding
      end subroutine s
```

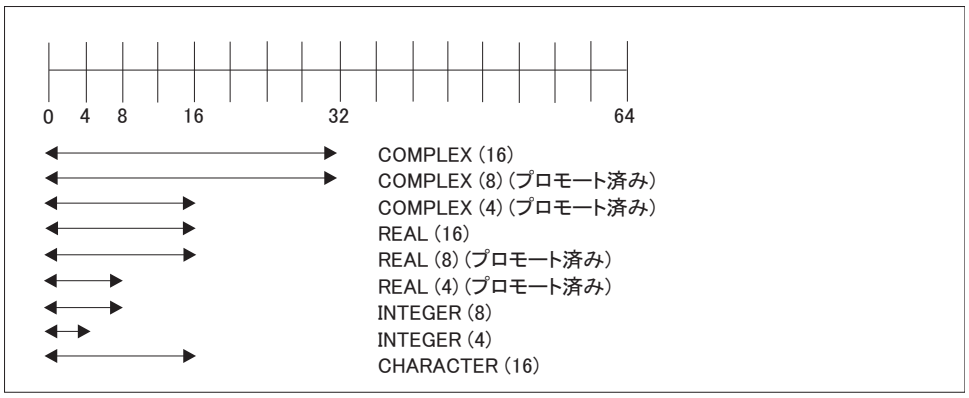


図 *6. -qautodbl=dbl* を指定した場合のストレージの関係

```
@process autodbl(dbl)
      block data
      complex(4) x8
      real(16) r16(2)    /1.123q0,2.123q0/
      real(8) r8
      real(4) r4         /1.123456789e0/
      integer(8) i8(2)   /1000,2000/
      character*5 c(2)  /"abcde","12345"/
      equivalence (x8,r8)
      common /named/ r16,i8,c,r4
!     Storage relationship between r8 and x8 is preserved.
!     Data values are NOT preserved between r8 and x8.
      end

      subroutine s()
         real(16) r16(2)
         real(8) r4
         integer(8) i8(2)
         character*5 c(2)
         common /named/ r16,i8,c,r4
!         r16(1) = 1.123q0                    ! no padding
!         r16(2) = 2.123q0                    ! no padding
!          r4    = 1.123456789d0              ! promotion occurred
!          i8(1) = 1000                       ! no padding
!          i8(2) = 2000                       ! no padding
!           c(1) = "abcde"                    ! no padding
!           c(2) = "12345"                    ! no padding
      end subroutine s
```

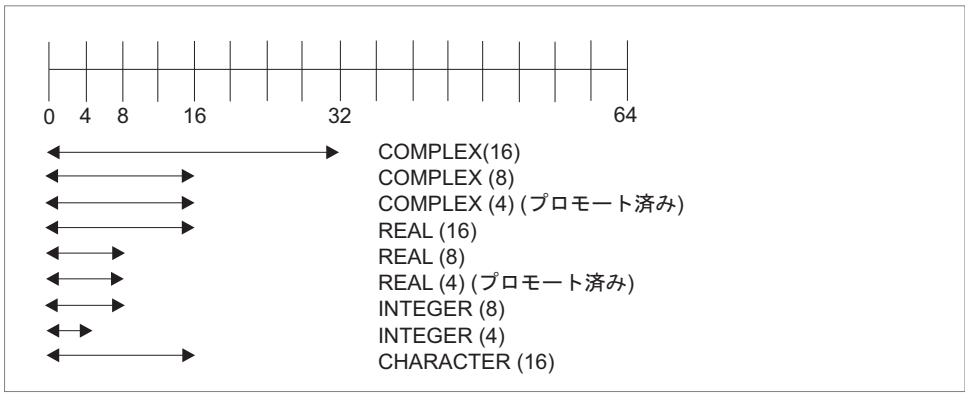


図 *7. -qautobl=dbl4* を指定した場合のストレージの関係

```
@process autodbl(dbl4)
      complex(8) x16    /(1.123456789d0,2.123456789d0)/
      complex(4) x8
      real(4) r4(2)
      equivalence (x16,x8,r4)
!     Storage relationship between r4 and x8 is preserved.
!     Data values between r4 and x8 are preserved.
!     x16   = (1.123456789d0,2.123456789d0)      ! promotion did not occur
!     x8    = (1.123456789d0,2.123456789d0)      ! promotion occurred
!     r4(1) = 1.123456789d0                      ! promotion occurred
!     r4(2) = 2.123456789d0                      ! promotion occurred
      end
```

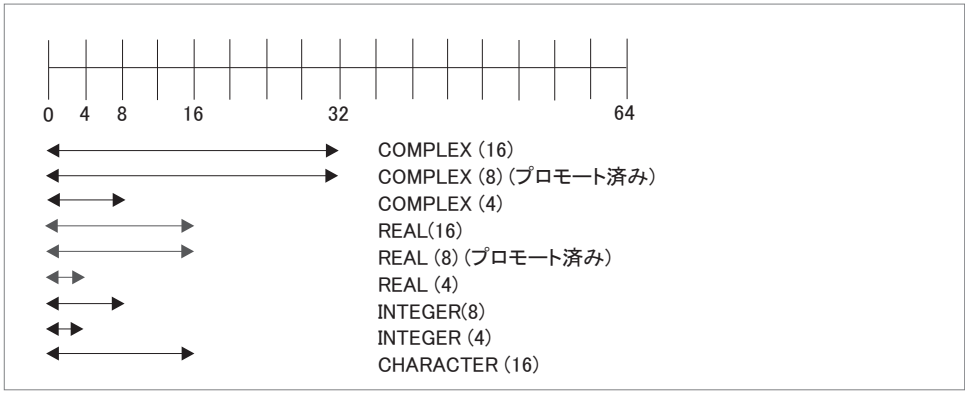


図 *8. -qautodbl=dbl8* を指定した場合のストレージの関係

```
@process autodbl(dbl8)
      complex(8) x16    /(1.12345678912345678 9d0,2.12345678912345678 9d0)/
      complex(4) x8
      real(8) r8(2)
      equivalence (x16,x8,r8)
!     Storage relationship between r8 and x16 is preserved.
!     Data values between r8 and x16 are preserved.
!     x16   = (1.12345678912345678 9q0,2.12345678912345678 9q0)
!                                                ! promotion occurred
!     x8    = upper 8 bytes of r8(1)             ! promotion did not occur
!     r8(1) = 1.12345678912345678 9q0            ! promotion occurred
!     r8(2) = 2.12345678912345678 9q0            ! promotion occurred
      end
```

第 13 章 XL Fortran 内部制限

言語の特徴	制限
INTEGER(n) のインデックス変数での、ループ制御による **DO** ループの最大反復実行回数 (*n* = 1、2、または 4)	(2**31)-1
INTEGER(8) のインデックス変数での、ループ制御による **DO** ループの最大反復実行回数	(2**63)-1
文字形式フィールドの最大幅	(2**31)-1
形式仕様の最大長	(2**31)-1
ホレリス定数および文字定数編集記述子の最大長	(2**31)-1
固定ソース形式文の最大長	65,000
自由ソース形式文の最大長	65,000
最大継続行数	n/a **1**
最大ネスト **INCLUDE** 行数	64
最大ネスト・インターフェース・ブロック数	1024
計算型 **GOTO** 内のステートメント番号の最大数	999
形式コードの最大繰り返し回数	(2**31)-1
入出力ファイルの許容レコード数とレコード長	レコード番号は最大 (2**63)-1 で、レコード長は最大 (2**63)-1 バイトです。 ただし、不定形式の順次ファイルでは、レコード長が (2**31)-1 を超えて (2**63)-1 までの場合、 **uwidth=64** 実行時オプションを使用しなければなりません。デフォルトの **uwidth=32** 実行時オプションを使用する場合、不定形式の順次ファイル内の最大レコード長は (2**31)-1 バイトです。
配列次元の許容境界範囲	配列次元の境界は、までの範囲内、-(2**63) から 2**63-1 までの範囲内で、正、負、またはゼロのいずれかの値をとります。
外部装置の許容数	0 から (2**31)-1 **2**
数値形式フィールドの最大幅	2000
同時にオープンできるファイルの最大数	1024 **3**

1 1 つの文 (最大サイズ 65,000 バイト) を作成する場合、継続行を必要なだけ指定できます。

2 この値は **INTEGER(4)** オブジェクトで表せるものでなければなりません。値を変数 **INTEGER(8)** で指定している場合でも同様です。

3 ランタイム・システムがファイル (例えば、事前接続されている装置 0、5、 6 など) をオープンする場合があるため、この値は実際には表中の値よりもいくらか小さくなります。

用語集

この用語集では、本書で頻繁に使用する用語を解説します。この用語集には、米国規格協会 (ANSI) によって作成された定義、および Web サイトの「***IBM*** 用語」から抜粋した項目項目が含まれています。

A

抽象インターフェース **(abstract interface)**
ABSTRACT INTERFACE は、仮引数のプロシージャー特性および名前で構成されます。プロシージャーおよび据え置きバインディングのインターフェースを宣言するために使用される。

抽象タイプ **(abstract type)**
ABSTRACT 属性を持つ型。非ポリモアフィック・オブジェクトは、抽象型として宣言できません。ポリモアフィック・オブジェクトを、抽象である動的タイプを持つように構成または割り振りすることはできません。

アクティブ・プロセッサー **(active processor)**
オンライン・プロセッサー を参照。

実引数 **(actual argument)**
プロシージャー参照で指定される式、変数、プロシージャー、代替戻り指定子のいずれか。

別名 **(alias)**
単一の名前を超える名前を介してアクセス可能な 1 つのストレージ。それぞれの名前はそのストレージの別名になる。

英字 **(alphabetic character)**
言語で使用される文字またはその他の記号 (数字を除く)。通常は、英大文字、小文字の A から Z に加え、特定の言語で使用可能なその他の特殊記号 (例えば $ や _ など) を指す。

英数字 **(alphanumeric)**
文字セットに関するもの。この文字セットには、文字、数字に加え、通常はその他の文字 (例えば、句読符号、数学記号など) が含まれる。

情報交換用米国標準コード **(American National Standard Code for Information Interchange)**
ASCII を参照。

引数 **(argument)**
関数やサブルーチンへ引き渡される式。実引数 *(actual argument)*、仮引数 *(dummy argument)* も参照。

引数関連付け **(argument association)**
プロシージャー起動時の実引数と仮引数の関係。

算術定数 **(arithmetic constant)**
整数、実数、複素数のいずれかの型の定数。

算術式 **(arithmetic expression)**
1 つ以上の算術演算子と算術 1 次子からなり、計算結果が単一の数値として表される。算術式は、符号なし算術定数、算術定数の名前、算術変数への参照、関数参照、算術演算子または括弧を使ったこのような 1 次子の組み合わせ。

算術演算子 **(arithmetic operator)**
算術演算を実行させる記号。組み込み算術演算子は次のとおり。

+ 加算

- 減算

***** 乗算

/ 除算

****** 指数

配列 **(array)**
順序付けられたスカラー・データのグループを含むエンティティー。配列内のオブジェクトはすべて、同一のデータ型と型付きパラメーターを持つ。

配列宣言子 **(array declarator)**
文の一部であり、プログラム単位内で使用される配列について記述するもの。配列宣言子では、配列の名前、含まれる次元数、各次元のサイズを指定する。

配列エレメント **(array element)**
配列名と 1 つ以上の添え字で識別される配列中の単一データ項目。添え字も参照。

配列名 **(array name)**
順序付けられたデータ項目のセットの名前。

配列セクション **(array section)**
配列であり、構造体コンポーネントではないサブオブジェクトのこと。

ASCII
1 文字が 7 ビット (パリティー検査ビットも含め 8 ビット) によって構成されるコード化文字セットを用いて、データ処理システム、データ通信システム、およびそれらの関連装置の間で情報交換をおこなうのに使用される標準コード。この ASCII セットを構成する文字の種類として、制御文字と図形文字とが含まれる。 *Unicode* も参照。

非同期 **(asynchronous)**
時間が同期していないか、通常のまたは予測可能な時間間隔で生起しないイベントについて形容する。

割り当てステートメント **(assignment statement)**
式の計算結果に基づいて、変数を定義または再定義する実行可能ステートメント。

関連名 **(associate name)**
SELECT TYPE または **ASSOCIATE** 構成体のセレクターが構成体内で知られる名前。

想定サイズ配列 **(assumed-size array)**
ダミー配列のサイズは、関連付けられる実引数から想定される。その最後の上部のバウンドは、アスタリスクで指定される。

想定型オブジェクト **(assumed-type object)**
TYPE(*) を使用して宣言されたエンティティー。想定型オブジェクトに宣言型はなく、その動的な型および型パラメーターは、対応する実引数から想定される。

属性 **(attribute)**
データ・オブジェクトの特性。型宣言ステートメント、属性仕様ステートメント、デフォルト設定のいずれかで指定される。

自動並列化 **(automatic parallelization)**
明示的にコーディングされた **DO** ループ、および配列言語のためのコンパイラーが生成した **DO** ループとを、コンパイラーが並列化しようとする処理。

B

基本オブジェクト **(base object)**
左端の *part_name* によって指定されるオブジェクト。

基本タイプ **(base type)**
別の型の拡張ではない拡張可能な型。

2 進定数 **(binary constant)**
1 つ以上の 2 進数字 (0 と 1) からの定数。

バインド **(bind)**
識別子をプログラム内の別のオブジェクトに関係させること。例えば、識別子を値、アドレス、または別の識別子に関係させること、または仮パラメーターと実パラメーターを関連させることなど。

バインディング・ラベル **(binding label)**
変数、共通ブロック、サブルーチン、または関数が比較プロセッサーで認識される方法を単一に識別する、型デフォルト文字の値。

無名共通ブロック **(blank common)**
名前のない共通ブロック。

ブロック・データ・サブプログラム **(block data subprogram)**
BLOCK DATA 文が先頭にあるサブプログラム。名前付き共通ブロックにおいて、変数の初期化に使用される。

bounds_remapping
ユーザーが、フラットでランク -1 の配列をマルチディメンション配列として表示できる。

BSS ストレージ **(bss storage)**
初期化されていない静的ストレージ。

busy-wait
スレッドが短いループ内で実行されていて、作業をすべて終了したので行うべき新しい作業がないため、他の作業を探しているときの状態。

バイト定数 **(byte constant)**
バイト型の名前付き定数。

バイト型 **(byte type)**
1 バイトのストレージを表すデータ型。**LOGICAL(1)**、**CHARACTER(1)**、**INTEGER(1)** のいずれかを使用できる場合に使用可能。

C

C 記述子 **(C descriptor)**
ISO_Fortran_binding.h ヘッダー・ファイルで定義されているタイプ `CFI_cdesc_t` の C 構造体。

文字定数 **(character constant)**
1 つ以上の英字からなる文字ストリング。アポストロフィまたは二重引用符で囲まれる。

文字式 **(character expression)**
文字オブジェクト、文字によって評価される関数参照のいずれか。また、連結演算子 (括弧は任意) で分離されるこれらの順序列の場合もある。

文字演算子 **(character operator)**
文字データに対して実行される操作を表す記号 (例えば、連結 (//) など)。

文字セット **(character set)**
プログラミング言語用またはコンピューター・システム用のすべての有効文字。

文字ストリング **(character string)**
連続した文字の列。

文字サブストリング **(character substring)**
文字ストリングの連続する一部分。

文字型 **(character type)**
英数字で構成されるデータ型。データ型 *(data type)* も参照。

チャンク **(chunk)**
連続するループ反復のサブセット。

クラス **(class)**
1 つの基本タイプと、その基本タイプから拡張されたすべての型で構成される一連の型。

照合順序 **(collating sequence)**
複数の文字が、ソート、マージ、比較、および索引付きのデータの順番処理の目的で並べられるときの順序。

コメント **(comment)**
プログラムにテキストを含めるための言語構文。プログラムの実行内容には関係ない。

共通ブロック **(common block)**
呼び出し側プログラムと 1 つ以上のサブプログラムによって参照されることのあるストレージ域。

コンパイル **(compile)**
ソース・プログラムを実行可能プログラム (オブジェクト・プログラム) へ変換すること。

コンパイラー・コメント・ディレクティブ **(compiler comment directive)**
ソース・コード中で、Fortran 文ではないが、コンパイラーが認識し、動作させる行。

コンパイラー・ディレクティブ **(compiler directive)**
ユーザー・プログラムの実行内容ではなく、XL Fortran の実行内容を制御するソース・コード。

複素定数 **(complex constant)**
順序付けられた 1 対の実定数または整定数。コンマで区切られ、括弧で囲まれて示される。最初の定数が複素数の実数部で、2 番目の定数が虚数部である。

複素数 **(complex number)**
順序付けられた 1 対の実数からなる数値。**a+bi** の書式で表される。 **a** および **b** は実数で、**i** の平方は -1 である。

複素数型 **(complex type)**
複素数の値を表すデータ型。この値は順序付けられた 1 対の実数データ項目であり、コンマで区切られ、括弧で囲まれて示される。最初の項目が複素数の実数部で、2 番目の項目が虚数部である。

コンポーネント **(component)**
派生型の構成要素。

コンポーネント・オーダー **(component order)**
構造コンストラクターの組み込み定様式入出力で使用できる、派生型のコンポーネントの順序付け。

準拠 **(conform)**
普及している標準に従うこと。実行可能プログラムが Fortran 95 標準に記述されているフォームとリレーションシップのみを使用しており、かつこの実行可能プログラムが Fortran 95 標準に従った解釈を持つのであれば、実行可能プログラムは Fortran 95 標準に適応している。実行可能プログラムが標準適応となるようにプログラム単位が実行可能プログラムに含まれている場合、このプログラム単位は Fortran 95 標準に適応している。標準に規定されている解釈を満たすようにプロセッサーが標準適応プログラムを実行する場合、このプロセッサーは標準に適応している。

接続装置 **(connected unit)**
XL Fortran では、**OPEN** 文による名前付きファイルへの明示的接続、暗黙的接続、事前接続といった 3 つの方法のいずれかでファイルに接続された装置。

定数 **(constant)**
不変の値を持つデータ・オブジェクト。定数には 4 つのクラスがあり、数字 (算術)、真理値 (論理)、文字データ (文字)、型なしのデータ (16 進値、8 進値、2 進値) がこれらに当たる。変数 *(variable)* も参照。

構文 **(construct)**
例えば、**SELECT CASE**、**DO**、**IF**、**WHERE** のいずれかの文で始まり、対応する終端ステートメントで終わるステートメントの順序列。

連続 **(contiguous)**
順序内の各配列エレメントが他のデータ・オブジェクトによって分離されて

いない場合、配列は連続している。順序内の各部分が他のデータ・オブジェクトによって分離されていない場合、複数の部分を含むデータ・オブジェクトは連続している。

継続行 **(continuation line)**
文をその最初の行を越えて継続させる行。

制御ステートメント **(control statement)**
文の連続的な順次呼び出しを変更するのに使用される文。制御文は、条件文 (**IF** など) の場合と、命令文 (**STOP** など) の場合がある。

D

データ・オブジェクト **(data object)**
変数、定数、または定数のサブオブジェクト。

データ・ストライピング **(data striping)**
データを複数の記憶装置に分散すること。これによって I/O 操作を並列実行でき、パフォーマンスが向上する。ディスク・ストライピング *(disk striping)* とも呼ばれる。

データ転送ステートメント **(data transfer statement)**
READ、**WRITE**、**PRINT** の各文。

データ型 **(data type)**
データと機能の特徴を定義する特性および内部表現。 組み込みの型としては、整数、実数、複素数、論理、文字の各型がある。組み込み *(intrinsic)* も参照。

デバッグ行 **(debug line)**
デバッグ用のソース・コードを含む行。修正するソース・フォームにだけ含めることが許可される。デバッグ行は、1 桁目の `D` または `X` で定義される。デバッグ行の処理は、**-qdlines** および **-qxlines** コンパイラー・オプションで制御される。

10 進記号 **(decimal symbol)**
実数の整数と小数部分を分離する記号。

宣言型 **(declared type)**
データ・エンティティーが持つように宣言されている型。ポリモアフィック・データ・エンティティーの場合、実行中の型 (動的タイプ) とは異なる場合がある。

デフォルトの初期化 **(default initialization)**
派生型の定義の一部として指定された値を持つオブジェクトの初期化。

据え置きバインディング **(deferred binding)**
DEFERRED 属性を持つバインディング。抽象型定義でのみ見られる据え置きバインディング。

定義可能変数 **(definable variable)**
割り当てステートメントの左側に名前または指定子を表示することによって値を変更可能な変数。

区切り文字 **(delimiters)**
構文のリストを囲むために使用する括弧またはスラッシュ (あるいはその両方) の組。

非正規数 **(denormalized number)**
非常に小さな絶対値と低精度の IEEE 数。非正規数は、ゼロの指数とゼロ以外の小数部で表される。

派生型 **(derived type)**
データがコンポーネントを持つ型。各コンポーネントは、組み込み型または別の派生型のいずれかである。

数字 **(digit)**
負数ではない整数を表す文字。例えば、0 から 9 のいずれかの数字。

ディレクティブ **(directive)**
コンパイラーに指示や情報を与えるコメントの型。

ディスク・ストライピング **(disk striping)**
データ・ストライピング *(data striping)* を参照。

DO ループ **(DO loop)**
DO 文で繰り返し呼び出されるステートメントの範囲。

DO 変数 **(DO variable)**
DO 文で指定される変数。 **DO** ループ内にある 1 つ以上の文の各オカレンスに先立ち、初期化または増分される。範囲内のステートメントの実行回数の制御に使用される。

DOUBLE PRECISION 定数 **(DOUBLE PRECISION constant)**
デフォルトの実際の精度の 2 倍の精度を持つ実数型の定数。

仮引数 **(dummy argument)**
括弧で囲まれたリストに名前が記述されたエンティティー。 **FUNCTION**、**SUBROUTINE**、**ENTRY**、文関数のいずれかの文のプロシージャー名の後ろに存在する。

動的ディメンション **(dynamic dimensioning)**
配列が参照される度にその境界を再評価するプロセス。

動的エクステント **(dynamic extent)**
ディレクティブについての動的エクステントとは、ディレクティブの字句エクステントおよび字句エクステント内から呼び出されたすべてのサブプログラムである。

動的タイプ **(dynamic type)**
プログラム実行中のデータ・エンティティーの型。ポリモアフィックでないデータ・エンティティーの動的タイプが、宣言された型と同じである。

E

編集記述子 **(edit descriptor)**
整数、実数、および複素数データの形式設定を制御する省略形のキーワード。

有効な項目 **(effective item)**
入出力リストの拡張から生成されたスカラー・オブジェクト。

エレメント型 **(elemental)**
組み込み演算、プロシージャー、割り当てを修飾する形容詞で、配列のエレメントまたは規格対応の配列とスカラーのセットの対応したエレメントに対して個別に適用される。

埋め込まれたブランク **(embedded blank)**
前後をブランク以外の文字で挟まれたブランク。

エンティティー **(entity)**
次のものを表す一般用語。プログラム単位、プロシージャー、演算子、インターフェース・ブロック、共通ブロック、外部装置、ステートメント関数、型、名前付き変数、式、構成のコンポーネント、名前付き定数、ステートメント・ラベル、構文、名前リスト・グループなど。

環境変数 **(environment variable)**
プロセスの操作環境を記述する変数。

エポック **(epoch)**
POSIX での時間に使用される開始日時 1970 年 1 月 1 日 00:00:00 (グリニッジ標準時)。

実行可能プログラム **(executable program)**
自己完結型プロシージャーとして実行できるプログラム。メインプログラムと、オプションで、モジュール、サブモジュール、サブプログラム、Fortran 以外の外部プロシージャーからなる。

実行可能ステートメント **(executable statement)**
プログラムにある処置、例えば、計算、条件のテスト、通常の順次実行の変更などを引き起こさせるステートメント。

明示的初期化 **(explicit initialization)**
データ・ステートメント初期値リスト、ブロック・データ・プログラム単位、型宣言ステートメント、または配列コンストラクターの値を持つオブジェクトの初期化。

明示的インターフェース **(explicit interface)**
有効範囲単位内で参照されるプロシージャーのためのもので、内部プロシージャー、モジュール・プロシージャー、組み込みプロシージャー、インターフェース・ブロックを持つ外部プロシージャー、有効範囲単位内の再帰的プロシージャー参照、インターフェース・ブロックを持つダミー・プロシージャーのいずれかのプロパティー。

式 **(expression)**
オペランド、演算子、括弧の順序列。変数、定数、関数参照、または、計算を指す。

拡張精度定数 **(extended-precision constant)**
連続的な 16 バイトのストレージに記憶される実数値に対するプロセッサー近似値。

拡張された型 **(extended type)**
別のタイプの拡張である拡張可能なタイプ。**EXTENDS** 属性で宣言される型。

拡張可能な型 **(extensible type)**
EXTENDS 属性を使用して派生された新規型の元の型。**BIND** 属性を持たない非シーケンス型。

拡張型 **(extension type)**
基本タイプは、それ自身が拡張型である。拡張された型は、それ自身および親タイプが拡張であるすべての型の拡張型である。

外部ファイル **(external file)**
入出力装置にある一連のレコード。内部ファイル *(internal file)* も参照。

外部名 **(external name)**
リンカーが、1 つのコンパイル単位から別のもう 1 つのコンパイル単位への参照を解決するのに使用する共通ブロック、サブルーチン、またはその他のグローバル・プロシージャーの名前。

外部プロシージャー **(external procedure)**
外部サブプログラムまたは Fortran 以外の手段で定義されるプロシージャー。

F

フィールド **(field)**
データの特定のカテゴリーを保管するのに使用されるレコード内の領域。

ファイル **(file)**
レコードの順序列。外部ファイル *(external file)*、内部ファイル *(internal file)* も参照。

ファイル索引 **(file index)**
*i-*ノード *(i-node)* を参照。

最終サブルーチン **(final subroutine)**
ファイナライズ中に自動的に呼び出されるサブルーチン。

ファイナライズ可能 **(finalizable)**
最終サブルーチンを持っている型、またはファイナライズ可能なコンポーネントを持つ型。ファイナライズ可能な型のオブジェクト。

最終化 **(finalization)**
オブジェクトを破棄する直前に、ユーザー定義の最終サブルーチンを呼び出すプロセス。

浮動小数点数 **(floating-point number)**
異なる数表示の対で表される実数。実数は、数表示の 1 つである小数部と、暗黙的な浮動小数点の基数を 2 番目の数表示で示される数値でべき乗することによって得られる値との積。

フォーマット **(format)**
文字、フィールド、行などの配置を定義すること。通常は、表示、印刷出力、ファイルなどのために使用される。

文字、フィールド、行などを配置すること。

定様式データ **(formatted data)**
指定の形式に従って、主記憶装置と入出力装置間で転送されるデータ。リスト指示 *(list-directed)* および 不定形式レコード *(unformatted record)* を参照。

関数 **(function)**
単一の変数またはオブジェクトの値を戻すプロシージャー。通常は単一の出口を持つ。組み込みプロシージャー *(intrinsic procedure)*、サブプログラム *(subprogram)* も参照。

G

総称識別子 **(generic identifier)**
INTERFACE 文に存在する字句トークン。インターフェース・ブロック内のプロシージャーすべてに関連する。

H

ハード制限 **(hard limit)**
ルート権限を使用することによって上下のみができる、またはシステムや稼働環境のインプリメンテーション固有の問題であるため変更ができないシステム・リソースの限界。ソフト限界 *(soft limit)* も参照。

16 進 **(hexadecimal)**
システムに関連する基数が 16 の数字。 16 進数は、0 から 9 と A (10) から F (15) の範囲にある。

16 進定数 **(hexadecimal constant)**
通常は、特殊文字で始まる定数。16 進数字のみを含む。

上位変換 **(high order transformations)**
最適化の一種で、ループおよび配列言語を構造化し直す。

ホレリス定数 **(Hollerith constant)**
XL Fortran による表現が可能な任意の文字のストリングで、 *n***H** で始まるもの。ここで、*n* はストリング内の文字数を示す。

ホスト **(host)**
内部プロシージャーを含むメインプログラムまたはサブプログラムは、内部プロシージャーのホストと呼ばれる。モジュール・プロシージャーを含むモジュールまたはサブモジュールは、モジュール・プロシージャーのホストと呼ばれる。モジュールまたはサブモジュールは、その下位サブモジュールのホストと呼ばれる。

親子結合 **(host association)**
内部サブプログラム、モジュール・サブプログラム、派生型の定義、またはサブモジュールが、ホストのエンティティーにアクセスするためのプロセス。

ホスト・インスタンス **(host instance)**
内部プロシージャーのホスト環境を提供するホスト・プロシージャーのインスタンス。

I

IPA プロシージャー間分析 (Interprocedural analysis)。最適化の一種で、プロシージャーの境界を超えて、また、別々のソース・ファイルに入ったプロシージャー呼び出しにまたがって最適化を行うことができる。

暗黙インターフェース **(implicit interface)**
プロシージャーそのものからではなく、有効範囲単位から参照されるプロシージャーは、暗黙インターフェースを持つといわれる。ただしこれは、このプロシージャーが、インターフェース・ブロックを持たない外部プロシージャー、インターフェース・ブロックを持たないダミー・プロシージャー、ステートメント関数のいずれかである場合のみ。

暗黙 **DO (implied DO)**
指標付け仕様 (**DO** 文と似ているが、**DO** という語の指定はない)。この範囲は、ステートメントのセットではなく、データ・エレメントのリストである。

無限大 **(infinity)**
オーバーフローまたはゼロ割り算で作成された IEEE 数 (正または負)。無限大は、すべてのビットが 1 の指数部とゼロの小数部で表される。

継承 **(inherit)**
親から取得すること。拡張された型において明示宣言なしで、親タイプから自動的に取得される拡張された型の型パラメーター、コンポーネント、またはプロシージャー・バインディングは、継承と呼ばれる。

継承関係 **(inheritance association)**
拡張された型での継承されたコンポーネントと親コンポーネント間の関係。

i-ノード **(i-node)**
オペレーティング・システム内の個別のファイルを説明する内部構造体。各ファイルには少なくとも 1 つの i ノードがある。 i ノードには、ファイルのノード、タイプ、所有者、および位置が含まれる。 i ノードのテーブルは、ファイル・システムの先頭近くに格納される。ファイル索引 *(file index)* とも呼ばれる。

入出力 **(input/output)**
入力または出力、あるいはその両方に関するもの。

入出力リスト **(input/output list)**
入力または出力文内の変数のリスト。読み取りまたは書き込みを行うデータを指定する。出力リストには、定数、演算子、または関数参照を含む式、括弧で囲まれた式のいずれかが含まれることがある。

整定数 **(integer constant)**
任意で符号が付けられる数字ストリング。小数点は付けない。

インターフェース・ブロック **(interface block)**
INTERFACE 文から、対応する **END INTERFACE** 文までの文の順序列。

インターフェース本体 **(interface body)**
FUNCTION、**SUBROUTINE** のいずれかの文から、対応する **END** 文までの、インターフェース・ブロック内の文の順序列。

妨害 **(interference)**
DO ループ内の 2 つの反復内容が互いに依存している状態。

内部ファイル **(internal file)**
内部記憶域にある一連のレコード。 外部ファイル *(external file)* も参照。

プロシージャー間分析 **(interprocedural analysis)**
IPA を参照。

組み込み **(intrinsic)**
Fortran 言語標準によって定義済みでこれ以上の定義や仕様がなくてもどの有効範囲単位内でも使用できる型、演算、割り当てステートメント、プロシージャーを修飾する形容詞。

組み込みモジュール **(intrinsic module)**
コンパイラーによって提供され、どのプログラムでも使用可能なモジュール。

組み込みプロシージャー **(intrinsic procedure)**
コンパイラーによって提供され、どのプログラムでも使用可能なプロシージャー。

K

キーワード **(keyword)**
ステートメント・キーワードは、ステートメント (またはディレクティブ) の構文の一部の語で、ステートメントを識別するために使用する。

引数キーワードは、仮引数の名前を指定する。

kind 型パラメーター **(kind type parameter)**
組み込み型の使用可能な種類にラベルを付ける値を持つパラメーター、または **KIND** 属性を持つように宣言された派生型パラメーター。

L

字句エクステント **(lexical extent)**
ディレクティブ構成内に直接現れる全てのコード。

字句トークン **(lexical token)**
不可分の解釈を持つ文字の順序列。

リンク・エディット **(link-edit)**
ロード可能なコンピューター・プログラムをリンカーによって作成すること。

リンカー **(linker)**
別々にコンパイルまたはアセンブルされたオブジェクト・モジュール間の相互参照を解決し、最終アドレスを割り当て、再配置可能ロード・モジュールを作成するプログラム。単一のオブジェクト・モジュールがリンクされる場合には、リンカーはただ単純にそのモジュールを再配置可能にする。

リスト指示 **(list-directed)**
事前定義の入出力形式。データ・リスト内の型、型付きパラメーター、エンティティーの値に応じて異なる。

リテラル **(literal)**
ソース・プログラム内の記号または数量。データへの参照ではなく、データそのものを指す。

リテラル定数 **(literal constant)**
組み込み型のスカラー値を直接表す字句トークン。

ロード・バランシング **(load balancing)**
作業負荷を複数のプロセッサー間で均等に配分することを目的とした最適化ストラテジー。

論理定数 **(logical constant)**
真 (`true`) または偽 (`false`) (つまり、T または F) の値を持つ定数。

論理演算子 **(logical operator)**
次のような論理式の演算を表す記号。

```
.NOT.    (論理否定)
.AND.    (論理積)
.OR.     (論理和)
.EQV.    (論理等価)
.NEQV.   (論理非等価)
.XOR.    (排他的論理和)
```

ループ **(loop)**
繰り返し実行されるステートメント・ブロック。

M

_main
プログラマーがメインプログラムに名前を付けていない場合に、コンパイラーが割り当てるデフォルトの名前。

メインプログラム **(main program)**
プログラムの実行時に最初に制御が渡されるプログラム・ユニット。サブプログラム *(subprogram)* も参照。

マスター・スレッド **(master thread)**
スレッドのチームの先頭のプロセス。

モジュール **(module)**
ほかのプログラム単位からアクセスされる定義を含むプログラム単位、またはこの定義にアクセスするプログラム単位。

モジュール・プロシージャー・インターフェース本体 **(module procedure interface body)**
初期ステートメントに **MODULE** プレフィックス指定子が含まれるインターフェース本体。モジュール・プロシージャー・インターフェース本体は、個別モジュール・プロシージャー・インターフェース本体を指定する。

モジュール・サブプログラム **(module subprogram)**
モジュールまたはサブモジュールに含まれているものの、内部サブプログラムではないサブプログラム。モジュール・サブプログラムは、関数サブプログラム、サブルーチン・サブプログラム、または個別モジュール・サブプログラムである。

mutex
スレッド間の相互排他を提供するプリミティブ・オブジェクト。相互排他 (mutex) は、一度に連携スレッドのうちの確実に 1 つだけが共用データへのアクセスやアプリケーション・コードの実行を許されるようにするために、複数のスレッド間で調整的に使用される。

N

NaN (not-a-number)
数値に対応しない浮動小数点形式にエンコードされたシンボリック・エンティティー。静止 *NaN (quiet NaN)*、シグナル通知 *(signaling NaN)* も参照。

名前 **(name)**
最初が英文字で、その後に 249 文字までの英数字 (英文字、数字、下線) が続く字句トークン。 FORTRAN 77 では、シンボル名と呼ばれていたので注意。

名前付き共通ブロック **(named common)**
複数個の変数で構成される個別の名前付き共通ブロック。

名前リスト・グループ名 **(namelist group name)**
READ、WRITE、および PRINT 文で使用する名前のリストを指定する NAMELIST 文内の最初のパラメーター。

負のゼロ **(negative zero)**
指数および小数部が両方ともゼロであるが、符号ビットが 1 である IEEE 表記。負のゼロは正のゼロと等しいとして扱われる。

ネスト **(nest)**
ある種類の 1 つ以上の構造体を、同じ種類の構造体に組み込むこと。例えば、あるループ (ネストされるループ) を別のループ (ネストするループ) 内にネストしたり、あるサブルーチン (ネストされるサブルーチン) を別のサブルーチン (ネストするサブルーチン) 内にネストしたりする。

NEWUNIT 値 **(NEWUNIT value)**
-2 より小さく、現在接続されているどのファイルの装置番号とも等しくない負の数値。これは、**NEWUNIT=** 指定子によって指定された変数にランタイム・ライブラリーが割り当てる装置の値である。

実行不能ステートメント **(nonexecutable statement)**
プログラム単位、データ、編集情報、文関数のいずれかの特性を記述する文で、プログラムの実行処理には関係がないもの。

非既存ファイル **(nonexisting file)**
アクセス可能なストレージ・メディアに物理的には存在しないファイル。

正規 **(normal)**
非正規、無限大、または NaN でない浮動小数点数。

非数字 **(not-a-number)**
NaN を参照。

数値定数 **(numeric constant)**
整数、実数、複素数、バイト数のいずれかを表す定数。

数値記憶単位 **(numeric storage unit)**
デフォルト整数、デフォルト実数、およびデフォルト論理のタイプの非ポインター・スカラー・オブジェクトにより、専有される空間。

O

8 進 **(octal)**
システムに関連する基数が 8 の数字。8 進数は、0 から 7 の範囲にある。

8 進定数 **(octal constant)**
8 進数からなる定数。

1 トリップ **DO** ループ **(one-trip DO-loop)**
反復カウントが 0 でも、到達したら 1 回は実行される **DO** ループ。 (このループ・タイプは FORTRAN 66 からのものである。)

オンライン・プロセッサー **(online processor)**
マルチプロセッサー・マシンにおいて、活動化されている (オンラインにされている) 方のプロセッサー。オンライン・プロセッサーの個数は、マシンに実際にインストール済みの物理プロセッサーの個数より小か等しい。アクティブ・プロセッサー *(active processor)* とも呼ばれる。

演算子 **(operator)**
1 つまたは 2 つのオペランドが関係する個々の計算の仕様要素。

P

埋め込み **(pad)**
フィールドまたは文字ストリングの未使用の位置を、ダミー・データ (通常は、ゼロまたはブランク) で埋めること。

ページ・スペース **(paging space)**
仮想記憶域内に常駐しているが、現在はアクセスされていない情報を保管するためのディスク・ストレージ。

親コンポーネント **(parent component)**
継承された部分に対応する拡張された型のエンティティーのコンポーネント。

親タイプ **(parent type)**
拡張された型の派生元の拡張可能な型。

passed-object 仮引数 **(passed-object dummy argument)**
プロシージャーが呼び出されるオブジェクトと関連する type-bound プロシージャーまたはプロシージャー・ポインター・コンポーネントの仮引数。

PDF *Profile-Directed Feedback* を参照。

ポインティング先配列 **(pointee array)**
整数 **POINTER** 文またはその他の仕様ステートメントにより宣言されている、明示的形状配列、または、想定サイズ配列。

ポインター **(pointer)**
POINTER の属性を持つ変数。ポインターは、ターゲットに関連するものでなければ、参照したり、定義したりしてはならない。ポインターが配列である場合、関連するポインターでなければ、形状を持たない。

ポリモアフィック **(polymorphic)**
プログラム実行中に異なる型になれる。**CLASS** キーワードで宣言されるオブジェクトはポリモアフィックである。

事前接続ファイル **(preconnected file)**
実行可能プログラムの実行時に、最初に装置に接続されるファイル。標準エラー、標準入力、および標準出力はすべて事前接続ファイルである (それぞれ、装置 0、5、6 に接続される)。

暗黙規定 **(predefined convention)**
暗黙に指定されたデータ・オブジェクトの型指定と長さ指定。明示的な指定がない場合、名前の最初の文字が基準となる。最初の文字が I から N の場合、長さが 4 の整数型となる。最初の文字が A から H、O から Z、$、_ の場合、長さが 4 の実数型となる。

存在 **(present)**
ある仮引数が実引数と関連しており、かつ、この実引数が呼び出しプロシージャーに存在する仮引数である場合、または呼び出しプロシージャーの仮引数でない場合、この仮引数はサブプログラムのインスタンスに存在する。

1 次子 (primary)
式の最も単純な形式。オブジェクト、配列コンストラクター、構造体コンストラクター、関数参照、括弧で囲まれた式のいずれか。

プロシージャー (procedure)
プログラムの実行時に呼び出されることのある計算。プロシージャーは、関数またはサブルーチンの場合もある。また、組み込みプロシージャー、外部プロシージャー、モジュール・プロシージャー、内部プロシージャー、ダミー・プロシージャー、ステートメント関数などの場合もある。サブプログラムに **ENTRY** 文が含まれていると、このサブプログラムは複数のプロシージャーを定義することがある。

プロシージャー・バインディング (procedure binding)
「type-bound プロシージャー (type-bound procedure)」を参照。

プロシージャー・ポインター (procedure pointer)
EXTERNAL および **POINTER** 属性を持つプロシージャー・エンティティー。外部プロシージャー、モジュール・プロシージャー、ダミー・プロシージャーまたは別のプロシージャー・ポインターに関連するポインターとなる。

profile-directed feedback (PDF)
条件付き分岐や頻繁に実行されるコード・セクションのパフォーマンスを、アプリケーションの実行中に収集された情報を使用して改善する最適化の型。

プログラム状態 (program state)
プログラムの実行中の特定のポイントでのユーザー変数の値。

プログラム単位 (program unit)
メインプログラムまたはサブプログラム。

pure 副次作用がないことを示す、プロシージャーの属性。

Q

静止 NaN (quiet NaN)
例外をシグナル通知しない NaN (非数字) 値。静止 NaN の意図は、NaN の結果を後続の計算に伝えることにある。 *NaN*、シグナル通知 *(signaling NaN)* も参照。

R

ランダム・アクセス (random access)
ファイルからのまたはファイルへの、レコードの読み取り、書き込み、除去を、任意の順序で行うことができるアクセス方式。順次アクセス *(sequential access)* も参照。

ランク (rank)
配列のディメンション数。

実定数 (real constant)
実数を表す 10 進数のストリング。実定数には、小数点または 10 進指数、あるいはその両方が含まれている。

レコード (record)
ファイル内でまとめて扱われる値の順序列。

関係式 **(relational expression)**
算術式または文字式の次に関係演算子が続き、その次に別の算術式または文字式が続く式。

関係演算子 **(relational operator)**
関係条件または関係式を表すのに使用される語または記号。

```
.GT.        より大
.GE.        より大か等しい
.LT.        より小
.LE.        より小か等しい
.EQ.        等しい
.NE.        等しくない
```

結果変数 **(result variable)**
関数の値を戻す変数。

戻り指定子 **(return specifier)**
文 (例えば CALL 文) のために指定される引数で、サブルーチンが RETURN 文中に指定したアクションに応じて、どのステートメント・ラベルに制御を戻すべきかを示す引数。

S

スカラー **(scalar)**
配列ではない単一のデータ。

配列となるための特性を持たないもの。

スケール因数 **(scale factor)**
実数内での小数点の位置を示す番号 (入力の際に、指数がなければ、数の大きさを示す数字)。

有効範囲 **(scope)**
実行可能プログラムの一部分。この部分では、字句トークン 1 つにつき 1 つの解釈がある。

有効範囲属性 **(scope attribute)**
実行可能プログラムの一部分。この範囲内では、字句トークンには、特定の指定プロパティーまたはエンティティーの 1 つの解釈が与えられる。

有効範囲単位 **(scoping unit)**
派生型の定義。

BLOCK 構文 (ただし、これらにネストされる **BLOCK** 構文、派生型の定義、インターフェース本体は含まない)。

インターフェース本体。

プログラム単位またはサブプログラム (ただし、これらに含まれる派生型の定義、**BLOCK** 構文、インターフェース本体、サブプログラムは除く)。

セレクター **(selector)**
ASSOCIATE 構文内の関連名に関連付けられるオブジェクト。

セマンティクス **(semantics)**
複数の文字や複数文字の集合における、その意味上の関係。これは解釈方法や使用法からは独立している。構文規則 *(syntax)* も参照。

個別モジュール・プロシージャー **(separate module procedure)**
個別モジュール・サブプログラムによって定義されるモジュール・プロシー

ジャー、あるいは初期ステートメントに **MODULE** プレフィックス指定子が含まれる関数またはサブルーチン・サブプログラムによって定義されるモジュール・プロシージャー。個別モジュール・プロシージャーは、モジュール・プロシージャー・インターフェース本体を定義する。

個別モジュール・サブプログラム **(separate module subprogram)**
初期ステートメントに **MODULE PROCEDURE** ステートメントが含まれるモジュール・サブプログラム。個別モジュール・サブプログラムは、モジュール・プロシージャー・インターフェース本体を定義する。

順次アクセス **(sequential access)**
ファイル内のレコードの論理順序に従って、ファイルの読み取り、書き込み、除去を行うアクセス方式。ランダム・アクセス *(random access)* も参照。

シグナル **NaN (signaling NaN)**
オペランドとして現れるといつもそれを無効な演算例外としてシグナル通知する NaN (非数字)。シグナル NaN の意図は、初期化されていない変数の使用などのプログラム・エラーをキャッチすることにある。 *NaN*、静止 *NaN (quiet NaN)* も参照。

スリープ **(sleep)**
別のスレッドがそのスレッドに作業を実行するようにシグナルを送るまで実行が完全に中断されている状態。

SMP 対称マルチプロセッシング *(symmetric multiprocessing)* を参照。

ソフト制限 **(soft limit)**
処理に対して現在有効なシステム・リソースの制限。ソフト制限の値は、ルート権限がなくても処理によって拡大または緩和できる。リソースに対するソフト制限は、ハード制限の設定値を超えて拡大することはできない。ハード限界 *(hard limit)* も参照。

スピル・スペース **(spill space)**
レジスターに保持する変数の数が多過ぎて、プログラムがレジスターの内容用の一時ストレージを必要とする場合に備えて、個々のサブプログラムに確保するスタック空間。

仕様ステートメント **(specification statement)**
ソース・プログラムで使用されているデータについての情報を提供する文。この文は、データ・ストレージを割り振るための情報も提供する。

スタンザ **(stanza)**
ファイル内の行グループのことで、この行グループは共通の機能を持っているか、あるいはシステムの一部を定義している。スタンザは通常ブランク行かコロンで分離されており、各スタンザには名前が付いている。

ステートメント **(statement)**
実行処理の順序列または宣言のセット内で、1 つのステップを表す言語構文。文には大きく分けて、実行可能と実行不能の 2 つのクラスがある。

文関数 **(statement function)**
後ろに仮引数のリストが続く名前。これは、組み込み式または派生型の式と等価であり、プログラム全体にわたってこれらの式の代わりに使用することができる。

ステートメント・ラベル **(statement label)**
ステートメントを判別するために使用される 1 から 5 桁の番号。ステートメント・ラベルは、制御権の移動、**DO** の範囲の定義、**FORMAT** 文への参照のために使用することができる。

ストレージ関連付け **(storage association)**
2 つのストレージ順序列間の関係 (ただしこれは、一方の記憶装置がもう一方の記憶装置と同一の場合のみ)。

構造体 **(structure)**
派生型のスカラー・データ・オブジェクト。

構造体コンポーネント **(structure component)**
その型のコンポーネントに対応する、派生型のデータ・オブジェクトの一部。

サブモジュール **(submodule)**
モジュールまたは別のサブモジュールを拡張するプログラム単位。サブモジュールは、その上位モジュールまたはサブモジュールから親子結合経由で定義にアクセスする。その下位サブモジュールによって親子結合経由でアクセスされる定義が含まれる場合がある。さらに、上位モジュールまたはサブモジュール内で宣言されたモジュール・プロシージャー・インターフェース本体を定義する個別モジュール・プロシージャーが含まれる場合もある。

サブオブジェクト **(subobject)**
名前付きデータ・オブジェクトの一部。ほかの部分とは別々に参照されたり、定義されたりすることがある。配列エレメント、配列セクション、構造体コンポーネント、サブストリングのいずれか。

サブプログラム **(subprogram)**
関数サブプログラムまたはサブルーチン・サブプログラム。 FORTRAN 77 では、ブロック・データ・プログラム単位は、サブプログラムと呼ばれていたので注意。メインプログラム *(main program)* も参照。

サブルーチン **(subroutine)**
CALL 文または定義された割り当てステートメントから呼び出されるプロシージャー。

添え字 **(subscript)**
括弧で囲まれた添え字エレメントまたは添え字エレメントのセット。特定の配列エレメントを識別する配列名とともに使用される。

サブストリング **(substring)**
スカラー文字ストリングの連続する一部分。 (配列セクションでは、サブストリング・セレクターを指定することができるが、結果はサブストリングにはならない。)

対称マルチプロセッシング **(symmetric multiprocessing、SMP)**
機能的に同一の複数プロセッサーを並列に使用して、単純で効率的なロード・バランシングを提供するシステム。

同期 **(synchronous)**
別のプロセス中の指定されたイベントの出現に合わせて、定期的に出現または出現を予見できる操作の形容。

構文 **(syntax)**
文の構造に関する規則。セマンティクス *(semantics)* も参照。

T

ターゲット **(target)**
TARGET 属性を持つように指定された名前付きのデータ・オブジェクト。ポインター用に **ALLOCATE** 文によって作成されるデータ・オブジェクト、またこのようなオブジェクトのサブオブジェクト。

スレッド **(thread)**
プロセスを制御している、コンピューター命令ストリーム。マルチスレッドのプロセスは、1 ストリームの命令 (1 スレッド) で開始して、その後、タスクを実行するために他の命令ストリームを作成することができる。

thread-visible 変数 **(thread-visible variable)**
複数のスレッドからアクセス可能な変数。

タイム・スライス **(time slice)**
タスクを実行するために割り当てられる、処理装置上の時間間隔。その時間間隔が満了すると、処理装置時間は別のタスクに割り振られるため、 1 つのタスクが一定の制限時間を超えて処理装置を独占することはできなくなる。

トークン **(token)**
プログラム言語において、特別な形式の中に、或る定義済みの重みを持つ文字ストリング。

トリガー定数 **(trigger constant)**
コメント行をコンパイラーのコメント・ディレクティブとして識別する文字列。

type-bound プロシージャー **(Type-bound procedure)**
型定義でのプロシージャー・バインディング。プロシージャーは、binding-name によって、その動的タイプのオブジェクトを介して、定義された演算子として、定義された割り当てによって、または最終化プロセスの一部として参照される。

互換性のある型 **(type compatible)**
すべてのエンティティーは、同じ型の他のエンティティーと互換性のある型である。制限のないポリモアフィック・エンティティーは、すべてのエンティティーと互換性のある型である。他のポリモアフィック・エンティティーは、動的タイプが、ポリモアフィック・エンティティーの宣言された型の拡張型であるエンティティーと互換性のある型である。

型宣言ステートメント **(type declaration statement)**
オブジェクトまたは関数の、型、長さ、および属性を指定する文。オブジェクトには、初期値を割り当てることができる。

型パラメーター **(type parameter)**
データ型のパラメーター。**KIND** および **LEN** は、組み込み型の型パラメーターである。派生型の型パラメーターには、 **KIND** または **LEN** 属性のいずれかがある。

注**:** 派生型の型パラメーターは、派生型の定義で定義される。

U

不定形式レコード **(unformatted record)**
内蔵記憶装置と外部記憶装置間で変更されずに伝送されるレコード。

ユニコード **(Unicode)**
現代のどの言語で書かれたテキストについても、交換、処理、表示をサポートする、汎用文字エンコード標準。この標準は、多数の言語による多くの古典的でヒストリカルなテキストもサポートしている。ユニコード標準は、ISO 10646 で定義済みの 16 ビットの国際文字セットを持っている。*ASCII* も参照。

装置 **(unit)**
入出力ステートメントで使用するためにファイルを参照する手段。装置は、ファイルに接続されるものと接続されないものとがある。接続されている場合には、ファイルを参照する。この接続は対称的である。つまり、装置がファイルに接続されていると、このファイルは装置に接続されていることになる。

アンセーフ・オプション **(unsafe option)**
不正なコンテキストで使用した場合に望ましくない結果を生じる可能性があるオプション。それ以外のオプションは、デフォルトの結果とあまり異ならない、通常は許容される結果を生じる。通常、アンセーフ・オプションを使用すると、該当するコードがそのオプションをアンセーフにする条件に制約されないと断言していることになる。

参照結合 **(use association)**
別々の有効範囲単位内での名前の関連付け。**USE** 文で指定される。

V

変数 **(variable)**
定義可能な値を持つデータ・オブジェクト。この値は、実行可能プログラムの実行時に再定義することができる。名前付きデータ・オブジェクト、配列エレメント、配列セクション、構造体コンポーネント、サブストリングのいずれか。 FORTRAN 77 では、変数は必ずスカラーで、名前が付けられていたことに注意。

X

XPG4 X/Open Common Applications Environment (CAE) Portability Guide Issue 4。XPG3 から POSIX 標準への拡張機能が含まれている、POSIX.1-1990、POSIX.2-1992、および POSIX.2a-1992 のスーパーセットである X/Open Common Applications Environment のインターフェースを定義する文書。

Z

ゼロ長文字 **(zero-length character)**
長さが 0 の文字オブジェクト。常に定義される。

ゼロ・サイズ配列 **(zero-sized array)**
下限を持つ配列。これは、対応する上限より大きい。この配列は、常に定義される。

特記事項

プログラミング・インターフェース: 所定のプログラミング・インターフェースを使用することにより、お客様は IBM XL Fortran for Linux のサービスを使用するためのプログラムを書くことができます。

本書は米国 IBM が提供する製品およびサービスについて作成したものです。本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

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

以下の保証は、国または地域の法律に沿わない場合は、適用されません。 IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

Intellectual Property Dept. for Rational Software
IBM Corporation
5 Technology Park Drive
Westford, MA 01886
U.S.A.

本プログラムに関する上記の情報は、適切な使用条件の下で使用することができますが、有償の場合もあります。

本書で説明されているライセンス・プログラムまたはその他のライセンス資料は、IBM 所定のプログラム契約の契約条項、IBM プログラムのご使用条件、またはそれと同等の条項に基づいて、IBM より提供されます。

この文書に含まれるいかなるパフォーマンス・データも、管理環境下で決定されたものです。そのため、他の操作環境で得られた結果は、異なる可能性があります。一部の測定が、開発レベルのシステムで行われた可能性がありますが、その測定値が、一般に利用可能なシステムのものと同じである保証はありません。さらに、一部の測定値が、推定値である可能性があります。実際の結果は、異なる可能性があります。お客様は、お客様の特定の環境に適したデータを確かめる必要があります。

IBM 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。 IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。 IBM 以外の製品の性能に関する質問は、それらの製品の供給者にお願いします。

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

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、

利便性もしくは機能性があることをほのめかしたり、保証することはできません。これらのサンプル・プログラムは特定物として現存するままの状態で提供されるものであり、いかなる保証も提供されません。 IBM は、お客様の当該サンプル・プログラムの使用から生ずるいかなる損害に対しても一切の責任を負いません。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、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. 1998, 2018.

このソフトウェアならびに文書は、その一部をカリフォルニア大学評議員の承諾のもとに提供された「Fourth Berkeley Software Distribution」に基づきます。これの開発にあたった次の研究機関に対し、敬意を表します。: Electrical Engineering and Computer Sciences Department、バークレー・キャンパス。

プライバシー・ポリシーに関する考慮事項:

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

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

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

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

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、http://www.ibm.com/legal/copytrade.shtml をご覧ください。

Adobe および Adobe 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。

Linux は、Linus Torvalds の米国およびその他の国における登録商標です。

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。

Windowsは、Microsoft Corporation の米国およびその他の国における商標です。

索引

日本語, 数字, 英字, 特殊文字の順に配列されています。なお, 濁音と半濁音は清音と同等に扱われています。

[ア行]

[カ行]

[サ行]

［タ行］

［ナ行］

［ハ行］

［マ行］

［ヤ行］

［ラ行］

［数字］

A

B

C

F

L

P

Q

S

X

［特殊文字］

プログラム番号: 5765-J10; 5725-C75

Printed in Japan

SC43-4663-00

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