IBM XL Fortran Advanced Edition V10.1 for Linux

コンパイラー解説書

SD88-6732-00
(英文原典：SC09-8019-00)

IBM XL Fortran Advanced Edition V10.1 for Linux

コンパイラー解説書

SD88-6732-00
(英文原典：SC09-8019-00)

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、 295 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 Advanced Edition V10.1 for Linux の Version 10.1 (プログラム番号 5724-M17) および新しい版で明記されていない限り、以降のすべてのリリースおよびモディフィケーションに適用されます。製品のレベルに合った版であることを確かめてご使用ください。

本マニュアルに関するご意見やご感想は、次の URL からお送りください。今後の参考にさせていただきます。

`http://www.ibm.com/jp/manuals/main/mail.html`

なお、日本 IBM 発行のマニュアルはインターネット経由でもご購入いただけます。詳しくは

`http://www.ibm.com/jp/manuals/` の「ご注文について」をご覧ください。

(URL は、変更になる場合があります)

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

原　典： SC09-8019-00
IBM XL Fortran Advanced Edition V10.1 for Linux
Compiler Reference

発　行： 日本アイ・ビー・エム株式会社

担　当： ナショナル・ランゲージ・サポート

第1刷 2005.10

この文書では、平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、平成角ゴシック体™W5、および平成角ゴシック体™W7を使用しています。この(書体*)は、(財) 日本規格協会と使用契約を締結し使用しているものです。フォントとして無断複製することは禁止されています。

注* 平成明朝体™W3、平成明朝体™W7、平成明朝体™W9、平成角ゴシック体™W3、
平成角ゴシック体™W5、平成角ゴシック体™W7

目次

本書について

本書は、IBM® XL Fortran Advanced Edition V10.1 for Linux® コンパイラーについて記述し、Fortran 言語で作成されるプログラムのコンパイル環境の設定、コンパイル方法、リンク方法、および実行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、XL Fortran コンパイラーを使用して作業する必要がある方々を対象としていますが、Linux オペレーティング・システムに精通し、ある程度 Fortran でのプログラミング経験があることを前提としています。 XL Fortran 慣れていないユーザーは 本書を使用して XL Fortran に固有の可能性とフィーチャーについての情報を検索できます。本書は、コンパイラーの各フィーチャー (特にオプション) について理解し、それらを使用して効率的にソフトウェアを開発する方法を理解するのに役立ちます。

本書の使用方法

本書では、XL Fortran プログラムのコンパイラーの構成、コンパイル、リンクと実行についての情報をカバーしますが、他の資料で扱われる次のトピックは、説明していません。

- インストール、システム要件、タイムリミット寸前の更新。「*XL Fortran インストール・ガイド*」および製品の README を参照してください。
- XL Fortran 機能の概説。「*XL Fortran Advanced Edition V10.1 for Linux 入門*」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 言語解説書*」を参照してください。
- 最適化、移植、OpenMP/ SMP プログラミング。「*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するオペレーティング・システムのコマンド。ご使用の Linux 固有の配布のマニュアル・ページのヘルプおよび資料を参照してください。

本書の構成

本書ではコンパイラーの概説から開始し、コンパイラーを起動する前に実行する必要があるタスクについての情報を提供します。次に、コンパイラー・オプションとデバッグ問題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。

- 『第 1 章 はじめに』 から 『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、コンパイル環境の設定、異なるコンパイル・モードで必要となる環境変数、構成ファイルのカスタマイズ、入出力ファ

イルのタイプ、コンパイラー・リストとメッセージおよびプリプロセッサーとリンケージ・エディターの起動に固有の情報です。

- 『第 5 章 XL Fortran コンパイラー・オプションの解説』 には、機能カテゴリー毎にコンパイラー・オプションを要約したテーブルが含まれています。名前によるオプションの検索、またはその代わりに機能カテゴリー・テーブルのリンクおよび機能性によるルックアップ・オプションの使用が可能です。このセクションには、アルファベット順にソートされた、個々のコンパイラー・オプションの説明も含まれています。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。
- 『第 6 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。
- 『第 7 章 問題判別とデバッグ』 では、デバッグおよびコンパイラー・リストの情報を扱います。
- 『付録 A. XL Fortran 技術情報』 および 『付録 B. XL Fortran 内部制限』 では、上級のプログラマーが異常な問題の診断、特殊な環境でコンパイラーの実行の際に必要となる情報を提供します。

本書で使用される規則と用語

書体の規則

次のテーブルでは本書で使用される表記上の規則を説明します。

表 1. 書体の規則

書体	示す	例
太字	コマンド、実行可能プログラム名、およびコンパイラー・オプション。	デフォルトで、これらの呼び出しコマンドとともに **-qsmp** コンパイラー・オプションを使用する場合、オプション **-qdirective=IBM*:SMP$:$OMP:IBMP:IBMT** が指定されます。
イタリック体	パラメーターまたは変数で、実際の名前または値をユーザーが提供する必要があります。イタリック体は、新規の用語を照会するためにも使用されます。	固定ソース・フォームの *trigger_constant* の最大長は、ディレクティブに対しては 4 であり、1 つ以上の行に続けられます。
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。	**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。
小文字の太字	小文字のプログラミング・キーワードおよびライブラリー関数、コンパイラー組み込みプロシージャー、ファイル名およびディレクトリー名、プログラム・コードの例、コマンド・ストリング、またはユーザー定義の名前。	未初期化のロック変数を使用して **omp_destroy_lock** を呼び出した場合、呼び出しの結果は未定義です。

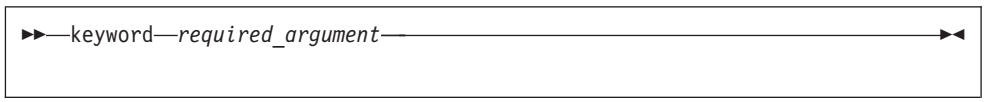
構文図の読み方

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、これらのダイアグラムの解釈と使用に役立ちます。

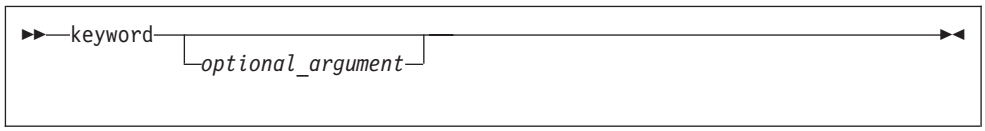
変数名またはユーザー指定の名前が *_list* で終わっている場合は、これらの項のリストをコンマで区切って指定できます。

句読記号、括弧、算術演算子、その他の特殊文字は、構文の一部として入力する必要があります。

- 構文図は線の経路に沿って、左から右へ、上から下へ読みます。
  - ▶▶── 記号は文の始まりを示します。
  - ──▶ 記号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。
  - ▶── 記号は文が前の行から続いていることを示します。
  - ──▶◀ 記号は文の終わりを示します。
  - プログラム単位、プロシージャー、構造体、インターフェース・ブロック、および派生型定義は、複数の個別の文から構成されています。そのような項目の場合、構文表記は囲み線で囲まれ、個々の構文図は、同等の Fortran ステートメントの必須の指定順序を示します。
  - 言語標準への IBM XL Fortran 拡張および言語標準の実装は構文図に番号でマークされ、ダイアグラムの直後に説明的な注記が書かれます。
- 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。

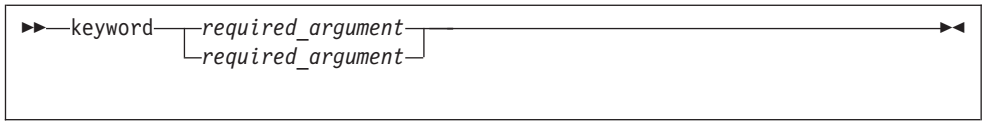


- オプション項目は、次のようにメインパスの下側に表示されます。

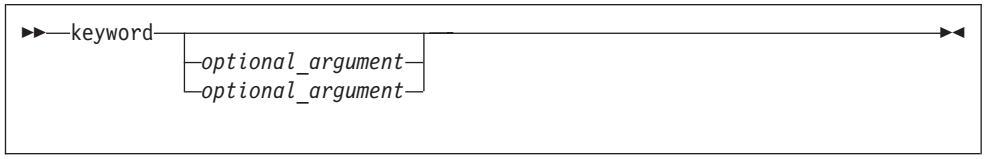


  注: オプション項目 (構文図にはない) は大括弧 ([および]) で囲まれます。たとえば、`[UNIT=]u`
- 複数の項目から選択できる場合は、縦方向に積み重ねて表示されます。

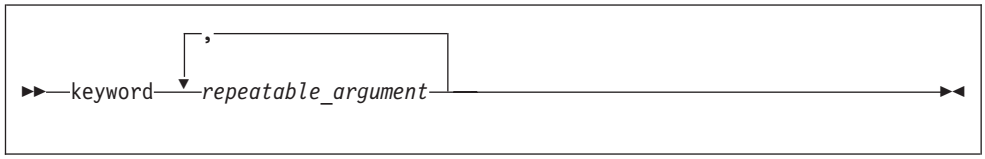
  項目のうちの 1 つを選択する必要がある場合、スタックのうちの 1 項目がメインパス上に表示されます。



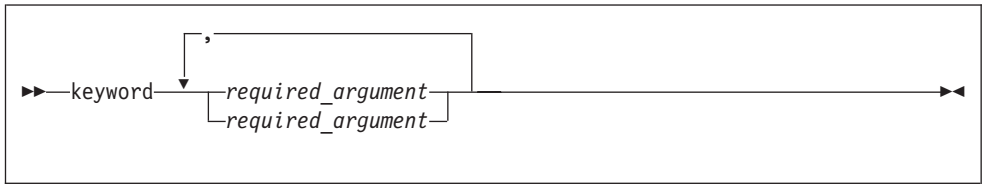
複数の項目からの選択がオプションの場合は、スタック全体をメインパスの下側に記述します。



- メインパスの上側に記述してある左へ戻る矢印 (反復矢印) は、繰り返し可能な項目を示し、ブランクでない場合は区切り文字が記述されます。

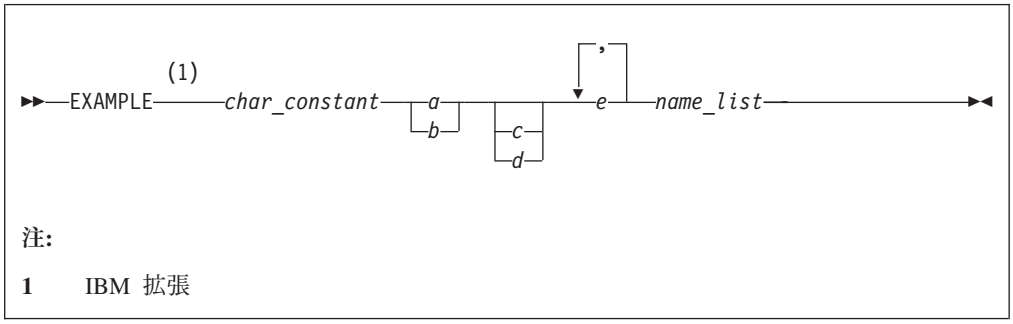


縦の並びの上側にある繰り返し矢印は、縦に積み重ねられている項目から複数の項目を選択できることを示しています。



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。



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 (...) が示されます。

構文図の例

```
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。

例

本書で使用する例は、ストレージの節約、エラーのチェック、実行の高速化、推奨される実務の実行方法を示すことなどは目的としていないので、簡易なスタイルでコーディングされています。

本書中の例では、コンパイラー呼び出しコマンド **xlf90**、**xlf90_r**、**xlf95**、**xlf95_r**、**xlf**、**xlf_r**、**f77**、**fort77**、**f90**、および **f95** のどれを使用してもかまいません。実際のソース・ファイルの場合は、17 ページの『XL Fortran プログラムのコンパイル』に説明されているように、コマンドによっては、他のものより適切なものもあります。

本書のサンプル・プログラム、および本書の記載事項を例証しているプログラムの一部は、 **/opt/ibmcmp/xlf/10.1/samples** ディレクトリーに入っています。

パス名についての注意事項

本書で使用されているパス名は、XL Fortran コンパイラーのデフォルト・インストール・パスを想定しています。デフォルトでは、XL Fortran は、選択したディスク上の次のディレクトリーにインストールされます。

`/opt/ibmcmp/xlf/10.1`

これとは異なる宛先 (*relocation-path*) をコンパイラーに対して選択することができます。異なるパスを選択した場合、コンパイラーは次のディレクトリーにインストールされます。

relocation-path`/opt/ibmcmp/xlf/10.1`

使用される用語についての注意事項

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。

- *自由ソース形式フォーマット*という用語は、*自由ソース形式*と表現している場合があります。
- *固定ソース形式フォーマット*という用語は、*固定ソース形式*と表現している場合があります。
- *XL Fortran* という用語は、*XLF* と表現している場合があります。

関連情報

IBM XL Fortran の資料

XL Fortran は、以下の形式の製品資料を提供します。

- Readme ファイル

  Readme ファイルには最新の情報が含まれており、その中には製品の資料への変更と修正が組み込まれています。README ファイルは、デフォルトで、インストール CD の `/opt/ibmcmp/xlf/10.1` ディレクトリー、およびルート・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、製品で提供されているコンパイラー呼び出し、およびすべてのコマンド行ユーティリティー用に提供されます。 man ページのインストールとアクセスの説明は、「*IBM XL Fortran Advanced Edition V10.1 for Linux インストール・ガイド*」で提供されています。

- インフォメーション・センター

  検索可能な HTML ファイルのインフォメーション・センターは、ネットワークで起動でき、リモート側でまたはローカル側でアクセスされます。インフォメーション・センターのインストールとアクセスの説明は 「*XL Fortran インストール・ガイド*」で提供されています。インフォメーション・センターは次の Web サイトでも表示可能です。

  http://publib.boulder.ibm.com/infocenter/lnxpcomp/index.jsp

- PDF 文書

  PDF 文書は、デフォルトで /opt/ibmcmp/xlf/10.1/doc/*language*/pdf ディレクトリーにあり、ここで *language* は次のサポートされる言語です。
  - en_US (米国英語)
  - ja_JP (日本語)

  PDF 文書は次の Web サイトでも入手可能です。
  http://www.ibm.com/software/awdtools/fortran/xlfortran/library

  この資料に加えて、次のファイルは XL Fortran 製品マニュアル全セットを構成します。

表 2. XL Fortran PDF ファイル

資料タイトル	PDF ファイル名	説明
IBM XL Fortran Advanced Edition V10.1 for Linux インストール・ガイド	install.pdf	XL Fortran のインストール、基本コンパイルおよびプログラム実行のユーザーの環境の構成の情報が含まれています。
XL Fortran Advanced Edition V10.1 for Linux 入門	getstart.pdf	XL Fortran 製品の紹介を含み、ご使用の環境の設定と構成、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ついての概要情報もあります。
IBM XL Fortran Advanced Edition V10.1 for Linux コンパイラー解説書	cr.pdf	ご使用のコンパイル環境の設定と構成、プログラムのコンパイル、リンクおよび実行、コンパイル・エラーのトラブルシューティング、および種々のコンパイラー・オプションの説明が含まれています。
IBM XL Fortran Advanced Edition V10.1 for Linux 言語解説書	lr.pdf	IBM によってサポートされる Fortran プログラム言語の情報が含まれ、移植性に対する言語拡張、非機密の標準規格への準拠、コンパイラー指示、および組み込みプロシージャー含まれています。
IBM XL Fortran Advanced Edition V10.1 for Linux 最適化およびプログラミング・ガイド	opg.pdf	アプリケーションの移植、上級プログラミング・トピックが含まれ、その中にはアプリケーションの移植、言語間呼び出し、浮動小数点操作、入出力、アプリケーションの最適化と並列化、および XL Fortran のハイパフォーマンス・ライブラリーなども解説されてされています。

  PDF ファイルは、Adobe Reader を使用して表示可能および印刷可能です。
  Adobe Reader をインストールしていない場合は、http://www.adobe.com からダウンロードすることができます。

追加資料

Redbooks、ホワイトペーパー、チュートリアル、およびその他の記事を含む、XL Fortran に関連する追加の資料は、次の Web サイトで入手できます。

http://www.ibm.com/software/awdtools/fortran/xlfortran/library

オペレーティング・システムおよびその他の資料は次のとおり入手可能です。

- URL http://www.rpm.org/ の ***RPM home page*** では、RPM Package Manager (RPM) を使用しての標準 Linux インストール手順を扱います。
- 一般情報および Linux についての資料は、URL http://www.tldp.org/ の ***The Linux Documentation Project*** にアクセスしてください。ご使用のオペレーティング・システムおよびそのフィーチャーの使用法についての情報は、ご使用の Linux 固有の配布のマニュアル・ページのヘルプおよび資料を参照してください。
- Linux の IBM 関連のオファリングについては、URL http://www.ibm.com/linux/ の ***Linux at IBM*** にアクセスしてください。
- 「*System V Application Binary Interface: PowerPC Processor Supplement*」は汎用 System V ABI についての補足であり、32 ビット・モードで作動している PowerPC アーキテクチャー™に構築された System V インプリメンテーションに固有の情報が含まれています。
- 「*64-bit PowerPC ELF Application Binary Interface Supplement*」は汎用 System V ABI についての補足であり、32 ビット・モードで作動している PowerPC アーキテクチャーに構築された System V インプリメンテーションに固有の情報が含まれています。

関連資料

本書でも言及される次の資料を参照したい場合があります。

- *OpenMP アプリケーション・プログラム・インターフェース バージョン 2.5*, available at http://www.openmp.org
- *ESSL for Linux on POWER V4.2 ガイドおよび解説*

規格資料

XL Fortran は次の規格に従って設計されています。本書で見られるいくつかの機能の厳密な定義については、以下の規格資料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. (本書では Fortran 90 という略式名を使用します。)
- *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 (E).
- *Information technology - Programming languages - Fortran - Part 1: Base language*, ISO/IEC 1539-1:1997. (本書では Fortran 95 という略式名を使用します。)
- *Information technology - Programming languages - Fortran - Floating-Point Exception Handling*, ISO/IEC JTC1/SC22/WG5 N1379.
- *Information technology - Programming languages - Fortran - Enhance Data Type Facilities*, ISO/IEC JTC1/SC22/WG5 N1378.
- *Military Standard Fortran DOD Supplement to ANSI X3.9-1978*, MIL-STD-1753 (United States of America, Department of Defense standard). Note that XL Fortran

supports only those extensions documented in this standard that have also been subsequently incorporated into the Fortran 90 standard.

- *OpenMP Application Program Interface, Version 2.5, (May 2005)* specification.

テクニカル・サポート

XL Fortran サポート・ページから追加のテクニカル・サポートが使用可能です。このページでは、テクニカル・サポート FAQ およびその他のサポート資料の大規模選択への検索機能を持つポータルが提供されます。次の Web サイトで XL Fortran サポート・ページを検索できます。

http://www.ibm.com/software/awdtools/fortran/xlfortran/support

必要とするものが検索できない場合、E メールを送信することができます。

compinfo@ca.ibm.com

XL Fortran の最新情報については、以下の製品情報サイトにアクセスしてください。

http://www.ibm.com/software/awdtools/fortran/xlfortran

第 1 章 はじめに

IBM XL Fortran Advanced Edition V10.1 for Linux は Linux オペレーティング・システム用の最適化を行う、標準ベースの、コマンド行コンパイラーであり、PowerPC® アーキテクチャーを使用して PowerPC ハードウェア上で実行されます。XL Fortran コンパイラーにより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 32 ビットおよび 64 ビット・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。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 PowerPC 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 プログラム言語のインプリメンテーションは、言語標準への規格合致の強制により、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。自身の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つでしょう。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 Advanced Edition V10.1 for Linux は大規模、複雑な、および計算的に集中的なプログラムに使用できます。また C/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。

第 2 章 XL Fortran の機能の概要

本節では、XL Fortran コンパイラー、言語、開発環境の機能について詳しく説明します。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

XL Fortran Advanced EditionVersion 10.1 コンパイラーは、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、「*IBM XL Fortran Advanced Edition V10.1 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。

コンパイラー、コンパイラーで生成されたオブジェクト・プログラム、およびランタイム・ライブラリーは、必要なソフトウェア、ディスク・スペース、および仮想記憶域を備えたすべての POWER3™、POWER4™、POWER5™、PowerPC 970、および PowerPC システムで稼動します。

POWER3、POWER4、あるいは POWER5、または POWER5+™ プロセッサーは、PowerPC の 1 つのタイプです。本書において、PowerPC に関する記述または参照は、POWER3、POWER4、あるいは POWER5、または POWER5+ 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

XL Fortran 言語は、次のもので構成されます。

- 米国規格協会 Fortran 90 言語の完全版 (Fortran 90 または F90 と呼ばれています)。「*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) に定義されています。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。
- 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。
- Fortran 95 標準に対する新規拡張機能
  - 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

- Fortran 2003 規格の部分的サポート

「*XL Fortran 言語解説書*」では、Fortran 95 言語の拡張機能は、『*書体の規則*』のトピックの説明のとおりにマークが付けられています。

マイグレーション・サポート

XL Fortran コンパイラーは、Fortran コンパイラー間でのソース・コードの移植またはマイグレーションに役立ちます。つまり、Fortran 90 および Fortran 95 の言語サポート (完全版)、および多種多様なコンパイラー・ベンダーから選択された言語拡張機能 (組み込み関数、データ型など) を提供します。本書では、このような拡張を「業界用拡張機能」と呼びます。

FORTRAN 77 ソース・コードへの投資を保護するために、XL Fortran の初期バージョンとの後方互換性を提供する一連のデフォルト値でコンパイラーを簡単に呼び出すことができます。 **xlf**、**xlf_r**、**f77**、および **fort77** コマンドは、既存の FORTRAN 77 プログラムに最大限の互換性を提供します。 **f90**、**xlf90** および **xlf90_r** コマンドと一緒に提供されるデフォルト・オプションを使用すると、Fortran 90 言語機能全体にアクセスできます。 **f95**、**xlf95** および **xlf95_r** コマンドと一緒に提供されるデフォルト・オプションを使用すると、Fortran 95 言語機能全体にアクセスできます。

さらに、**.f77**、**.f90**、または **.f95** のような拡張子を持つご使用のソース・ファイルの名前を付けることができ、**xlf** または **xlf_r** などの汎用コンパイラー呼び出しを使用して言語レベルの適切なデフォルト自動的に選択できます。

ソース・コードの適合性検査

Fortran 2003、 FORTRAN 77、Fortran 90、または Fortran 95 コンパイラーへの移植時、またはこれらのコンパイラーからの移植時に、問題発生の原因となるものをプログラム内から見つける手掛かりとして、 XL Fortran コンパイラーは、特定の Fortran 定義に準拠していない機能に関してユーザーに警告する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。

- 全 Fortran 2003 Standard (**-qlanglvl=2003std** オプション)、すべての米国標準規格 (ANS) FORTRAN 77 (**-qlanglvl=77std** オプション)。すべての米国標準規格 (ANS) Fortran 90 (**-qlanglvl=90std** オプション)。すべての Fortran 95 標準 (**-qlanglvl=95std** 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)
- 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 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)
- IBM SAA® FORTRAN (**-qsaa** オプション)

また、**langlvl** 環境変数を使って、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 2003 規格合致検査は、XL Fortran の現行、この標準のサブセット・インプリメンテーションに基づいています。

高度な構成が可能なコンパイラー

コンパイラーを起動するには、 **xlf**、**xlf_r**、**xlf90**、**xlf90_r**、**xlf95**、**xlf95_r**、 **f77**、または **fort77** コマンドを使用します。 **xlf**、**xlf_r**、 および **f77** コマンドは、XL Fortran バージョン 2 の動作および I/O 形式と最大限の互換性を維持しています。**f90**、**xlf90** および **xlf90_r** コマンドを使用すると、Fortran 90 への準拠性が高まり、効率と使いやすさの向上に役立つインプリメンテーションの選択肢が提供されます。 **f95**、**xlf95** および **xlf95_r** コマンドを使用すると、95 への準拠性が高まり、効率と使いやすさの向上に役立つインプリメンテーションの選択肢が提供されます。 **f77** または **fort77** コマンドは、XPG4 動作との最大の互換性を提供します。

一連の **xlf_r**、**xlf90_r**、および **xlf95_r** コマンドと、一連の **xlf**、**xlf90**、**xlf95**、**f77**、 **fort77**、**f90**、および **f95** コマンドの主な相違点は、前者のコマンド群は、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) にリンクしてバインドするという点です。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。たとえば、次のようになります。

```
xlf -F/etc/opt/ibmcmp/xlf/10.1/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、単一のファイル **/etc/opt/ibmcmp/xlf/10.1/xlf.cfg** をカスタマイズすることができます。

関連情報

- 11 ページの『構成ファイルのカスタマイズ』
- 17 ページの『XL Fortran プログラムのコンパイル』
- 45 ページの『XL Fortran コンパイラー・オプションの概要』 および 73 ページの『XL Fortran コンパイラー・オプションの詳細記述』
- 268 ページの『XL Fortran メッセージに関する情報』

診断リスト作成

コンパイラー出力リスト作成の項は、お読みになっても、省略してもどちらでもかまいません。適用できるコンパイラー・オプションおよびリスト作成そのものについては、 58 ページの『リストとメッセージを制御するオプション』および 277 ページの『第 8 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、真のアセンブラー・ソース・ファイルが得られます。

シンボリック・デバッガーのサポート

プログラムに対して、**dbxgdb**、およびその他のシンボリック・デバッガーを使用することができます。

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制御するのに役立ちます。

- 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

本節では、 XL Fortran の設定値をあらゆるユーザーに合わせてカスタマイズする方法を説明します。フルインストール手順は、本節の範囲を超えており、本節では詳細なインストール手順をカバーする資料を紹介します。

さらに本節は、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. **/opt/ibmcmp/xlf/10.1/doc/en_US/README** 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配信す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。
3. この製品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。

   Linux 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。

ネットワーク・ファイル・システム上でのコンパイラーの使用

マシンのネットワーク・クラスターに対してネットワーク・ファイル・システム・サーバーにインストールされている XL Fortran コンパイラーを使用するには、ネットワーク・インストール・マネージャーを使用してください。

XL Fortran のコンポーネントは、次のディレクトリーに入っています。

- **/opt/ibmcmp/xlf/10.1/bin** には、コンパイラー呼び出しコマンドが入っています。
- **/opt/ibmcmp/xlf/10.1/exe** には、コンパイラーが必要とする実行モジュールおよびファイルが入っています。
- **/opt/ibmcmp/xlf/10.1/lib** と **/opt/ibmcmp/xlf/10.1/lib64** には、ライブラリー (再配布不可) が入っています。
- **/opt/ibmcmp/lib/** と **/opt/ibmcmp/lib64/** には、ライブラリー (再配布可能) が入っています。

- **/usr/include/opt/ibmcmp/xlf/10.1/include** には、インクルード・ファイルと、提供された **.mod** ファイルが含まれています。
- **/opt/ibmcmp/msg** には、再分散可能ランタイム・ライブラリーのメッセージ・カタログが入っています。
- **/opt/ibmcmp/xlf/10.1/doc/en_US/pdf** には、PDF 形式の XL Fortran 資料 (英語) が入っています。

  /opt/ibmcmp/xlf/10.1/doc/ja_JP/pdf には、PDF 形式の XL Fortran 資料 (日本語) が入っています。
- **/opt/ibmcmp/xlf/10.1/doc/en_US/html** には、HTML 形式の XL Fortran 資料 (英語) が入っています。

  /opt/ibmcmp/xlf/10.1/doc/ja_JP/html には、HTML 形式の XL Fortran 資料 (日本語) が入っています。

/etc/opt/ibmcmp/xlf/10.1/xlf.cfg ファイルをサーバーからクライアントにコピーする必要があります。 **/etc/opt/ibmcmp/xlf/10.1** ディレクトリーには、マシン固有の構成ファイルが入っているため、サーバーからはこのディレクトリーをマウントしないでください。

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。

各国語サポートのための環境変数

コンパイラーからの診断メッセージおよびリストは、オペレーティング・システムのインストール時に指定したデフォルトの言語で表示されます。メッセージおよびリストを別の言語で表示したい場合は、コンパイラーを実行する前に以下の環境変数を設定してエクスポートすることができます。

LANG これは*ロケール* を指定します。ロケールはカテゴリーに分類されます。個々のカテゴリーにはロケール・データの特定な面が含まれています。 **LANG** を設定すれば、すべてのカテゴリーに対して各国語を変更することができます。

NLSPATH これは、メッセージ・カタログを見つけるのに必要なディレクトリー名のリストを示します。

たとえば、日本語ロケールを指定するには、**LANG** 環境変数を **ja_JP** に設定します。

関連のあるメッセージ・カタログがインストールされている場合は、**ja_JP** の代わりに有効な各言語コードを使用してください。

オペレーティング・システムがインストールされると、これらの環境変数は初期化され、コンパイラーで使用したい環境変数とは異なる場合があります。

各カテゴリーは、それぞれ関連付けられている環境変数を持っています。特定のカテゴリーの各国語を変更したいが、他のカテゴリーの各国語は変更したくないという場合は、対応する環境変数を設定してエクスポートすることができます。

たとえば、次のようになります。

LC_MESSAGES
送出されるメッセージに対して各国語を指定します。これは、コンパイラーおよび XLF コンパイル済みプログラムからのメッセージに影響を与えます。これらのメッセージは画面に表示することもできますし、リスト、モジュール、またはその他のコンパイラー出力ファイルに格納することもできます。

LC_TIME
時刻形式カテゴリーに対して各国語を指定します。主にコンパイラー・リストに影響を与えます。

LC_CTYPE
文字の分類、大文字/小文字の変換、およびその他の文字属性を定義します。 XL Fortran の場合は、主にマルチバイト文字の処理に影響を与えます。

LC_NUMERIC
数値の入出力に使用する形式を指定します。この変数をシェルで設定しても、コンパイラーにも XLF コンパイル済みプログラムにも影響を与えません。 **LC_NUMERIC** カテゴリーは、プログラムの最初の I/O ステートメントによって **POSIX** に設定されているので、別の設定が必要なプログラムは、この箇所の後でリセットしてから、すべての I/O ステートメントに対して **POSIX** への設定を復元します。

注:

1. **LC_ALL** 環境変数を指定すると、 **LANG** およびその他の **LC_** 環境変数の値がオーバーライドされます。
2. XL Fortran コンパイラーまたはアプリケーション・プログラムがメッセージ・カタログにアクセスできなかったり、特定のメッセージを検索できない場合は、英語でメッセージが表示されます。
3. バックスラッシュ \ は、¥ 記号と同じ 16 進コード X'5C' を持っており、ロケールが日本語の場合はディスプレイに ¥ 記号として表示できます。

関連情報: 33 ページの『実行時メッセージ用の言語の選択』

各国語サポート環境変数およびロケールの概念についての詳細は、 Linux 固有の資料および man ページ・ヘルプを参照してください。

ライブラリー検索パスの設定

実行可能プログラムが共用ライブラリーにリンクされている場合、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、2 つの方法で設定できます。以下のいずれかです。

- **-R** (または **-rpath**) コンパイル/リンク・オプション
- **LD_LIBRARY_PATH** および **LD_RUN_PATH** 環境変数

コンパイル/リンク **-R** (または **-rpath**) オプションで検索パスを指定すると、指定のランタイム・ライブラリー検索パスが実行可能プログラムに書き込まれます。 **-L** オプションを使用した場合、ライブラリー検索パスはリンク時に検索されますが、実行可能プログラムへの、ランタイム・ライブラリー検索パスとしての書き込みは行われません。たとえば、次のようになります。

```
# 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.
# -R directories are searched at run time.
```

また、**LD_LIBRARY_PATH** および **LD_RUN_PATH** 環境変数を使用してライブラリー検索パスを指定することもできます。実行時にライブラリーの検索が行われるディレクトリーを指定するには、**LD_RUN_PATH** を使用します。リンク時と実行時の両方にライブラリーの検索が行われるディレクトリーを指定するには、**LD_LIBRARY_PATH** を使用します。

リンカー・オプションおよび環境変数の詳細については、**ld** コマンドの man ページを参照してください。

PDFDIR: PDF プロファイル情報用ディレクトリーの指定

-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、**-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、それらのファイルは更新されます。

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るため、次のガイドラインに従うことをお勧めします。

- **-qpdf** オプションを使用する場合は、常に **PDFDIR** 変数を設定する。
- 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するか、あるいは **-qipa=pdfname=[filename]** 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白に指定します。
- **PDFDIR** 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、それから 1 つ以上の環境変数に、それらのユニットがスクラッチ・ファイルとしてオープンされたときに使用されるファイル名を **XLFSCRATCH_***unit* の形式で設定します。例は「*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。

構成ファイルのカスタマイズ

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 **/etc/opt/ibmcmp/xlf/10.1/xlf.cfg** を提供します。

単一ユーザー・システム上で実行している場合、またはコンパイル・スクリプトや makefiles を持つコンパイル環境をすでに持っている場合は、デフォルトの構成ファイルをそのままにしておくこともできます。

それ以外の場合、特に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。たとえば、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/ibmcmp/xlf/10.1/bin/xlf95 /home/username/bin/my_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:

1. 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。

2. デフォルトの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。
3. 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。

属性

構成ファイルには、以下の属性が含まれています。

use 属性の値は、ローカル・スタンザだけではなく、指定したスタンザからも与えられます。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。

crt 32 ビット・モードで呼び出された場合、デフォルト (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)。リンケージ・エディターに第 1 パラメーターとして渡されます。

crt_64 64 ビット・モードで呼び出された場合、 **-q64** (たとえば、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) を使用します。リンケージ・エディターに第 1 パラメーターとして渡されます。

mcrt **crt** 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。

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 **crt** 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。

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 32 ビット・モードで呼び出された場合、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_libs_64 64 ビット・モードで呼び出された場合、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 32 ビット・ツール・チェーンのパスを指定します。

gcc_path_64 64 ビット・ツール・チェーンのパスを指定します。

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。

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。

code 最適化コード生成プログラムの絶対パス名。

xlfopt たとえば、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。

as アセンブラーの絶対パス名。

asopt たとえば、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。

ld リンカーの絶対パス名。

ldopt たとえば、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。

認識されないオプションのほとんどは、いずれの場合もリンカーに渡されますが、**-W** コンパイラー・オプションによっ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です。

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。

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・オプションは **-C** で、これは出力に C スタイルのコメントを保持します。

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。デフォルトでは **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、たとえば **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、 229 ページの『-qsuffix オプション』を参照してください。

cppsuffix XL Fortran でコンパイルする前に、 C プリプロセッサー (**cpp**) でファイルをプリプロセス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。

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。

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。

libraries コンマで区切られている **-l** オプションで、すべての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。

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。

hot 配列言語の変換を行うプログラムの絶対パス名。

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。

bolt バインド・プログラムの絶対パス名。

defaultmsg デフォルト・メッセージ・ファイルの絶対パス名。

include コンパイル・インクルード・ファイルおよびモジュール・ファイルに使用する検索パスを指示します。

include_32 32 ビット・コンパイル・インクルード・ファイルに使用される検索パスを指示します。

include_64 64 ビット・コンパイル・インクルード・ファイルに使用する検索パスを指示します。

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, ...
```

関連情報: 81 ページの『-F オプション』は、別の構成ファイルか構成ファイルの特定のスタンザ、あるいはその両方を選択するのに使用することができます。

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。

```
rpm -qa | grep xlf.cmp-10.1 | xargs rpm -qi
```

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およびリリース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。このサイクルの何らかの部分で問題を検出したら、最適化、デバッグなどのヘルプに関して、本節以降を参照することが必要な場合があります。

前提条件の情報:

1. 必須の Linux 設定 (たとえば、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、8 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 言語解説書*」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。

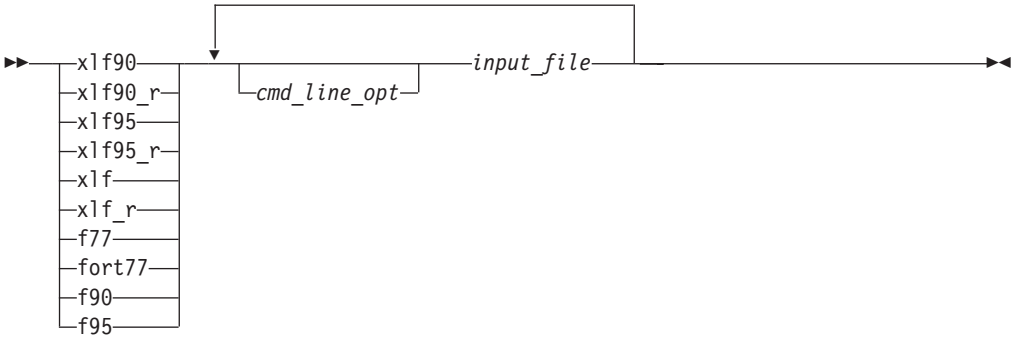
XL Fortran ソース・ファイルの編集

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** がなければなりませんが、構成ファイルの **fsuffix** 属性が異なるサフィックスを指定しない限り、つまり **-qsuffix** コンパイラー・オプションを使用している場合は、そうする必要はありません。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** も使うことができます。接尾部 **.f77**、**.f90**、または **.f95** 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 言語解説書*」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。

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、**xlf90**、**xlf90_r**、**xlf95**、**xlf95_r**、**xlf**、**xlf_r**、 **f77**、**fort77**、**f90**、または **f95** コマンドを次の形式で使用します。



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 (**/etc/opt/ibmcmp/xlf/10.1/xlf.cfg** ファイルを参照) 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するのに必要なステップを実行し、すべての **.s** ファイルをアセンブルして、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します。特に、**xlf_r**、**xlf90_r**、 および **xlf95_r** コマンドは、スレッド・セーフ・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・ファイルをリンクしたりバインドしたりします。

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 3. XL Fortran 呼び出しコマンド

ドライバー呼び出し	パスまたは位置	主な機能	リンクされるライブラリー
xlf90	/opt/ibmcmp/xlf/10.1/bin	Fortran 90	libxlf90.so
xlf90_r	/opt/ibmcmp/xlf/10.1/bin	スレッド・セーフ Fortran 90	libxlf90_r.so
xlf95	/opt/ibmcmp/xlf/10.1/bin	Fortran 95	libxlf90.so
xlf95_r	/opt/ibmcmp/xlf/10.1/bin	スレッド・セーフ Fortran 95	libxlf90_r.so
xlf	/opt/ibmcmp/xlf/10.1/bin	FORTRAN 77	libxlf90.so
xlf_r	/opt/ibmcmp/xlf/10.1/bin	スレッド・セーフ FORTRAN 77	libxlf90_r.so
f77 あるいは **fort77**	/opt/ibmcmp/xlf/10.1/bin	FORTRAN 77	libxlf90.so
f90	/opt/ibmcmp/xlf/10.1/bin	Fortran 90	libxlf90.so
f95	/opt/ibmcmp/xlf/10.1/bin	Fortran 95	libxlf90.so

呼び出しコマンドには、ディレクティブ・トリガーが以下のようになるという意味があります。

- **f77**、**fort77**、**f90**、**f95**、 **xlf**、 **xlf90**、および **xlf95** の場合、ディレクティブ・トリガーは、デフォルトでは **IBM*** です。
- それ以外のコマンドの場合、ディレクティブ・トリガーはデフォルトでは **IBM*** および **IBMT** です。 **-qsmp** を指定すると、コンパイラーは **IBMP**、**SMP$**、および **$OMP** トリガー定数も認識します。 **-qsmp=omp** オプションを指定すると、コンパイラーは **$OMP** トリガー定数だけを認識します。

-qsmp コンパイラー・オプションを指定すると、以下のようになります。
- コンパイラーは、自動並列化をオンにします。
- コンパイラーは、**IBMP**、**IBMT**、**IBM***、**SMP$**、および **$OMP** ディレクティブ・トリガーを認識します。

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。ライブラリー **libxlf90_r.so** は、**libxlf90_t.so** のスーパーセットです。 **xlf90_r**、**xlf95_r**、および **xlf_r** コマンドを使用するとき、ファイル **xlf.cfg** は自動的にセットアップされ、**libxlf90_r.so** にリンクされます。

libxlf90_t.so は部分的スレッド・サポート・ランタイム・ライブラリーです。このライブラリーは、使用に関する制限が 1 つ付いた状態で **/opt/ibmcmp/lib/libxlf90_t.so** としてインストールされます。これは、ライブラリーのルーチンはスレッド再入可能ルーチンではなく、1 つの Fortran のスレッドだけが、入出力操作を実行したり、ライブラリーを使用するマルチスレッド・アプリケーションに Fortran の組み込み機能を呼び出すことができるからです。アプリケーションがスレッド化されているときスレッドの同期によるオーバーヘッドを **libxlf90_r.so** で避けるために、Fortran スレッドが 1 つだけ存在しているマルチスレッド・アプリケーションで **libxlf90_t.so** を使用できます。

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。 **xlf_r**、 **xlf90_r**、 または **xlf95_r** 呼び出しコマンド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。

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

f90、**xlf90**、および **xlf90_r** コマンドを使用するほうが、**xlf**、**xlf_r**、および **f77/fort77** コマンドを使用するよりも、Fortran 90 標準にプログラムをより準拠させることができます。 **f95**、**xlf95** および **xlf95_r** コマンドを使用するほうが、**xlf**、**xlf_r**、および **f77/fort77** コマンドを使用するよりも、Fortran 95 標準にプログラムをより準拠させることができます。 **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、 **f90**、**xlf90** および **xlf90_r** コマンドのセットと、 **f95**, **xlf95** および **xlf95_r** コマンド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、**f90**、**xlf90** および **xlf90_r** コマンドは Fortran 90 標準に完全に準拠しているわけではありません。さらに、**f95**、**xlf95** および **xlf95_r** コマンドもデフォルトで Fortran 95 標準に完全に準拠しているわけではありません。 Fortran 90 または Fortran 95 完全準拠が必要な場合、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 -qextname -qfloat=nomaf:nofold -qnoswapomp
-qlanglvl=90std
-qlanglvl=95std
```

また、プログラムを実行する前に、次のようなコマンドの 1 つ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、パフォーマンスと使いやすさ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。たとえば、**-qextname** は、共通ブロックやサブプログラムなどの外部シンボルの 1 つに **main** という名前が付いている場合にだけ必要になります。

XL Fortran SMP プログラムのコンパイル

xlf_r、 **xlf90_r**、または **xlf95_r** 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。。主な違いは、**xlf_r**、、 **xlf90_r**、 または **xlf95_r** コマンドを指定した場合、スレッド・セーフ・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。または、これらの 6 つの呼び出しコマンドのいずれか 1 つと一緒に **-qsmp** オプションを指定することは可能です。**-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 サポートのレベル

XL Fortran では、IEEE 1003.1-2001 (POSIX) 標準 pthreads API を使用したスレッド・プログラミングがサポートされます。

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**、または **xlf95_r** コマンドを使用します。たとえば、次のように指定します。

```
xlf95_r test.f
```

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単位、サブプログラム、またはインターフェース本体がある場合、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。

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。

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 (.f または .F サフィックス)
: **.f**、**.f77**、**.f90**、**.f95** および **.F** ファイルはすべて、コンパイル用のソース・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。サフィックス **.F** を持つファイルは、コンパイルされる前にまず、 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: 27 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。

11 ページの『構成ファイルのカスタマイズ』および 229 ページの『-qsuffix オプション』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: 69 ページの『リンクを制御するオプション』および 30 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 (11 ページの『構成ファイルのカスタマイズ』および 229 ページの『-qsuffix オプション』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 (11 ページの『構成ファイルのカスタマイズ』および 229 ページの『-qsuffix オプション』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。

共用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。リンク時に共用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、共用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。

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 **/etc/opt/ibmcmp/xlf/10.1/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、 11 ページの『構成ファイルのカスタマイズ』および 81 ページの『-F オプション』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: 83 ページの『**-I** オプション』 と 181 ページの『-qmoddir オプション』を参照してください。

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***」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: 191 ページの『-qpdf オプション』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 91 ページの『-o オプション』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 78 ページの『-c オプション』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 **.f** 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 **.s** 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: 78 ページの『-c オプション』および 30 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、 91 ページの『-o オプション』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 **.f** 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: 254 ページの『-S オプション』および 30 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、91 ページの『-o オプション』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、拡張子 **.lst** を持っています。

関連情報: 58 ページの『リストとメッセージを制御するオプション』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、 181 ページの『-qmoddir オプション』を参照してください。

cpp プリプロセス済みソース・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: 27 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 80 ページの『-d オプション』を参照してください。

プロファイル・データ・ファイル (.*pdf*)

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: 191 ページの『-qpdf オプション』を参照してください。

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 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。)

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 73 ページの『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
```

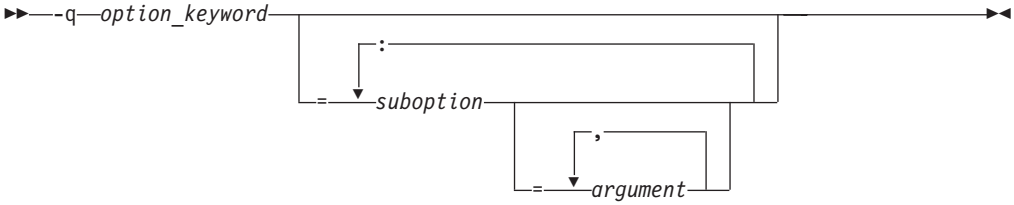
(例外がいくつかあります。たとえば、**-pg**。これは単一オプションで、**-p -g** と同じではありません。)

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、すで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。

たとえば、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 (たとえば **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。

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

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

  ```
  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 **/etc/opt/ibmcmp/xlf/10.1/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。

  たとえば、 **/etc/opt/ibmcmp/xlf/10.1/xlf.cfg** の **xlf95** スタンザに以下の行を入れて、

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

  を入れて、次のコマンド を発行すると、

  ```
  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。 を発行すると、

関連情報: 260 ページの『-W オプション』および 11 ページの『構成ファイルのカスタマイズ』を参照してください。

バイナリー・ファイル内の情報の表示 (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・バイナリー実行可能ファイルおよびライブラリー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** ファイルにエンコードされています。

たとえば、**/opt/ibmcmp/xlf/10.1/exe/xlfentry** にエンコードされた情報を見るには、次のコマンドを発行します。

```
strings /opt/ibmcmp/xlf/10.1/exe/xlfentry | grep "@(#)"
```

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特定のコードを生成するようにコンパイラーに指示するプログラムを作成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、103 ページの『-qarch オプション』を参照してください。 88 ページの『-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。

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プリプロセスすることができます。

特定のファイルについて **cpp** を呼び出すには、ファイル・サフィックス **.F** を使用してください。 **-d** オプションを指定すると、個々の **.F** ファイル *filename*.**F** は、中間ファイル **F***filename*.**f** にプリプロセスされて、これがコンパイルされます。 **-d**

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は **/tmp** になります。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。プリプロセスは行いたいが、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、プリプロセス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の詳細については、「*XL Fortran 言語解説書*」の言語エレメントの節にある『*条件付きコンパイル*』を参照してください。

cpp プリプロセス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: 80 ページの『-d オプション』、 255 ページの『-t オプション』、 260 ページの『-W オプション』、および 11 ページの『構成ファイルのカスタマイズ』を参照してください。

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 (たとえば、**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。

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** オプションを使用して渡す必要があります。たとえば、 **LNXV1** という名前のシンボルの存在をテストする **#ifdef** ディレクティブがプログラムに含まれている場合は、次のような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
```

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 **/opt/ibmcmp/xlf/10.1/exe/cpp** で見つかった **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。

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。たとえば、次のコマンドを実行します。

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

これにより、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、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、32 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。

別個のステップのコンパイルとリンク

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

XL Fortran デフォルトの一部ではない **ld** オプションでリンクしなければならない場合は、それらのオプションをコンパイラー・コマンド行に入れることができます。

```
xlf95 -Wl,<options...> file.f # xlf95 passes all these options to ld
```

コンパイラーは、**-q** オプション以外の認識されないオプションを **ld** コマンドに渡します。

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。いくつかのプログラムが同時に同じ共用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。ライブラリー・ルーチンとの命名の競合を回避するための、リンク中に行われなければならない特別な予防措置は必要とされません。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・ルーチンとの命名の競合を回避したい場合は、プログラム内のデータ・オブジェクトおよびルーチンの名前を選択するときに、何らかの予防措置をとる必要があります (『リンク中の命名競合の回避』で説明しています)。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、**--static** オプションをリンカー・コマンドに追加します。たとえば、次のようになります。

  ```
  xlf95 -Wl,--static test.f
  ```

リンク中の命名競合の回避

実行時サブプログラム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。これはデフォルトです。

-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。

- プログラムが、XLF 提供のあるルーチンを呼び出すと、次のように、使用できる共通ブロック名およびサブプログラム名に、いくつかの制約事項が適用されます。

XLF が提供する関数名	使用できない共通ブロック名または サブプログラム名
mclock	times
rand	irand

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 (たとえば、 **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。非 SMP プログラムの場合、**libxlf90.so**、**libxlfmath.so**、および **libxlomp_ser.so** のみが通常必要な XL Fortran ライブラリーです。 SMP プログラムの場合、通常、少なくとも **libxlf90_r.so**、**libxlfmath.so**、 および **libxlsmp.so** ライブラリーが必要出会う。SMP プログラムの場合、通常は少なくとも **libxlf90_r.so**, **libxlfmath.so**、および **libxlsmp.so** ライブラリーが必要です。**libxlfpmt*.so** と **libxlfpad.so** が必要となるのは、プログラムが **-qautodbl** オプションを使用してコンパイルされている場合のみです。

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: 30 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、Fortran ランタイム・ライブラリーのスレッド・セーフであるバージョンです。 **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。たとえば、次のようになります。

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/ibmcmp/lib/libxlf90_r.so** **/opt/ibmcmp/lib64/libxlf90_r.so** **/opt/ibmcmp/lib/libxlsmp.so** **/opt/ibmcmp/lib64/libxlsmp.so**	/opt/ibmcmp/xlf/10.1/include

実行時メッセージ用の言語の選択

XL Fortran プログラムが作成する実行時メッセージ用の言語を選択するには、プログラムの実行前に環境変数 **LANG** と **NLSPATH** を設定してください。

環境変数の設定の他にも、プログラムは C ライブラリー・ルーチン **setlocale** を呼び出して、実行時にプログラムのロケールを設定する必要があります。たとえば、次のプログラムは、実行時メッセージのカテゴリーを環境変数 **LC_ALL**、**LC_MESSAGES**、**LANG** に応じて設定することを指定します。

```
PROGRAM MYPROG
PARAMETER(LC_MESSAGES = 5)
EXTERNAL SETLOCALE
CHARACTER NULL_STRING /Z'00'/
CALL SETLOCALE (%VAL(LC_MESSAGES), NULL_STRING)
END
```

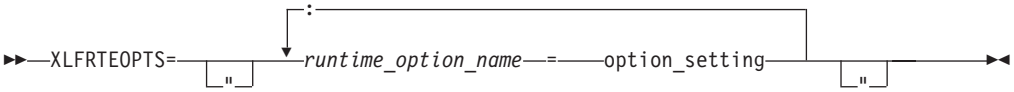
関連情報: 8 ページの『各国語サポートのための環境変数』を参照してください。

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 の実行時オプションの設定を指定することができます。

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、および乱数発生ルーチンの指定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 ステートメントが実行された。
- **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 言語解説書*」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。

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、入出力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能にすると、Fortran のサービス・ルーチン **flush_** を呼び出して、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。非同期入出力を実行する Fortran プログラムを使用していない場合は、 **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。

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 ステートメントの **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 言語解説書*」の『データ転送ステートメント』を参照してください。 **IOSTAT** 値に関する詳細は、「*XL Fortran 言語解説書*」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。

default_recl 実行時オプションは 64 ビット・モードでのみ適用されます。 32 ビット・モードでは、**default_recl** は無視され、レコード・サイズは 32 ビットになります。

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、64 ビット・モードでは上記のコード・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 言語解説書*」の **OPEN** 文の説明を参照してください。

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 ステートメント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。

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。

たとえば、**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 値の設定および使用の詳細については、「*XL Fortran 言語解説書*」の **READ** および **WRITE** の説明、ならびに『*条件と IOSTAT 値*』の節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={extended| 90std | 95std | 2003std}

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 ステートメント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定します。

95std Fortran 95 標準の I/O ステートメント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定します。

2003std XL Fortran がサポートする Fortran 2003 形式とともに、Fortran 95 標準が指定するすべての標準の I/O ステートメントおよび形式をコンパイラーが受け入れるように指定します。それ以外は、エラーとしてフラグが付けられます。

たとえば、**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.
```

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

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**
- **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}
複数の論理装置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のみです。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'**)
- 非同期入出力
- 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のみです。
- 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: 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。たとえば、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。たとえば、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=95std**、**langlvl=90std**、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 に関する詳細は、「*XL Fortran 言語解説書*」の『*名前リストの形式設定*』を参照してください。

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* に、指定した装置番号へ関連付けたいファイルの名前を指定します。 例は「*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。

unit_vars={yes | no}
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}
不定様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。不定様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。実行時オプションの **uwidth** は、64 ビット・モードのアプリケーションでしか使用できません。

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 基礎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。たとえば、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。

```
export XL_BLAS_LIB=/usr/lib/libblas.a
```

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。

XL_NOCLONEARCH

XL_NOCLONEARCH を使用してプログラムに汎用コードのみを実行するように指示してください。ここで、汎用コードとはアーキテクチャー用に使用されないコードです。**XL_NOCLONEARCH** 環境変数は、デフォルトで設定されません。ユーザーはアプリケーションでデバッグの目的用に設定できます。(**-qipa=clonearch** オプションも参照してください。)

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、8 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フローが、プログラムのコンパイル時に重大度 **S** の意味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の呼び出し

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:

- 77 ページの『-C オプション』
- 144 ページの『-qflttrap オプション』
- 215 ページの『-qsigtrap オプション』

「*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。

- これらの例外について詳しくは、『*浮動小数点例外の検出およびトラッピング*』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*』。

第 5 章 XL Fortran コンパイラー・オプションの解説

本節には以下のものがあります。

- コンパイラー・オプションの表。この表は、使用する分野別に構成されており、各オプションの構文と目的についてのハイレベルな情報が収められています。
- 73 ページの『XL Fortran コンパイラー・オプションの詳細記述』にある各コンパイラー・オプションについての詳細情報。

XL Fortran コンパイラー・オプションの概要

以降に記載されている表は、XL Fortran コンパイラーで使用できるコンパイラー・オプションを示しています。これらのオプションは、コンパイラー指示 **@PROCESS** を使用して、構成ファイル、コマンド行、Fortran ソース・コードにも入力できます。

-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。ただし、本節の「構文」節とサマリー表の「コマンド行オプション」列では、**-q** オプションとサブオプションの名前および **@PROCESS** ディレクティブの名前には大文字を使用しており、オプション・キーワードの最小の省略形を表します。 たとえば、**-qOPTimize** の有効な省略形は、 **-qopt**、**-qopti** などです。

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。

コンパイラーへの入力を制御するオプション

下のオプションは、コンパイラー入力に高い影響を与えます。これらはどのソース・ファイルが処理されるかを決定し、大/小文字の区別、桁の区別、その他のグローバルな形式の問題についての選択を行います。

関連情報: 20 ページの『XL Fortran 入力ファイル』および 48 ページの『出力ファイルの位置を指定するオプション』を参照してください。

60 ページの『互換性を維持するためのオプション』に記載されているオプションの多くは、許可されている入力形式を多少変えます。

表 4. コンパイラーへの入力を制御するオプション

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-I*dir*		インクルード・ファイルおよび **.mod** ファイルの検索パスにディレクトリーを追加します。 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・ファイルおよび **.mod** ファイルのデフォルト・ディレクトリーを検査する前に、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。 **デフォルト:** 以下のディレクトリーは、次のような順序で検索されます。 1. 現行ディレクトリー 2. ソース・ファイルが入っているディレクトリー 3. **/usr/include**. また、**/opt/ibmcmp/xlf/10.1/include** も検索されます。コンパイラーとともに出荷されるインクルード・ファイルと **.mod** ファイルはここに入っています。	83
-qci=*numbers*	CI(*numbers*)	指定された **INCLUDE** 行を活動化します。 **デフォルト:** デフォルト値なし。	119

表 4. コンパイラーへの入力を制御するオプション (続き)

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-qcr -qnocr		コンパイラーが CR (復帰) 文字をどのように解釈するかを制御することができます。これにより、Mac OS または DOS/Windows の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。 **デフォルト: -qcr**	121
-qdirective [=*directive_list*] -qnodirective [=*directive_list*]	DIRECTIVE [(*directive_list*)] NODIRECTIVE [(*directive_list*)]	トリガー定数として知られる文字列を指定します。これらの文字列は、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します。 **デフォルト: IBM*** で始まっているコメント行はディレクティブと見なされます。 **-qsmp=omp>** が指定されている場合は、**$OMP** だけがディレクティブ・トリガーであると見なされます。他のディレクティブ・トリガーはすべて、明示的にオンに切り替えない限り、オフになります。 **-qsmp=noomp** (-qsmp には noomp がデフォルト) が指定されている場合、**IBMP**、**$OMP** および **SMP$** はディレクティブ・トリガーと見なされ、オンになっているその他のディレクティブ・トリガー (**IBM*** や **IBMT** など) と同様に見なされます。 **-qthreaded** が指定されている場合は、**IBMT** で始まっているコメント行もディレクティブと見なされます。	126
-qenum=*value*		列挙型定数の範囲を指定し、判別されるストレージ・サイズを使用可能にします。	133
-qfixed [=*right_margin*]	FIXED [(*right_margin*)]	入力ソース・プログラムが固定ソース形式になっていることを示し、任意で行の最大長を指定します。 **デフォルト: f90**、**xlf90**、**xlf90_r**、**f95**、**xlf95**、および **xlf95_r** コマンドの場合は **-qfree=f90**、 **xlf**、**xlf_r**、および **f77/fort77** コマンドの場合は **-qfixed=72** です。	139

表 4. コンパイラーへの入力を制御するオプション (続き)

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-qfree[={f90|ibm}] -k	FREE[({F90|IBM})]	ソース・コードがフリー・フォームになっていることを示します。 ibm および f90 サブオプションは、それぞれ VS FORTRAN と Fortran 90/Fortran 95 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。 -k と -qfree は、-qfree=f90 の短縮形です。 **デフォルト: f90、f95、xlf90、xlf90_r、xlf95、および xlf95_r** コマンドの場合は **-qfree=f90**、 **xlf、xlf_r、**および **f77/fort77** コマンドの場合は **-qfixed=72** です。	146
-qmbcs -qnombcs	MBCS NOMBCS	文字リテラル定数、ホレリス定数、H 編集記述子、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) 文字または Unicode 文字を含めることができるかどうかをコンパイラーに示します。 **デフォルト: -qnombcs**	178
-U -qmixed -qnomixed	MIXED NOMIXED	コンパイラーが、名前内の文字の大文字と小文字を区別するようにします。 **デフォルト: -qnomixed**	256
-qsuffix={suboptions}		コマンド行でのソース・ファイル・サフィックスを指定します。	229

出力ファイルの位置を指定するオプション

これらのオプションは、コンパイラーが出力ファイルを格納するディレクトリーの名前を指定します。

この表では、* は、XL Fortran コンパイラーではなく **ld** コマンドによってオプションが処理されることを示します。これらのオプションの詳細については、**ld** コマンドに関する Linux 情報に記載されています。

関連情報: 22 ページの『XL Fortran 出力ファイル』および 46 ページの『コンパイラーへの入力を制御するオプション』を参照してください。

表 5. 出力ファイルの位置を指定するオプション

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-d		**cpp** によって作成されたプリプロセス済みソース・ファイルを削除せずに残します。 **デフォルト: cpp** で作成された一時ファイルは削除されます。	80

表 5. 出力ファイルの位置を指定するオプション (続き)

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-o *name**		出力オブジェクト・ソース・ファイル、実行可能ソース・ファイル、アセンブラー・ソース・ファイルなどの名前を指定します。 **デフォルト:** -o a.out	91
-qmoddir=*directory*		コンパイラーが書き込むモジュール・ファイル (**.mod**) の位置を指定します。 **デフォルト: .mod** ファイルは現行ディレクトリーに置かれます。	181

パフォーマンスの最適化のためのオプション

これらのオプションは、XL Fortran プログラムの実行速度を速めたり、パフォーマンスが低下している部分を見つけ、調整するのに役立ちます。このようなオプションの中で最も重要なのは **-O** です。一般に、他のパフォーマンス関連のオプションは、**-O** と組み合わせることにより、効果が上がります。 **-O** を組み合わせないと、まったく効果のないオプションもあります。

関連情報: 「*XL Fortran 最適化およびプログラミング・ガイド*」の『*XL コンパイラー・プログラムの最適化*』を参照してください。

68 ページの『浮動小数点処理のためのオプション』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、エラー条件や誤った結果が起きないように十分注意して使用してください。

表 6. パフォーマンスの最適化のためのオプション

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-O[*level*] -qoptimize[=*level*] -qnooptimize	OPTimize[(*level*)] NOOPTimize	コンパイル時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、最適化のレベル (0、2、 3、4 または 5) を指定します。 **デフォルト: -qnooptimize**	88
-p -pg		プロファイル用のオブジェクト・ファイルをセットアップします。 **デフォルト:** プロファイルなし。	92
-Q -Q! -Q+*names* -Q-*names*		プロシージャーをインライン化するかどうか、または、特定のプロシージャーをインライン化しなければならないかインライン化してはならないかを指定します (これらを両方指定する場合もあります)。*names* はプロシージャー名をコロンで区切ったリストです。 **デフォルト:** インライン化なし。	93

表 6. パフォーマンスの最適化のためのオプション (続き)

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-qalias={[no]aryovrlp | [no]intptr | [no]pteovrlp| [no]std}...]	ALIAS({[NO]ARYOVRLP |[NO]INTPTR |[NO]PTEOVRLP |[NO]STD}...)	ある種の別名付けが、プログラムに含まれているかどうかを示します。コンパイラーは、同じストレージ・ロケーションにさまざまな名前が別名として付けられている可能性がある場合、最適化の有効範囲を制限します。 **デフォルト: xlf95、 xlf95_r、f90**、および **f95** コマンドの場合は **-qalias=aryovrlp:nointptr:pteovrlp:std**、**xlf**、**xlf_r**、 **f77**、および **fort77** コマンドの場合は **-qalias=aryovrlp:intptr:pteovrlp:std** です。	97
-qalign={[no]4k| struct {=subopt}| bindc {=subopt}}	ALIGN({[NO]4K| STRUCT{(subopt)}| BINDC{(subopt)}})	誤って位置合わせされたデータによるパフォーマンス上の問題を回避する、ストレージ内でのデータ・オブジェクトの位置合わせを指定します。 **[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。 **デフォルト: -qalign=no4k:struct=natural:bindc=linuxppc**	100
-qarch=*architecture*		コンパイラーが生成する命令を制御します。デフォルトを変更すると、パフォーマンスを向上させることができますが、特定のマシンでしか実行できないコードが生成される可能性があります。選択項目は com、 ppc、 ppcgr、 ppc64、 ppc64gr、 ppc64grsq、 rs64b、 rs64c、 pwr3、 pwr4、 pwr5、 pwr5x、 ppc64v、および ppc970 です。 **デフォルト: -qarch=ppc64grsq**	103
-qassert={deps | nodeps | itercnt=*n*}	ASSERT(DEPS | NODEPS | ITERCNT(N))	最適化を微調整するのに役立つファイルの特性に関する情報を指定します。 **デフォルト: -qassert=deps:itercnt=1024**	108

表 6. パフォーマンスの最適化のためのオプション (続き)

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-qcache={ auto | assoc=*number* | cost=*cycles* | level=*level* | line=*bytes* | size=*Kbytes* | type={C|c|D|d| I|i}}[:...]		特定の実行マシンに対して、キャッシュ構成を指定します。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。 **デフォルト:** コンパイラーは、 **-qtune** 設定値または **-qarch** 設定値、あるいは、その両方に基づいて一般的な値を使用します。	114
-qcompact -qnocompact	COMPACT NOCOMPACT	コード・サイズを大きくする最適化を抑制します。 **デフォルト: -qnocompact**	120
-qdirectstorage -qnodirectstorage		指定のコンパイル単位がライトスルー使用可能またはキャッシュ使用禁止のストレージを参照できることをコンパイラーに通知します。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。 **デフォルト: -qnodirectstorage**	129
-qenablevmx -qnoenablevmx		ベクトル・マルチメディア拡張機能 (Vector Multimedia eXtension (VMX)) 命令の生成を使用可能にします。 **デフォルト: -qenablevmx**	132
-qessl		Fortran 90 組み込みプロシージャーの代わりに科学技術計算サブルーチン・ライブラリー (ESSL) ルーチンを使用することができます。 **-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。 **デフォルト: -qnoessl**	135
-qhot[=*suboptions*] -qnohot	HOT[=*suboptions*] NOHOT	最適化中に**高位**ループ分析および**変**換を実行するようにコンパイラーに指示します。 **デフォルト: -qnohot**	149
-qinlglue | -qnoinlglue		このオプションは、**-q64** および **-O2** およびより高位のものでコンパイルしているとき、ご使用のアプリケーションで外部関数呼び出しを最適化する Glue コードをインライン化します。 **デフォルト: -qnoinlglue**	156

表 6. パフォーマンスの最適化のためのオプション (続き)

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-qipa[=*suboptions*] | -qnoipa		プロシージャー間で詳細な分析 (プロシージャー間分析、つまり IPA) を行うことによって、**-O** 最適化を増大させます。 **デフォルト:** サブプログラムの境界全域に適用されるある一定の最適化を除外しながら、 **-O** は個々のサブプログラムを分析します。 **-O5** を指定することと、 **-O4** と **-qipa=level=2** を指定することは同じであることに注意してください。	160
-qkeepparm -qnokeepparm		最適化しているときでも、着信するプロシージャー・パラメーターがスタックに保管されることを保証します。 **デフォルト: -qnokeepparm**	167
-qmaxmem=*Kbytes*	MAXMEM (*Kbytes*)	コンパイラーが特定のメモリー集中の最適化を実行するときに、割り振るメモリーの量を指定キロバイト数に制限します。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。 **デフォルト: -qmaxmem=8192**。 **-O3**、**-O4**、および **-O5** では、**-qmaxmem=-1**。	176
-qpdf{1|2}		プロファイル指示フィードバック (*profile-directed feedback* (PDF)) によって最適化を調整します。その場合、条件付き分岐の近辺および頻繁に実行されるコード・セクション内の最適化が、サンプル・プログラムの実行結果を使用して改善されます。 **デフォルト:** 最適化では、分岐の頻度とその他の統計に関しては、固定の想定を使用します。	191
-qprefetch | -qnoprefetch		プリフェッチ命令がコンパイラーによって自動的に挿入されるかどうかを指示します。 **デフォルト: -qprefetch**	202
-qshowpdf -qnoshowpdf		追加の呼び出しとブロック数プロファイル情報を実行可能ファイルに追加します。このオプションは、**-qpdf1** オプションとともに使用します。 **デフォルト: -qnoshowpdf**	214
-qsmallstack[=dynlenonheap | nodynlenonheap] -qnosmallstack		可能な限りコンパイラーがスタック使用を最小化するように指定します。 **デフォルト: -qnosmallstack**	216

表 6. パフォーマンスの最適化のためのオプション (続き)

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-qsmp[=*suboptions*] -qnosmp		**xlf_r**、 **xlf90_r**、 または **xlf95_r** で使用するとき、ループの自動並列化、ループおよび他の項目のユーザー指定の並列化、およびアルゴリズムのチャンク化の選択を制御します。 **デフォルト: -qnosmp**	217
-NS*bytes* -qSPILLsize=*bytes*	SPILLsize (*bytes*)	レジスター予備スペースのサイズを指定します。 **デフォルト:** -NS512	87
-qstacktemp={0 | -1 | *positive_integer*}	STACKTEMP={0 | -1 | *positive_integer*}	実行時に適用できる XL Fortran コンパイラーのテンポラリー (temporary) をどこに割り振るかを判別します。 **デフォルト: 0** コンパイラーは適当なテンポラリー (temporary) をヒープまたはスタックに任意に割り振ります。	225
-qstrict -qnostrict	STRICT NOSTRICT	-O3、-O4、および -O5 オプションで行われた最適化によってプログラムのセマンティクスが変更されないことを保証します。 **デフォルト: -O3** 以上のレベルの最適化が有効になっている場合は、結果または例外が、最適化していないプログラムの結果または例外と異なるように、コードを再配置できます。 **-O2** の場合、デフォルトは **-qstrict** です。このオプションは、**-qnoopt** の場合は無視されます。	226
-qstrict_induction -qnostrict_induction		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てしまわないようにします。そのような最適化を実行した場合、帰納変数が関係した整数オーバーフローの動作が発生したときにアンセーフ になる可能性があります (プログラムのセマンティクスが変更される可能性があります)。 **デフォルト: -qnostrict_induction**	228
-qthreaded		コンパイラーがスレッド・セーフ・コードを生成するように指定します。**xlf_r**、**xlf90_r**、 および **xlf95_r** コマンドの場合、これはデフォルトでオンになります。	235

表 6. パフォーマンスの最適化のためのオプション (続き)

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-qtune=*implementation*		ハードウェア・アーキテクチャーの特定のインプリメンテーションに対する命令の選択、スケジューリング、その他のインプリメンテーションに依存するパフォーマンス拡張機能を調整します。auto、rs64b、 rs64c、pwr3、pwr4、pwr5、または ppc970 の設定が有効です。**デフォルト: -qtune=pwr4**	236
-qunroll[=auto | yes] -qnounroll		自動的に **DO** ループをアンロールすることを、コンパイラーに許可するかどうかを指定します。**デフォルト: -qunroll=auto**	239
-qunwind -qnounwind	UNWIND NOUNWIND	プロシージャー呼び出し中に、コンパイラーが、揮発性レジスターの保存と復元のデフォルト動作を保持するように指定します。**デフォルト: -qunwind**	241
-qzerosize -qnozerosize	ZEROSIZE NOZEROSIZE	サイズがゼロの文字ストリングおよび配列の検査を行わせないことによって、FORTRAN 77 プログラムと、一部の Fortran 90 および Fortran 95 プログラムのパフォーマンスを向上させます。**デフォルト: xlf90**、**xlf90_r**、**xlf95**、**xlf95_r**、 **f90**、および **f95** コマンドの場合は **-qzerosize** であり、**xlf**、**xlf_r**、**f77**、および **fort77** コマンドの場合は **-qnozerosize** です (サイズがゼロのオブジェクトを含むプログラムには、これらのコマンドを使用できないことを示しています)。	253

エラー・チェックおよびデバッグのためのオプション

これらのオプションを使用すると、XL Fortran プログラム内の問題を回避、検出、修正するのに役立ち、 267 ページの『第 7 章 問題判別とデバッグ』を頻繁に参照する必要もなくなります。

特に、**-qlanglvl** は、Fortran 標準の潜在的な違反を警告することによって、コンパイル・プロセスの初期の段階で移植性の問題を検出するのに役立ちます。これはプログラムで拡張が行われていたり、そのような拡張ができるようにするコンパイラー・オプションが備えられているためです。

-C および **-qflttrap** のようなその他のオプションは、実行時の計算エラーの検出もしくは回避、またはその両方を行います (これを行わないと誤った出力結果が作成される可能性があります)。

これらのオプションではコンパイル時に追加チェックを必要とし、また実行速度の低下をもたらす実行時エラー・チェックを使用する場合もあるので、チェックの必要性、コンパイル速度の低下、実行パフォーマンスという 3 要素間の適切なバランスを見つけ出すために実験が必要な場合があります。

これらのオプションを使用すると、行わなければならない問題判別とデバッグの量を最小限にとどめることができます。デバッグ中に役立つオプションには、他にも次のものがあります。

- 74 ページの『-# オプション』、 258 ページの『-v オプション』、および 259 ページの『-V オプション』
- 97 ページの『-qalias オプション』
- 119 ページの『-qci オプション』
- 186 ページの『-qobject オプション』
- 208 ページの『-qreport オプション』
- 223 ページの『-qsource オプション』

表 7. デバッグおよびエラー・チェックのためのオプション

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-C -qcheck -qnocheck	CHECK NOCHECK	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が正しいかどうかをチェックします。コンパイル時に境界外参照が検出されると、重大エラーとして報告され、実行時に **SIGTRAP** シグナルを生成します。 **デフォルト: -qnocheck**	77
-D -qdlines -qnodlines	DLINES NODLINES	桁 1 に D を持つ固定ソース形式行がコンパイルされるか、コメントとして扱われるかを指定します。 **デフォルト: -qnodlines**	79
-g -qdbg -qnodbg	DBG NODBG	シンボリック・デバッガーで使用するデバッグ情報を生成します。 **デフォルト: -qnodbg**	82
-qflttrap [=*suboptions*] -qnoflttrap	FLTTRAP [(*suboptions*)] NOFLTTRAP	実行時に検出する浮動小数点例外条件のタイプを決定します。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。 **デフォルト: -qnoflttrap**	144
-qfullpath -qnofullpath		ソース・ファイルとインクルード・ファイルの完全なパス名、つまり絶対パス名は、コンパイルされたオブジェクト・ファイルの中にデバッグ情報と一緒に記録されます (-g オプション)。 **デフォルト:** ソース・ファイルの相対パス名はオブジェクト・ファイルの中に記録されます。	147

表 7. デバッグおよびエラー・チェックのためのオプション (続き)

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-qhalt=*sev*	HALT(*sev*)	コンパイル時メッセージの最大の重大度が、指定した重大度と等しいか、それを上回る場合、オブジェクト・ファイル、実行可能ファイル、アセンブラー・ソース・ファイルを作成する前に動作を停止します。 *severity* (重大度) は、i (通知)、l (言語)、w (警告)、e (エラー)、s (重大エラー)、u (回復不能エラー)、 q (「停止しない」を示す重大度) のいずれかです。 **デフォルト: -qhalt=S**	148
-qinitauto[=*hex_value*] -qnoinitauto		*hex_value* の長さに応じて、自動変数用のストレージの個々のバイトまたはワード (4 バイト) を、特定の値に初期化します。これにより、定義前に参照される変数を見つけることができます。たとえば、**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。 **デフォルト: -qnoinitauto**。 *hex_value* を付けずに **-qinitauto** を指定する場合、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。	154
-qlanglvl={ 77std | 90std | 90pure | 95std | 95pure | 2003std | 2003pure | extended}	LANGLVL({ 77STD | 90STD | 90PURE | 95STD | 95PURE | 2003STD | 2003PURE | EXTENDED})	非準拠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) を決定します。非準拠のソース・コード、およびそのような非準拠を許可するオプションを識別します。 **デフォルト: -qlanglvl=extended**	168
-qlinedebug -qnolinedebug	LINEDEBUG NOLINEDEBUG	デバッガーに対して行番号およびソース・ファイル名情報を生成します。 **デフォルト: -qnolinedebug**	172
-qOBJect -qNOOBJect	OBJect NOOBJect	オブジェクト・ファイルを作成するか、または、ソース・ファイルの構文をチェックした直後に停止するかを指定します。 **デフォルト: -qobject**	186

表 7. デバッグおよびエラー・チェックのためのオプション (続き)

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-qsaa -qnosaa	SAA NOSAA	SAA FORTRAN 言語定義に従っているかどうかをチェックします。非準拠のソース・コード、およびそのような非準拠を許可するオプションを識別します。 **デフォルト: -qnosaa**	210
-qsaveopt -qnosaveopt		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を、該当するオブジェクト・ファイルに保管します。 **デフォルト: -qnosaveopt**	212
-qsigtrap[= *trap_handler*]		xl__trce、事前定義トラップ・ハンドラーまたはユーザー作成トラップ・ハンドラーをメイン・プログラムにインストールします。 **デフォルト:** トラップ・ハンドラーがインストールされません。 **trap** 命令が実行されたとき、プログラムはメモリー・ダンプします。	215
-qtbtable={none | small | full}		64 ビット環境にのみ適用されます。オブジェクト・ファイル内のトレースバック情報のデバッグ量を制限し、プログラムのサイズを小さくします。 **デフォルト:** 最適化なしで (**-O** を指定しない) コンパイルする場合、またはデバッグ用に (**-g** を指定して) コンパイルする場合、トレースバックの全情報をオブジェクト・ファイルに入れます。それ以外の場合、トレースバックの限定情報をオブジェクト・ファイルに入れます。	234
-qwarn64 -qnowarn64		8 バイト整数ポインターの 4 バイトへの切り捨てを検出します。通知メッセージを使って 32 ビットから 64 ビットへマイグレーションするときに問題を起こす可能性のある文を識別します。 **デフォルト: -qnowarn64**	243
-qxflag=dvz		**-qxflag=dvz** を指定すると、コンパイラーは、浮動小数点ゼロ除算演算を検出するコードを生成します。 **デフォルト:** 浮動小数点ゼロ除算演算を検出するコードは生成されません。	244
-qxlines -qnoxlines	XLINES NOXLINES	桁 1 に X を持つ固定ソース形式行をソース・コードと見なしてコンパイルするか、コメントとして扱うかを指定します。 **デフォルト: -qnoxlines**	250

リストとメッセージを制御するオプション

これらのオプションは、コンパイラーがリスト (**.lst** ファイル) を作成するかどうか、どのような種類の情報がリストに入るか、エラー条件を検出したらコンパイラーはそれについて何を行うかなどを決定します。

54 ページの『エラー・チェックおよびデバッグのためのオプション』に記載されているオプションの中には、コンパイラー・メッセージも作成できるものがあります。

表 8. リストとメッセージを制御するオプション

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-#		個々のコンポーネントを実際には実行せずに、コンパイルの進行に関する情報を生成します。 **デフォルト:** 進捗メッセージは生成されません。	74
-qattr[=full] -qnoattr	ATTR[(FULL)] NOATTR	属性の属性コンポーネントおよびリストの相互参照セクションを作成するかどうかを指定します。 **デフォルト: -qnoattr**	109
-qflag= *listing_severity*: *terminal_severity* -w	FLAG (*listing_severity*, *terminal_severity*)	診断メッセージを指定されたレベルまたはそれ以上のレベルに限定します。 *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。 -w は、-qflag=e:e の短い形式です。 **デフォルト: -qflag=i:i**	140
-qlist[={[no]offset}] -qnolist	LIST[([NO]OFFSET)] NOLIST	リストのオブジェクト・セクションを作成するかどうかを指定します。 **デフォルト: -qnolist**	173
-qlistopt -qnolistopt	LISTOPT NOLISTOPT	リスト・ファイル内のすべてのコンパイラー・オプションの設定を表示するか、または、選択したオプションだけを表示するかを決定します。これらの選択したオプションには、コマンド行またはディレクティブに指定されているオプションと、常にリストにあるオプションが含まれます。 **デフォルト: -qnolistopt**	174

表 8. リストとメッセージを制御するオプション (続き)

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-qnoprint		他のリスト・オプションの設定とは関係なく、リスト・ファイルが作成されないようにします。 **デフォルト:** -qattr、-qlist、-qlistopt、-qphsinfo、 -qreport、 -qsource、または -qxref のいずれかを指定したとき、1 つの リストが作成されます。	183
-qphsinfo -qnophsinfo	PHSINFO NOPHSINFO	各コンパイラー・フェーズのタイミング情報が端末に表示されるかどうかを決定します。 **デフォルト: -qnophsinfo**	196
-qreport[={smplist | hotlist}...] -qnoreport	REPORT [({SMPLIST | HOTLIST}...)] NOREPORT	プログラムを並列化する方法とループを最適化する方法を示す変換報告書を作成するかどうかを決定します。 **デフォルト: -qnoreport**	208
-qsource -qnosource	SOURCE NOSOURCE	リストのソース・セクションを作成するかどうかを指定します。 **デフォルト: -qnosource**	223
-qsuppress [= *nnnn-mmm* [:*nnnn-mmm*...] | cmpmsg] | -qnosuppress		出力ストリームから抑止されるメッセージを指定します。	230
-qversion -qnoversion		起動されているコンパイラーのバージョンとリリースを表示します。 **デフォルト: -qnoversion**	242
-qxref -qnoxref -qxref=full	XREF NOXREF XREF(FULL)	属性の相互参照コンポーネントおよびリストの相互参照セクションを作成するかどうかを決定します。 **デフォルト: -qnoxref**	252
-S		個々の Fortran ソース・ファイルに対して同等のアセンブラー・ソースを示す 1 つ以上の **.s** ファイルを作成します。 **デフォルト:** 同等のアセンブラー・ソースは作成されません。	254
-v		呼び出しコマンドで実行される個々のコンパイラー・コンポーネントの名前とパラメーターを表示することによって、コンパイルの進捗状況をトレースします。 **デフォルト:** 進捗メッセージは生成されません。	258

表 8. リストとメッセージを制御するオプション (続き)

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-V		呼び出しコマンドで実行される個々のコンパイラー・コンポーネントの名前とパラメーターを表示することによって、コンパイルの進捗状況をトレースします。これらの情報は、シェル実行可能形式で表示されます。 **デフォルト:** 進捗メッセージは生成されません。	259

互換性を維持するためのオプション

これらのオプションは、過去、現在、将来のハードウェア・プラットフォーム上の XL Fortran ソース・コード間の互換性を維持するのに役立つほか、変更をできる限り最小にとどめてプログラムを XL Fortran に移植するのに役立ちます。

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 へのプログラムの移植*』 でこの主題を詳細に説明しています。「*XL Fortran 最適化およびプログラミング・ガイド*」の 『*他のシステムの浮動小数点結果の複写*』では、他のシステムと互換性のある浮動小数点結果を得るためには、 68 ページの『浮動小数点処理のためのオプション』に記載されているオプションをどのように使用したらよいかを説明しています。

146 ページの『-qfree オプション』の **-qfree=ibm** 形式もまた、VS FORTRAN 自由ソース形式との互換性を提供します。

表 9. 互換性を維持するためのオプション

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-qautodbl=*setting*	AUTODBL(*setting*)	単精度浮動小数点計算を倍精度へ自動的に変換する方法、そして倍精度計算を拡張精度へ自動的に変換する方法を提供します。 none、 dbl、 dbl4、 dbl8、 dblpad、 dblpad4、または dblpad8 の設定のいずれかを使用します。 **デフォルト: -qautodbl=none**	110

表 9. 互換性を維持するためのオプション (続き)

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-qbigdata -qnobigdata		32 ビット・モードでは、初期化されたデータの 16 MB を超えるプログラム (gcc 制限) および共用ライブラリーの呼び出しルーチン (like open、close、 printf など) 用に、このオプションを使用します。 **デフォルト:** -qnobigdata	179
-qcclines -qnocclines	CCLINES NOCCLINES	コンパイラーが条件付きコンパイル行を認識するかどうかを決定します。 **デフォルト: -qsmp=omp** オプションを指定した場合は **-qcclines**、指定しなかった場合は **-qnocclines**。	117
-qctyplss [=([no]arg)] -qnoctyplss	CTYPLSS [([NO]ARG)] NOCTYPLSS	型が指定されていない定数を使用できる場合に、必ず文字定数式が許可されるかどうかを指定します。他のプラットフォームからプログラムを移植するとき、この言語拡張機能が必要となる場合があります。サブオプション **arg** は、実引数として使用されるホレリス定数が、整数の実際の引数として扱われることを指定します。 **デフォルト: -qnoctyplss**	122
-qddim -qnoddim	DDIM NODDIM	配列が参照されるたびに、pointee 配列の境界が再評価されることを指定し、pointee 配列用の境界式に対する制約事項をいくつか除去します。 **デフォルト: -qnoddim**	125

表 9. 互換性を維持するためのオプション (続き)

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-qdpc -qdpc=e -qnodpc	DPC DPC(E) NODPC	実定数を **DOUBLE PRECISION** 変数に割り当てるときに、最大の精度を得られるように実定数の精度を高めます。 他のプラットフォームからプログラムを移植するとき、この言語拡張機能が必要となる場合があります。 **デフォルト: -qnodpc**	131
-qescape -qnoescape	ESCAPE NOESCAPE	文字ストリング、ホレリス定数、H 編集記述子、ストリング編集記述子で、円記号がどのように扱われるかを指定します。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。 他のプラットフォームからプログラムを移植するとき、この言語拡張機能が必要となる場合があります。 **デフォルト: -qescape**	134

表 9. 互換性を維持するためのオプション (続き)

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-qextern=*names*		ユーザー作成のプロシージャーを、XL Fortran 組み込み機能の代わりに呼び出せるようにします。*names* はプロシージャー名をコロンで区切ったリストです。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。**デフォルト:** 組み込みプロシージャー名がユーザー作成のプロシージャー名と同じであると、前者が後者をオーバーライドします。	136
-qextname [=*name*:*name*...] -qnoextname	EXTNAME [(*name*:*name*...)] NOEXTNAME	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します (これが混合言語プログラムに対する規則であるシステムの場合、システムからプログラムを移植するのに役立ちます)。**デフォルト: -qnoextname**	137
-qinit=f90ptr	INIT(f90ptr)	ポインターの初期関連付け状況を関連付け解除にします。**デフォルト:** ポインターのデフォルト時関連付け状況は未定義です。	153
-qintlog -qnointlog	INTLOG NOINTLOG	式および文内に整数と論理値を混在させることができることを指定します。**デフォルト: -qnointlog**	157

表 9. 互換性を維持するためのオプション (続き)

コマンド行 オプション	@PROCESS ディレクティブ	説明	ページ 参照
-qintsize=*bytes*	INTSIZE(*bytes*)	デフォルトの **INTEGER** 値および **LOGICAL** 値のサイズを設定します。 **デフォルト: -qintsize=4**	158
-qlog4 -qnolog4	LOG4 NOLOG4	論理オペランドを持つ論理演算の結果が、**LOGICAL(4)** であるか、それともオペランドの最大長を持つ **LOGICAL** であるかを指定します。 **デフォルト: -qnolog4**	175
-qminimaltoc -qnominimaltoc		コンパイラーは、各コンパイル単位ごとに、1 つの TOC エントリーのみ作成します。デフォルトでは、プログラム内の固有の非自動変数参照ごとに、少なくとも 1 つの TOC エントリーをコンパイラーが割り当てます。8192 の TOC エントリーのみが使用可能で、重複エントリーは廃棄されません。これは、プログラムが 8192 の TOC エントリーを超える場合、64 ビット・モードで大きなプログラムをリンクしている際に、エラーが起きる原因になります。 **デフォルト:** -qnominimaltoc	179
-qmodule=mangle81		コンパイラーが非組み込みモジュール・ファイルに対して XL Fortran バージョン 8.1 の命名規則を使用するように指定します。 **デフォルト:** コンパイラーは、非組み込みモジュール名に対して現在の命名規則を使用します。この規則は、前のバージョンのコンパイラーで使用されていた規則とは互換性がありません。	182

表 9. 互換性を維持するためのオプション (続き)

コマンド行 オプション	@PROCESS ディレクティブ	説明	ページ 参照
-qnullterm -qnonullterm	NULLTERM NONULLTERM	仮引数として渡される文字定数式に NULL 文字を付加することによって、ストリングを C 関数に渡しやすくします。 **デフォルト:** **-qnonullterm**	184
-1 -qonetrip -qnoonetrip	ONETRIP NOONETRIP	**DO** 文が実行される場合、反復回数が 0 であったとしても、コンパイル済みプログラム内の個々の **DO** ループを最低 1 回実行します。 **デフォルト: -qnoonetrip**	75
-qport [=*suboptions*] -qnoport	PORT [(*suboptions*)] NOPORT	他の Fortran 言語拡張機能を収容するためにいくつかのオプションを提供して、 XL Fortran にプログラムを移植すると柔軟性が増加します。 **デフォルト: -qnoport**	199
-qposition= {appendold | appendunknown}	POSITION({APPENDOLD | APPENDUNKNOWN})	**POSITION=** 指定子を持たない **OPEN** 文の後にデータが書き込まれ、対応する **STATUS=** 値 (**OLD** または **UNKNOWN**) が指定されると、ファイル・ポインターをファイルの終わりに置きます。 **デフォルト: OPEN** 文の I/O 指定子とコンパイラー呼び出しコマンドに応じて、 **xlf**、 **xlf_r**、および **f77/fort77** コマンドの場合は **-qposition=appendold**、**xlf90**、**xlf90_r**、**xlf95**、**xlf95_r**、**f90**、および **f95** コマンドの場合は、定義済みの Fortran 90 および Fortran 95 の動作になります。	201

表 9. 互換性を維持するためのオプション (続き)

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-qqcount -qnoqcount	QCOUNT NOQCOUNT	拡張精度 **Q** 編集記述子 (**Q***w.d*) だけではなく、**Q** 文字カウント編集記述子 (**Q**) を受け入れます。-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。 **デフォルト: -qnoqcount**	203
-qoldmod=compat		このオプションは、V9.1.1 より前の XL Fortran のバージョンでコンパイルされたモジュール・データを含むオブジェクト・ファイルが未初期化のモジュール・データを含まないかまたは **convert_syms** スクリプトを使用して移植するかを指定します。	187
-qrealsize=*bytes*	REALSIZE(*bytes*)	**REAL**、**DOUBLE PRECISION**、**COMPLEX**、および **DOUBLE COMPLEX** 値のデフォルト・サイズを設定します。 **デフォルト: -qrealsize=4**	204
-qsave[={all | defaultinit}] -qnosave	SAVE{(ALL | DEFAULTINIT)} NOSAVE	ローカル変数のデフォルト・ストレージ・クラスを指定します。 **-qsave**、**-qsave=all**、あるいは **-qsave=defaultinit** は、デフォルト・ストレージ・クラスを **STATIC** に設定し、**-qnosave** は、**AUTOMATIC** に設定します。 **デフォルト: -qnosave** FORTRAN77 コマンドの動作と同一にするために、**xlf**、**xlf_r**、**f77**、または **fort77** では、**-qsave** がデフォルトでオンになります。	211

表 9. 互換性を維持するためのオプション (続き)

コマンド行 オプション	@PROCESS ディレクティブ	説明	ページ 参照
-qsclk=[centi | micro]		**SYSTEM_CLOCK** 組み込みプロシージャーを使用して値を戻すとき、コンパイラーがセンチ秒レゾリューションを使用するように指定します。 **-qsclk=micro** を使用することによりマイクロ秒レゾリューションを指定することができます。 **デフォルト: -qsclk=centi**	213
-qswapomp -qnoswapomp	SWAPOMP NOSWAPOMP	コンパイラーが、XL Fortran プログラムにある OpenMP ルーチンを認識して置換するように指定します。 **デフォルト: -qswapomp**	232
-u -qundef -qnoundef	UNDEF NOUNDEF	変数名の暗黙の型指定が許可されるかどうかを指定します。 -u および -qundef は、暗黙のステートメントを許可する個々の有効範囲にある **IMPLICIT NONE** 文と同じ効果を持っています。 **デフォルト: -qnoundef**	257
-qxflag=oldtab	XFLAG(OLDTAB)	桁 1 から 5 のタブを単一文字として解釈します (固定ソース形式のプログラムの場合)。 **デフォルト:** タブは 1 つ以上の文字として解釈されます。	245

表 9. 互換性を維持するためのオプション (続き)

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-qxlf77=*settings*	XLF77(*settings*)	変更された言語セマンティクスと I/O データ形式について、 FORTRAN 77 との互換性を提供します。これらの変更のほとんどは、Fortran 90 標準で必要です。 **デフォルト:** デフォルト・サブオプションは、**xlf90**、**xlf90_r**、**xlf95**、**xlf95_r**、 **f90**、および **f95** コマンドの場合は `blankpad`、`nogedit77`、`nointarg`、`nointxor`、`leadzero`、`nooldboz`、`nopersistent`、 および `nosofteof` で、**xlf**、**xlf_r**、および **f77/fort77** コマンドの場合は正確にこの逆です。	246
-qxlf90= {[no]signedzero | [no]autodealloc}	XLF90({[no]signedzero | [no]autodealloc})	言語の特定の機能について、コンパイラーが Fortran 90 または Fortran 95 レベルのサポートを提供しているかどうかを判別します。 **デフォルト:** **xlf95**、**xlf95_r**、 および **f95** 呼び出しコマンドでは、デフォルト・サブオプションは **signedzero** と **autodealloc** です。他のすべての呼び出しコマンドでは、デフォルト・サブオプションは **nosignedzero** と **noautodealloc** です。	248

浮動小数点処理のためのオプション

システムの浮動小数点のパフォーマンスと精度を最大限に利用するために、コンパイラーおよび XLF コンパイル済みプログラムが浮動小数点計算をどのように実行するかを詳しく指定しなければならない場合もあります。

関連情報: 144 ページの『-qflttrap オプション』および「*XL Fortran 最適化およびプログラミング・ガイド*」の『*他のシステムの浮動小数点結果の複写*』を参照してください。

表 10. 浮動小数点処理のためのオプション

コマンド行オプション	@PROCESSディレクティブ	説明	ページ参照
-qfloat=*options*	FLOAT(*options*)	特定のタイプの浮動小数点計算を処理するために、コンパイラーがどのようにコードの生成または最適化を行うかを決定します。**デフォルト:** デフォルト・サブオプションは、**nocomplexgcc**、**nofltint**、**fold**、**nohsflt**、**maf**、**nonans**、**norrm**、**norsqrt**、および **nostrictnmaf** で、これらの設定の中には、**-O3** 最適化がオンの場合、または **-qarch=ppc** の場合は異なるものもあります。	141
-qieee={ Near | Minus | Plus | Zero} -y{n | m | p | z}	IEEE({Near | Minus | Plus | Zero})	コンパイル時に定数浮動小数点式を評価するときにコンパイラーが使用する丸めモードを指定します。**デフォルト: -qieee=near**	152
-qstrictieeemod -qnostrictieeemod	STRICTIEEE-MOD NOSTRICTIEEE-MOD	**ieee_arithmetic** および **ieee_exceptions** 組み込みモジュール用の Fortran 2003 IEEE 演算規則をコンパイラーに順守させるかどうかを指定します。**デフォルト: -qstrictieeemod**	227

リンクを制御するオプション

これらのオプションは、コンパイル中に **ld** コマンドがオブジェクト・ファイルを処理する方法を制御します。これらのオプションの中には、**ld** に渡されて、コンパイラーによる処理がまったく行われないものもあります。

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。

この表では、***** は、XL Fortran コンパイラーではなく **ld** コマンドによってオプションが処理されることを示します。これらのオプションの詳細については、**ld** コマンドに関する man ページに記載されています。

表 11. リンクを制御するオプション

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-c		実行可能ファイルの代わりに、オブジェクト・ファイルを作成します。 **デフォルト:** 実行可能ファイルを作成するコンパイルおよびリンク・エディット	78
-L*dir**		指定されたディレクトリーの **-l** オプションで指定されたライブラリーを調べます。 **デフォルト: /usr/lib**	85
-l*key**		指定されたライブラリー・ファイル (*key* はファイル **lib*key*.so**、または **lib*key*.a** を選択します) を検索します。 **デフォルト: xlf.cfg** にリストされているライブラリー	86
-qpic -qnopic		共用ライブラリーで使用できる位置独立コード (PIC) を生成します。 **デフォルト:** 32 ビット・モードでは **-qnopic**。 64 ビット・モードでは **-qpic=small**。	198

他のコンパイラー操作を制御するオプション

これらのオプションを使用すると、次を実行する際に役立ちます。

- コンパイラーの内部サイズ限界の制御
- コンパイル時に実行されるコマンドの名前とオプションの決定
- ターゲット・アーキテクチャーのビット・モードおよび命令セットの決定

表 12. コンパイラーの内部操作を制御するためのオプション

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-B*prefix*		コンパイル中に使用する実行可能ファイル (コンパイラー、リンカーなど) の代替パス名を決定します。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。 **デフォルト:** これらのコンポーネントのパスは、構成ファイルと **$PATH** 環境変数、またはその両方に定義されます。	76

表 12. コンパイラーの内部操作を制御するためのオプション (続き)

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-F*config_file* -F*config_file*:*stanza* -F:*stanza*		代替構成ファイルを指定するか、その構成ファイル内で使用するスタンザを指定します (またはその両方を指定します)。 **デフォルト:** 構成ファイルは **/etc/opt/ibmcmp/xlf/10.1/xlf.cfg** ですが、スタンザはコンパイラーを実行するコマンドの名前によって異なります。	81
-q32		32 ビット・ターゲット・アーキテクチャーのビット・モードと命令セットを設定します。	94
-q64		64 ビット・ターゲット・アーキテクチャーのビット・モードと命令セットを設定します。	95
-t*components*		-B オプションで指定されたプレフィックスを、指定されたコンポーネントに適用します。 *components* には、セパレーターを持たない 1 つ以上の F、 c、 d、 I、 a、 h、 b、 z、または l を指定でき、それぞれ、 C プリプロセッサー、コンパイラー、 -S 逆アセンブラー、プロシージャー間分析 (IPA) ツール、アセンブラー、ループ・最適化プログラム、コード生成プログラム、バインド・プログラム、およびリンカーに対応します。 **デフォルト: -B** プレフィックスがあれば、それがすべてのコンポーネントに適用されます。	255
-W*component*, *options*		リストされたオプションを、コンパイル中に実行されるコンポーネントに渡します。 *component* は、 F、c、 d、I、a、z、または l のいずれかで、それぞれ、 C プリプロセッサー、コンパイラー、 -S 逆アセンブラー、プロシージャー間分析 (IPA) ツール、アセンブラー、バインド・プログラム、およびリンカーに対応します。 **デフォルト:** これらのプログラムに渡されるオプションは次のとおりです。 • 構成ファイルにリストされているオプション • コマンド行上の認識されないオプション (リンカーに渡される)	260

廃止、または不適オプション

次に示すオプションは、以下の理由のいずれかまたは両方のために廃止されています。

- よりよいと思われる代替オプションに置き換えられました。通常、制限されているオプションまたは特殊な目的を持つオプションは、より一般的な目的と追加機能を持つオプションと入れ替えられ、前のオプションは廃止オプションとされます。
- その機能を使用する顧客がほとんど、またはまったくいないと予想されました。また、将来の製品からこの機能を除去しても現在のユーザーにほとんど影響がないと予想されました。

注:

1. これらのオプションのいずれかを既存の makefile またはコンパイル・スクリプトで使用している場合は、将来起こり得る問題を回避するために、できる限り早急に新しい代替オプションへマイグレーションする必要があります。
2. **-qposition** の **append** サブオプションは、**appendunknown** に代わりました。

表 13. 廃止、または不適オプション

コマンド行オプション	@PROCESS ディレクティブ	説明	ページ参照
-qcharlen=*length*	CHARLEN (*length*)	これは廃止されたオプションです。依然として受け入れますが、無効です。文字定数および定数のサブオブジェクトの最大長は 32 767 バイト (32 KB) です。文字変数の最大長は、268 435 456 バイト (256 MB) です (32 ビット・モードの場合)。また文字変数の最大長は 64 ビット・モードで 2**40 バイトです。この限界は常に有効で、長ストリングを含むプログラムに移植性の問題が生じるのを防ぐのに十分な大きさとなっています。	
-qrecur -qnorecur	RECUR NORECUR	このオプションの使用はお勧めできません。外部サブプログラムを再帰的に呼び出すことができるかどうかを指定します。 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。	207

XL Fortran コンパイラー・オプションの詳細記述

以下のオプションのアルファベット順リストには、これらのオプションを効率的に使用するのに必要な全情報が記載されています。

構文情報の読み方

- 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 (適用可能な場合)。
- 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。
- { } 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 (...) が示されます。

-# オプション

構文

-#

個々のコンポーネントを実際には実行せずに、コンパイルの進行に関する情報を生成します。

規則

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、システム呼び出しと、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。

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
- 個々のステップに、どのオプションが有効であるか

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。)

-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

258 ページの『-v オプション』と 259 ページの『-V オプション』は、同じ出力を作成しますが、コンパイルも実行します。

-1 オプション

構文

-1
ONETRIP | **NOONETRIP**

DO 文が実行される場合、反復回数が 0 であったとしても、コンパイル済みプログラム内の個々の **DO** ループを最低 1 回実行します。このオプションを指定すると、FORTRAN 66 との互換性を保つことができます。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。

制限

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。

関連情報

-qonetrip は、**-1** の長い形式です。

-B オプション

構文

-B*prefix*

コンパイル中に使用する実行可能ファイル (コンパイラー、リンカーなど) の代替パス名を決定します。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。

引数

prefix は、代替の実行可能ファイルが常駐してい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。

規則

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。

例

この例では、初期レベルの XL Fortran コンポーネントが **/opt/ibmcmp/xlf/9.1/exe** ディレクトリーにインストールされています。アップグレードした製品をテストして誰でも使用できるようにするために、システム管理者は **/home/jim** ディレクトリー下に最新のインストール・イメージを復元して、次のようなコマンドでテストします。

```
/home/jim/xlf/10.1/bin/xlf95 -tchIbdz -B/home/jim/xlf/9.1/exe/ test_suite.f
```

アップグレードがいったん受け入れ基準を満たすと、システム管理者はそれを **/opt/ibmcmp/xlf/10.1** の古いレベルにインストールします。

関連情報

255 ページの『-t オプション』、 81 ページの『-F オプション』、 11 ページの『構成ファイルのカスタマイズ』、および 15 ページの『2 つのレベルの XL Fortran の実行』を参照してください。

-C オプション

構文

-C
CHECK | **NOCHECK**

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が正しいかどうかをチェックします。

規則

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は、報告されるエラーの重大度が、このオプションの指定時に **S** (重大) に上がります。

実行時に、参照が境界外に及ぶと、プログラムは **SIGTRAP** シグナルを発生させます。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。これは予想通りの動作であり、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。

関連情報

-C オプションは、 149 ページの『-qhot オプション』によって行われる最適化をいくらか抑止します。コードのデバッグが完了した後に **-C** オプションを除去し、**-qhot** オプションを追加すれば、より徹底的な最適化を実行することができます。

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。 253 ページの『-qzerosize オプション』を参照してください。

「*XL Fortran 最適化およびプログラミング・ガイド*」の 215 ページの『-qsigtrap オプション』および 『*例外ハンドラーのインストール*』には、**SIGTRAP** シグナルを検出してプログラムを終了させずに回復する方法が記述されています。

-qcheck は、**-C** の長い形式です。

-c オプション

構文

-c

完成したオブジェクト・ファイルを、リンク・エディットのために **ld** コマンドに送りません。このオプションを指定すると、出力は個々のソース・ファイルの **.o** ファイルになります。

-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

91 ページの『-o オプション』を参照してください。

-D オプション

構文

```
-D
DLINES | NODLINES
```

コンパイラーが桁 1 に D を持つ固定ソース形式行をコンパイルするか、コメントとして扱うかを指定します。

-D 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。

C 形式の **-D** マクロ定義を C プリプロセッサー に受け渡す (たとえば、**.F** で終わるファイルをコンパイルする) ためには、**-W** オプションを使用します。たとえば、次のようになります。

```
-WF,-DDEFINE_THIS
```

関連情報

-qdlines は **-D** の長い形式です。

-d オプション

構文

```
-d
```

cpp によって生成されるプリプロセス後のソース・ファイルを、削除せずに保持します。

規則

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。

関連情報

27 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。

-F オプション

構文

-F{*config_file* | *config_file:stanza* | :*stanza*}

代替構成ファイルを指定するか、その構成ファイル内で使用するスタンザを指定します (またはその両方を指定します)。

構成ファイルは、さまざまな種類のデフォルト、たと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。デフォルトの構成ファイル (**/etc/opt/ibmcmp/xlf/10.1/xlf.cfg**) は、インストール時に作成されます。デフォルトのスタンザは、コンパイラーを呼び出すために使用するコマンドの名前 (**xlf90**、**xlf90_r**、**xlf95**、**xlf95_r**、**xlf**、**xlf_r**、**f77**、または **fort77**) によって異なります。

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、 **/etc/opt/ibmcmp/xlf/10.1/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。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。

制限

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。

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

11 ページの『構成ファイルのカスタマイズ』では、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。

-g オプション

構文

-g
DBG | **NODBG**

シンボリック・デバッガーで使用するデバッグ情報を生成します。

-qdbg は **-g** の長い形式です。

-g は **-Q!** オプションを暗黙指定します。

関連情報

- 172 ページの『-qlinedebug オプション』
- 275 ページの『Fortran 90 または Fortran 95 プログラムのデバッグ』
- 6 ページの『シンボリック・デバッガーのサポート』

-I オプション

構文

-Idir

インクルード・ファイルおよび **.mod** ファイルの検索パスにディレクトリーを追加します。 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・ファイルおよび **.mod** ファイルのデフォルト・ディレクトリーを検査する前に、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。

引数

dir は有効なパス名 (たとえば **/home/dir**、**/tmp**、または **./subdir** など) でなければなりません。

規則

検索を行う前に、コンパイラーは *dir* 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 *dir* 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、コンパイラーとともに出荷されるインクルード・ファイルおよび **.mod** ファイルが入っている **/opt/ibmcmp/xlf/10.1/include** も検索します。

関連情報

181 ページの『-qmoddir オプション』は、モジュールを含んでいるファイルをコンパイルするときに **.mod** ファイルを特定のディレクトリーに置きます。

-k オプション

構文

```
-k
FREE(F90)
```

プログラムが自由ソース形式になることを指定します。

該当する製品レベル

このオプションの意味は、XL Fortran バージョン 2 とは異なっています。 **-k** の前の動作を実行するには、代わりに **-qfree=ibm** オプションを使用してください。

関連情報

146 ページの『-qfree オプション』および「*XL Fortran 言語解説書*」の『*自由ソース形式*』を参照してください。

このオプションは **-qfree=f90** の短い形式です。

-L オプション

構文

-Ldir

指定されたディレクトリーの **-l** オプションで指定されたライブラリーを調べます。デフォルト・ライブラリー以外のライブラリーを **/opt/ibmcmp/xlf/10.1/lib** または **/opt/ibmcmp/xlf/10.1/lib64** で使用する場合は、その他のライブラリーの位置を指し示す 1 つ以上の **-L** オプションを指定することができます。また、ライブラリーの検索パスのために **LD_LIBRARY_PATH** および **LD_RUN_PATH** 環境変数を指定することもできます。

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

69 ページの『リンクを制御するオプション』および 30 ページの『XL Fortran プログラムのリンク』を参照してください。

-l オプション

構文

-l*key*

指定されたライブラリー・ファイルを検索します (*key* はライブラリー **lib*key*.so** または **lib*key*.a** を選択します)。

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

69 ページの『リンクを制御するオプション』および 30 ページの『XL Fortran プログラムのリンク』を参照してください。

-NS オプション

構文

-NS*bytes*
SPILLSIZE(*bytes*)

レジスター の予備スペース (レジスターからストレージへの流出物で、内部最適化プログラムによって使用されるプログラムストレージ域) のサイズ (バイト単位) を指定します。

規則

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のバイト数を定義します。

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。

関連情報

-qspillsize は **-NS** の長い形式です。

-O オプション

構文

-O[*level*]
OPTimize[(*level*)] | **NOOPTimize**

コンパイル中にコードを最適化するかどうかを指定し、最適化する場合はそのレベルも指定します。

引数

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。現在、**-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。このフォームでは、現在は、最適化は実施されず、無視されます。

-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 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 **-qfloat** オプションの設定にも影響を与え、デフォルト時には **fltint** オプションと **rsqrt** サブオプションをオンにして、**-qmaxmem=-1** を設定します。

-O3 を指定すると、明示的に **-qhot** または **-qhot=level=1** を指定しない限り、**-qhot=level=0** を暗黙指定します。

-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。
- **-qhot**
- **-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。 **-O4 -qarch=com** をたとえば、指定すると、積極的に内部手順の最適化を行えるだけでなく、コードの移植性を保持することもできます。

-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: **-O2** と **-qsmp=omp** による高い最適化を組み合わせると、プロシージャー間分析 (IPA) を含む、追加の最適化アルゴリズムが呼び出されます。 IPA 最適化は、コンパイラーが追加の **fmadd** 命令を生成できるようにします。

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。

制限

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と、潜在的な変更や例外が起こるのをなくすことができます。

@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。

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

226 ページの『-qstrict オプション』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。

160 ページの『-qipa オプション』、 149 ページの『-qhot オプション』と 191 ページの『-qpdf オプション』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 『*XL コンパイラー・プログラムの最適化*』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、ご使用のコードから最高のパフォーマンスを得るために行う方法が説明されています。

-qOPTimize は **-O** の長い形式です。

-o オプション

構文

-o *name*

出力オブジェクト・ソース・ファイル、実行可能ソース・ファイル、アセンブラー・ソース・ファイルなどの名前を指定します。

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。

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。

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 オプション

構文

```
-p[g]
```

プロファイル用のオブジェクト・ファイルをセットアップします。

-p または **-pg** はプロファイル用のプログラムを用意します。プログラムを実行すると、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。

規則

プロファイルのために、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、個々のサブプログラムの始動ルーチンを、開始時にモニター・サブルーチンを呼び出すルーチンと置き換えます。プログラムが正常に終了すると、記録された情報が **gmon.out** ファイルに書き込まれます。

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、このコマンドの man ページを参照してください。

-Q オプション

構文

-Q+*names* | -Q-*names* | -Q | **-Q!**

Fortran 90 または Fortran 95 プロシージャーをインライン化するかどうか、および/または、インライン化しなければならないかインライン化してはならない特定のプロシージャーの名前を指定します。 *names* はプロシージャー名をコロンで区切ったリストです。

規則

デフォルトでは、**-Q** 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します。別の有効範囲のプロシージャーへの呼び出しに対するインライン展開をオンにするには、**-qipa** オプションも使用する必要があります。

引数

リストなしの **-Q** オプションは、インライン化される呼び出し数の制限、および結果としてのコード・サイズの増加量に従って、適切なすべてのプロシージャーをインライン化します。 **+***names* は、インライン化するプロシージャー名をコロンで区切って指定し、それぞれのプロシージャーに対する限界値を上げます。 **-***names* は、インライン化しないプロシージャー名をコロンで区切って指定します。これらのオプションを複数指定して、どのプロシージャーが最もインライン化されやすいかを厳密に制御することができます。

-Q! オプションは、インライン化をオフにします。

プロシージャーは非常に小さくない限り、基本的な **-Q** オプションによってはインライン化されません。 (正確なカットオフを判別するのは難しいが、) 通常は数行以内のソース・ステートメントを含みます。 **-Q+** によって命名されたプロシージャーは、最大では約 20 倍大きくでき、依然としてインライン化できます。

制限

-Q を使用してインライン化を有効にするには、少なくとも **-02** の最適化レベルを指定する必要があります。

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

```
xlf95  -O -Q many_small_subprogs.f   # Compiler decides what to inline.
xlf95  -O -Q+bigfunc:hugefunc test.f # Inline even though these are big.
xlf95  -O -Q -Q-only_once pi.f       # Inline except for this one procedure.
```

関連情報

- 160 ページの『-qipa オプション』
- 「*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の利点*』

-q32 オプション

構文

-q32

64 ビット環境で 32 ビットのコンパイル・モード (簡単に言えば、 32 ビット・モード) を使用できるようにします。 **-q32** オプションではコンパイル・ビット・モードが指定され、**-qarch** オプションとの組み合わせで、32 ビット実行モジュールが実行されるターゲット・マシンが決まります。

規則

- 32 ビット・モードでは、デフォルト整数およびデフォルト実サイズは 4 バイトです。
- 32 ビット・モードでは、デフォルト整数のポインター・サイズは 4 バイトです。
- 32 ビット・モードをターゲットとしたときは、32 ビットのオブジェクト・モジュールが作成されます。
- **-q32** がデフォルトです 。
- **-q64** は、**-q32** をオーバーライドすることができます。
- **-qarch** の設定値はすべて、**-q32** と互換性があります。 **-q32** を指定すると、デフォルトのサブオプションは **ppc64grsq** になり、 **-q32** のデフォルトの **-qtune** サブオプションは **pwr4** になります。
- **LOC** 組み込み関数は **INTEGER(4)** 値を戻します。

例

- 32 ビット・コンパイル・モードを使用し、汎用 PowerPC アーキテクチャーをターゲットにする場合。

  ```
  -qarch=ppc -q32
  ```

- 現在は同じコンパイル・モードを保持しているが、ターゲットを RS64II に変更する場合。

  ```
  -qarch=ppc -q32 -qarch=rs64b
  ```

  -qarch の最後の設定値が優先される点に注意してください。

- 現在は同じターゲットを保持しているが、コンパイル・モードを 64 ビットに変更する場合。

  ```
  -qarch=ppc -q32 -qarch=rs64b -q64
  ```

  -q64 を指定すると、以前のインスタンスである **-q32** がオーバーライドされる点に注意してください。

関連情報

- 103 ページの『-qarch オプション』
- 236 ページの『-qtune オプション』
- 243 ページの『-qwarn64 オプション』
- 265 ページの『第 6 章 64 ビット環境での XL Fortran の使用』
- 95 ページの『-q64 オプション』

-q64 オプション

構文

-q64
64 ビットのコンパイル・ビット・モードが指定され、**-qarch** オプションとの組み合わせで、64 ビット実行モジュールが実行されるターゲット・マシンが決まります。 **-q64** オプションを指定すると、オブジェクト・モジュールが 64 ビットのオブジェクト形式で作成され、 64 ビットの命令セットが生成されます。注意点として、32 ビット環境でコンパイルを行って 64 ビット・オブジェクトを作成することもできますが、そのオブジェクトは、**-q64** オプションを使って 64 ビット環境にリンクする必要があります。

規則

- **-q64** と互換性のある **-qarch** の設定値は、以下のとおりです。
  - **-qarch=auto** (64 ビット・システムでコンパイルする場合)
  - **-qarch=com** (**-q64** および **-qarch=com** を使用すると、コンパイラーはサイレントに arch 設定を **ppc64grsq** にアップグレードします。)
  - **-qarch=ppc** (**-q64** および **-qarch=ppc** を使用すると、コンパイラーはサイレントに arch を **ppc64grsq** にアップグレードします。)
  - **-qarch=ppcgr** (**-q64** および **-qarch=ppcgr** を使用すると、コンパイラーはサイレントに arch を **ppc64grsq** にアップグレードします。)
  - **-qarch=ppc64**
  - **-qarch=ppc64v**
  - **-qarch=ppc64gr**
  - **-qarch=ppc64grsq**
  - **-qarch=rs64b**
  - **-qarch=rs64c**
  - **-qarch=pwr3**
  - **-qarch=pwr4**
  - **-qarch=pwr5**
  - **-qarch=pwr5x**
  - **-qarch=ppc970**
- **-q64** のデフォルトの **-qarch** 設定は **ppc64** です。
- 64 ビット・モードをターゲットとしたときは、64 ビットのオブジェクト・モジュールが作成されます。
- **-q32** が **-q64** をオーバーライドすることがあります。
- **-q64** は、**-qarch** の競合設定値をオーバーライドし、その結果、設定値が **-q64 -qarch=ppc64** になり、警告メッセージが表示されます。
- **-q64** のデフォルトとなるチューニング設定値は **-qtune=pwr4** です。
- 64 ビット・モードでは、デフォルト整数およびデフォルト実サイズは 4 バイトです。
- 64 ビット・モードでは、デフォルト整数のポインター・サイズは 8 バイトです。
- 最大配列サイズは、約 2**40 バイト (静的ストレージの場合) または 2**60 バイト (ヒープでの動的割り振りの場合) まで増大します。バインドされる最大次元範囲は、 2**63, 2**63-1 バイトまで拡張されます。理論上の最大配列サイズは、2**60 ですが、これはオペレーティング・システムにより課される制限に制約さ

れます。配列定数の最大配列サイズは拡張されておらず、32 ビット・モードでの最大サイズと同じままです。初期化できる最大配列サイズは 2**28 バイトです。
- 配列コンストラクター暗黙 DO ループの最大反復カウントは 2**63-1 バイトまで拡大されます。
- 最大文字変数の長さは約 2**40 バイトまで拡張されます。文字定数および定数のサブオブジェクトの最大長は、32 767 バイト (32 KB) の 32 ビット・モードの場合と同じままです。
- **LOC** 組み込み関数は **INTEGER(8)** 値を戻します。
- **-qautodbl=dblpad** を 64 ビット・モードで使用する必要がある場合は、 8 バイトの整数演算用として、 **INTEGER(4)** を **INTEGER(8)** にプロモートするため **-qintsize=8** を使用してください。

例

下の例は 64 ビット・モードで RS64II (RS64b とも呼ばれます) をターゲットとしています。

```
-q32 -qarch=rs64b -q64
```

次の例では、64 ビット・アーキテクチャーの共通グループ (現在は RS64II、RS64III、POWER3、POWER4、POWER5、 POWER5+、および PowerPC 970 のみからなる) をターゲットとする 64 ビット・コンパイルを行います。

```
-q64 -qarch=com
```

このarch 設定はサイレントに、もっとも一般的な 64 ビット・モード・コンパイル・ターゲットである **ppc64grsq** にアップグレードされます。

関連情報

- 103 ページの『-qarch オプション』
- 236 ページの『-qtune オプション』
- 265 ページの『第 6 章 64 ビット環境での XL Fortran の使用』
- 243 ページの『-qwarn64 オプション』

-qalias オプション

構文

```
-qalias={argument_list}
ALIAS( {ARGUMENT_LIST} )
```

ある種の別名付けが、プログラムに含まれているかどうかを示します。コンパイラーは、同じストレージ・ロケーションにさまざまな名前が別名として付けられている可能性がある場合、最適化の有効範囲を制限します。考慮すべきエイリアス計画については、「*XL Fortran 最適化およびプログラミング・ガイド*」の『*XL コンパイラー・プログラムの最適化*』を参照してください。

引数

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
pointee 変数でないデータ・オブジェクトを参照するために pointee 変数を使用できるかどうか、または 2 つの pointee 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。使用できない場合は、**NOPTEOVRLP** を指定します。

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。

規則

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 および Fortran 95 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。

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うことができます。

```
@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
```

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。

```
! 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。

```
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)
```

制限

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。 **xlf90**、**xlf90_r**、**xlf95**、**xlf95_r**、**f90**、および **f95** コマンドは、プログラムに標準的な別名 (**-qalias=aryovrlp:pteovrlp:std:nointptr**) のみが入っているものと想定しますが、**xlf_r**、**xlf**、および **f77/fort77** コマンドは整数 **POINTER** が存在する可能性がある (**-qalias=aryovrlp:pteovrlp:std:intptr**) ものと想定します。

-qalign オプション

構文

```
-qalign={[no]4k|struct={suboption}|bindc={suboption}}
ALIGN({[NO]4K|STRUCT{(suboption)}|BINDC{(suboption)}})
```

誤って位置合わせされたデータによるパフォーマンス上の問題を回避する、ストレージ内でのデータ・オブジェクトの位置合わせを指定します。 **[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。

デフォルト

デフォルト設定は **-qalign= no4k:struct=natural:bindc=linuxppc** です。

引数

[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。

制限

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、不定様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。

関連情報

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、 109 ページの『-qattr オプション』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。

-qarch オプション

構文

-qarch=*architecture*

コンパイラーが生成する命令を制御します。デフォルトを変更すると、パフォーマンスを向上させることができますが、特定のマシンでしか実行できないコードが生成される可能性があります。

一般に、**-qarch** オプションを指定すると、コンパイルで特定のアーキテクチャーをターゲットにすることができます。 特定の **-qarch** 設定では、コンパイラーは特定の一致する **-qtune** 設定にデフォルト指定されて、さらにパフォーマンスを向上させることができます。 その結果生成されるコードはほかのアーキテクチャーでは稼動しない可能性がありますが、選択したアーキテクチャーには最高のパフォーマンスを提供します。複数のアーキテクチャーで実行可能なコードを生成するには、アーキテクチャーのグループをサポートする **-qarch** サブオプション (**com**、**ppc**、または **ppc64** など) を指定します。これにより、すべてのサポートされているアーキテクチャー、PowerPC、または 64 ビット PowerPC アーキテクチャー上でそれぞれコードを生成します。 **-qarch** サブオプションをグループ引数とともに指定すると、**-qtune** を **auto** として指定するか、またはそのグループ内の特定アーキテクチャーを指定することができます。 **-qtune=auto** の場合、コンパイラーは、**-qarch** サブオプションで指定されたグループ内のすべてのアーキテクチャー上で稼動するコードを生成しますが、コンパイルに使用したマシンのアーキテクチャー上で最高のパフォーマンスが得られる命令シーケンスを選択します。代わりに、特定アーキテクチャーをパフォーマンス・チューニングのターゲットとすることもできます。

引数

Y-HPC 以外の場合、選択項目は次のとおりです。

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。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。**-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。

com コンパイラーが生成した実行可能ファイルは、すべてのマシンに共通の命令のみが入っているため、コンパイラーがサポートするどのハードウェア・プラットフォームでも実行できます。この選択は、デフォルトと同じ **-qarch=ppc64grsq** になります。

-q64 オプションと **-qarch=com** オプションを同時に指定した場合、ターゲット・プラットフォームは 64 ビットであり、**-qarch** オプションはサイレントに **ppc64grsq** にアップグレードされます。命令セットは、すべての 64 ビット・マシンに共通の命令に制限されます。詳細については、265 ページの『第 6 章 64 ビット環境での XL Fortran の使用』を参照してください。

ppc 実行可能ファイルは、RS64II、RS64III、POWER3、POWER4、POWER5、PowerPC 970、および将来の PowerPC チップをベースにしたプラットフォームを含む、任意の PowerPC ハードウェア・プラットフォームで稼動させることできます。コンパイラー・オプション **-q64** を指定する場合、ターゲット・プラットフォームは 64 ビットの PowerPC であり、コンパイラーは

サイレントに **-qarch** 設定を **ppc64grsq** にアップグレードします。詳細については、 265 ページの『第 6 章 64 ビット環境での XL Fortran の使用』を参照してください。

ppcgr 32 ビット・モードでは、PowerPC ハードウェア・プラットフォーム用のオプションのグラフィック命令を含む可能性のあるオブジェクト・コードを生成します。

64 ビット・モードでは、64 ビット PowerPC プラットフォームでは実行されるが 32 ビットのみのプラットフォームでは実行されない、オプションのグラフィックス命令を含むオブジェクト・コードを生成し、**-qarch** オプションはサイレントに **-qarch=ppc64grsq**にアップグレードされます。

ppc64 実行可能ファイルは、任意の 64 ビット PowerPC ハードウェア・プラットフォーム上で実行できます。このサブオプションは、32 ビット・モードでのコンパイル時に選択できますが、その結果のオブジェクト・コードには、64 ビット・モードをサポートしない PowerPC プラットフォームで実行したときには認識されないか、または動作が異なる命令が含まれる可能性があります。

ppc64gr
実行可能ファイルは、オプションのグラフィック命令をサポートする任意の 64 ビット PowerPC ハードウェア・プラットフォーム上で実行できます。

ppc64grsq
実行可能ファイルは、オプションのグラフィック命令および平方根命令をサポートする任意の 64 ビット PowerPC ハードウェア・プラットフォーム上で実行できます。これはデフォルト・オプションです。

rs64b 実行可能ファイルは、任意の RS64II マシン上で実行できます。

rs64c 実行可能ファイルは、任意の RS64III マシン上で実行できます。

pwr3 実行可能ファイルは、任意の POWER3、POWER4、POWER5、POWER5+、または PowerPC 970 ハードウェア・プラットフォーム上で実行できます。前のリリースでは、**pwr3** 設定は、POWER3 および POWER4 のプロセッサー・グループをターゲットにするために使用されました。コンパイル・ターゲットをより一般的なプロセッサー・グループにするには、**ppc64grsq** 設定を使用して、POWER3、POWER4、POWER5、POWER5+、または PowerPC 970 プロセッサー・グループを含めます。これらのプラットフォーム用の実行可能ファイルはほかの PowerPC システム上で使用できない命令を含んでいる場合があると、それらのシステムとの互換性がなくなることがあります。

pwr4 実行可能ファイルは、任意の POWER4、POWER5、POWER5+、または PowerPC 970 ハードウェア・プラットフォーム上で実行できます。**-qarch=pwr4** を使用すると、以前の PowerPC インプリメンテーションでは稼動しない 2 進のものになります。

pwr5 実行可能ファイルは、任意の POWER5 または POWER5+ハードウェア・プラットフォーム上で実行できます。

pwr5x 実行可能ファイルは、任意の POWER5+ ハードウェア・プラットフォーム上で実行できます。

ppc64v

実行可能ファイルは、オプションの VMX 命令をサポートする任意の 64 ビット PowerPC ハードウェア・プラットフォーム (たとえば PowerPC 970 など) 上で実行できます。

ppc970

実行可能ファイルは、任意の PowerPC 970 ハードウェア・プラットフォーム上で実行できます。

注:

1. **-qarch** 設定は、**-qtune** 設定に対して許可される選択値とデフォルトを決定します。 **-qarch** および **-qtune** を使用して、プログラムを特定マシンで実行することができます。
2. **-qarch=com**、**-qarch=ppc**、または **-qarch= ppcgr** を使用して **-q64** を指定すると設定はサイレントに **-qarch=ppc64grsq** にアップグレードされます。

プログラムを特定マシン上でのみ稼動させたい場合は、**-qarch** オプションを使用して、該当するアーキテクチャーに特定のコードを生成する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、特定のチップ・モデルを指定する引数を提供します。たとえば、**-qarch=pwr3** を指定して、プログラムが POWER3 ハードウェア・プラットフォームで実行されるように指定することができます。

所定のアプリケーション・プログラムに対しては、それぞれのソース・ファイルをコンパイルするときに必ず同じ **-qarch** 設定を指定してください。

さらに、**-qcache** および **-qhot** オプションのようなその他のパフォーマンス関連オプションを使用して、特定のマシンを対象としたプログラムのパフォーマンスを向上させることができます。

以下の指針を使用して、このオプションを使用するかどうかを決定する一助としてください。

- プログラムを広範囲に分散可能にすることが第一の関心事である場合は、デフォルト (**ppc64grsq**) をそのまま使用してください。プログラムがすべてのタイプのプロセッサー上で均等に稼動することが多い場合は、 **-qarch** または **-qtune** オプションを指定しないでください。デフォルトでは、すべてのプロセッサーに共通の命令サブセットのみをサポートします。
- プログラムを複数のアーキテクチャーで稼動させるが、特定のアーキテクチャーに調整したい場合は、 **-qarch** および **-qtune** オプションを組み合わせて使用します。 **-qarch** 設定が、プログラムを実行する予定のすべてのタイプのプロセッサーを対象としていることを確認してください。
- プログラムを単一マシン上でのみ使用する場合、または別のマシン上で使用する前に再コンパイルする場合は、適用可能な **-qarch** 設定を指定してください。そのようにするとパフォーマンスが向上する場合があり、コンパイル時間も増加しません。 **rs64b**、**rs64c**、 **pwr3**、**pwr4**、**pwr5**、**pwr5x**、または **ppc970** サブオプションを指定した場合は、別に **-qtune** オプションを指定する必要はありません。

- 実行パフォーマンスが第一の関心ごとである場合は、適切な **-qarch** サブオプションを指定すると (そして、おそらく **-qtune** および **-qcache** オプションも指定すると)、スピードアップが可能です。この方法を使用すると、さまざまなマシンに対してさまざまなバージョンの実行可能ファイルを作成しなければならない場合があり、構成管理が複雑になります。追加の努力が正当であるかどうかを確認するために、パフォーマンス向上のテストが必要になります。
- 通常は、プログラムがターゲット・マシンの特性を利用できるように特定のアーキテクチャーをターゲットにすることをお勧めします。たとえば、POWER4 マシンをターゲットとするときに **-qarch=pwr4** を指定すると、浮動小数点中心のプログラム、または整数の乗算が含まれるプログラムにとって有利です。PowerPC システムでは、主にアンプロモートの単精度変数を処理するプログラムで **-qarch=ppc** を指定すると、より効果的です。 POWER3 システムでは、**-qarch=pwr3**、**-qarch=pwr4** および **-qarch=pwr5** を指定すると、主に倍精度変数 (またはいずれかの **-qautodbl** オプションを指定して倍精度にプロモートされる単精度変数) を処理するプログラムの効率が向上します。

その他の考慮事項

PowerPC 命令セットには、特定のハードウェア・プラットフォームにインプリメントされる 2 つのオプショナル命令グループが入っていますが、これらは必須ではありません。この 2 つのグループとは、グラフィックス命令グループと sqrt 命令グループです。特定の **-qarch** オプションを指定してコンパイルされるコード (そのすべては特定の PowerPC マシンを参照する) は、同じ命令グループがある、任意の同等な PowerPC マシン上で実行されます。以下の表では、さまざまな PowerPC マシンに組み込まれる命令グループを示します。

表 14. PowerPC プラットフォームの命令グループ

プロセッサー	グラフィックス・グループ	sqrt グループ	64 ビット
ppc	なし	なし	なし
ppcgr	あり	なし	なし
ppc64	なし	なし	あり
ppc64v	あり	あり	あり
ppc64gr	あり	なし	あり
ppc64grsq	あり	あり	あり
rs64b	あり	あり	あり
rs64c	あり	あり	あり
pwr3	あり	あり	あり
pwr4	あり	あり	あり
pwr5	あり	あり	あり
pwr5x	あり	あり	あり
ppc970	あり	あり	あり

関連情報

- 27 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』
- 236 ページの『-qtune オプション』
- 114 ページの『-qcache オプション』

- 149 ページの『-qhot オプション』
- 「*XL Fortran 最適化およびプログラミング・ガイド*」の『最良の -qarch サブオプションの選択』

-qassert オプション

構文

```
-qassert={deps | nodeps | itercnt=n}
ASSERT(DEPS) | NODEPS | ITERCNT(N))
```

最適化を微調整するのに役立つファイルの特性に関する情報を指定します。

引数

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*　不明ループの反復カウント値を指定します。

関連情報

次を参照してください。

- 背景情報およびこれらの申告の使用の説明に対して「*XL Fortran 最適化およびプログラミング・ガイド*」の『*ハイ・オーダー変換 (HOT) の利点*』
- 「*XL Fortran 言語解説書*」の **ASSERT** ディレクティブ。

-qattr オプション

構文

-qattr[=full] | **-qnoattr**
ATTR[(FULL)] | **NOATTR**

属性の属性コンポーネントおよびリストの相互参照セクションを作成するかどうかを指定します。

引数

-qattr 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。 **-qattr=full** が指定されると、参照されてもされなくても、すべての識別子が報告されます。

-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

58 ページの『リストとメッセージを制御するオプション』および 281 ページの『属性および相互参照セクション』を参照してください。

-qautodbl オプション

構文

-qautodbl=*setting*
AUTODBL(*setting*)

単精度浮動小数点計算を倍精度へ自動的に変換する方法、そして倍精度計算を拡張精度へ自動的に変換する方法を提供します。

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。たとえば、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。

引数

-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 (たとえば **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)** にプロモートします。
- **COMPLEX(4)** は **COMPLEX(8)** にプロモートします。

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)** にプロモートします。
- **COMPLEX(8)** は **COMPLEX(16)** にプロモートします。

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。

dblpad4 **dbl4** と同じプロモーションを実行し、ストレージをプロモートしたオブジェクトと共用できる場合は、他の型のオブジェクト (**CHARACTER** は除く) の埋め込みも行います。

このサブオプションは、リンク中に **libxlfpmt4.a** および **libxlfpad.a** ライブラリーを必要とします。

dblpad8 **dbl8** と同じプロモーションを実行し、ストレージをプロモートしたオブジェクトと共用できる場合は、他の型のオブジェクト (**CHARACTER** は除く) の埋め込みも行います。

このサブオプションは、リンク中に **libxlfpmt8.a** および **libxlfpad.a** ライブラリーを必要とします。

dblpad **dbl4** と **dbl8** が行ったプロモーションを結合し、ストレージをプロモートしたオブジェクトと共用できる場合は、他の型のオブジェクト (**CHARACTER** は除く) の埋め込みも行います。

このサブオプションは、リンク中に **libxlfpmt4.a**、**libxlfpmt8.a** および **libxlfpad.a** ライブラリーを必要とします。

規則

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。プロモートしたオブジェクトに対してストレージの関係を維持する必要がある場合は、**dblpad4** を使用してください。 **REAL(8)** 計算がほとんどないか、まったくない場合は、**dblpad** を使用することもできます。
- すべての結果について最高の精度にする場合、**dbl** または **dblpad** を使用できます。 **dbl4**、**dblpad4**、**dbl8**、および **dblpad8** は、精度を高める実数型のサブセットを選択します。

dbl4 または **dblpad4** を使用することにより、**REAL(8)** オブジェクトを **REAL(16)** オブジェクトに変換せずに、**REAL(4)** のサイズを増やすことができます。**REAL(16)** は、計算処理の点で、**REAL(8)** ほど効率がよくありません。

-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。たとえば、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。

制限

- 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。
- pointee 用のストレージ・スペースがシステム・ルーチン **malloc** によって獲得される場合、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、pointee 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、 285 ページの『-qautodbl のプロモーションと埋め込みの実行の詳細』を参照してください。

204 ページの『-qrealsize オプション』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。

-qbigdata オプション

構文

-qbigdata | **-qnobigdata**

32 ビット・モードでは、初期化されたデータの 16 MB を超えるプログラム (gcc 制限) および共用ライブラリーの呼び出しルーチン (like open、close、printf など) 用に、このコンパイラー・オプションを使用します。

-qcache オプション

構文

```
-qcache=
{
   assoc=number |
   auto |
   cost=cycles |
   level=level |
   line=bytes |
   size=Kbytes |
   type={C|c|D|d|I|i}
}[:...]
```

特定の実行マシンに対して、キャッシュ構成を指定します。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。

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** サブオプションと **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。
- 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、システムの持つキャッシュより大きなキャッシュ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適切です。

引数

assoc=*number*

キャッシュのセット連想度を指定します。

0 直接マップされるキャッシュ

1 完全に連想のキャッシュ

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**

制限

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

236 ページの『-qtune オプション』、 103 ページの『-qarch オプション』、および 149 ページの『-qhot オプション』を参照してください。

-qcclines オプション

構文

```
-qcclines | -qnocclines
CCLINES |  NOCCLINES
```

コンパイラーが条件付きコンパイル行を、固定ソース形式および F90 自由ソース形式で認識するかどうかを決定します。 IBM 自由ソース形式はサポートされていません。

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 言語解説書*」の言語エレメントの節の『*条件付きコンパイル*』を参照してください。

-qcheck オプション

構文

-qcheck | **-qnocheck**
CHECK | **NOCHECK**

-qcheck は、 77 ページの『-C オプション』の長い形式です。

-qci オプション

構文

-qci=*numbers*
CI(*numbers*)

処理する **INCLUDE** 行の識別番号 (1 から 255) を指定します。 **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ユニオンです。

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 (たとえば **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。

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。

制限

INCLUDE 行の任意の数字は広く行き渡っている 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。

関連情報

「*XL Fortran 言語解説書*」の **INCLUDE** ディレクティブについての節を参照してください。

-qcompact オプション

構文

-qcompact | **-qnocompact**
COMPACT | **NOCOMPACT**

コード・サイズを大きくする最適化を抑制します。

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成要素がある場合、**-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。

-qcr オプション

構文

-qcr | -qnocr

コンパイラーが CR (復帰) 文字をどのように解釈するかを制御することができます。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。これにより、Mac OS または DOS/Windows の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。

-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 オプション

構文

-qctyplss[(=[no]arg)] | **-qnoctyplss**
CTYPLSS[([NO]ARG)]| **NOCTYPLSS**

型が指定されていない定数を使用できる場合に、必ず文字定数式が許可されるかどうかを指定します。他のプラットフォームからプログラムを移植するとき、この言語拡張機能が必要となる場合があります。

引数

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、実際の引数を整数として扱われることを指定します。

規則

-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。

制限

- **-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 言語解説書*」の『ホレリス定数』と、「*XL Fortran 最適化およびプログラミング・ガイド*」の『参照によるまたは値による引数の引き渡し』を参照してください。

-qdbg オプション

構文

-qdbg | **-qnodbg**
DBG | **NODBG**

-qdbg は 82 ページの『-g オプション』の長い形式です。

-qddim オプション

構文

-qddim | **-qnoddim**
DDIM | **NODDIM**

配列が参照されるたびに、pointee 配列の境界が再評価されることを指定し、pointee 配列用の境界式に対する制約事項をいくつか除去します。

規則

デフォルト時には、pointee 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。

- pointee が参照されるたびに、その pointee 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、pointee 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 pointee 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
```

-qdirective オプション

構文

-qdirective[=*directive_list*] | **-qnodirective**[=*directive_list*]
DIRECTIVE[(*directive_list*)] | **NODIRECTIVE**[(*directive_list*)]

トリガー定数として知られる文字列を指定します。これらの文字列は、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します。

背景情報

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。最大限の柔軟性を持たせるようにするため、将来の XL Fortran コンパイラーに付属する可能性がある新しいディレクティブは、コメント行に含められます。これにより、他のコンパイラーがそれらのディレクティブを認識しなければ、移植性の問題を避けられます。

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** の指定には暗黙的に **-qdirective=smp¥$:¥$omp:ibmp** が含まれており、デフォルトで、トリガー定数 **SMP$**、**$OMP**、および **IBMP** もオンになります。 **-qsmp=omp** を指定する場合、コンパイラーはその時点までに指定したすべてのトリガー定数を無視し、 **$OMP** トリガー定数だけを認識します。 **-qthreaded** の指定には暗黙的に **-qdirective=ibmt** が含まれており、デフォルト時にはトリガー定数 **IBMT** もオンになります。

引数

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。

注

- 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。たとえば、次のようになります。

```
xlf95 -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。

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、プログラムは自由形式ソースに戻ります。

関連情報

「*XL Fortran 言語解説書*」の **SOURCEFORM** ディレクティブについての節を参照してください。

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、「*XL Fortran 言語解説書*」の『ディレクティブ』の節にある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。

-qdirectstorage オプション

構文

-qdirectstorage | **-qnodirectstorage**

指定のコンパイル単位がライトスルー使用可能またはキャッシュ使用禁止のストレージを参照できることをコンパイラーに通知します。

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 PowerPC 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。

-qdlines オプション

構文

-qdlines | **-qnodlines**
DLINES | **NODLINES**

-qdlines は 79 ページの『-D オプション』の長い形式です。

-qdpc オプション

構文

-qdpc[=e] | **-qnodpc**
DPC[(E)] | **NODPC**

実定数を **DOUBLE PRECISION** 変数に割り当てるときに、最大の精度を得られるように実定数の精度を高めます。 他のプラットフォームからプログラムを移植するとき、この言語拡張機能が必要となる場合があります。

規則

-qdpc を指定すると、すべての基本実定数 (たとえば 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 型付き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。

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。

関連情報

110 ページの『-qautodbl オプション』と 204 ページの『-qrealsize オプション』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。

-qenablevmx オプション

構文

-qenablevmx | -qnoenablevmx

ベクトル・マルチメディア拡張機能 (Vector Multimedia eXtension (VMX)) 命令の生成を使用可能にします。 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アルゴリズム集約的なタスクで使用されるとき、より高い性能を提案できます。

-qenablevmx は、コンパイラーが VMX 組み込み関数および **-qhot=simd** 最適化の両方を使用可能にするためにオンである必要があります。**-qenablevmx** は、PowerPC 970 プロセッサーのように、VMX 命令をサポートする **-qarch** ターゲットにのみ互換性があります。

関連情報

- 103 ページの『-qarch オプション』
- 149 ページの『-qhot オプション』
- 「*XL Fortran 言語解説書*」の VMX 組み込みプロシージャー
- 「*XL Fortran 言語解説書*」の Pixel、ベクター、および unsigned データ型

-qenum オプション

構文

-qenum=*value*

列挙型定数の範囲を指定し、判別されるストレージ・サイズを使用可能にします。

引数

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 15. 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 型付きパラメーター
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 言語解説書*」の ENUM / ENDENUM 文

-qescape オプション

構文

-qescape | -qnoescape
ESCAPE | NOESCAPE

文字ストリング、ホレリス定数、H 編集記述子、ストリング編集記述子で、円記号がどのように扱われるかを指定します。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。 他のプラットフォームからプログラムを移植するとき、この言語拡張機能が必要となる場合があります。

デフォルト

デフォルト時には、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、たとえば、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 オプション

構文

-qessl | **-qnoessl**

Fortran 90 組み込みプロシージャーの代わりに科学技術計算サブルーチン・ライブラリー (ESSL) ルーチンを使用することができます。

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。

規則

-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。 ESSL は、V4.1.1 以降が推奨されます。ライブラリーは、32 ビット環境と 64 ビット環境をサポートします。

また、**libessl.so** および **libesslsmp.so** は **libxlf90_r.so** に依存するので、**xlf_r**、**xlf90_r**、または **xlf95_r** を使用してコンパイルしてください (これらは、デフォルトでリンクするために **libxlf90_r.so** を使用します)。直接にリンカーまたはリンク用に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、次の URL を参照してください。
http://publib.boulder.ibm.com/clresctr/windows/public/esslbooks.html

-qextern オプション

構文

-qextern=*names*

ユーザー作成のプロシージャーを、XL Fortran 組み込み機能の代わりに呼び出せるようにします。*names* はプロシージャー名をコロンで区切ったリストです。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。

引数

プロシージャー名をコロンで区切ってください。

該当する製品レベル

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 オプション

構文

-qextname[=*name1*[:*name2*...]] | **-qnoextname**
EXTNAME[(*name1: name2:...*)] | **NOEXTNAME**

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します (これが混合言語プログラムに対する規則であるシステムの場合、システムからプログラムを移植するのに役立ちます)。 **-qextname=name1[:name2...]** を使用して、特定のグローバル・エンティティーを識別します。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。

メインプログラムの名前は影響を受けません。

-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。

制限

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。

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

このオプションは、他のオプションに指定された名前にも影響を与えます。したがって、コマンド行上の名前に下線を入れる必要はありません。影響を受けるオプションは、 136 ページの『-qextern オプション』、 93 ページの『-Q オプション』、および 215 ページの『-qsigtrap オプション』です。

-qfixed オプション

構文

```
-qfixed[=right_margin]
FIXED[(right_margin)]
```

入力ソース・プログラムが固定ソース形式になっていることを示し、任意で行の最大長を指定します。

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。

デフォルト

-qfixed=72 は、 **xlf**、**xlf_r**、**f77**、および **fort77** コマンドのデフォルトです。
-qfree=f90 は、 **f90**、**f95**、 **xlf90**、**xlf90_r**、**xlf95**、および **xlf95_r** コマンドのデフォルトです。

関連情報

146 ページの『-qfree オプション』を参照してください。

このソース形式の正確な仕様については、「*XL Fortran 言語解説書*」の『*固定ソース形式*』を参照してください。

-qflag オプション

構文

```
-qflag=listing_severity:terminal_severity
FLAG(listing_severity,terminal_severity)
```

listing_severity と *terminal_severity* の両方を指定する必要があります。

診断メッセージを指定されたレベルまたはそれ以上のレベルに限定します。 *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。 **-w** は、**-qflag=e:e** の短い形式です。

引数

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** オプション下で作成されたメッセージなど)。移植不可能な言語構成要素を示します。

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。 **-qhalt** 設定値を変更して、この種のエラーの発生時にコンパイラーがオブジェクト・ファイルを生成するように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。

-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。

デフォルト

このオプションのデフォルトは **i:i** で、これによりすべてのコンパイラー・メッセージを表示されます。

関連情報

168 ページの『-qlanglvl オプション』および 267 ページの『XL Fortran エラー・メッセージに関する情報』を参照してください。

-qfloat オプション

構文

```
-qfloat=options
FLOAT(options)
```

浮動小数点計算のスピードを上げて精度を改善するための、別の方法を選択します。

-qfloat 設定を変更する場合は、その前に 「*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*』に記述されている事柄と IEEE 標準を熟知しておかなければなりません。

デフォルト

デフォルト設定では、サブオプション **nocomplexgcc**、 **nofltint**、 **fold**、 **nohsflt**、 **maf**、 **nonans**、 **norrm** 、 **norsqrt**、および **nostrictnmaf** を使用します。以下に示すように、このデフォルトを変更するオプションもいくつかあります。

個々のサブオプションのデフォルト設定は、明示的に変更されない限り有効です。例えば、**-qfloat=nofold** を選択すると、**nohsflt**、または関連オプションの設定は影響を受けません。

引数

使用可能なサブオプションにはそれぞれ、 **fold** と **nofold** のような肯定の形式と否定の形式があります。否定の形式は肯定の形式の反対です。

サブオプションは以下のとおりです。

complexgcc | **nocomplexgcc**

複素数を渡すとき、または戻すときに、Linux 規則を使用します。このオプションは、gcc コンパイル・コードと Linux 上で IBM XL C/C++ コンパイラー のデフォルトの設定値と互換性を保持します。

注: このサブオプションの場合、コンパイル・コードを非 XL Fortran コンパイル・コードと混合するのは、モジュール変数など、ランタイム・ライブラリー情報またはグローバル・データに依存しない、小さい、内蔵タイプの、数学的に指向したサブプログラムに制限してください。異なる環境からコンパイルされたプログラムに例外処理または I/O がスムーズに作動することを期待しないでください。

fltint | **nofltint**

ライブラリー関数の呼び出しではなく、コードのインライン・シーケンスを使用することによって浮動小数点と整数との間の変換をスピードアップします。

ライブラリー関数 (**-qfloat=fltint** が指定されていない場合、または別のオプションによって示されていない場合、デフォルトで呼び出される) は、整数の表現可能範囲外の浮動小数点値をチェックし、範囲外の浮動小数点値が渡された場合は、最小または最大の表現可能整数を戻します。

Fortran 言語では、整数の表現可能範囲外の浮動小数点値をチェックする必要がありません。効率を向上させるために、**-qfloat=fltint** によって使用され

るインライン・シーケンスはこのチェックを行いません。範囲外の値が渡された場合、インライン・シーケンスは未定義の結果を生成します。

このサブオプションは、デフォルトではオフになりますが、 **-O3** の最適化レベルでは、**-qstrict** も指定されている場合を除いてオンになります。

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。

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。また、浮動小数点と整数間の変換に対して、**fltint** サブオプションと同じ手法を使用します。

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。他のアプリケーション・プログラムをコンパイルするときにこのオプションを使用すると、予期しない結果が起きても警告を出さない場合があります。慎重に使用してください。詳細については、 284 ページの『-qfloat=hsflt オプションの技術情報』を参照してください。

maf | nomaf
適切な場所で乗加算命令を使用することにより、浮動小数点計算をより速く正確に行います。考えられる欠点は、コンパイル時に実行した、または他のタイプのコンピューターで実行した同様の計算の結果とまったく同じにはならない場合があることです。また、負のゼロが生成される可能性があります。

nans | nonans
-qflttrap=invalid:enable オプションを使用してシグナル NaN (非数値) 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。

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コンパイラーの最適化をオフにします。いずれかの手段、たとえば **fpsets** プロシージャーを呼び出すことによって、プログラムが丸めモードを変更する場合のみ、このオプションを使用してください。それ以外の場合にこのオプションを使用すると、プログラムが誤った結果を算出する場合があります。

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。

このサブオプションは、デフォルトではオフになりますが、 **-O3** を指定すると、**-qstrict** も指定されている場合を除いてオンになります。

strictnmaf | nostrictnmaf
負の MAF 命令を導入するために使用する浮動小数点変換をオフにします。

それの変換は値ゼロの符号を保存することができないからです。デフォルトでは、コンパイラーはこのタイプの変換を使用可能にします。

セマンティクスを厳密にするには、**-qstrict** と **-qfloat=strictnmaf** を両方とも指定します。

-qflttrap オプション

構文

-qflttrap[=*suboptions*] | **-qnoflttrap**
FLTTRAP[(*suboptions*)] | **NOFLTTRAP**

実行時に検出する浮動小数点例外条件のタイプを決定します。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。 発生することをトラッピングするために

引数

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。

IMPrecise 指定された例外のチェックをサブプログラムの入り口と出口のみで行います。このサブオプションを指定すると、パフォーマンスは改善されますが、例外の正確なスポットが見つけにくくなることがあります。

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** または **imprecise** 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。

デフォルト

サブオプションなしの **-qflttrap** オプションは **-qflttrap=ov:und:zero:inv:inex** と同等ですが、このデフォルトには **enable** が含まれていません。このため、**fpsets** または類似のサブルーチンがソースに既存している場合のみ、有効になる可能性があります。サブオプションを使ったり、使わなかったりして **-qflttrap** を複数回指定すると、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。

例

次のプログラムをコンパイルします。

```
REAL X, Y, Z
DATA X /5.0/, Y /0.0/
Z = X / Y
PRINT *, Z
END
```

次のコマンドを使用します。

```
xlf95 -qflttrap=zerodivide:enable -qsigtrap divide_by_zero.f
```

除算が実行されると、プログラムは停止します。

zerodivide サブオプションは、ガードすべき例外のタイプを識別します。 **enable** サブオプションは、例外が発生すると **SIGFPE** シグナルを出します。 **-qsigtrap** オプションは、シグナルがプログラムを停止すると、通知出力を出します。

関連情報

- 215 ページの『-qsigtrap オプション』
- 103 ページの『-qarch オプション』
- どの場合にどの方法で **-qflttrap** オプションを使用するかについての詳細な説明は、「*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およびトラッピング*』に詳細に説明されています。特に、このオプションを使用し始めたばかりのときは、これを参照してください。

-qfree オプション

構文

```
-qfree[={f90|ibm}]
FREE[({F90|IBM})]
```

ソース・コードが自由ソース形式になっていることを示します。 **ibm** サブオプションと **f90** サブオプションは、それぞれ VS FORTRAN と 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 にも適用されることに注意してください。

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。

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、 **xlf**、**xlf_r**、**f77**、および **fort77** コマンドのデフォルトです。
-qfree=f90 は、 **f90**、**f95**、 **xlf90**、**xlf90_r**、**xlf95**、および **xlf95_r** コマンドのデフォルトです。

関連情報

139 ページの『-qfixed オプション』を参照してください。

-k は **-qfree=f90** と同等です。

Fortran 90 自由ソース形式については、「*XL Fortran 言語解説書*」の『*自由ソース形式*』で説明されています。この形式は、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。この形式については、「*XL Fortran 言語解説書*」の『*IBM 自由ソース形式*』に説明されています。

-qfullpath オプション

構文

-qfullpath | **-qnofullpath**

ソース・ファイルとインクルード・ファイルの完全なパス名、つまり絶対パス名は、コンパイルされたオブジェクト・ファイルの中にデバッグ情報と一緒に記録されます。 (**-g** オプションを指定してコンパイルするとき、デバッグ情報は表示されます。)

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** 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。

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。

制限

-qfullpath は **-g** オプションがなくても機能しますが、 **-g** 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

82 ページの『-g オプション』を参照してください。

-qhalt オプション

構文

```
-qhalt=severity
HALT(severity)
```

コンパイル時メッセージの最大の重大度が、指定した重大度と等しいか、それを上回る場合、オブジェクト・ファイル、実行可能ファイル、アセンブラー・ソース・ファイルを作成する前に動作を停止します。 *severity* (重大度) は、**i** (通知)、**l** (言語)、**w** (警告)、 **e** (エラー)、**s** (重大エラー)、**u** (回復不能エラー)、**q** (「停止しない」を示す重大度) のいずれかです。

引数

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** オプション下で作成されたメッセージなど)。移植不可能な言語構成要素を示します。

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。 **-qhalt** 設定値を変更して、この種のエラーの発生時にコンパイラーがオブジェクト・ファイルを生成するように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。

デフォルト

デフォルトは **-qhalt=s** です。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。

制限

-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

- 『-qhalt オプション』
- 186 ページの『-qobject オプション』

-qhot オプション

構文

-qhot[=*suboptions*] | **-qnohot**
HOT[=*suboptions*] | **NOHOT**

最適化中に**高位**ループ分析および**変**換を実行するようにコンパイラーに指示します。

-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。

-O および **-qhot** を使用するときに最適化レベル **2** 以上を指定しないと、コンパイラーは **-O2** を暗黙指定します。

-O3 を指定する場合、コンパイラーは **-qhot=level=0** を想定します。**-O3** を使用してすべての HOT 最適化を防ぐため、**-qnohot** を指定する必要があります。

サブオプションを使用しないで **-qhot** を指定することは、**-qhot=nosimd**、**-qhot=noarraypad**、**-qhot=vector**、および **-qhot=level=1** を暗黙指定します。これらのサブオプションはオプション **-qsmp**、**-O4**、または**-O5** によっても暗黙指定されます。

-qhot、**-qenablevmx**、および **-qarch=ppc970** または **-qarch=ppc64v** のいずれかを指定する場合、**-qhot=simd** がデフォルトとして設定されます。

配列の埋め込み: XL Fortran では、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。 **arraypad** サブオプションを使用すると、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。これにより、配列処理プログラムを低下させるキャッシュ・ミスやページ不在を削減することができます。

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 **-qhot=arraypad** を指定してください。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。

-C オプションは、いくつかの配列最適化をオフにします。

ベクトル化: **-qhot** コンパイラー・オプションは、**simd** および **vector** サブオプションをサポートします。これらは、可能であれば演算を並列に実行することによって、配列データの演算のためのソース・コードのループを最適化することができます。どちらのサブオプションもパフォーマンスを向上させることができますが、各サブオプションはそれぞれ特定のタイプのベクトル化に最も適しています。

ショート・ベクトル化: -qhot=simd: **simd** サブオプションは、配列データを最適化して、ターゲット・アーキテクチャーで許可される場合、算術演算を並列に実行します。並列演算は、16 バイト・ベクトル・レジスターで行われます。コンパイラーは、レジスター長を超えるベクトルを 16 バイト単位に分割して、最適化を促進します。 16 バイトの 1 つの単位には、次のいずれかの型のデータを含めることができます。

- 4 個の 4 バイト単位 (たとえば、4 個の整数 (4)、または 4 個の実数 (4))
- 8 個の 2 バイト単位
- 16 個の 1 バイト単位

ショート・ベクトル化は、倍精度の浮動小数点算術演算をサポートしません。ユーザーは **-qarch=ppc970** のように VMX 命令をサポートするアーキテクチャーを指定する必要があります。ユーザーは、一般に、たとえば、**-qhot=simd** 最適化をイメージ処理アプリケーションに適用するときに利点を見いだします。

ロング・ベクトル化: -qhot=vector: **vector** サブオプションは、**-qnostrict** または **-O3** またはより高位のレベルの最適化のレベルと併せて使用されたとき、配列データを最適化して、可能な場合は、算術演算を並列に実行します。コンパイラーは、ベクトル・サイズの制限なしで標準レジスターを使用します。単精度および倍精度の浮動小数点算術演算のサポートは、通常、ユーザーが大きな算術演算の要件を持つアプリケーションに **-qhot=vector** を適用するときに有利です。

引数

arraypad
コンパイラーは、キャッシュ使用効率が向上する可能性があるすべての配列を埋め込みます。すべての配列を必ずしも埋め込む必要はなく、コンパイラーは異なる配列に異なる量を埋め込むことができます。

arraypad=*n*
コンパイラーはソース内の各配列を埋め込みます。埋め込み数は、正の整数値でなければなりません。各配列は、エレメントの整数値で埋め込まれます。整数値 *n* は、**arraypad** を有効に使用するために、最大配列エレメント・サイズの倍数でなければなりません。この値は通常、4、8、または 16 です。

arraypad および **arraypad=n** オプションを指定した場合、コンパイラーは再シェーピングまたはそれと同等のものをチェックしません。埋め込みが行われる場合、プログラムが予測不能な結果を生成することがあります。

level={0 | 1}

level=0
コンパイラーは高位の変換のサブセットを実行します。これらのいくつかには、例として、早期の配布、ループ交換、およびループ・タイル機能が含まれます。

最適化レベル **-O3** は **qhot=level=0** を暗黙指定します。

level=1
-qhot=level=0 が明示的に指定されない限り、 **-qhot=level=1** は **-qhot** と同等であり、**-qhot** を暗黙指定するコンパイラー・オプションもまた **-qhot=level=1** を暗黙指定します。

level 以外のすべての **-qhot** サブオプションのデフォルトの HOT レベルは 1 です。たとえば、**-O3 -qhot=novector** の指定により HOT レベルは 1 に設定されます。

-qhot または **-qhot=level=1** を明示的に指定しない限り、**-O3** の指定は **-qhot=level=0** を暗黙指定します。

明示的に **-qhot=level=0** を指定しない限り、**-O4**、**-O5**、または **-qsmp** は **-qhot=level=1** を暗黙指定します。

simd | **nosimd**

コンパイラーは、連続する配列エレメントに適用されるループ内の特定の操作を VMX (Vector Multimedia eXtension) 命令の呼び出しに変換します。この呼び出しは、1 度に複数の結果を計算するため、それぞれの結果を順番に計算するより処理は速くなります。

-qhot=nosimd を指定すると、コンパイラーはループと配列に対して最適化を実行しますが、特定のコードを VMX 命令への呼び出しに置き換えることはしません。

このサブオプションは、指定されたアーキテクチャーが VMX 命令をサポートし、**-qenablevmx** に効力があるときにのみ効果があります。

vector | **novector**

-qnostrict または **-O3** 以上の最適化レベルを指定したとき、コンパイラーはある種の操作 (たとえば、平方根、逆数平方根) を **libxlopt.a** ライブラリー内の MASS ライブラリー・ルーチンへの呼び出しに変換します。操作がループしている場合、ルーチンのベクター・バージョンが呼び出されます。操作がスカラーである場合、ルーチンのスカラー・バージョンが呼び出されます。この呼び出しは、1 度に複数の結果を計算するため、それぞれの結果を順次に計算するより処理は速くなります。

-O4 以上を使用している場合、ベクトル化は、プログラムの結果の精度に影響を及ぼすことがあるので、精度についての変更を受け入れられない場合は、**-qhot=novector** を指定する必要があります。

関連情報

- 103 ページの『-qarch オプション』
- 77 ページの『-C オプション』
- 226 ページの『-qstrict オプション』
- 217 ページの『-qsmp オプション』
- 132 ページの『-qenablevmx オプション』
- 88 ページの『-O オプション』
- 「*XL Fortran 言語解説書*」の『*ループ最適化の指示*』
- 「*XL Fortran 最適化およびプログラミング・ガイド*」の『*ハイ・オーダー変換 (HOT) の利点*』

-qieee オプション

構文

```
-qieee={Near | Minus | Plus | Zero}
IEEE({Near | Minus | Plus | Zero})
```

コンパイル時に定数浮動小数点式を評価するときにコンパイラーが使用する丸めモードを指定します。

引数

選択項目は次のとおりです。

Near 最も近い値に丸めます。

Minus マイナスの無限大方向に丸めます。

Plus プラスの無限大方向に丸めます。

Zero ゼロ方向に丸めます。

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することを想定しています。このオプションは、コンパイル時の演算 (たとえば、 **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 (最も近い値への丸め) モード以外に丸めモードを変更する場合は、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。

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*』
- 88 ページの『-O オプション』
- 141 ページの『-qfloat オプション』

-qinit オプション

構文

```
-qinit=f90ptr
INIT(F90PTR)
```

ポインターの初期関連付け状況を関連付け解除にします。これは、Fortran 90 以降に当てはまることに注意してください。

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 言語解説書*」の『*ポインター関連付け*』を参照してください。

-qinitauto オプション

構文

-qinitauto[=*hex_value*] | **-qnoinitauto**

hex_value の長さに応じて、自動変数用のストレージの個々のバイトまたはワード (4 バイト) を、特定の値に初期化します。これにより、定義前に参照される変数を見つけることができます。たとえば、**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、**-qfltrap** 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** 変数および **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。たとえば、 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。たとえば、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。

引数

- *hex_value* は 1 桁から 8 桁の 16 進数 (0-F) です。
- 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。たとえば、*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。

デフォルト

- 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。
- **-qinitauto** に *hex_value* サブオプションを指定しない場合、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。

制限

- 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

144 ページの『-qflttrap オプション』、および「*XL Fortran 言語解説書*」の **AUTOMATIC** ディレクティブについての節を参照してください。

-qinlglue オプション

構文

-qinlglue | **-qnoinlglue**
INLGLUE | **NOINLGLUE**

このオプションは、**-q64** および **-O2** およびより高位のものでコンパイルしているとき、ご使用のアプリケーションで外部関数呼び出しを最適化する Glue コードをインライン化します。

リンカーによって生成された *Glue コード*は、2 つの外部関数間で制御を引き渡すのに使用されます。パフォーマンスの補助のため、最適化プログラムは、ユーザーが適切なプロセッサーを持つマシン上で **-qtune=pwr4**、**-qtune=pwr5**、**-qtune=ppc970**、または **-qtune=auto** を指定してコンパイルするとき、Glue コードを自動的にインライン化します。**-qnoinlglue** はこれらのアーキテクチャー上で Glue コードの自動インライン化を防ぎます。

Glue コードのインライン化はユーザーのコードのサイズを増加させます。**-qcompact** を指定すると **-qinlglue** をオーバーライドしてコードの増大を防ぎます。

-qnoinlglue または **-qcompact** を指定するとパフォーマンスを低下させます。これらのオプションは慎重に使用してください。

関連情報

- 95 ページの『-q64 オプション』
- 120 ページの『-qcompact オプション』
- 236 ページの『-qtune オプション』
- 「*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』
- the 「*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』

-qintlog オプション

構文

-qintlog | **-qnointlog**
INTLOG | **NOINTLOG**

式および文内に整数と論理データ・エンティティーを混在させることができることを指定します。 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。

制限

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

関連情報

- **-qport=clogicals** オプション。

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
END
```

関連情報

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。

-qintsize オプション

構文

```
-qintsize=bytes
INTSIZE(bytes)
```

デフォルトの **INTEGER** および **LOGICAL** データ・エンティティー (つまり、長さまたは種類が指定されていないデータ・エンティティー) のサイズを設定します。

背景情報

指定されたサイズ[1] は、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。
- 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。

バイト に使用できるサイズは次のとおりです。

- 2
- 4 (デフォルト)
- 8

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。たとえば、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。このオプションのデフォルト値 4 は、多くの 32 ビット・コンピューター用に書かれたコードに適しています。**-q64** コンパイラー・オプションの指定は、**-qintsize** のデフォルト設定に影響しないことに注意してください。

制限

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。

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。

---

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。

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

204 ページの『-qrealsize オプション』および「*XL Fortran 言語解説書*」の 『*タイプ・パラメーターおよび指定子*』を参照してください。

-qipa オプション

構文

-qipa[=*suboptions*] | **-qnoipa**

プロシージャー間で詳細な分析 (プロシージャー間分析、つまり IPA) を行うことによって、**-O** 最適化を増大させます。

-qipa を指定するときには、**-O**、**-O2**、**-O3**、**-O4**、または **-O5** オプションも指定する必要があります。 (**-O5** オプションを指定することは、**-O4** オプションと **-qipa=level=2** を指定することと同じです。) パフォーマンスをさらに改善するために、**-Q** オプションを指定することもできます。 **-qipa** は、最適化実行中、および単一プロシージャーから複数プロシージャー (おそらく別のソース・ファイル内にある) へのインライン化実行中、そして、それらのリンク実行中に調べられる区域を拡張します。

サブオプションを指定することによって、実行される最適化を微調整することができます。

このオプションを使用するために必要なステップは、次のとおりです。

1. **-qipa** オプションを指定してコンパイルする前に、予備のパフォーマンス分析および調整を行います。これが必要なのは、プロシージャー間分析はリンク時間を長引かせる 2 パス方式 (コンパイル時間とリンク時間のフェーズ) を使用するためです。 (**noobject** サブオプションを使用してこのオーバーヘッドを削減することができます。)
2. アプリケーション全体またはできるだけ多くの部分で、コンパイル・ステップとリンク・ステップの両方に **-qipa** オプションを指定します。 **-qipa** を指定してコンパイルしないプログラムの部分に関して、何を前提事項にするかを示すサブオプションを指定します。 (アプリケーションに IBM XL C/C++ コンパイラーでコンパイルされた C または C++ コードが含まれている場合は、リンク時にさらに最適化の機会をもたらすために **-qipa** オプションを指定して C または C++ コードをコンパイルする必要もあります。)

   コンパイル中に、コンパイラーは **.o** ファイルにプロシージャー間分析情報を格納します。リンク中に、**-qipa** オプションはアプリケーション全体の完全な最適化を再発生させます。

-# 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。

引数

コンパイル時のサブオプション

IPA は、コンパイル時間のフェーズで以下のサブオプションを使用します。

object | **noobject**
: オブジェクト・ファイルに標準オブジェクト・コードを組み込むかどうかを指定します。 **noobject** サブオプションを指定すると、最初の IPA フェーズ中にオブジェクト・コードを生成しないことにより、全体的なコンパイル

時間は大きく短縮されます。 **-S** と **noobject** を同時に指定すると、**noobject** は無視されることに注意してください。

コンパイルとリンクを同じステップで実行した場合で、 **-S** またはリストされているオプションを指定しない場合、**-qipa=noobject** が暗黙指定されます。

プログラムに **noobject** サブオプションを使用して作成したオブジェクト・ファイルが含まれる場合、 **-qipa** を使用してプログラムをリンクする前に、エントリー・ポイント (実行可能プログラムの場合は主プログラム、ライブラリーの場合はエクスポートされたプロシージャー) を含むファイルがあれば **-qipa** オプションを指定してコンパイルする必要があります。

リンク時のサブオプション

IPA は、リンク時間のフェーズで以下のサブオプションを使用します。

clonearch=*arch***{,***arch***}** | **noclonearch**

同一の命令セットの複数のバージョンが作成されるアーキテクチャーを指定します。ご使用のアプリケーションの同一コピーを複数の異なる機械で最適のパフォーマンスを必要とする場合、このサブオプションを使用してください。

コンパイラーは効果のある **-qarch** 設定に基づき、命令セットの汎用版を生成し、該当する場合、**clonearch** サブオプションでユーザーが指定するアーキテクチャー用に *clones* 特殊バージョンを生成します。コンパイラーはコードをご使用のアプリケーションに挿入して実行時にプロセッサー・アーキテクチャーを検査します。実行時に、ランタイム環境に対して最良に最適化された、命令セットのアプリケーションのバージョンが選択されます。

arch は、コンマで区切られた、アーキテクチャー値のリストです。サポートされる clonearch 値は **pwr4**、**pwr5**、および **ppc970** です。値を指定しない、無効値を指定する、または **-qarch** 設定値に等しい値を指定する場合、このオプションに対して関数のバージョン管理は実行されません。

注:

1. 複数のプラットフォームにわたる互換性を確実にするには、**-qarch** 値は **-qarch=clonearch=***arch* によって指定されたアーキテクチャーのサブセットでなければなりません。
2. **-qcompact** が効力のある場合、**-qarch=clonearch=***arch* は使用不可です。

表 16. 互換性のあるアーキテクチャーおよび clonearch 設定値

-qarch 設定	**許可された clonearch 値**
com、ppc、pwr3、ppc64、ppcgr、ppc64gr、ppc64grsq	pwr4, pwr5, ppc970
pwr4	pwr5, ppc970
ppc64v	ppc970

cloneproc=*name***{,***name***}** | **nocloneproc[=***name***{,***name***}]**

clonearch サブオプションにより指定されたアーキテクチャーを複製するための機能の名前を指定します。ここで、*name* は関数名のコンマで区切られたリストです。

注: **-qipa=clonearch=***arch* を指定しない、または **-qipa=noclonearch** を指定する場合、**-qipa=cloneproc=***name* および **-qipa=nocloneproc=***name* は効果がありません。

exits=*procedure_names*

プロシージャーのリストを指定します。それぞれのプロシージャーがプログラムを終了させることになります。コンパイラーはこれらのプロシージャーの呼び出しを最適化する (たとえば、保管/復元手順の除去により) ことができます。これらのコードがプログラムに戻ることはないからです。これらのプロシージャーは、**-qipa** を指定してコンパイルされたプログラムの他の部分を呼び出してはいけません。

inline=*inline-options*

-qipa=inline= コマンドは、以下に示すインライン・オプションのリスト (コロンで区切る) を取得できます。

auto | **noauto**

プロシージャーを自動的にインライン化するかどうかを指定します。

limit=*number*

inline=auto オプションがインライン展開の程度を決定するために使用する限界サイズを変更します。呼び出し側のプロシージャーの限界サイズは、この確立された「limit」よりも下である必要があります。 *number* には、生成されるコードのバイト数を最適化プログラムに見合った概算で入れます。数字が大きいほど、コンパイラーはインライン化サブプログラムを大きくするか、インライン化サブプログラム呼び出しを増やすか、またはその両方を行うことができます。この引数は、「**inline=auto**」がオンになっているときにのみ実施されます。

procedure_names

インライン化を試行するプロシージャーのリストを指定します。

threshold=*number*

インライン化するプロシージャーの限界サイズの上限を指定します。 *number* は、インライン・サブオプション「limit」で定義された値です。この引数は、「inline=auto」がオンになっているときにのみ実施されます。

注: デフォルトでは、コンパイラーは、 **inline=** *procedure_names* サブオプションを使って指定したプロシージャーだけではなく、すべてのプロシージャーのインライン化を試行します。特定のプロシージャーだけをインライン化する場合は、 **inline=** *procedure_names* を指定してから、**inline=noauto** を指定してください。 (この順番でサブオプションを指定する必要があります。) たとえば、**sub1** のプロシージャー以外のすべてのプロシージャーのインライン化をオフにするには、**-qipa=inline=sub1:inline=noauto** と指定します。

isolated=*procedure_names*

-qipa を指定してコンパイルされていないプロシージャーのリストをコンマ

で区切って指定します。「isolated」として指定されたプロシージャーや、呼び出しチェーン内のプロシージャーは、グローバル変数を直接に参照することはできません。

level=*level*

実行されるプロシージャー間分析および最適化のレベルを決定します。

0 最小限のプロシージャー間分析と最適化のみを行います。

1 インライン化、限定された別名分析、限定された呼び出し側の調整をオンにします。

2 完全なプロシージャー間データ・フロー分析と完全な別名分析を行います。 **-O5** を指定することは、**-O4** と **-qipa=level=2** を指定することと同じです。

デフォルト・レベルは **1** です。

list[=*filename* **| short | long]**

-qlist コンパイラー・オプションまたは **-qipa=list** コンパイラー・オプションによってオブジェクト・リストが要求されたイベントにおいて、リンク・フェーズ中に出力リスト・ファイル名を指定することにより、ユーザーが出力のタイプを指示できるようにします。 *filename* サブオプションを指定しなかった場合、デフォルト・ファイル名は「a.lst」になります。

short を指定した場合は、オブジェクト・ファイル・マップ、ソース・ファイル・マップ、グローバル・シンボル・マップのセクションが組み込まれます。**long** を指定した場合は、オブジェクト解像度警告、オブジェクト参照マップ・セクション、インライン報告書、区画マップ・セクションに加えて、それに先行するセクションも表示されます。

-qipa オプションと **-qlist** オプションを同時に指定すると、IPA は `a.lst` ファイルを生成し、既存の `a.lst` ファイルがあればそれらを上書きします。 `a.f` というソース・ファイルがあるとすると、IPA のリストにより、通常のコンパイラー・リスト `a.lst` が上書きされます。代替のリスト・ファイル名を指定するときには、 **list=***filename* サブオプションを使用できます。

lowfreq=*procedure_names*

通常のプログラムの実行過程でまれに呼び出されるようなプロシージャーのリストを指定します。たとえば、初期化およびクリーンアップのためのプロシージャーは一度だけ呼び出されて、デバッグ・プロシージャーは運用レベルのプログラムではまったく呼び出されないこともあります。 これらのプロシージャーへの呼び出しに対して行う最適化を少なくすることによって、コンパイラーはプログラムの他の部分をより速くすることができます。

missing={unknown | safe | isolated | pure}

-qipa を指定してコンパイルされておらず、また **unknown**、**safe**、**isolated**、または **pure** サブオプション内で明示的に名前付けされていないプロシージャーの、プロシージャー間動作を指定します。デフォルトでは **unknown** を想定します。これにより、これらのプロシージャーの呼び出しに対するプロシージャー間最適化の量が大きく制限されます。

noinline=*procedure_names*

インライン化しないプロシージャーのリストを指定します。

partition={small | medium | large}

分析するプログラム内の領域サイズを指定します。区画が大きいほど多くのプロシージャーを入れることができ、結果としてより優れたプロシージャー間分析を行うことができますが、最適化するストレージがそれだけ多く必要になります。ページングのためにコンパイル時間が長すぎる場合は、区画サイズを小さくしてください。

pdfname=[*filename***]**

PDF プロファイル情報を含むプロファイル・データ・ファイルの名前を指定します。 ***filename*** を指定しない場合は、デフォルト・ファイル名は、**__pdf** になります。プロファイルは現行作業ディレクトリーか、**PDFDIR** 環境変数が指名しているディレクトリーに置かれます。これにより、ユーザーは、同時に複数の実行可能ファイルを同じ **PDFDIR** を使用して稼動することができます。こうすると、動的ライブラリーの **PDF** を使用したチューニングに特に有効です。 (チューニングの最適化についての詳細は、 191 ページの『-qpdf オプション』を参照してください。)

pure=*procedure_names*

-qipa を指定してコンパイルされていないプロシージャーのリストを指定します。「pure」として指定したプロシージャーは、「isolated」および「safe」でなければなりません。また、呼び出し元から見えるデータ・オブジェクト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=*procedure_names*

-qipa を指定してコンパイルされていないプロシージャーのリストを指定します。「safe」として指定されているプロシージャーは、グローバル変数および仮引数を修正する場合があります。「safe」プロシージャーの呼び出しチェーン内からは、**-qipa** を指定してコンパイルされたプロシージャーへの呼び出しが行われない場合があります。

stdexits | nostdexits

特定の事前定義ルーチンが、**exits** サブオプションを指定している場合のように最適化可能であることを指定します。該当するプロシージャーは、**abort**、**exit**、**_exit**、**_assert** です。

threads[=N] | nothreads

threads[=N] により、使用可能な数、あるいは *N* で指定した数の並列スレッドが稼動します。 *N* は正整数でなければなりません。 **nothreads** では、並列スレッドは稼動しません。これは、1 つのシリアル・スレッドを稼動するのと同じです。

-qipa=threads を指定すると、IPA 最適化時間を削減することができます。スレッド・サブオプションを指定すると、IPA 最適化プログラムは最適化処理の部分を並列スレッドで実行することができるため、マルチプロセッサー・システム上でコンパイル・プロセスを高速化することができます。

unknown=*procedure_names*

-qipa を指定してコンパイルされていないプロシージャーのリストを指定します。「unknown」として指定されたプロシージャーは、**-qipa** を指定してコンパイルされたプログラムの他の部分の呼び出しを行い、グローバル変数と仮引数を修正する場合があります。

isolated、**missing**、**pure**、**safe**、および **unknown** の主な使用目的は、 **-qipa** を指定してコンパイルしていないライブラリー・ルーチンへの呼び出しに対して、最適化をどの程度安全に実行するかを指定することです。

以下のコンパイラー・オプションは、**-qipa** のリンク時間のフェーズに影響があります。

-qlibansi | -qnolibansi
ANSI C が定義したライブラリー関数の名前を持つ関数はすべてライブラリー関数と見なすことを指定します。

-qlibposix | -qnolibposix
IEEE 1003.1-2001 (POSIX) が定義したライブラリー関数の名前を持つ関数はすべてシステム機能と見なすことを指定します。

-qthreaded
コンパイラーが、スレッド・セーフ・コードの生成を試行すると見なします。

規則

正規表現は、以下のサブオプションでサポートされています。

exits

inline

lowfreq

noinline

pure

safe

unknown

正規表現の構文規則について、以下に説明します。

表 17. 正規表現の構文

式	説明
`string`	`string` で指定したすべての文字と突き合わせます。たとえば、`test` は、`testimony`、 `latest`、`intestine` と突き合わせます。
`^string`	行の先頭に置かれている場合にのみ、 `string` によって指定されているパターンと突き合わせます。
`string$`	行の末尾に置かれている場合にのみ、 `string` によって指定されているパターンと突き合わせます。
`str.ing`	すべての文字と突き合わせます。たとえば、`t.st` は `test`、`tast`、`tZst`、および `t1st` と突き合わせます。
`string¥.$`	円記号 (¥) は、その文字を突き合わせるための特殊エスケープ文字に使用することができます。たとえば、ピリオドで終了する行を検索する場合、 `.$` という式では、最低でも 1 つの文字を含むすべての行が表示されます。ピリオド (.) をエスケープするには、`¥.$` と指定します。

表 17. 正規表現の構文 (続き)

式	説明
[string]	string で指定したすべての文字と突き合わせます。たとえば、t[a-g123]st は tast および test とは突き合わせますが、 t-st または tAst とは突き合わせません。
[^string]	string で指定した文字とは突き合わせません。たとえば、t[^a-zA-Z]st は t1st、 t-st、および t,st とは突き合わせますが、 test または tYst とは突き合わせません。
string*	string によって指定されたパターンの、ゼロ回以上のオカレンスと突き合わせます。たとえば、te*st は、tst、test、および teeeeeest と突き合わせます。
string+	string によって指定されたパターンの、1 回以上のオカレンスと突き合わせます。たとえば、t(es)+t は、test、tesest とは突き合わせますが、tt. との突き合わせは行いません。
string?	string によって指定されたパターンの、ゼロ回以上のオカレンスと突き合わせます。たとえば、te?st は、tst または test のいずれかと突き合わせます。
string{m,n}	string によって指定されたパターンの m 回と n 回の間のオカレンスを突き合わせます。たとえば、a{2} は aa と、また b{1,4} は b、bb、bbb、および bbbb とそれぞれ突き合わせます。
string1 | string2	string1 または string2 のいずれかによって指定されたパターンと突き合わせます。たとえば、s | o は s と o の両方の文字と突き合わせます。

関数名だけが考慮されるため、正規表現は自動的に ^ および $ 文字で囲まれます。たとえば、**-qipa=noinline=^foo$** は **-qipa=noinline=foo** と同一です。 そのため、 **-qipa=noinline=bar** の場合、**bar** は決してインライン化されませんが、**bar1**、**teebar**、**barrel** はインライン化される可能性があります。

例

一連のファイルをプロシージャー間分析でコンパイルする方法を次に示します。

```
xlf95 -O -qipa f.f
xlf95 -c -O3 *.f -qipa=noobject
xlf95 -o product *.o -qipa -O
```

次の例では、パフォーマンスを向上させるために正規表現を使って、上で示したのと同じファイルとプロシージャー間分析とをリンクする方法を示します。この例では、関数 **user_abort** はプログラムを終了し、その関数のルーチン **user_trace1**、**user_trace2**、および **user_trace3** はまれにしか呼び出されないものと想定します。

```
xlf95 -o product *.o -qipa=exit=user_abort:lowfreq=user_trace[123] -O
```

関連情報

88 ページの『-O オプション』、 92 ページの『-p オプション』、および 93 ページの『-Q オプション』を参照してください。

-qkeepparm オプション

構文

-qkeepparm | **-qnokeepparm**

背景情報

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、最適化を行ってコードをコンパイルすると、最適化プログラムは必要であれば、スタックに対するストアを除去する可能性があります。

-qkeepparm コンパイラー・オプションを指定すると、最適化していてもそのパラメーターがスタックに保管されることが保証されます。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。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。

-qlanglvl オプション

構文

```
-qlanglvl={suboption}
LANGLVL({suboption})
```

非準拠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) を決定します。非準拠のソース・コード、およびそのような非準拠を許可するオプションを識別します。

規則

コンパイラーは、言語レベルで許可されていない構文を指定していることを検出すると、重大度コード **L** のメッセージを送出します。

引数

77std ANSI FORTRAN 77 標準で指定されている言語を受け入れて、他はすべてエラーとして報告します。

90std ISO Fortran 90 標準で指定されている言語を受け入れて、他はすべてエラーとして報告します。

90pure 廃止 Fortran 90 機能が使用されたことに対するエラーを報告する以外は、**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、他はすべてエラーとして報告します。

95pure 廃止 Fortran 95 機能が使用されたことに対するエラーを報告する以外は、 **95std** と同じです。

2003std XL Fortran がサポートするすべての Fortran 2003 の機能とともに、ISO Fortran 95 標準で指定されている言語を受け入れて、他はすべてエラーとして報告します。

2003pure 廃止 Fortran 2003 機能が使用されたことに対するエラーを報告する以外は、 **2003std** 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 95 言語標準、XL Fortran がサポートするすべての Fortran 2003 の機能、およびすべての拡張機能をコンパイラーが受け入れるように指定します。

デフォルト

デフォルトは **-qlanglvl=extended** です。

制限

-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。

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
!------------------------------------------------------------
```

次の図には、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** 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 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

140 ページの『-qflag オプション』、 148 ページの『-qhalt オプション』、および 210 ページの『-qsaa オプション』を参照してください。

33 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。

-qlibansi オプション

構文

-qlibansi | **-qnolibansi**

ANSI C ライブラリー関数の名前を持つ関数はすべてシステム機能と見なすことを指定します。

このオプションにより、最適化プログラムは、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。

注: お客様のアプリケーションに、標準の C 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 C 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しないでください。

関連情報

160 ページの『-qipa オプション』を参照してください。

-qlibposix オプション

関連情報

160 ページの『-qipa オプション』を参照してください。

-qlinedebug オプション

構文

-qlinedebug | **-qnolinedebug**
LINEDEBUG | **NOLINEDEBUG**

デバッガーに対して行番号およびソース・ファイル名情報を生成します。

このオプションは最小のデバッグ情報を作り出すので、結果のオブジェクト・サイズは **-g** デバッグ・オプションが指定された場合に作り出すオブジェクト・サイズより小さいのです。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。

-g と同様に、使用される最適化レベルが高ければ高いほど、行番号を含め、デバッグ情報はより誤解を招きやすくなります。

-qlinedebug オプションを指定する場合、インライン・オプションはデフォルトの **-Q!** になります (関数はインラインされません)。

-qnolinedebug オプションは、**-g** に効果がありません。

関連情報

- 82 ページの『-g オプション』
- 88 ページの『-O オプション』

-qlist オプション

構文

```
-qlist[=offset | nooffset] | -qnolist
LIST[([NO]OFFSET)] | NOLIST
```

オブジェクト・リストを含むコンパイラー・リストを作成します。

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 オプションと **-qlist** オプションを同時に指定すると、IPA は `a.lst` ファイルを生成し、既存の `a.lst` ファイルがあればそれらを上書きします。 `a.f` というソース・ファイルがあるとすると、IPA のリストにより、通常のコンパイラー・リスト `a.lst` が上書きされます。これを回避するには、**-qipa** の **list=***filename* サブオプションを使用して、代わりのリストを生成してください。

注

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 **-qlist=offset** オプションは関係があります。たとえば、プログラムでネストされたプロシージャーが使用される場合、これは発生することがあります。

-qlist=offset を指定する場合、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。

PDEF ヘッダーのオフセットには、テキスト域の開始からのオフセットが含まれます。これにより、`.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、**-qlist=offset** または **-qlist=nooffset** が指定されたかを示す同一の値を生産します。

-qlist オプションを指定すると **-qlist=nooffset** と見なされます。

関連情報

- 58 ページの『リストとメッセージを制御するオプション』
- 282 ページの『オブジェクト・セクション』
- 254 ページの『-S オプション』
- 「*XL Fortran 言語解説書*」の『プログラム単位およびプロシージャー』

-qlistopt オプション

構文

-qlistopt | **-qnolistopt**
LISTOPT | **NOLISTOPT**

リスト・ファイル内のすべてのコンパイラー・オプションの設定を表示するか、または、選択したオプションだけを表示するかを決定します。これらの選択したオプションには、コマンド行またはディレクティブに指定されているオプションと、常にリストにあるオプションが含まれます。

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。

規則

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「オン/オフ」オプション。たとえば、**-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「オン/オフ」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。

関連情報

58 ページの『リストとメッセージを制御するオプション』および 278 ページの『オプション・セクション』を参照してください。

-qlog4 オプション

構文

-qlog4 | **-qnolog4**
LOG4 | **NOLOG4**

論理オペランドを持つ論理演算の結果が、**LOGICAL(4)** であるか、それともオペランドの最大長を持つ **LOGICAL** であるかを指定します。

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。

引数

-qlog4 は常に結果を **LOGICAL(4)** にし、 **-qnolog4** は結果をオペランドの長さに依存させます。

制限

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。

-qmaxmem オプション

構文

-qmaxmem=*Kbytes*
MAXMEM(*Kbytes*)

コンパイラーが特定のメモリー集中の最適化を実行するときに、割り振るメモリーの量を指定キロバイト数に制限します。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。

デフォルト

-O2 最適化レベルでは、デフォルトの **-qmaxmem** 設定は 8,192 KB です。 **-O3** 最適化レベルでは、デフォルトの設定は無制限 (-1) です。

規則

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。**マシン・ラン-O3** を指定してコンパイルするときには、マシン・ラン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 8,192 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・スペースを超えた場合は、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) を超えた場合は、自動的にはオフになりません。

制限

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、リソースを使い果たす場合があります。

関連情報

- 88 ページの『-O オプション』
- 「*XL Fortran 最適化およびプログラミング・ガイド*」の『*XL コンパイラー・プログラムの最適化*』

-qmbcs オプション

構文

-qmbcs | **-qnombcs**
MBCS | **NOMBCS**

文字リテラル定数、ホレリス定数、H 編集記述子、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) 文字または Unicode 文字を含めることができるかどうかをコンパイラーに示します。

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。

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)。

規則

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。

制限

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。

-qminimaltoc オプション

構文

-qminimaltoc | **-qnominimaltoc**

このコンパイラー・オプションは、目次 (TOC) の生成を変更できます。コンパイルを 64 ビット・モードで行った際、コンパイラーは各実行可能ファイルごとに作成します。デフォルトでは、プログラム内の固有の、非自動変数参照ごとに、少なくとも 1 つの TOC エントリーをコンパイラーが割り当てます。 現在、8192 の TOC エントリーのみが使用可能で、重複エントリーは廃棄されません。これは、プログラムが 8192 の TOC エントリーを超える場合、64 ビット・モードで大きなプログラムをリンクしている際に、エラーが起きる原因になります。

-qminimaltoc を指定すると、コンパイラーが、各コンパイル単位ごとに、1 つの TOC エントリーのみ作成します。このオプションの指定は、使用可能な TOC エントリーの使用を最小限にします。しかし、この使用はパフォーマンスに影響します。**-qminimaltoc** オプションは、特に頻繁に実行されるコードを含むファイルでは、個別に使用してください。

-qmixed オプション

構文

-qmixed | **-qnomixed**
MIXED | **NOMIXED**

これは、 256 ページの『-U オプション』の長い形式です。

-qmoddir オプション

構文

-qmoddir=*directory*

コンパイラーが書き込むモジュール・ファイル (**.mod**) の位置を指定します。

デフォルト

-qmoddir を指定しない場合、 **.mod** ファイルは現行ディレクトリーに置かれます。

モジュールを参照するファイルをコンパイルしているときに、このディレクトリーから **.mod** ファイルを読むには、 83 ページの『-I オプション』を使用してください。

関連情報

22 ページの『XL Fortran 出力ファイル』を参照してください。

モジュールは Fortran 90 または 95 の機能であり、「*XL Fortran 言語解説書*」の『モジュール』の節で説明されています。

-qmodule オプション

構文

```
-qmodule=mangle81
```

コンパイラーが非組み込みモジュール・ファイルに対して XL Fortran バージョン 8.1 の命名規則を使用するように指定します。

このオプションを使用すると、モジュールと Version 10.1 のコンパイラーに関連したオブジェクト・ファイルを生成して、これらのオブジェクト・ファイルと、バージョン 8.1 のコンパイラーでコンパイルされた他のオブジェクト・ファイルをリンクすることができます。

Version 8.1 コンパイラーでコンパイルされたアプリケーションをリンクする必要がある場合にのみこのオプションを使用してください。可能な場合、このコンパイラー・スイッチの使用を避け、コンパイラーの新規バージョンで古いコードおよびモジュールを再ビルドすることをお勧めします。そのようにすることによりご使用のモジュールと組み込みのコンパイラー・モジュールの間の命名の競合が避けられます。

関連情報

- 「*XL Fortran 言語解説書*」の『モジュール』 セクション。(モジュールは Fortran 90/95 フィーチャーです。)
- 「*XL Fortran 最適化およびプログラミング・ガイド*」の『*XL Fortran の外部名の規則*』
- 31 ページの『リンク中の命名競合の回避』

-qnoprint オプション

構文

```
-qnoprint
```

他のリスト・オプションの設定とは関係なく、コンパイラーがリスト・ファイルを作成しないようにします。

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。

規則

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

58 ページの『リストとメッセージを制御するオプション』を参照してください。

-qnullterm オプション

構文

-qnullterm | **-qnonullterm**
NULLTERM | **NONULLTERM**

仮引数として渡される文字定数式に NULL 文字を付加することによって、ストリングを C 関数に渡しやすくします。

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。

背景情報

このオプションの影響を受けるのは、基本文字定数、複数の文字定数を連結したもの、文字型の名前付き定数、ホレリス定数、2 進、8 進、16 進の型なし定数から構成された引数 (インターフェース・ブロックが使用可能な場合) か、それらのオブジェクトから全体が構成されているすべての文字式です。 **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初期設定式である場合は 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。

制限

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『*言語間の文字タイプの引き渡し*』を参照してください。

-qobject オプション

構文

-qOBJect | -qNOOBJect
OBJect | NOOBJect

オブジェクト・ファイルを作成するか、または、ソース・ファイルの構文をチェックした直後に停止するかを指定します。

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、コード生成のオーバーヘッドなしに、プログラムの構文をすばや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。

制限

-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

58 ページの『リストとメッセージを制御するオプション』および 282 ページの『オブジェクト・セクション』を参照してください。

283 ページの『コンパイラー・フェーズ』には、コンパイラー・フェーズに関する技術情報が記載されています。

-qoldmod オプション

構文

```
-qoldmod=compat
```

このオプションは、V9.1.1 より前の XL Fortran のバージョンでコンパイルされたモジュール・データを含むオブジェクト・ファイルが未初期化のモジュール・データを含まないかまたは **convert_syms** スクリプトを使用して移植するかを指定します。

XL Fortran が初期化されていないモジュール変数用に使用するマングリング・スキームは、Version 9.1.1 で変更されました。前のマングリング・スキームは @ 記号を使用していて、それが共用ライブラリーの作成時に Linux リンカーを混乱させました。Linux リンカーは、 @ 記号をシンボルのバージョン管理用に予約しています。

この変更の結果として、未初期化のモジュール変数を含む XL Fortran およびこれらのモジュールを使用するすべてのコンパイル単位は、**-qoldmod=compat** オプションを指定して再コンパイルする必要があります。

再コンパイルが実現可能でない場合、古いコンパイラーにより作成されたオブジェクト・ファイルを次のスクリプトの使用により移植することができます。

```
> cat convert_syms
#!/bin/bash
if [[ $# = 0 || "$1" = "-?" ]]
then
  echo "Usage: $0 object-files"
  echo "e.g.   $0 *.o *.a"
  exit
fi

for file in $*
do
  XLFSYMS=""
  for i in `nm $file | grep -o "&&¥([NI]&¥)¥?@.*$" | sort -u`
  do
     XLFSYMS="$XLFSYMS --redefine-sym $i=`echo $i | sed -e 's/@/¥&/'`"
  done
  if [[ -n "$XLFSYMS" ]]; then
    echo Converting symbols in $file...
    objcopy $XLFSYMS $file
  fi
done
```

モジュール・シンボル・ファイル (*.mod) は移植する必要はありません。

例

```
# If old_module.o and old_module.mod were produced using the original
# release of XL Fortran Advanced Edition V9.1 for Linux, you'll get
# the following error when you try to use them in new programs:
#
> xlf95 new_program.f old_module.o
"new_program.f", line 1.5: 1517-022 (S) Module old_module was compiled
using an incompatible version of the compiler.  Please see the -qoldmod
option for information on recovery.
1501-511  Compilation failed for file new_program.f.

# Convert the object file.  This needs to be done only once.
```

```
#
> ./convert_syms old_module.o
>

# You can now use the updated object file safely.
# Specify -qoldmod=compat
#
> xlf95 new_program.f old_module.o -qoldmod=compat
** _main   === End of Compilation 1 ===
1501-510  Compilation successful for file new_program.f.
>
```

-qonetrip オプション

構文

-qonetrip | **-qnoonetrip**
ONETRIP | **NOONETRIP**

これは、 75 ページの『-1 オプション』の長い形式です。

-qoptimize オプション

構文

-qOPTimize[=*level*] | **-qNOOPTimize**
OPTimize[(*level*)] | **NOOPTimize**

これは、 88 ページの『-O オプション』の長い形式です。

-qpdf オプション

構文

```
-qpdf{1|2}
```

プロファイル指示フィードバック (*profile-directed feedback* (PDF)) によって最適化を調整します。その場合、条件付き分岐の近辺および頻繁に実行されるコード・セクション内の最適化が、サンプル・プログラムの実行結果を使用して改善されます。

PDF を使用する場合、次のステップに従ってください。

1. **-qpdf1** オプションを使用してプログラム内の一部またはすべてのソース・ファイルをコンパイルします。最適化のために、**-O2** オプションか、あるいは、**-O3**、**-O4**、 **-O5** オプションのいずれか (こちらの方が望ましい) を指定する必要があります。ファイルのコンパイルに使用するコンパイラー・オプションには特に注意してください。後で同じオプションを使用する必要が生じます。

   大きなアプリケーションでは、最適化から最も利益を得られる範囲のコードを集中的に作成します。アプリケーションのコードすべてを **-qpdf1** オプション付きでコンパイルする必要はありません。

   -qpdf2 ステップでフル再コンパイルを避けたい場合、オブジェクト・ファイルが保管されるように、リンクなしで **-qpdf1** を指定して最初にコンパイルしてください。次にオブジェクト・ファイルを実行可能モジュールにリンクしてください。この手順を使用するには、以下と同様のコマンドを発行してください。

   ```
   xlf -c -qpdf1 -O2 file1.f file2.f
   xlf -qpdf2 -O2 file1.o file2.o
   ```

2. 一般的なデータ・セットを使用してプログラムをひととおり実行します。プログラムは終了時にプロファイル情報を記録します。さまざまなデータ・セットを使用してプログラムを何回も実行すると、プロファイル情報が累積し、分岐やコード・ブロックが実行される頻度の正確なカウントが得られます。

   重要: 完成したプログラムを通常実行するときに使用されるデータを代表するようなデータを使用してください。

3. 前と同じコンパイラー・オプションを使用し、 **-qpdf1** を **-qpdf2** に変更してプログラムを再コンパイルおよび再リンクします。 **-L** や **-l** などはリンカー・オプションであることに留意してください。それらのオプションはこの時点で変更できます。 この 2 回目のコンパイルでは、累積したプロファイル情報を使用して最適化を微調整できます。結果として生成されたプログラムにはプロファイルのオーバーヘッドは含まれておらず、フル・スピードで実行されます。

   あるいは、オブジェクト・ファイルを 1 から保管した場合、単にそれらを再リンクします。たとえば、次のコマンドを発行します。

   ```
   xlf -qpdf2 -O2 file1.o file2.o
   ```

   これにより、かなりの時間を節約でき、最適化のために大規模アプリケーションを微調整できます。**-qpdf2** パスで異なるオプションを使用することにより、PDF バイナリーを異なる最適化の方法で作成し、テストできます。

最善のパフォーマンスを得るため、PDF を使用する場合は (上記の例のように) あらゆるコンパイルに **-O3**、**-O4**、または **-O5** オプションを使用してください。 IBM XL C/C+ コンパイラーでコンパイルした C または C++ コードがアプリケーションに含まれている場合、これらのコンパイラーで使用可能な **-qpdf1** および **-qpdf2** オプションを指定して、追加の PDF 最適化を実行することができます。すべての Fortran および C/C++ コード上で **-qpdf1/-qpdf2** と **-qipa** または **-O5** オプションを組み合わせると (すなわち、IPA とリンクすると)、最適化に使用可能な最大 PDF 情報をもたらします。

規則

プロファイルは現行作業ディレクトリーか、**PDFDIR** 環境変数が設定されている場合はその変数が指定するディレクトリーに置かれます。

コンパイルと実行で時間を無駄にしないために、**PDFDIR** 環境変数には必ず絶対パスを指定してください。そうしなかった場合、アプリケーションを誤ったディレクトリーから実行し、アプリケーションがプロファイル・データ・ファイルを見つけられないこともあります。その場合、プログラムが正しく最適化されなかったり、セグメント化障害によって停止する可能性があります。 セグメント化障害は、**PDFDIR** 変数の値を変更し、 PDF プロセスを完了する前にアプリケーションを実行した場合にも起きることがあります。

背景情報

このオプションは、アプリケーション全体を 2 回コンパイルする必要があるので、他のデバッグと調整が済んだ後、アプリケーションを実動させる前の最終ステップの 1 つとして使用されるよう設計されています。

制限

- PDF 最適化には、少なくとも **-O2** の最適化レベルも必要です。
- 実行時にプロファイル情報を収集するため、メインプログラムは必ず PDF を使用してコンパイルしなければなりません。
- プロファイル情報のセットを識別するために **-qipa=pdfname** サブオプションを使用していない限り、同時に同じ **PDFDIR** ディレクトリーを使用する 2 つの別々のアプリケーションをコンパイルしたり、実行しないでください。
- ある特定のプログラムについて、すべてのコンパイル・ステップで同じセットのコンパイラー・オプションを使用する必要があります。そのようにしなかった場合、PDF はプログラムを正しく最適化できず、むしろ処理速度が落ちる場合さえあります。コンパイラーの設定は、構成ファイルによって供給されたものも含め、すべて同じでなければなりません。
- **-qipa** を直接あるいは他のオプションを通じて起動していない場合は、**-qpdf1** および **-qpdf2** は、**-qipa=level=0** オプションを起動します。
- **-qpdf1** を使用してプログラムをコンパイルする場合は、実行時にプロファイル情報が生成され、それにパフォーマンス・オーバーヘッドが含まれることに注意してください。このオーバーヘッドは、**-qpdf2** を使用するか、PDF をまったく使用しないで再コンパイルすると解消されます。

以下のコマンドは **/opt/ibmcmp/xlf/10.1/bin** ディレクトリーにあり、**PDFDIR** ディレクトリーの管理に使用できます。

`cleanpdf [pathname]`

pathname ディレクトリーから、または *pathname* が指定されていない場合は **PDFDIR** ディレクトリーから、すべてのプロファイル情報を削除します。また、**PDFDIR** が設定されていない場合は、現行ディレクトリーから削除します。

プログラムを変更して PDF プロセスを再度実行した場合、プロファイル情報を除去すると、実行時オーバーヘッドが低減されます。

このプログラムは、**-qpdf2** でコンパイルした後か、特定のアプリケーションについて PDF を済ませた後に実行してください。 **cleanpdf** を実行した後にアプリケーションで PDF を引き続き使用する場合は、 **-qpdf1** を指定してすべてのファイルを再コンパイルする必要があります。

`mergepdf`

2 つ以上の入力 PDF レコードから単一の PDF レコードを生成します。すべての PDF レコードは、同じ実行可能ファイルからのものでなければなりません。

mergepdf は、各 PDF レコード (すなわちファイル) をその「重み」に基づいて自動的にスケーリングします。スケーリング比率は、それぞれの PDF レコードごとに指定することができるため、重要性の高いトレーニング・ランには、重要性の低いものよりも重みを大きくすることができます。 **mergepdf** の構文は次のとおりです。

```
mergepdf [-r1] record1 [-r2] record2 ... -outputrecname [-n] [-v]
```

上記の意味は次のとおりです。

-r PDF レコードのスケーリング比率。入力レコードに対して -r を指定しない場合、デフォルトの比率は 1.0 で、外部スケーリングは適用されません。スケーリング比率は、ゼロ以上でなければなりません。また、浮動小数点数または整数でもかまいません。

record PDF プロファイルが含まれている入力ファイル、またはディレクトリー。

-o output_recordname
mergepdf がマージ済みレコードを書き込む先の PDF 出力ディレクトリー名、またはファイル名。ディレクトリーを指定する場合、ディレクトリーはコマンドを実行する前に存在していなければなりません。

-n PDF レコードを正規化しません。デフォルトでは、レコードは、**-r** でユーザー指定の比率を適用する前に、それぞれのプロファイルごとに内部的に計算された比率に基づいて正規化されます。 **-n** を指定すると、PDF レコードは、ユーザー指定の比率 **-r** によってスケーリングされます。 **-r** を指定しない場合、PDF レコードはまったくスケーリングされません。

-v 冗長モードは、使用されている内部およびユーザー指定のスケーリング比率を表示します。

`resetpdf [pathname]`

上で説明した **cleanpdf** [*pathname*] と同じです。このコマンドは、以前のバージョンとの互換性のために提供されています。

showpdf　　プログラム実行で実行されたすべてのプロシージャーの呼び出しおよびブロック数を表示します。このコマンドを使用するには、**-qpdf1** および **-qshowpdf** コンパイラー・オプションを両方とも指定して、まずアプリケーションをコンパイルする必要があります。

例

簡単な例を次に示します。

```
# Set the PDFDIR variable.
  export PDFDIR=$HOME/project_dir
# Compile all files with -qpdf1.
  xlf95 -qpdf1 -O3 file1.f file2.f file3.f
# Run with one set of input data.
  a.out <sample.data
# Recompile all files with -qpdf2.
  xlf95 -qpdf2 -O3 file1.f file2.f file3.f
# The program should now run faster than without PDF if
# the sample data is typical.
```

より複雑な例を次に示します。

```
# Set the PDFDIR variable.
  export PDFDIR=$HOME/project_dir
# Compile most of the files with -qpdf1.
  xlf95 -qpdf1 -O3 -c file1.f file2.f file3.f
# This file is not so important to optimize.
  xlf95 -c file4.f
# Non-PDF object files such as file4.o can be linked in.
  xlf95 -qpdf1 -O3 file1.o file2.o file3.o file4.o
# Run several times with different input data.
  a.out <polar_orbit.data
  a.out <elliptical_orbit.data
  a.out <geosynchronous_orbit.data
# Do not need to recompile the source of non-PDF object files (file4.f).
  xlf95 -qpdf2 -O3 file1.f file2.f file3.f
# Link all the object files into the final application.
  xlf95 -qpdf2 -O3 file1.o file2.o file3.o file4.o
```

-qpdf2 を指定してソースの再コンパイルを迂回する例を次に示します。

```
# Set the PDFDIR variable.
  export PDFDIR=$HOME/project_dir
# Compile source with -qpdf1.
  xlf -O3 -qpdf1 -c file.f
# Link in object file.
  xlf -O3 -qpdf1 file.o
# Run with one set of input data.
  a.out <sample.data
# Link in object file from qpdf1 pass.
# (Bypass source recompilation with -qpdf2.)
  xlf -O3 -qpdf2 file.o
```

pdf1 オブジェクトおよび pdf2 オブジェクトの使用の例を次に挙げます。

```
# Set the PDFDIR variable.
  export PDFDIR=$HOME/project_dir
# Compile source with -qpdf1.
  xlf -c -qpdf1 -O3 file1.f file2.f
# Link in object files.
  xlf -qpdf1 -O3 file1.o file2.o
# Run with one set of input data.
  a.out <sample.data
# Link in the mix of pdf1 and pdf2 objects.
  xlf -qpdf2 -O3 file1.o file2.o
```

関連情報

- 20 ページの『XL Fortran 入力ファイル』
- 22 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『*プロファイル指定フィードバック (PDF) の利点*』

-qphsinfo オプション

構文

-qphsinfo | **-qnophsinfo**
PHSINFO | **NOPHSINFO**

-qphsinfo コンパイラー・オプションは、各コンパイラー・フェーズのタイミング情報を端末に表示します。

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* はコンパイラー時間と CPU がシステム呼び出しの処理に要する時間の合計を表します。

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

283 ページの『コンパイラー・フェーズ』

-qpic オプション

構文

```
-qpic[=small|large] | -qnopic (in 32-bit mode)
-qpic[=small|large] | -qnopic (in 64-bit mode)
```

-qpic コンパイラー・オプションは、共用ライブラリーで使用できる位置独立コード (PIC) を生成します。

引数

small | large
small サブオプションは、グローバル・オフセット・テーブルのサイズが最大で 64K であると想定するようコンパイラーに指示します。 **large** サブオプションを使用すると、グローバル・オフセット・テーブルのサイズを 64K より大きくすることができます。このサブオプションによって、より多くのアドレスをグローバル・オフセット・テーブルに保管できます。ただし、このサブオプションは、通常、**small** サブオプションが生成するコードよりも大きいコードを生成します。

64 ビット・モードでは、**-qpic=small** がデフォルトです。

-qnopic
コンパイラーは位置独立コードを生成しません。

32 ビット・モードでは、**-qnopic** がデフォルトです。

関連情報

リンカー・オプションの詳細については、**ld** コマンドのマニュアル・ページを参照してください。

-qport オプション

構文

-qport[=*suboptions*] | **-qnoport**
PORT[(*suboptions*)] | **NOPORT**

-qport コンパイラー・オプションは、他の Fortran 言語拡張機能を収容するためにいくつかのオプションを提供して、XL Fortran にプログラムを移植するときの柔軟性を高めます。特定の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。

引数

clogicals | **noclogicals**

このオプションを指定した場合、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**

このオプションを指定する場合、型のない定数 16 進数ストリングは、**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定数 16 進数ストリングは、影響を受けることはありません。

mod | **nomod** このオプションを指定することにより、**MOD** 組み込み関数の既存の制約をなくし、同じデータ型のパラメーターの 2 つの引数は、別の種類の型付きパラメーターにすることができます。その結果、その引数は同じ型になりますが、より大きい種類の型付き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このオプションを指定すると、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、このオプション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。タイプが非整数であるタイプなし実引数に関連付けられている仮引数は、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

詳細については、「*XL Fortran 言語解説書*」の **INT** および **MOD** 組み込み関数に関する節を参照してください。

-qposition オプション

構文

```
-qposition={appendold | appendunknown} ...
POSITION({APPENDOLD | APPENDUNKNOWN} ...)
```

POSITION= 指定子を持たない **OPEN** 文の後にデータが書き込まれ、対応する **STATUS=** 値 (**OLD** または **UNKNOWN**) が指定されると、ファイル・ポインターをファイルの終わりに置きます。

デフォルト設定は **OPEN** 文の I/O 指定子とコンパイラー呼び出しコマンドに応じて、**xlf**、**xlf_r**、および **f77/fort77** コマンドの場合は **-qposition=appendold**、**xlf90**、 **xlf90_r**、**xlf95**、**xlf95_r**、 **f90**、および **f95** コマンドの場合は、定義済みの Fortran 90 および Fortran 95 の動作になります。

規則

最初の入出力操作がファイル・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。

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 言語解説書*」の **OPEN** 文

-qprefetch オプション

構文

-qprefetch | -qnoprefetch

コードのパフォーマンスを向上させる機会がある場合に、プリフェッチ命令を自動的に挿入するようにコンパイラーに指示します。

関連情報

プリフェッチ・ディレクティブについて詳しくは、「*XL Fortran 言語解説書*」の『**PREFETCH ディレクティブ**』および「*The POWER4 Processor Introduction and Tuning Guide*」を参照してください。トリガー制約を使用してプリフェッチ・ディレクティブを選択的に制御するには、126 ページの『-qdirective オプション』を参照してください。

-qqcount オプション

構文

-qqcount | **-qnoqcount**
QCOUNT | **NOQCOUNT**

拡張精度 **Q** 編集記述子 (**Q***w*.*d*) だけではなく、 **Q** 文字カウント編集記述子 (**Q**) を受け入れます。**-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。

規則

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 言語解説書*」の『*Q (文字カウント) 編集*』を参照してください。

-qrealsize オプション

構文

```
-qrealsize=bytes
REALSIZE(bytes)
```

REAL、**DOUBLE PRECISION**、 **COMPLEX**、および **DOUBLE COMPLEX** 値のデフォルト・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。

規則

このオプションは定数、変数、派生型コンポーネント、および kind 型付きパラメーターが指定されていない関数 (組み込み関数を含む) のサイズ[2]に影響を及ぼします。 **REAL(4)** や **COMPLEX*16** など、kind 型付きパラメーターまたは長さ (length) で宣言されたオブジェクトは影響を受けません。

引数

バイト に使用できる値は次のとおりです。

- 4 (デフォルト)
- 8

結果

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。

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。たとえば、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。このオプションのデフォルト値 4 は、多くの 32 ビット・コンピューター用に書かれたプログラムに適しています。

---

2. Fortran 90 および 95 の用語では、これらの値は *kind 型付き*パラメーター と呼ばれています。

-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。

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)
         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

158 ページの『-qintsize オプション』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。 110 ページの『-qautodbl オプション』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。

「*XL Fortran 言語解説書*」の『*型付きパラメーターおよび指定子*』では、kind 型付きパラメーターについて説明しています。

-qrecur オプション

構文

-qrecur | **-qnorecur**
RECUR | **NORECUR**

このオプションの使用はお勧めできません。外部サブプログラムを再帰的に呼び出すことができるかどうかを指定します。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。

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
```

制限

xlf、**xlf_r**、**f77**、または **fort77** コマンドを使用して再帰呼び出しを含む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・クラスを自動的に作成します。

-qreport オプション

構文

```
-qreport[={smplist | hotlist}...]
-qnoreport
REPORT[({SMPLIST | HOTLIST}...)] NOREPORT
```

プログラムを並列化する方法とループを最適化する方法を示す変換報告書を作成するかどうかを決定します。

smplist サブオプションを使用すると、低レベルの変換が調べられるため、 SMP プログラムのパフォーマンスをデバッグしたり調整したりできます。プログラムのデータ処理方法や、ループの自動並列化方法を知ることもできます。リスト内のコメントは、変換後のプログラムが元のソース・コードにどのように対応しているかを示したり、特定のループが並列化されなかった理由などを示したりします。

hotlist サブオプションを使用すると、ループの変換過程を示す報告書を生成することができます。

引数

smplist
: 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。 この報告書が作成されるのは、**-qsmp** オプションが有効な場合のみです。

hotlist
: 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。サブオプションを指定せずに **-qreport** を指定した場合は、このサブオプションがデフォルトになっています。

  また、**-qsmp** オプションが有効なときに **-qreport=hotlist** オプションが指定された場合は、SMP 実行時への呼び出し、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。

背景情報

変換リストはコンパイラー・リスト・ファイルの一部です。

制限

ループ変換および自動並列処理は、**-O5** (または **-qipa=level=2**) 最適化レベルを使用して、リンク・ステップで行われます。 **-qreport** オプションは、リンク・ステップで、リスト・ファイルに報告書を生成します。

ループ変換リストを生成する場合は、 **-qsmp** オプションか **-qhot** オプションを必ず指定します。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場合は、 **-qsmp** オプションを必ず指定します。

リストに示されるコードはコンパイル可能であるというわけではありません。プログラムにこのコードを組み込んだり、名前が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

並列の調整に使用できるリスト・ファイルを作成するには、次のようにします。

```
xlf_r -qsmp -O3 -qhot -qreport=smplist needs_tuning.f
```

並列調整とループ・パフォーマンス調整の両方に使用できるリスト・ファイルを作成するには、次のようにします。

```
xlf_r -qsmp -O3 -qhot -qreport=smplist:hotlist needs_tuning.f
```

ループのパフォーマンスの調整にだけ使用できるリスト・ファイルを作成するには、次のようにします。

```
xlf95_r -O3 -qhot -qreport=hotlist needs_tuning.f
```

関連情報

191 ページの『-qpdf オプション』を参照してください。

-qsaa オプション

構文

-qsaa | **-qnosaa**
SAA | **NOSAA**

SAA FORTRAN 言語定義に従っているかどうかをチェックします。非準拠のソース・コード、およびそのような非準拠を許可するオプションを識別します。

規則

これらの警告には、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。

制限

-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。

関連情報

168 ページの『-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。

-qsave オプション

構文

```
-qsave[={all|defaultinit}] | -qnosave
SAVE[({all|defaultinit})] NOSAVE
```

これには、ローカル変数のデフォルト・ストレージ・クラスを指定します。

-qsave=all を指定する場合は、デフォルト・ストレージ・クラスは **STATIC** です。 **-qnosave** を指定する場合は、デフォルト・ストレージ・クラスは **AUTOMATIC** です。 **-qsave=defaultinit** を指定する場合は、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。 **-qsave** オプションのデフォルト・サブオプションは、**all** です。 2 つのサブオプションは相互に排他的です。

このオプションのデフォルトは、使用される起動によって異なります。たとえば、**-qsave** を指定して、FORTRAN 77 プログラムの動作を再現しなければならない場合があります。 **xlf**、**xlf_r**、**f77**、および **fort77** コマンドは、前の動作を保持するために、 **/etc/opt/ibmcmp/xlf/10.1/xlf.cfg** にデフォルト・オプションとしてリストされている **-qsave** を持っています。

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

207 ページの『-qrecur オプション』でコンパイルされたマルチスレッド・アプリケーションおよびサブプログラムには、通常、**-qnosave** オプションが必要です。

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 言語解説書*」の『変数のストレージ・クラス』を参照してください。

-qsaveopt オプション

構文

-qsaveopt | **-qnosaveopt**

ソース・ファイル、およびその他の情報のコンパイルに使用するコマンド行オプションを該当するオブジェクト・ファイルに保管します。コンパイルでは、このオプションを有効にするためのオブジェクト・ファイルを生成する必要があります。各オブジェクトに複数のコンパイル単位が含まれている場合であっても、コマンド行オプションの 1 コピーだけが保管されます。

使用されているオプションをリストするには、オブジェクト・ファイルに **strings -a** コマンドを発行します。以下がリストされます。

```
opt source_type invocation_used compilation_options
```

たとえば、オブジェクト・ファイルが **t.o** である場合、**strings -a t.o** コマンドは、以下のような情報を生成します。

```
@(#) opt f /opt/ibmcmp/xlf/10.1/bin/xlf90 -qlist -qsaveopt t.f
```

ここで、**f** は Fortran として使用されたソースを識別し、**/opt/ibmcmp/xlf/10.1/bin/xlf90** は使用された呼び出しコマンドを示し、**-qlist -qsaveopt** はコンパイル・オプションを示します。

-qsclk オプション

構文

-qsclk[={**centi** | micro}]

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レゾリューションを指定します。デフォルトは、センチ秒レゾリューション (**-qsclk=centi**) です。マイクロ秒レゾリューションを使用するには、**-qsclk=micro** を指定します。

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関する詳細については、「*XL Fortran 言語解説書*」の『**SYSTEM_CLOCK**』を参照してください。

-qshowpdf オプション

構文

-qshowpdf | **-qnoshowpdf**

-qpdf1 および **-O** の最小最適化レベルとともに使用して、追加の呼び出しとブロック数プロファイル情報を実行可能ファイルに追加します。

-qpdf1 とともに指定すると、コンパイラーは、追加のプロファイル情報をコンパイル済みアプリケーションに挿入して、アプリケーション内のすべてのプロシージャーに対する呼び出しおよびブロック数を収集します。コンパイル済みアプリケーションを実行すると、呼び出しおよびブロック数を **._pdf** ファイルに記録します。

トレーニング・データを使用してアプリケーションを実行した後で、**showpdf** ユーティリティーを使用して **._pdf** ファイルの内容を検索することができます。このユーティリティーについては、191 ページの『-qpdf オプション』で説明します。

-qsigtrap オプション

構文

-qsigtrap[=*trap_handler*]

メインプログラムが入っているファイルをコンパイルするときに、このオプションは、指定されたトラップ・ハンドラーをセットアップして **SIGTRAP** および **SIGFPE** 例外をキャッチします。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。

引数

xl__trce トラップ・ハンドラーを使用可能にするには、ハンドラー名なしで **-qsigtrap** を指定してください。 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

- 43 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およびトラッピング*』では、浮動小数点計算の結果生じる例外を扱う多くの方法を説明しています。
- 「*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 で使用できる例外ハンドラーのリストが掲載されています。

-qsmallstack オプション

構文

```
-qsmallstack[=[no]dynlenonheap]
 | -qnosmallstack
```

可能な限りコンパイラーがスタック使用を最小化するように指定します。

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。

デフォルトで、**-qnosmallstack** は、すべてのサブオプションがオフであることを暗黙指定されます。

-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。

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。

規則

- 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** オプションおよび **-qstacktemp** オプションの両方が使用されるとき、一時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

- 225 ページの『-qstacktemp オプション』

-qsmp オプション

構文

-qsmp[=*suboptions*]
-qnosmp

コードを SMP システムに対応するようにして生成すべきかどうかを示します。デフォルトでは、単一処理装置マシンを想定してコードを生成します。このオプションを指定した場合、コンパイラーはトリガー定数 **SMP$**、**$OMP**、および **IBMP** にあるすべてのディレクティブを認識します (ただし **omp** サブオプションを指定した場合は認識されません)。

すべてのスレッド・セーフ・コンポーネントで自動的にリンクを行うものは、**xlf_r**、 **xlf90_r** 、および**xlf95_r** 呼び出しコマンドのみです。 **xlf**、 **xlf90**、**xlf95**、**f77**、および **fort77** 呼び出しコマンドと一緒に **-qsmp** オプションを使用することもできますが、適切なコンポーネントにリンクさせるのはユーザー側の責任で行います。 **-qsmp** オプションを使ってプログラム内のソース・ファイルをコンパイルする場合、 **ld** コマンドを使ってリンクしない限り、リンク時に **-qsmp** も一緒に指定する必要があります。

引数

auto | **noauto**　このサブオプションは自動並列化を制御します。デフォルトでは、明示的にコーディングされた **DO** ループだけでなく、配列言語として使用するためにコンパイラーが生成したループも、コンパイラーは並列化しようとします。サブオプション **noauto** が指定されている場合、自動並列化はオフになり、所定のディレクティブによってマークされた構文だけが並列化の対象になります。コンパイラーがサブオプション **omp** を検出し、 **-qsmp** または **-qsmp=auto** サブオプションがコマンド行で明示的に指定されていない場合、**noauto** サブオプションが暗黙指定されます。また、**-qsmp=noopt** は **-qsmp=noauto** を暗黙指定することに注意してください。自動並列化は **-qsmp=noopt** のもとでは行われません。ユーザー指定の並列化のみが行われます。

nested_par | **nonested_par**

nested_par サブオプションを指定する場合、コンパイラーは所定のネスト並列構文 (**PARALLEL DO**、**PARALLEL SECTIONS**) は並列化します。この場合の並列化の対象には、有効範囲単位内にあるネストされたループ構成体だけでなく、他の並列構成体から (直接であれ間接であれ) 参照されるサブプログラム内の並列構成体も含まれます。デフォルトでは、コンパイラーはネストされた並列構成体を続けます。このオプションは、自動的に並列化されるループには効果がないことに注意してください。この場合、多くても (有効範囲単位内にある) ループ・ネストに含まれる 1 ループしか並列化されません。

nested_par サブオプションを設定しても、これは OpenMP API に従っているわけではないことに注意してください。このサブオプションを指定する場合、ランタイム・ライブラリーは、 **PARALLEL**

構文を囲むのに使用したのと同じスレッドを、ネストされた **PARALLEL DO** および **PARALLEL SECTIONS** 構文にも使用します。

omp | **noomp**
-qsmp=omp を指定する場合、コンパイラーは OpenMP API での準拠事項を実施します。このオプションを指定すると、以下のような効果があります。

- 自動並列化はオフになります。
- 以前に認識されているすべてのディレクティブ・トリガーが無視されます。
- **-qsmp=omp** を指定すると、 **-qcclines** コンパイラー・オプションはオンになります。
- **-qnocclines** および **-qsmp=omp** を指定すると、 **-qcclines** コンパイラー・オプションはオンになりません。
- 認識されているディレクティブ・トリガーは **$OMP** だけになります。しかし、その後の **-qdirective** オプションで別のトリガーを指定することができます。
- OpenMP API に準拠しない言語構成要素がコードに含まれている場合、コンパイラーは警告メッセージを発行します。

C プリプロセッサーの起動時にこのオプションを指定すると、**_OPENMP** C プリプロセッサー・マクロも自動的に値 *200505* と一緒に定義されます。これは、条件付きコンパイルをサポートするのに役立ちます。このマクロは、C プリプロセッサーの呼び出し時にのみ定義されます。

詳細については、「*XL Fortran 言語解説書*」の*言語エレメント*の節の『*条件付きコンパイル*』を参照してください。

opt | **noopt**
-qsmp=noopt サブオプションを指定すると、コンパイラーは、コードの並列化に必要な最小の最適化量を実行します。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**
このサブオプションは、 **CRITICAL** 構文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指定します。 **rec_locks** サブオプションを指定すると、スレッドは同じ名前を持つ別の **CRITICAL** 構文の動的範囲内から、 **CRITICAL** 構文を実行することができます。 **norec_locks** を指定すると、こうした状況ではデッドロックが生じます。

デフォルトは、**norec_locks** または正規のロックです。

schedule=*option*
schedule サブオプションは、以下に示されているサブオプションのいずれかをとることができます。

affinity[=*n*]

ループの反復は、最初に、**CEILING**(number_of_iterations / number_of_threads) 反復を含む、*number_of_threads* 区画に分割されます。各区画は最初はスレッドに割り当てられており、その後それぞれが *n* 反復を含むチャンクに再分割されていきます。 *n* が指定されなかった場合は、**CEILING**(number_of_iterations_left_in_partition / 2) ループ反復でチャンクが構成されます。

スレッドが解放されると、スレッドが最初に割り当てられた区画から次のチャンクを取ります。その区画の中にチャンクが 1 つも無くなったら、最初に別のスレッドに割り当てられた区画から使用可能な次のチャンクを探します。

スリープ状態のスレッドに最初から割り当てられていた区画の処理は、活動状態にある別のスレッドにより完了されます。

dynamic[=*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。 *n* が指定されなかった場合は、**CEILING** (number_of_iterations / number_of_threads) 反復でチャンクが構成されます。

活動状態のスレッドがチャンクに割り当てられる仕方は、「先着順実行」の原則に基づいています。残りの処理のチャンクは、すべての処理の割り当てが終了するまで、活動状態のスレッドに割り当てられていきます。

スリープ状態のスレッドに割り当てられた処理は、そのスレッドが使用可能にならない限り、活動状態の別のスレッドに引き継がれます。

guided[=*n*]

ループの反復は、*n* ループ反復の最小チャンク・サイズに到達するまで、より小さなチャンクへと漸進的に分割されていきます。 *n* が指定されなかった場合、 *n* のデフォルト値である 1 反復が適用されます。

最初のチャンクには **CEILING**(number_of_iterations / number_of_threads) 回の反復が含まれます。次のチャンクには **CEILING**(number_of_iterations_left / number_of_threads) 回の反復が含まれます。活動状態のスレッドがチャンクに割り当てられる仕方は、「先着順実行」の原則に基づいています。

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。各スレッドは、「ラウンドロビン」方式でチャンクに割り当てられます。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、必然的に、スケジューリング・タイプが*巡回スケジューリング* として参照されます。

n を指定しない場合、チャンクには **CEILING** (number_of_iterations / number_of_threads) 反復が入ります。各スレッドは、これらのチャンクのいずれかに割り当てられます。これを*ブロック・スケジューリング* と言います。

スリープ状態のスレッドに処理が割り当てられている場合、その処理を完了できるようにするため、そのスレッドは活動状態にさせられます。

チャンク入れのアルゴリズムと **SCHEDULE** の詳細については、「*XL Fortran 言語解説書*」の『ディレクティブ』という節を参照してください。

threshold=*n* 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、ループに示されているレベル「work」に基づいて、ループの並列化をどこまで許可するかの下限を示します。現在、「work」の計算の大部分は、ループ内の反復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。このサブオプションが指定されなかった場合、プログラムはデフォルト値 *n*=100 を使用します。

規則

- **-qsmp** を複数回指定した場合、後続のサブオプション設定によってオーバーライドされない限り、すべてのサブオプションの直前の設定が保存されます。コンパイラーは、以前に指定したサブオプションをオーバーライドすることはありません。サブオプションがない **-qsmp** のバージョンでも、同じことが当てはまり、デフォルト・オプションが保存されます。
- **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。
- **noomp** サブオプションを指定すると、**auto** が暗黙指定されます。
- **omp** および **noomp** サブオプションは、明示的に設定した場合に限り、コンパイラー・リストに含められます。
- サブオプションのない **-qsmp** を指定すると、**-qsmp=opt** がデフォルトの設定になります。**-qsmp=noopt** サブオプションを設定した後 **-qsmp** を使用すると、**-qsmp=noopt** 設定は常に無視されます。
- コマンド行で、**-qsmp** オプションをサブオプションなしで指定し、その後に **-qsmp=noopt** サブオプションを指定すると、**-qsmp=opt** および **-qsmp=auto** オプションが使用できます。
- **-qsmp=noopt** サブオプションを指定すると、**-qsmp=noauto** と見なされます。また **-qnoopt** も想定されます。このオプションは、コマンド行上のどこにあっても **-O2**、**-O3**、 **-qhot** のような パフォーマンス・オプションをオーバーライドします (**-qsmp** の前に **-qsmp=noopt** が現れる場合を除く)。
- **-qsmp** と指定すると、事実上 **-qhot** であると見なされます。

- **-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。

制限

-qsmp=noopt サブオプションは、プログラムのパフォーマンスに影響することがあります。

-qsmp を指定してある状態では、特定のサブオプションの前後に **omp** サブオプションを指定することはできません。 **omp** を使用してそれらのサブオプションを指定しようとする場合、コンパイラーは警告メッセージを発行します。

auto
このサブオプションは自動並列化を制御しますが、 **omp** は自動並列化をオフにしてしまいます。

nested_par
nested_par サブオプションを設定しても、これは OpenMP API に従っているわけではないことに注意してください。このサブオプションを指定する場合、ランタイム・ライブラリーは、 **PARALLEL** 構文を囲むのに使用したのと同じスレッドを、ネストされた **PARALLEL DO** および **PARALLEL SECTIONS** 構文にも使用します。

rec_locks
このサブオプションは、OpenMP API との整合性のない **CRITICAL** 構成体の動作を指定します。

schedule=affinity=*n*
類縁性スケジューリング・タイプは、OpenMP API 標準にはありません。

例

-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)。次の例では、**-qsmp=noopt** の後にあるすべての最適化オプションが、通常の有効範囲と優先順位の規則に従って処理されることを示します。

例 1

```
     xlf90 -qsmp=noopt -O3...
is equivalent to
     xlf90 -qsmp=noopt...
```

例 2

```
     xlf90 -qsmp=noopt -O3 -qsmp...
is equivalent to
     xlf90 -qsmp -O3...
```

例 3

```
     xlf90 -qsmp=noopt -O3 -qhot -qsmp -O2...
is equivalent to
     xlf90 -qsmp -qhot -O2...
```

次を指定すると、コンパイラーは **$OMP** ディレクティブ・トリガーと **SMP$** ディレクティブ・トリガーの両方を認識し、いずれかのトリガーで指定したディレクティブが OpenMP では許可されていない場合に警告を発行します。

```
-qsmp=omp -qdirective=SMP$
```

次を指定すると、**noauto** サブオプションが使用されます。コンパイラーは警告メッセージを出し、**auto** サブオプションを無視します。

```
-qsmp=omp:auto
```

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** を指定する必要があります。

```
   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

xlf、**xlf_r**、**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。

-qsource オプション

構文

-qsource | **-qnosource**
SOURCE | **NOSOURCE**

リストのソース・セクションを作成するかどうかを指定します。

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 16.12: 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 ===
        16 |   call mult(i,j)
              ............a...
a - 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

58 ページの『リストとメッセージを制御するオプション』および 278 ページの『ソース・セクション』を参照してください。

-qspillsize オプション

構文

```
-qspillsize=bytes
SPILLSIZE(bytes)
```

-qspillsize は **-NS** の長い形式です。 87 ページの『-NS オプション』を参照してください。

-qstacktemp オプション

構文

```
-qstacktemp={0 | -1 | positive signed integer value}
STACKTEMP={0 | -1 | positive signed integer value}
```

実行時に適用できる XL Fortran コンパイラーのテンポラリー (temporary) をどこに割り振るかを判別します。

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一時変数です。もっとも一般的には、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。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。たとえば、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。

注: **-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。

引数

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。

正符号の整数値
スタックにこの*値*より小さい適当なテンポラリー (temporary) を、ヒープにこの*値* より大か等しいものを割り当てます。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。

-qstrict オプション

構文

```
-qstrict | -qnostrict
STRICT | NOSTRICT
```

デフォルトにより **-O3**、**-O4**、および **-O5** オプションで行われた最適化と、オプショナルで **-O2** オプションで行われた最適化によって、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。

デフォルト

-O3、**-O4**、および **-O5** の場合、デフォルトは **-qnostrict** です。 **-O2** の場合、デフォルトは **-qstrict** です。このオプションは、**-qnoopt** の場合は無視されます。 **-qnostrict** では、最適化はコードを再調整して、結果または例外が、最適化されていないプログラムのものとは異なるようにします。

このオプションは、最適化されているプログラムでのプログラムの実行における変更が、最適化されていないプログラムの場合とは異なった結果を発生させる状況を意図したオプションです。 IEEE 浮動小数点算術計算用のほとんど使用されない規則と関連があるので、このような状況はめったに発生しません。

規則

-qnostrict が有効である場合は、**-qstrict** も指定されていない限り、以下の最適化がオンになります。

- 例外を発生させる可能性のあるコードを再配置することができます。該当する例外は、実行の別の時点で発生することもありますし、まったく発生しないこともあります。 (コンパイラーは依然として、そういう状況を最小限にとどめようとします。)
- 浮動小数点演算は、値ゼロの符号を保存することができません。 (この符号が保存されるようにするには、**-qfloat=rrm**、**-qfloat=nomaf**、または **-qfloat=strictnmaf** も指定する必要があります。)
- 浮動小数点式は、再関連付けすることができます。たとえば、結果が同一とはなりませんが、 **(2.0*3.1)*4.2** は **2.0*(3.1*4.2)** になります (その方が速い場合)。
- **-qfloat** オプションの **fltint** サブオプションおよび **rsqrt** サブオプションはオンになります。 **-qstrict** オプションか、 **-qfloat** の **nofltint** サブオプションおよび **norsqrt** サブオプションも使用すると、再びオフにすることができます。レベルの低い最適化が指定されている場合や最適化が指定されていない場合は、これらのサブオプションはデフォルト時にはオフになります。

関連情報

88 ページの『-O オプション』、 149 ページの『-qhot オプション』、および 141 ページの『-qfloat オプション』を参照してください。

-qstrictieeemod オプション

構文

-qstrictieeemod | -qnostrictieeemod
STRICTIEEEMOD | NOSTRICTIEEEMOD

ieee_arithmetic および **ieee_exceptions** 組み込みモジュール用の Fortran 2003 IEEE 演算規則をコンパイラーに順守させるかどうかを指定します。 **-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うプロシージャーから、それらを使わ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 オプション

構文

-qSTRICT_INDUCtion | **-qNOSTRICT_INDUCtion**

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てしまわないようにします。そのような最適化を実行した場合、帰納変数が関係した整数オーバーフローの動作が発生したときにアンセーフ になる可能性があります (プログラムのセマンティクスが変更される可能性があります)。

-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、コンパイラーはプログラムのセマンティクスを変えることのある最適化を実行する可能性があります。

-qsuffix オプション

構文

-qsuffix=*option=suffix*

xlf.cfg ファイルの代わりに、コマンド行でソース・ファイルのサフィックスを指定します。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るだけでなく、**xlf.cfg** ファイルの修正に関連した問題のリスクを削減できます。 どのファイル・タイプの場合にも、 1 度に 1 つの設定だけがサポートされます。

引数

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。
- 新しいサフィックスに関する設定は、 **xlf.cfg** ファイルの対応するデフォルト設定をオーバーライドします。
- **-qsuffix** と **-F** の両方が指定された場合、 **-qsuffix** は最後に処理されるため、その設定が **xlf.cfg** ファイルの設定をオーバーライドします。

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 が呼び出され、サフィックスが .F2K のファイルを処理します。

-qsuppress オプション

構文

```
-qsuppress[=nnnn-mmm[:nnnn-mmm ...] | cmpmsg]
-qnosuppress
```

引数

nnnn-mmm[:nnnn-mmm ...]
: 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。*nnnn-mmm* はメッセージ番号です。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。

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。

背景情報

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。

- 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、**-qextname** コンパイラー・オプションで指定された場合は無視されます。
- **-qipa** コンパイラー・オプションも一緒に指定する場合は、コマンド行で **-qextname** コンパイラー・オプションの前に **-qipa** を入力する必要があります。そのようにしないと、IPA メッセージの表示は抑止できません。

制限

- 値 *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るからです。
- 値 *mmm* は、3 桁の任意の整数です (必要であれば先行ゼロを付けます)。

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

その他のタイプのメッセージ表示抑止については、 140 ページの『-qflag オプション』を参照してください。

-qswapomp オプション

構文

-qswapomp | -qnoswapomp
SWAPOMP | NOSWAPOMP

コンパイラーが、XL Fortran プログラムにある OpenMP ルーチンを認識して置換するように指定します。

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。

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。

制限

-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 言語解説書*」にある『*OpenMP 実行環境ルーチンおよびロック・ルーチン*』の節を参照してください。

-qtbtable オプション

構文

```
-qtbtable={none | small | full}
```

注: 64 ビット環境にのみ適用されます。

オブジェクト・ファイル内のトレースバック情報のデバッグ量を制限し、プログラムのサイズを小さくします。

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。この場合、**-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 (**-qtbtable=full**)、 **-O** 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**)。

引数

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。

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。

背景情報

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。

関連情報

- 82 ページの『-g オプション』
- 120 ページの『-qcompact オプション』
- 88 ページの『-O オプション』
- 「*XL Fortran 最適化およびプログラミング・ガイド*」の『*最適化コードのデバッグ*』

-qthreaded オプション

構文

```
-qthreaded
```

コンパイラーがこのオプションを使用することにより、スレッド・セーフ・コードを生成する必要があるのはいつかを判断します。

-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には、これらの両方のオプションを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。 サブプログラム用に代替エントリー・ポイントを持つため **ENTRY** 文、およびコンパイル用に **xlf_r** コマンドを使用する場合、スレッド・セーフであるために **-qxlf77=nopersistent** オプションも指定してください。適切なロック機構もインプリメントする必要があります。

デフォルト

-qthreaded は、**xlf90_r**、 **xlf95_r**、 および **xlf_r** コマンド用のデフォルトです。

-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。

-qtune オプション

構文

-qtune=*implementation*

ハードウェア・アーキテクチャーの特定のインプリメンテーションに対する命令の選択、スケジューリング、その他のインプリメンテーションに依存するパフォーマンス拡張機能を調整します。コンパイラーは、ターゲット・アーキテクチャーと互換性のある **-qtune** 設定を使用します。これは、**-qarch**、**-q32**、および **-q64** オプションによって制御されます。

プログラムを複数のアーキテクチャーで稼動させるけれど、特定のアーキテクチャーで調整したい場合は、 **-qarch** および **-qtune** オプションを組み合わせて使用することができます。これらのオプションは、基本的には 浮動小数点中心のプログラムに有効です。

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・インストラクションを配置 (スケジューリング) することによって、**-qtune** オプションはパフォーマンスを改善することができます。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。

引数

選択項目は次のとおりです。

auto どのプロセッサー・タイプに属するコンパイル・マシンであるか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。

rs64b RS64II プロセッサー用に、最適化が調整されます。

rs64c RS64III プロセッサー用に、最適化が調整されます。

pwr3 POWER3 プロセッサー用に、最適化が調整されます。

pwr4 POWER4 プロセッサー用に、最適化が調整されます。

pwr5 POWER5 プロセッサー用に、最適化が調整されます。

ppc970 PowerPC 970 プロセッサー用に最適化が調整されます。

-qtune を指定しないと、設定は **-qarch** オプションによって決定されます。

-qarch 設定	**許可されている -qtune 設定**	**デフォルトの -qtune 設定**
ppc	**-qarch=ppc64** エントリーで、受け入れ可能な **-qtune** 設定のリストを参照してください。	pwr4

-qarch 設定	**許可されている -qtune 設定**	**デフォルトの -qtune 設定**
ppcgr	**-qarch=ppc64gr** エントリーで、受け入れ可能な **-qtune** 設定のリストを参照してください。	pwr4
ppc64	rs64b、 rs64c、 pwr3、 pwr4、 pwr5、 ppc970、 auto	pwr4
ppc64gr	rs64b、 rs64c、 pwr3、 pwr4、 pwr5、 ppc970、 auto	pwr4
ppc64grsq	rs64b、 rs64c、 pwr3、 pwr4、 pwr5、 ppc970、 auto	pwr4
rs64b	rs64b、 auto	rs64b
rs64c	rs64c、 auto	rs64c
pwr3	pwr3、 pwr4、 pwr5、 ppc970 、 auto	pwr3
pwr4	pwr4、 pwr5、 ppc970 、 auto	pwr4
pwr5	pwr5、 auto	pwr5
pwr5x	pwr5、 auto	pwr5
ppc970	ppc970、 auto	ppc970

これで、**-qtune** サブオプションと互換性のあるマシン上でコンパイルしている限り、**-qarch=auto** とともに **-qtune** サブオプションを指定できるようになりました。たとえば、**-qarch=auto** と **-qtune=pwr5** を指定する場合、POWER3、POWER4、または POWER5 マシン上でコンパイルする必要があります。

関連情報

- 27 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』
- 103 ページの『-qarch オプション』
- 114 ページの『-qcache オプション』

-qundef オプション

構文

-qundef | **-qnoundef**
UNDEF | **NOUNDEF**

-qundef は、 257 ページの『-u オプション』の長い形式です。

-qunroll オプション

構文

-qunroll[=**auto** | yes] | -qnounroll

DO ループのアンロールをプログラム内で許可するかどうかを指定します。アンロールは、外部および内部 **DO** ループで許可されます。

引数

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。 **-qunroll** をコマンド行で指定していない場合は、これがデフォルトです。

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。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。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。

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。

規則

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 言語解説書*」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『*ハイ・オーダー変換 (HOT) の利点*』を参照してください。

-qunwind オプション

構文

-qunwind |-qnounwind
UNWIND | NOUNWIND

プロシージャー呼び出し中に、コンパイラーが、揮発性レジスターの保存と復元のデフォルト動作を保持するように指定します。 **-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・オプションと結合して使用するときは、例外処理操作に関するデバッグ情報は、プログラム・スタックをアンワインドするとき不正確になる可能性があります。

-qversion オプション

構文

-qversion | **-qnoversion**

起動されているコンパイラーのバージョンとリリースを表示します。

コンパイラー・コマンドで、このオプションを単独で指定します。たとえば、次のようになります。

```
xlf90 -qversion
```

-qwarn64 オプション

構文

-qwarn64 | **-qnowarn64**

32 ビット環境から 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。 **-qwarn64** オプションでは、通知メッセージを使用して、32 ビットから 64 ビットへのマイグレーションで問題の原因となり得る文が識別されます。

規則

- デフォルト設定値は **-qnowarn64** です。
- **-qwarn64** オプションは、 32 ビットおよび 64 ビット・モードのどちらでも使用できます。
- 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。

関連情報

- 94 ページの『-q32 オプション』
- 95 ページの『-q64 オプション』
- 265 ページの『第 6 章 64 ビット環境での XL Fortran の使用』

-qxflag=dvz オプション

構文

```
-qxflag=dvz
```

-qxflag=dvz を指定すると、コンパイラーは、浮動小数点ゼロ除算演算を検出するコードを生成します。

このオプションは、最適化レベル **-O** 以上を指定した場合にのみ有効です。

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
```

上記の意味は次のとおりです。

x は被除数です。

y 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*』
- 144 ページの『-qflttrap オプション』
- 267 ページの『XL Fortran エラー・メッセージに関する情報』

-qxflag=oldtab オプション

構文

```
-qxflag=oldtab
XFLAG(OLDTAB)
```

桁 1 から 5 のタブを単一文字として解釈します (固定ソース形式のプログラムの場合)。

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。

規則

-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 オプション

構文

```
-qxlf77=settings
XLF77(settings)
```

変更された言語セマンティクスと I/O データ形式について、 FORTRAN 77 との互換性を提供します。これらの変更のほとんどは、Fortran 90 標準で必要です。

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 95、Fortran 90、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。したがって、デフォルトのサブオプションは、**blankpad**、**nogedit77**、**nointarg**、 **nointxor**、**leadzero**、**nooldboz**、**nopersistent**、**nosofteof** です。ただし、これらのデフォルトは、新しいプログラムのコンパイルに使用しなければならない **xlf95**、**xlf95_r**、**xlf90**、**xlf90_r**、**f90**、および **f95** コマンドのみによって使用されます。

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。このオプションについては、Fortran 90 または Fortran 95 で既存のソースまたはデータ・ファイルを使用し、**xlf90**、 **xlf90_r**、 **xlf95**、 **xlf95_r**、 **f90**、または **f95** コマンドを使用する場合で、動作またはデータ形式が変更されたために一部の互換性が失われる場合にのみ必要です。最終的には、古い動作への依存性を除去するために、データ・ファイルを再作成するか、またはソース・ファイルを変更できなければなりません。

引数

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 (たと

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 (たとえば、**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**
との互換性を得るために、**ENTRY** 文を持つサブプログラムの引数のアドレスを静的ストレージに保管します。これはインプリメンテーション選択項目であり、パフォーマンスの増加のため変更されています。

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。

-qxlf90 オプション

構文

```
-qxlf90={settings}
XLF90({settings})
```

Fortran 言語上の理由で、 Fortran 90 標準との 互換性を提供します。

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。 **f95**、**xlf95** または **xlf95_r** コマンドの場合、デフォルトのサブオプションは **signedzero** と **autodealloc** 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** と **noautodealloc** です。

引数

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、ゼロ以外の数字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** 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。

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確実な場合、このサブオプションをオフにして、性能低下の可能性を避けることができます。

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。たとえば、次のようになります。

呼び出しコマンド / xlf90 サブオプション	**出力**
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 言語解説書*」の『*組み込みプロシージャー*』の節および『*配列の概念*』の節にある『**SIGN**』の情報を参照してください。

-qxlines オプション

構文

-qxlines | **-qnoxlines**
XLINES | **NOXLINES**

桁 1 に X を持つ固定ソース形式行がコンパイルされるか、コメントとして扱われるかを指定します。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。 **-D** 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。

デフォルト

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。 **-qxlines** オプションは、**-D** から独立しているので、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。

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

79 ページの『-D オプション』と、「*XL Fortran 言語解説書*」の言語エレメントの節にある『*条件付きコンパイル*』を参照してください。

-qxref オプション

構文

-qxref[=full] | **-qnoxref**
XREF[(FULL)] | **NOXREF**

属性の相互参照コンポーネントおよびリストの相互参照セクションを作成するかどうかを決定します。

-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

58 ページの『リストとメッセージを制御するオプション』および 281 ページの『属性および相互参照セクション』を参照してください。

-qzerosize オプション

構文

-qzerosize | -qnozerosize
ZEROSIZE | NOZEROSIZE

サイズがゼロの文字ストリングおよび配列の検査を行わせないことによって、FORTRAN 77 プログラムと、一部の Fortran 90 および Fortran 95 プログラムのパフォーマンスを向上させます。

このようなオブジェクトを処理する可能性がある Fortran 90 および Fortran 95 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。

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。 **xlf90**、 **xlf90_r**、 **xlf95**、 **xlf95_r**、 **f90**、および **f95** コマンドの場合は **-qzerosize** で、**xlf**、**xlf_r**、および **f77/fort77** コマンドの場合は **-qnozerosize** です。(FORTRAN 77 との互換性のため)

規則

-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。

-S オプション

構文

-S

個々の Fortran ソース・ファイルに対して同等のアセンブラー・ソースを示す 1 つ以上の **.s** ファイルを作成します。

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。

制限

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

91 ページの『-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。

アセンブラー言語形式については、「*Assembler Language Reference*」を参照してください。

-t オプション

構文

-tcomponents

-B オプションで指定されたプレフィックスを、指定されたコンポーネントに適用します。 *components* には、分離文字なしで 1 つ以上の **a**、 **F**、 **c**、 **h**、 **I**、 **b**、 **z**、 **l**、または **d** を指定でき、それぞれアセンブラー、C プリプロセッサー、コンパイラー、配列言語最適化プログラム、プロシージャー間分析 (IPA) ツール/ループ、コード生成プログラム、バインド・プログラム、リンカー、および **-S** 逆アセンブラーに対応します。

規則

-t が指定されないと、プレフィックス **-B** がすべてのコンポーネントに適用されます。

Component	-t Mnemonic	Standard Program Name
assembler	a	as
C preprocessor	F	cpp
compiler front end	c	xlfentry
array language optimizer	h	xlfhot
IPA/loop optimizer	I	ipa
code generator	b	xlfcode
binder	z	bolt
linker	l	ld
disassembler	d	dis

関連情報

76 ページの『-B オプション』 (この項には例も記載されています)。

-U オプション

構文

```
-U
MIXED | NOMIXED
```

コンパイラーが、名前内の文字の大文字と小文字を区別するようにします。

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。

規則

-U 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。たとえば、`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、モジュールの名前に影響を与え、したがって、 **.mod** ファイルの名前にも影響を与えます。

デフォルト

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。たとえば、`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。

制限

-U 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。

関連情報

これは、**-qmixed** の短い形式です。 180 ページの『-qmixed オプション』を参照してください。

-u オプション

構文

```
-u
UNDEF | NOUNDEF
```

変数名の暗黙の型指定が許可されないことを指定します。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。

デフォルト

デフォルト時には、暗黙のタイプ指定は許可されます。

関連情報

「*XL Fortran 言語解説書*」の『**IMPLICIT**』を参照してください。

これは、**-qundef** の短い形式です。 238 ページの『-qundef オプション』を参照してください。

-v オプション

構文

-v

コンパイルの進捗状況を生成します。

規則

コンパイラーがコマンドを実行してさまざまなコンパイル・ステップを実行するときに、このオプションは、コンパイラーが呼び出すコマンドおよび、コンパイラーが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。

特定のコンパイルの関して、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

74 ページの『-# オプション』 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。

-V オプション

構文

-V

表示から直接切り貼りによってコマンドを作成できるという点を除き、このオプションは **-v** と同じです。

-W オプション

構文

-Wcomponent,options

リストされたオプションを、コンパイル中に実行されるコンポーネントに渡します。*component* は、**a**、**F**、 **c**、**h**、**I**、**b**、**z**、**l**、または **d** の 1 つで、それぞれアセンブラー、C プリプロセッサー、コンパイラー、配列言語最適化プログラム、プロシージャー間分析 (IPA) ツール/ループ、コード生成プログラム、バインド・プログラム、リンカー、 および **-S** 逆アセンブラーに対応します。

Component	-W Mnemonic	Standard Program Name
assembler	a	as
C preprocessor	F	cpp
compiler front end	c	xlfentry
array language optimizer	h	xlfhot
IPA/loop optimizer	I	ipa
code generator	b	xlfcode
binder	z	bolt
linker	l	ld
disassembler	d	dis

-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。たとえば、次のように指定します。

```
-Wcomponent,option_1[,option_2,...,option_n]
```

背景情報

このオプションの主な目的は、コンパイラー・オプションのシーケンスを作成して、最適化プリプロセッサーの 1 つに渡すことです。また、**ld** コマンドにパラメーターを渡すことによって、リンク・エディット・ステップを微調整するのにも使用できます。

デフォルト

ほとんどのオプションは、リンカーに渡す際に **-W** 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るからです。 **-W** (または構成ファイル内の **ldopts** スタンザ) が絶対に必要なオプションは、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 (たとえば **-v** または **-S**) のみです。

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

26 ページの『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。

¥ を使用して、**-W** オプションで提供されたストリングにリテラル・コンマを組み込むことができます。

次の例の **¥** は、**-WF** ストリングにリテラル・コンマを組み込み、4 つではなく、3 つの引数が生じ、C プリプロセッサーに提供されます。

```
$ xlf -qfree=f90  '-WF,-Dint1=1,-Dint2=2,-Dlist=3¥,4'  a.F
$ cat a.F
print *, int1
print *, int2
print *, list
end
```

プログラムからの出力は以下のとおりです。

```
$ ./a.out
 1
 2
 3 4
```

-w オプション

構文

-w

140 ページの『-qflag オプション』の同義語です。**-qflag=e:e** を設定して、言語レベル検査によって生成されるメッセージだけではなく、警告メッセージおよび通知メッセージも抑止します。

-y オプション

構文

```
-y{n | m | p | z}
IEEE(Near | Minus | Plus | Zero)
```

コンパイル時に定数浮動小数点式を評価するときにコンパイラーが使用する丸めモードを指定します。これは **-qieee** オプションと同等です。

引数

n 最も近い値に丸めます。

m マイナスの無限大方向に丸めます。

p プラスの無限大方向に丸めます。

z ゼロ方向に丸めます。

関連情報

88 ページの『-O オプション』および 141 ページの『-qfloat オプション』 を参照してください。

-y は 152 ページの『-qieee オプション』の短い形式です。

第 6 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・スペースと 64 ビット整数の使用時に 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。

64 ビット・アドレス・スペース内に収まる、より大きな実行可能モジュールをサポートするように、 64 ビット実行可能モジュールの要件を満たす個別の 64 ビット・オブジェクト・フォームが使用されます。リンカーは、64 ビット実行モジュールを作成するために 64 ビット・オブジェクトをバインドします。バインドされるオブジェクトは、すべて同じオブジェクト形式である必要があることに注意してください。 以下のシナリオは認められておらず、ロードまたは実行、あるいはその両方が失敗します。

- 32 ビット・ライブラリーまたは共用ライブラリーからシンボルに対する参照を持つ 64 ビット・オブジェクトまたは実行可能ファイル
- 64 ビット・ライブラリーまたは共用ライブラリーからシンボルに対する参照を持つ 32 ビット・オブジェクトまたは実行可能ファイル
- 32 ビット・モジュールを明示的にロードしようとする 64 ビット実行モジュール
- 64 ビット・モジュールを明示的にロードしようとする 32 ビット実行モジュール
- 32 ビット・プラットフォームで 64 ビット・アプリケーションの実行を試みること

64 ビット・プラットフォームでも 32 ビット・プラットフォームでも、32 ビット実行モジュールは引き続き、現行の 32 ビット・プラットフォームの場合と同様に実行されます。32 ビット・プラットフォームでは、64 ビットの実行可能プログラムは **-q64** オプションを指定することによって生成することができます。

XL Fortran コンパイラーは、主に 64 ビット・モードのサポートを、コンパイラー・オプション **-qarch** とともに使用されるコンパイラー・オプション **-q64** で提供します。ターゲット・アーキテクチャーのビット・モードおよび命令セットはこの組み合わせによって決まります。 **-q32** および **-q64** オプションは、**-qarch** オプションの設定よりも優先されます。 **-q64** オプションは 32 ビット・モードの **-qarch** 設定にのみ優先し、コンパイラーは **-qarch** 設定を、64 ビット・モードを扱う項目にアップグレードします。**-q32** オプションと **-q64** オプションとの競合は、「最後のオプションが優先される」という規則で解決されます。 **-qarch=com** の設定は、32 ビット・モードでのアプリケーションの将来の互換性を保証します。 64 ビット・モード・アプリケーションの場合、**-qarch=ppc64** を使用すれば、現在されている、または将来サポートされる 64 ビット・モード・システムで同じ効果が得られます。 **rs64b**、**rs64c**、**pwr3**、**pwr4**、**pwr5**、**pwr5x**、**ppc970**、 および **auto** 設定のように、特定のアーキテクチャーを対象とする **-qarch** 設定は、よりシステムに依存するようになります。

64 ビット環境のコンパイラー・オプション

-q32、**-q64**、および **-qwarn64** コンパイラー・オプションは、主に 64 ビット・プラットフォームを対象にする開発者用です。それらを使用して次のことが可能になります。

- 64 ビット環境のアプリケーションの開発
- 32 ビット環境から 64 ビット環境へのソース・コードのマイグレーション

第 7 章 問題判別とデバッグ

本節では、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、標準エラー出力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・エラーには以下のような重大度レベルがあり、これはエラー・メッセージの一部として表示されます。

U 回復不能エラー。内部コンパイラー・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・エラーを修正し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:
1. 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。
2. メッセージ・レベル **I**、**L**、**W**、**E** は、コンパイルが成功したことを示しています。

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。 **-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。たとえば、**-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低レベルのメッセージを制限することができます。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、 230 ページの『-qsuppress オプション』を参照してください。

コンパイラーの戻りコード

コンパイラーのリターン・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 コンパイラーが重大度 E または *halt_severity* (どちらか重大度の低い方) 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

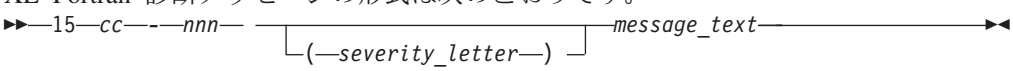
XLF コンパイル・プログラムが異常終了した場合は、オペレーティング・システムへの戻りコードは 1 です。

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 メッセージに関する情報

-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。



上記の意味は次のとおりです。

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 コード生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。

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

ユーザーがすでに気付いていたり、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
```

メッセージの言語の選択

XL Fortran の出荷時のデフォルト・メッセージは、英語だけになっています。さらに、翻訳されたメッセージ・カタログを注文することもできます。

- 日本語でのコンパイラー・メッセージ
- 日本語での実行時メッセージ

コンパイル時メッセージが別の言語で表示されるべき時に英語で表示されている場合は、正しいメッセージ・カタログがインストールされていること、環境変数 **LANG** と **LC_MESSAGES** と **LC_ALL**、あるいは、そのいずれかが適切に設定されていることを確認してください。

実行時メッセージが別の言語で表示される場合は、お使いのプログラムが **setlocale** ルーチンを呼び出すことも確認してください。

関連情報: 8 ページの『各国語サポートのための環境変数』および 33 ページの『実行時メッセージ用の言語の選択』を参照してください。

インストールされている XL Fortran メッセージ・カタログを判別するには、以下のコマンドを使用して、インストールされているメッセージ・カタログのリストを表示してください。

```
rpm -ql xlf.cmp        # compile-time messages
rpm -ql xlf.msg.rte    # run-time messages
rpm -ql xlsmp.msg.rte  # SMP run-time messages
```

メッセージ・カタログのファイル名は、サポートされているすべての各国語で同一です (入っているディレクトリーは別)。

注: XL Fortran プログラムを XL Fortran メッセージ・カタログがないシステム上で実行した場合、ランタイム・エラー・メッセージ (ほとんどは I/O 問題に関するもの) が正しく表示されず、プログラムはメッセージ番号を出しますが、それに関連したテキストは表示されません。この問題を回避するには、XL Fortran メッセージ・カタログを **/opt/ibmcmp/msg** から、実行システムで設定された **NLSPATH** 環境変数の一部となっているディレクトリーへコピーします。

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。

---

xlf90: not found
xlf90_r: not found
xlf95: not found
xlf95_r: not found
xlf: not found
xlf_r: not found
f77: not found
fort77: not found
f90: not found
f95: not found
徴候: 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。
解決策: **PATH** 環境変数にディレクトリー **/opt/ibmcmp/xlf/10.1/bin** が入っ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。

Could not load program *program*
Error was: not enough space
徴候: 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。
解決策: 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。たとえば、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 program *program*
Could not load library *library_name*.**so**
Error was: no such file or directory
解決策: XL Fortran ライブラリーが **/opt/ibmcmp/xlf/10.1/lib** と **/opt/ibmcmp/xlf/10.1/lib64** 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 **libxlf90.so** がインストールされている場合は、そのディレクトリーが組み込まれるように **LD_LIBRARY_PATH** と **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。

徴候: コンパイラーまたは XL Fortran アプリケーション・プログラムからのメッセージが別の言語で表示されます。
解決策: 正しい各国語環境を設定してください。 **env** コマンドを使用して各ユーザー用の各国語を設定することができます。あるいは、各ユーザーが環境変数 **LANG**、**NLSPATH**、**LC_MESSAGES**、**LC_TIME**、および **LC_ALL** のうちの 1 つ以上を設定することもできます。これらの変数の目的がよくわからない場合は、8 ページの『各国語サポートのための環境変数』に詳細が記載されているので参照してください。

徴候: 入出力エラーでコンパイルが失敗します。
解決策: **/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。

徴候: 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。
解決策: 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。これらのオプションのリストに関しては 60 ページの『互換性を維持するためのオプション』を参照し、移植に関する概要は「*XL Fortran 最適化およびプログラミング・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

たとえば **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。 (サンプル構成ファイルとその内容の説明については、 11 ページの『構成ファイルのカスタマイズ』 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・スペースの詳細については、 man ページ情報 **man 8 mkswap swapon** を参照してください。

たとえば、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。

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。

filename.o: In function 'main':
filename.o(.text+0x14): undefined reference to 'p'
filename.o(.text+0x14): relocation truncated to fit: R_PPC_REL24 p

徴候: 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。

解決策: 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、-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入ってい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、33 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、43 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。これらのオプションのリストに関しては

60 ページの『互換性を維持するためのオプション』を参照し、移植に関する概要は「*XL Fortran 最適化およびプログラミング・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、したがって、変数に影響を与える最適化を防止できる場合もあります。 (「*XL Fortran 言語解説書*」の『**VOLATILE**』を参照。) 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で、ユーザーがエラーの入出力ステートメントに **IOSTAT** を指定していた場合は、 **IOSTAT** 変数に「*XL Fortran 言語解説書*」の『*条件および IOSTAT 値*』に従って値が割り当てられます。

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、ある一定の入出力エラーに対して、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。 33 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** の **ulimit** などのシェル・コマンド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
```

例外ハンドラーに関する指示については、「*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、実行時例外の原因の詳細については、 43 ページの『XL Fortran 実行時例外』を参照してください。

Fortran 90 または Fortran 95 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、**dbx** 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。

関連情報:

- 54 ページの『エラー・チェックおよびデバッグのためのオプション』

第 8 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: 58 ページの『リストとメッセージを制御するオプション』を参照してください。

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。

- 以下のものから構成されるコンパイラー識別子
  - コンパイラー名
  - バージョン名
  - 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、1 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、次の節は、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・プロセス中に生成されるエラー・メッセージには、次のものがあります。

- 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。以下のもので構成されます。
  - 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

たとえば、次のようになります。

```
         2 |          equivalence (i,j,i)
       ................................a.
a - 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。以下のもので構成されます。
  - 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

たとえば、次のようになります。

```
"doc.f", line 6.11: 1513-039 (S) Number of arguments is not
permitted for INTRINSIC function abs.
```

変換報告書セクション

-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 XL Fortran がプログラムを最適化した方法が示されます。本節では、元のソース・コードに対応する擬似 Fortran コードを表示し、 **-qhot** または **-qsmp** オプション (あるいは、その両方) 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。

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
                  @LoopIV0 = 0
       Id=1       DO @LoopIV0 = @LoopIV0, 99
     5|             IF (.FALSE.) GOTO lab_11
                    @LoopIV1 = 0
       Id=2         DO @LoopIV1 = @LoopIV1, 99
                      ! DIR_INDEPENDENT loopId = 0
     6|               a((@LoopIV1 + 1),(@LoopIV0 + 1)) = (@LoopIV0 + 1)
     7|             ENDDO
                    lab_11
     8|           ENDDO
                  lab_9
     9|         END PROGRAM _main


Source        Source        Loop Id       Action / Information
File          Line

----------    ----------    ----------    ------------------------------------
         0             4             1    Loop interchanging applied to loop nest.

>>>>> FILE TABLE SECTION <<<<<
```

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。
  - タイプ
  - 名前のクラス
  - 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
  - private、public、protected、module
- 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 (割り振り可能オブジェクトの場合)
- 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
- pointee
- プライベート・コンポーネント
- プログラム
- 参照パラメーター
- 名前変更
- 静的
- 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。そして最後に、合計サイクル・タイム (直線的実行時間) と、作成されたマシン・インストラクションの合計数が表示されます。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。このセクションは必ず存在します。

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 時間です。このセクションは、リストに必ず存在します。

付録 A. XL Fortran 技術情報

本節では、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。

コンパイラー・フェーズ

典型的なコンパイラー呼び出しコマンドは、次のプログラムの一部またはすべてを順に実行します。リンク時最適化の場合は、フェーズのいくつかがコンパイル中に複数回実行されます。 各プログラムが実行されるたびに、その実行結果が次のプログラムに送られます。

1. プリプロセッサー
2. 以下の段階で構成されるコンパイラー。
   a. フロントエンドの構文解析とセマンティック分析
   b. ループ変換
   c. プロシージャー間分析
   d. 最適化
   e. レジスターの割り振り
   f. 最終アセンブリー
3. アセンブラー (任意の **.s** ファイルを対象とする)
4. リンカー **ld**

XL Fortranライブラリー内の外部名

ユーザー定義の名前と実行時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、実行時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

XL Fortran ランタイム環境は、システムの環境に対応する各国語で診断メッセージを作成します。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、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 141 ページの『-qfloat オプション』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。

たと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。たとえば、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行の詳細

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。たとえば、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ストレージ・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ストレージ・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、pointee 先配列、pointee **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。たとえば、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、不定様式の 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

本節に示す例は、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。

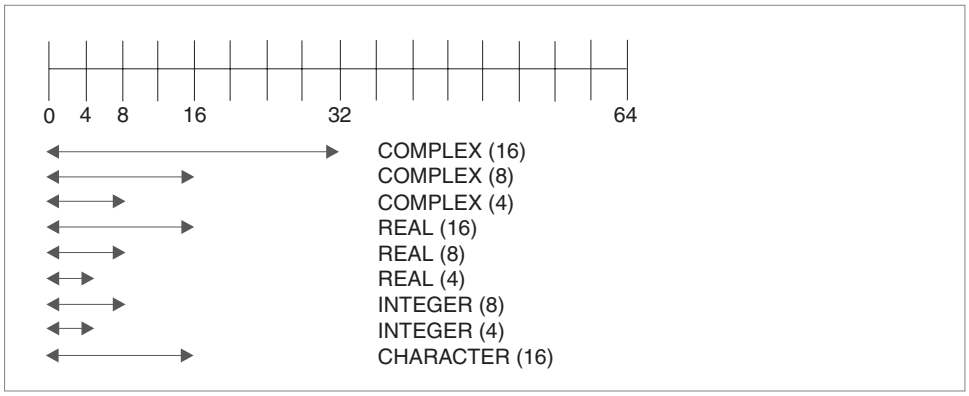


図 1. -qautodbl オプションを指定しなかった場合のストレージの関係

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
```

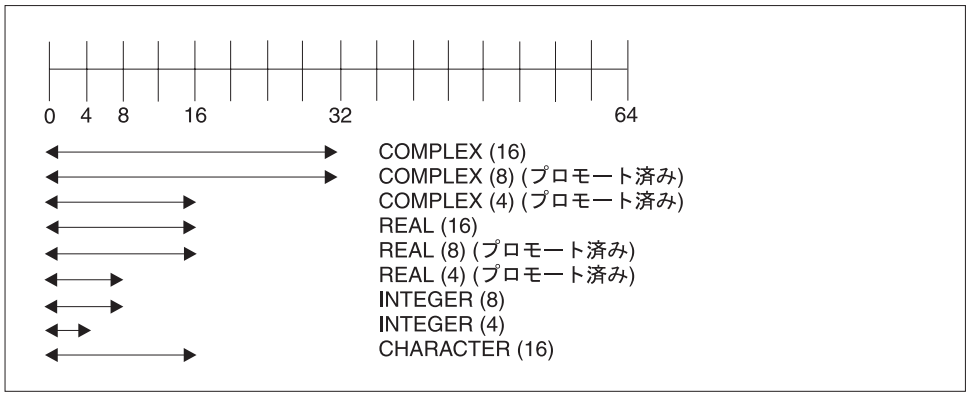


図 2. -qautodbl=dbl を指定した場合のストレージの関係

```
@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
```

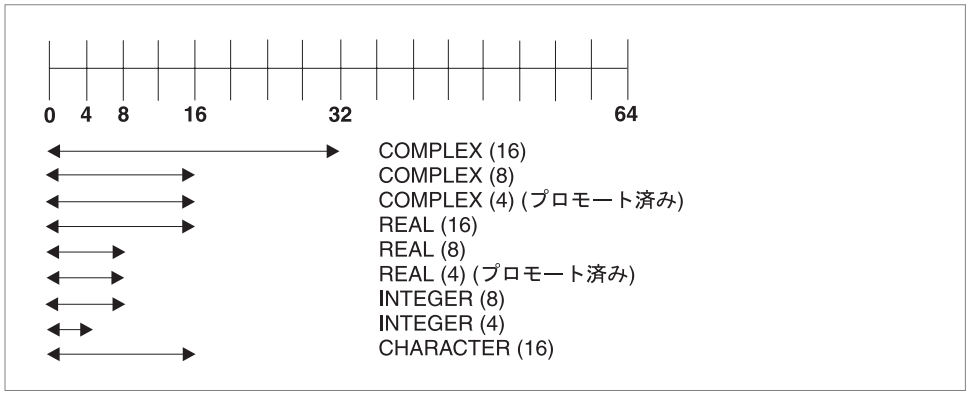


図 3. -qautobl=dbl4 を指定した場合のストレージの関係

```
@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
```

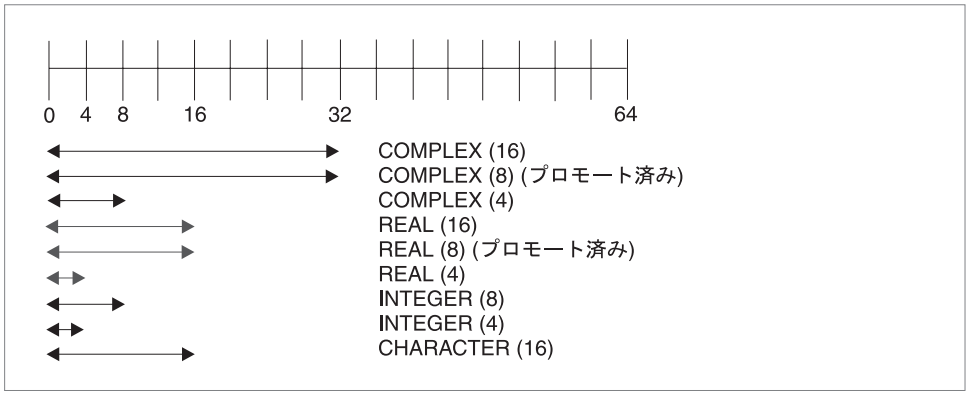


図 4. -qautodbl=dbl8 を指定した場合のストレージの関係

```
@process autodbl(dbl8)
      complex(8) x16    /(1.12345678912345678d0,2.12345678912345678d0)/
      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   = (1.12345678912345678q0,2.12345678912345678q0)
!                                                ! promotion occurred
!     x8    = upper 8 bytes of r8(1)             ! promotion did not occur
!     r8(1) = 1.12345678912345678q0              ! promotion occurred
!     r8(2) = 2.12345678912345678q0              ! promotion occurred
      end
```

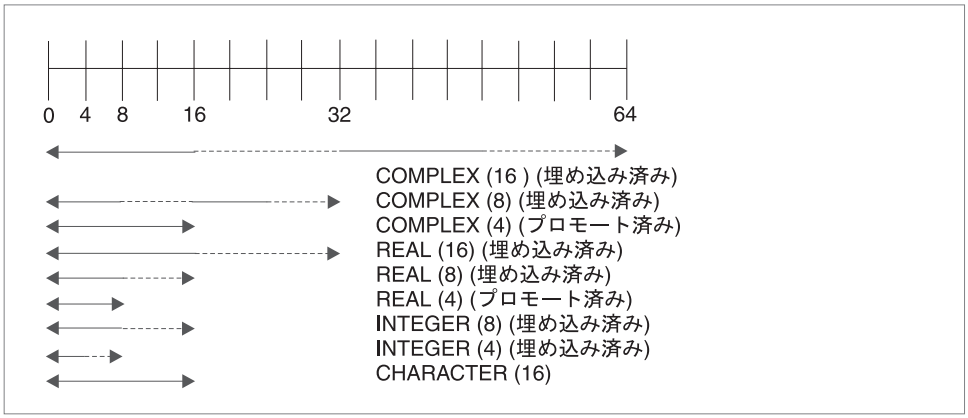


図 5. -qautodbl=dblpad4 を指定した場合のストレージの関係

上の図で、破線は埋め込みを表します。

```
@process autodbl(dblpad4)
      complex(8) x16  /(1.123456789d0,2.123456789d0)/
      complex(4) x8
      real(4) r4(2)
      integer(8) i8(2)
      equivalence(x16,x8,r4,i8)
!     Storage relationship among all entities is preserved.
!     Date values between x8 and r4 are preserved.
!     x16   = (1.123456789d0,2.123456789d0)      ! padding occurred
!     x8    = (upper 8 bytes of x16, 8 byte pad) ! promotion occurred
!     r4(1) = real(x8)                           ! promotion occurred
!     r4(2) = imag(x8)                           ! promotion occurred
!     i8(1) = real(x16)                          ! padding occurred
!     i8(2) = imag(x16)                          ! padding occurred
      end
```

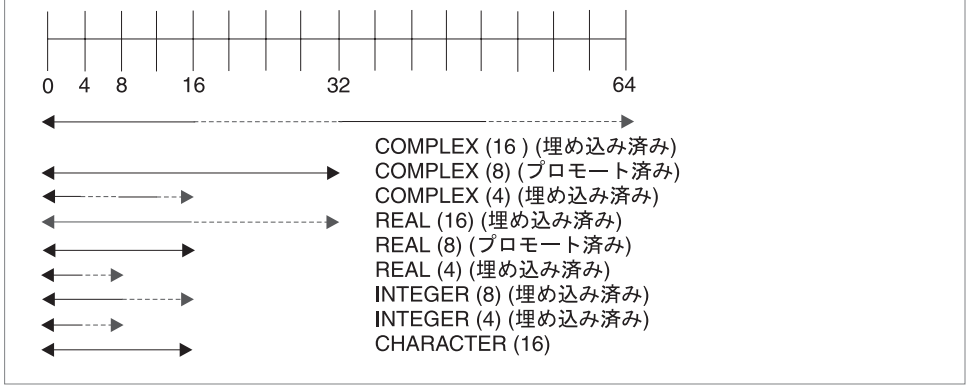


図 6. -qautodbl=dblpad8 を指定した場合のストレージの関係

上の図で、破線は埋め込みを表します。

```
@process autodbl(dblpad8)
      complex(8) x16    /(1.12345678912345678 9d0,2.12345678912345678 9d0)/
      complex(4) x8
      real(8) r8(2)
      integer(8) i8(2)
      byte b(16)
      equivalence (x16,x8,r8,i8,b)
```

```
!     Storage relationship among all entities is preserved.
!     Data values between r8 and x16 are preserved.
!     Data values between i8 and b are preserved.
!     x16 = (1.123456789123456789q0,2.123456789123456789q0)
!                                              ! promotion occurred
!     x8 = upper 8 bytes of r8(1)              ! padding occurred
!     r8(1) = real(x16)                        ! promotion occurred
!     r8(2) = imag(x16)                        ! promotion occurred
!     i8(1) = upper 8 bytes of real(x16)       ! padding occurred
!     i8(2) = upper 8 bytes of imag(x16)       ! padding occurred
!     b(1:8)= i8(1)                            ! padding occurred
!     b(9:16)= i8(2)                           ! padding occurred
      end
```

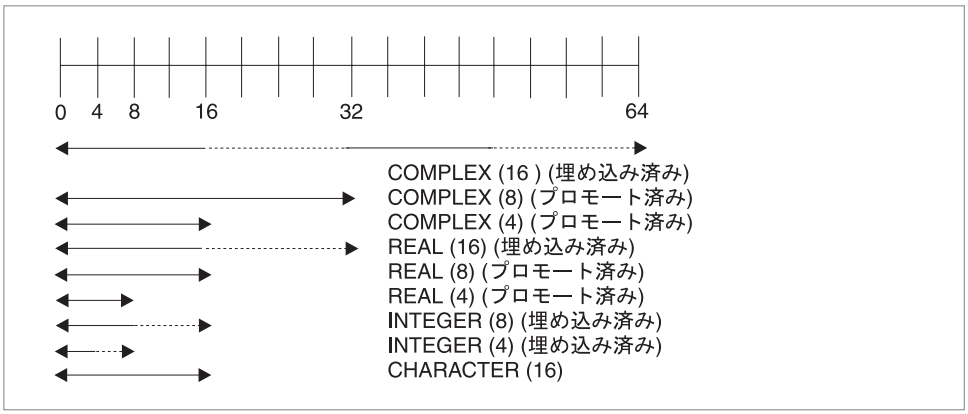


図 7. -qautodbl=dblpad を指定した場合のストレージの関係

上の図で、破線は埋め込みを表します。

```
@process autodbl(dblpad)
      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(8) x8
         real(16) r16(4)
         integer(8) i8(4)
         character*5 c(2)
         common /named/ x8,r16,i8,c
!          x8    = (1.123456789d0,2.123456789d0)   !  promotion occurred
!         r16(1) = 1.123q0                         !  padding occurred
!         r16(3) = 2.123q0                         !  padding occurred
!          i8(1) = 1000                            !  padding occurred
!          i8(3) = 2000                            !  padding occurred
!           c(1) = "abcde"                         !  no padding occurred
!           c(2) = "12345"                         !  no padding occurred
      end subroutine s
```

付録 B. XL Fortran 内部制限

言語の特徴	制限
INTEGER(n) の索引付き変数での、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) の索引付き変数での、ループ制御による **DO** ループの最大反復実行回数	(2**63)-1
文字フォーマット・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	34,000
自由ソース形式文の最大長	34,000
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	1,024
計算済み **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
32 ビット・モードでの入出力ファイルの許容レコード番号とレコード長	レコード番号は最大 (2**63)-1 です。最大レコード長は (2**31)-1 バイトです。
64 ビット・モードでの入出力ファイルの許容レコード番号とレコード長	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、不定様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、不定様式の順次ファイル内のレコードの最大長は (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、 32 ビット・モードでは -(2**31) から 2**31-1 までの範囲、64 ビット・モードでは -(2**63) から 2**63-1 までの範囲内の正、負、またはゼロのいずれかの値をとります。
外部装置番号の許容数	0 から (2**31)-1 **2**
数値フォーマット・フィールドの最大幅	2,000
同時にオープンできるファイルの最大数	1 024 **3**

1 1 つの文 (最大サイズ 34,000 バイト) を作成する場合、継続行数の制限はないので、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 実際には、実行時にシステムがオープンするファイル (事前接続した装置 0、5、6 など) のために、この値は表中の値よりも小さくなります。

特記事項

本書は米国 IBM が提供する製品およびサービスについて作成したものです。

本書に記載の製品、サービス、または機能が日本においては提供されていない場合があります。日本で利用可能な製品、サービス、および機能については、日本 IBM の営業担当員にお尋ねください。本書で IBM 製品、プログラム、またはサービスに言及していても、その IBM 製品、プログラム、またはサービスのみが使用可能であることを意味するものではありません。これらに代えて、IBM の知的所有権を侵害することのない、機能的に同等の製品、プログラム、またはサービスを使用することができます。ただし、IBM 以外の製品とプログラムの操作またはサービスの評価および検証は、お客様の責任で行っていただきます。

IBM は、本書に記載されている内容に関して特許権 (特許出願中のものを含む) を保有している場合があります。本書の提供は、お客様にこれらの特許権について実施権を許諾することを意味するものではありません。実施権についてのお問い合わせは、書面にて下記宛先にお送りください。

〒106-0032
東京都港区六本木 3-2-31
IBM Corporation
IBM World Trade Asia Corporation
Licensing

以下の保証は、国または地域の法律に沿わない場合は、適用されません。

IBM およびその直接または間接の子会社は、本書を特定物として現存するままの状態で提供し、商品性の保証、特定目的適合性の保証および法律上の瑕疵担保責任を含むすべての明示もしくは黙示の保証責任を負わないものとします。国または地域によっては、法律の強行規定により、保証責任の制限が禁じられる場合、強行規定の制限を受けるものとします。

この情報には、技術的に不適切な記述や誤植を含む場合があります。本書は定期的に見直され、必要な変更は本書の次版に組み込まれます。 IBM は予告なしに、随時、この文書に記載されている製品またはプログラムに対して、改良または変更を行うことがあります。

本書において IBM 以外の Web サイトに言及している場合がありますが、便宜のため記載しただけであり、決してそれらの Web サイトを推奨するものではありません。それらの Web サイトにある資料は、この IBM 製品の資料の一部ではありません。それらの Web サイトは、お客様の責任でご使用ください。

IBM は、お客様が提供するいかなる情報も、お客様に対してなんら義務も負うことのない、自ら適切と信ずる方法で、使用もしくは配布することができるものとします。

本プログラムのライセンス保持者で、(i) 独自に作成したプログラムとその他のプログラム (本プログラムを含む) との間での情報交換、および (ii) 交換された情報の相互利用を可能にすることを目的として、本プログラムに関する情報を必要とする方は、下記に連絡してください。

IBM Corporation
Lab Director
IBM Canada Limited
8200 Warden Avenue
Markham, Ontario, Canada
L6G 1C7

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。

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。お客様は、IBM 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。

この情報をソフトコピーでご覧になっている場合は、写真やカラーの図表は表示されない場合があります。

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。

OpenMP は、OpenMP Architecture Review Board の商標です。本書の一部は、*OpenMP Application Program Interface, Version 2.5, (May 2005)*の仕様から抜粋されたものです。 Copyright 1997-2005 OpenMP Architecture Review Board.

プログラミング・インターフェース情報

プログラミング・インターフェース情報は、プログラムを使用してアプリケーション・ソフトウェアを作成する際に役立ちます。

一般使用プログラミング・インターフェースにより、お客様はこのプログラム・ツール・サービスを含むアプリケーション・ソフトウェアを書くことができます。

ただし、この情報には、診断、修正、および調整情報が含まれている場合があります。診断、修正、調整情報は、お客様のアプリケーション・ソフトウェアのデバッグ支援のために提供されています。

注: 診断、修正、調整情報は、変更される場合がありますので、プログラミング・インターフェースとしては使用しないでください。

商標およびサービス・マーク

以下は、IBM Corporation の商標です。

IBM	IBM (ロゴ)	POWER3
POWER4	POWER5	PowerPC
PowerPC Architecture	pSeries	z/OS

Linux は、Linus Torvalds の米国およびその他の国における商標です。

UNIX は、The Open Group の米国およびその他の国における登録商標です。

Windows は、Microsoft Corporation の米国およびその他の国における商標です。

他の会社名、製品名およびサービス名などはそれぞれ各社の商標または登録商標です。

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「*IBM Dictionary of Computing*」から抜粋した項目が含まれています。

［ア行］

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。

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。

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。

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 ～ N の場合、長さが 4 の整数型となる。最初の文字が A ～ H、O ～ Z、$、_ の場合、長さが 4 の実数型となる。

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。

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。

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。

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。

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 ～ Z に加え、特定の言語で使用可能なその他の特殊記号 (たとえば $ 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 (たとえば、句読符号、数学記号など) が含まれる。

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。

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。

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。

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。

［カ行］

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。

外部名 (external name). リンカーが、一つのコンパイル単位から別のもうひと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。

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。

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。

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。

環境変数 (environment variable). プロセスの操作環境を記述する変数。

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.        に等しくない
```

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。

関連名 (associate name). **ASSOCIATE** 構文内で、この構文の選択子が認識される名前。

キーワード (keyword). (1) 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。 (2) 引数キーワードは、仮引数の名前を指定する。

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。

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。

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。

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。

形式 (format). (1) 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。 (2) 文字、フィールド、行などを配置すること。

継続行 (continuation line). 文をその最初の行を越えて継続させる行。

結果変数 (result variable). 関数の値を戻す変数。

高位変換 (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。

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。

構文 (construct). たとえば、**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。

構文 (syntax). 文の構造に関する規則。セマンティクス *(semantics)* も参照。

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。

コンパイラー・コメント・ディレクティブ. 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。

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。

［サ行］

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。)

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。

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
```

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。

算術定数 (arithmetic constant). 整数、実数、複素数のいずれかの型の定数。

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。

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。

シグナル 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。

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)。

実行可能ステートメント (executable statement). プログラムにある処置、たとえば、計算する、条件をテストする、通常の順次実行を変更するなど、を引き起こさせるステートメント。

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、サブプログラム、Fortran 以外の外部プロシージャーからなる。

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。

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。

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、選択戻り指定子のいずれか。

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。

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。

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。

純粋 (pure). 副次作用がないことを示す、プロシージャーの属性。

使用関連付け (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。

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。

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。

情報交換用米国標準コード(American National Standard Code for Information Interchange). *ASCII* を参照。

数字 (digit). 負数ではない整数を表す文字。たとえば、0 ～ 9 のいずれかの数字。

数値記憶単位. 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。

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。

スカラー (scalar). (1) 配列ではない単一のデータ。 (2) 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)。

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・ラベル (statement label). 1 ～ 5 桁の番号。文の識別に使用される。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)。

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。

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。

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。

スレッド可視変数 (thread visible variable). 複数のスレッドからアクセス可能な変数。

正規 (normal). 非正規、無限大、または NaN でない浮動小数点数。

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。

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 NaN (signalling NaN)* も参照。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

[タ行]

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。

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。

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。

チャンク (chunk). 連続するループ反復のサブセット。

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。

データ転送ステートメント (data transfer statement). **READ**、**WRITE**、**PRINT** の各文。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

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。

［ナ行］

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。

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。

入出力 (input/output (入出力)). 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。

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。たとえば、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。

［ハ行］

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。

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。

バイト定数 (byte constant). バイト型の名前付き定数。

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。

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。

配列名 (array name). 順序付けられたデータ項目のセットの名前。

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。たとえば、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。

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。

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。

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。

非数字 (not-a-number). *NaN* を参照。

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。

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。

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。

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。

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。

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。

不定様式レコード (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。

浮動小数点数 (floating-point number). 異なる数表示の対で表される実数。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。

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。

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。

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。

プロシージャー間分析 (interprocedural analysis). *IPA* を参照。

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。

プロファイル指示フィードバック (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。

文 (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。

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。

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。

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。

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。

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。

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。

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は、モジュール・プロシージャーのホストと呼ばれる。

ホスト関連付け (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。

［マ行］

マスター・スレッド (master thread). スレッドのグループのヘッド・プロセス。

無限大. 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。

無名共通ブロック (blank common). 名前のない共通ブロック。

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。

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。

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。

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 (たとえば、連結 (*//*) など)。

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。

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。

文字ストリング (character string). 連続した文字の列。

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。

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。

戻り指定子 (return specifier). 文 (たとえば 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。

［ヤ行］

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). (1) 派生型の定義。 (2) インターフェース本体 (ただし、インターフェース本体に含まれる派生型の定義とインターフェース本体は除く)。 (3) 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、インターフェース本体、サブプログラムは除く)。

［ラ行］

ランク (rank). 配列のディメンション数。

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。

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。

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。

ループ (loop). 繰り返し実行されるステートメント・ブロック。

レコード (record). ファイル内でまとめて扱われる値の順序列。

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)
```

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。

[ワ行]

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。

[数字]

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。

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 からものである。)

16 進 (hexadecimal). システムに関連する基数が 16 の数字。 16 進数は、0 ～ 9 と A (10) ～ 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。

2 進定数 (binary constant). 1 つ以上の 2 進数字 (0 と 1) からの定数。

8 進 (octal). システムに関連する基数が 8 の数字。8 進数は、0 ～ 7 の範囲にある。

8 進定数 (octal constant). 8 進数からなる定数。

A

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。

B

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。

D

DO 変数 (DO variable). **DO** 文で指定される変数。 **DO** のループ内にある 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。

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。

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。

I

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。

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。

K

kind 型付きパラメーター (kind type parameter). 組み込み型の使用可能な種類のラベルを付けたパラメーターの値。

M

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 NaN (signalling NaN)* も参照。

P

PDF. *プロファイル指示フィードバック (profile-directed feedback)* を参照。

pointee 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。

S

SMP. *対称マルチプロセッシング (symmetric multiprocessing)* を参照。

U

Unicode. 最近の世界のいかなる言語で書かれたテキストの交換、処理、表示をもサポートする汎用文字エンコード標準。この標準は、いくつか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-bit 国際文字セットを持っている。*ASCII* も参照。

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。

［特殊文字］

_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。

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］

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

プログラム番号: 5724-M17

SD88-6732-00

日本アイ・ビー・エム株式会社
〒106-8711　東京都港区六本木3-2-12