IBM XL Fortran Advanced Edition V9.1 for Linux

ユーザーズ・ガイド

SC88-9699-01
(英文原典：SC09-7948-01)

IBM XL Fortran Advanced Edition V9.1 for Linux

ユーザーズ・ガイド

SC88-9699-01
(英文原典：SC09-7948-01)

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、 405 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 V9.1 for Linux™ の バージョン 9.1.1 (プログラム番号 5724-K76) 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。本書の前版からの変更点は、左側の余白にある縦線 (|) で示されています。

•

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： SC09–7948–01
IBM XL Fortran Advanced Edition V9.1 for Linux
User's Guide

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 2005.1

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

図

XL Fortran の新機能

XL Fortran バージョン 9.1 では、次の新しい機能と変更された機能を提供します。

新しいまたは変更されたコンパイラー・オプションおよびサブオプションは、以下のとおりです。

- **-qcr** オプションを使用すると、コンパイラーが CR (復帰) 文字をどのように解釈するのかを制御することができます。
- **-qflttrap=nanq** 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。
- **-qport=nullarg** サブオプションは、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き数をヌル引き数として扱います。
- **-qmodule=mangle81** オプションは、非組み込みモジュールに対するバージョン 8.1 モジュールの命名規則との互換性を提供します。
- **-qsaveopt** オプションは、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。
- **-qversion** オプションは、呼び出しコンパイラーのバージョンとリリースを示します。
- **-qhot=vector** サブオプションは、連続する配列エレメントに対してループで実行される特定の操作を、ベクトル・ライブラリー・ルーチンに対する呼び出しに変換します。

以下の XL Fortran 拡張が Fortran 2003 のドラフト標準から変更されました。

- **2003std** 実行時オプションは、ドラフト標準に合致しているかどうかについてのコードの規格合致検査を提供します。
- **ISO_C_BINDING** 組み込みモジュール、**BIND** 属性およびステートメント、モジュール変数、共通ブロック、サブルーチン/関数、および **-qalign=bindc** コンパイラー・サブオプションは、C との相互運用についてのサポートを提供します。
- 派生型コンポーネントの **PUBLIC/PRIVATE** 属性。
- **ASSOCIATE** 構文は、エンティティーを変数または式の値に関連付けます。
- 以下のコマンド行引き数組み込み機能。
  - **COMMAND_ARGUMENT_COUNT**
  - **GET_COMMAND_ARGUMENT**
  - **GET_ENVIRONMENT_VARIABLE**
- **FLUSH** ステートメントは、他のプロセスで使用可能な外部ファイルからデータを作成します。
- データ転送操作の **IOMSG=** 指定子、ファイル位置指定、**FLUSH**、およびファイル照会ステートメント。
- **ISO_FORTRAN_ENV** 組み込みモジュールは、Fortran 環境に関連するパブリック・エンティティーを提供します。
- **NEW_LINE** 組み込み機能は、改行文字を返します。

- **IMPORT** ステートメントは、ホスト関連付けによって、インターフェース本体でアクセス可能なホスト有効範囲単位から名前付きエンティティーを作成します。
- **PROCEDURE** ステートメントは、ダミー・プロシージャーまたは外部プロシージャーを宣言します。

以下のパフォーマンス関連ディレクティブおよびコンパイラー・オプション/サブオプションが追加されました。

- POWER5 および PowerPC 970 アーキテクチャ (ppc64gr、ppc64grsq、pwr5、および ppc970) に対するサポートを提供する **-qarch** および **-qtune** コンパイラー・サブオプション。
- **-qenablevmx** オプションは、コンパイラーに対して、これらの PowerPC 970 プロセッサーが処理できるように Vector Multimedia eXtension (VMX) 命令を生成するように指示します。
- **-qhot=simd** サブオプションは、Altivec 命令を使用してループを自動的に SIMD ベクトル化します。
- **-qshowpdf** オプションは、**-qpdf1** とともに使用して、追加の呼び出しとブロック・ カウント・プロファイル情報を実行可能ファイルに提供します。
- 最適化ユーティリティーである **showpdf** および **mergepdf** は、PDF 指示コンパイルに関する詳細情報を提供します。
- **-qdirectstorage** オプションは、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。
- ディレクティブ **NOVECTOR**、**NOSIMD**、および **ALIGNX** 組み込みサブルーチンは、自動ベクトル化機能と自動 SIMD ベクトル化機能の非常に細かい制御をコンパイラーで提供します。
- **LOOPID** ディレクティブは、ループに有効範囲固有の ID を付けます。ID は、**BLOCK_LOOP** および他のディレクティブで、ループ特有の変換を制御するために使用できます。ループ変換に関する情報は、オプションの **-qreport** コンパイラーを使用して表示することができます。
- **EIEIO** ディレクティブは、キャッシュおよびメモリー管理に役立ちます。
- **PROTECTED STREAM** ディレクティブを使用すると、保護ストリームを管理できるため、これらのストリームがハードウェア検出ストリームによって置き換えられることはありません。
- **SWDIV** および**SWDIV_NOCHK** 組み込み機能は、ソフトウェアに浮動小数点除算アルゴリズムを提供します。

その他の機能:

- **FRE** および **FRSQRTES** PowerPC 浮動小数点組み込み関数。
- **POPCNT**、および **POPCNTB** 組み込み機能はデータ・オブジェクトのレジスターの設定ビットのカウントを提供し、 **POPPAR** 組み込み機能はデータ・オブジェクトのパリティーを判断します。
- 32 ビットおよび 64 ビット・モジュールは、現在は 1 つのファイルに組み込まれています。
- 複数のインクルード・パスの許可を許可します。
- ベクトル化アプリケーションで使用するための MASS ベクター・ライブラリーが利用可能です。
- コンパイラー呼び出しコマンドおよび各コマンド行ユーティリティーの man ページが提供されています。以前のバージョンで提供されていたヘルプ・ファイルは、コンパイラー呼び出しの man ページに置き換えられました。

はじめに

本書は、IBM® XL Fortran Advanced Edition V9.1 for Linux™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。

本書の使用法

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。

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。

本書は、以下の事項については説明していません。

インストール

「*IBM XL Fortran Advanced Edition V9.1 for Linux インストール・ガイド*」に記載されています。

Fortran プログラムの記述方法

「*XL Fortran ランゲージ・リファレンス*」に記載されています。

本書の第 1 部は、プログラムのコンパイル、リンク、実行に必要なステップに従って編成されていて、その後に XL Fortran コンパイラーの各機能と、このコンパイラーが生成するプログラムについて説明しています。

第 2 部では、ソフトウェア開発について総括的に説明しています。

経験の程度や目的に応じて、お好きな箇所から読み始めたり、お好きな順序でお読みいただくこともできます。以下に特定の作業を行いたい場合に参照すべき箇所を示します。

不特定のユーザーのためにコンパイラーをセットアップしたい場合

11 ページの『インストール手順の指示が記載されている資料』を参照します。

カスタマイズされたコンパイラー・デフォルトを作成したい場合

15 ページの『構成ファイルのカスタマイズ』を参照します。

すべてのコンパイラー・オプションの用途と、それらのオプションの相互の関連を理解したい場合

57 ページの『XL Fortran コンパイラー・オプションの概要』の該当箇所を参照します。

特定のオプションを名前で調べたい場合

85 ページの『XL Fortran コンパイラー・オプションの詳細記述』をアルファベット順に探します。

プログラムを XL Fortran に移植したい場合

72 ページの『互換性を維持するためのオプション』を参照して必要なオプションを確認してから、 379 ページの『XL Fortran へのプログラムの移植』を参照してその他の移植情報を参照します。

構文図およびステートメントの読み方

本書では、Fortran ステートメントおよび Linux コマンドの構文の図示に、「構文図」と呼ばれる表記法を使用しています。コンパイラー・オプションの構文は、「中括弧と大括弧」と呼ばれる表記法を使用して、ステートメントによって図示されています。

Fortran のキーワードは、**OPEN**、**COMMON**、**END** などのように大文字で記述されます。大文字と小文字の区別がない場合でも、構文図に記述されているとおりに入力する必要があります。

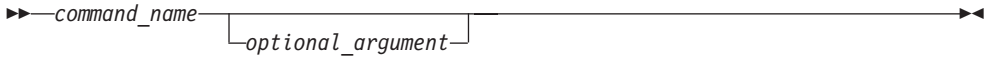
変数名およびユーザー指定の名前は、たとえば *array_element_name* のように小文字で示されます。

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ることを示しています。

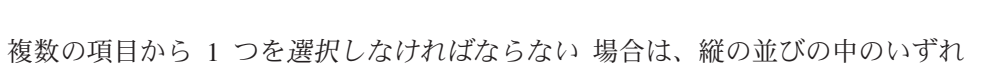
句読記号、括弧、算術演算子、その他の特殊文字は、構文の一部として入力する必要があります。

構文図

- 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。
  - ▸▸── 記号はコマンドの始まりを示します。
  - ──▸ 記号はコマンド構文が次の行に続いていることを示します。
  - ▸── 記号はコマンドが前の行から続いていることを示します。
  - ──▸◂ 記号はコマンドの終わりを示します。
  - 完全なステートメントではなく、小さい構文単位で示す図表は、▸── 記号で始まり、 ──▸ 記号で終わります。
  - 構造体、インターフェース・ブロック、および派生型定義は、複数の個別のステートメントから構成されています。そのような項目の場合、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。
- 必須項目は、次のように横線 (メインパス) 上に記述されます。

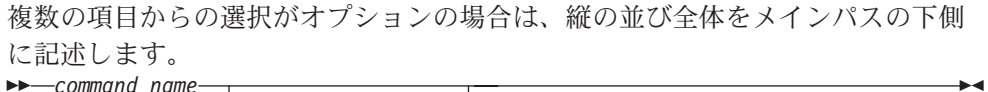


- オプションの選択項目は、次のようにメインパスの下側に記述されます。



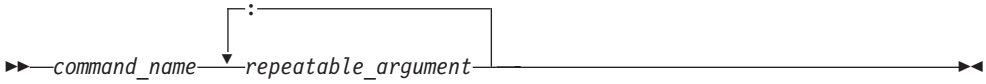
- 複数の項目から選択できる場合は、縦に並べて記述します。

  複数の項目から 1 つを選択しなければならない 場合は、縦の並びの中のいずれか 1 つの項目をメインパスに記述します。



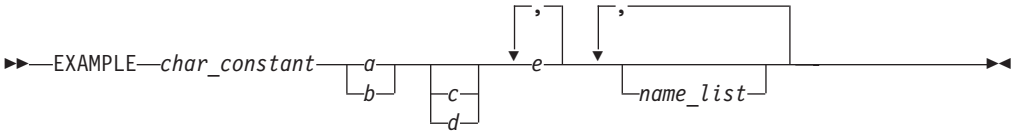
  複数の項目からの選択がオプションの場合は、縦の並び全体をメインパスの下側に記述します。

- 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記号が記述されます。



縦の並びの上側にある繰り返し矢印は、縦に並べて記述さ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。

構文図の例



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。
- *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。

構文ステートメント

構文ステートメントは、左から右に読みます。

- 個々の必須引き数は、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。
- オプションの引き数は、[ ] 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。
- 繰り返せる引き数の後には、省略符号 (...) が示されます。

構文ステートメントの例

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

次のリストは、構文ステートメントを説明しています。

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

注: 同じ例が構文ステートメントと構文図の両方で使用されています。

本書の例についての注意事項

- 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、考えられるすべての実行方法を示すことなどは目的としていないため、単純な形式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、23 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、および本書で提供する考え方の具体例を示すプログラムの一部は、 **/opt/ibmcmp/xlf/9.1/samples** ディレクトリーに入っています。

本書の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/9.1
```

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/9.1
```

本書の用語についての注意事項

本書中の用語には、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。

書体の規則

この文書では、以下のようなテキスト区別方式を使用します。

- Fortran キーワード、コマンド、ステートメント、ディレクティブ、組み込みプロシージャー、コンパイラー・オプション、およびファイル名は**太字**で表示します。たとえば、**COMMON**、**END**、および **OPEN** などです。
- 他の情報源への参照は、*イタリック* で表示します。
- 変数名とユーザー指定名は、小文字のイタリックで表示します。たとえば、*array_element_name* などです。

関連資料

さらに詳しく知りたい場合は、以下の資料を参照してください。

XL Fortran およびオペレーティング・システムの資料

- 「*IBM XL Fortran Advanced Edition V9.1 for Linux ランゲージ・リファレンス*」には、 XL Fortran プログラム言語が説明されています。
- URL `http://www.rpm.org/` の ***RPM ホーム・ページ*** には、RPM Package Manager (RPM) を使用した標準 Linux インストール手順のすべての面が記載されています。 XL Fortran では、一般的なインストール手順をこのライセンス・プログラムに適用する方法を説明している、簡潔なインストールの指示が「*IBM XL Fortran Advanced Edition V9.1 for Linux インストール・ガイド*」に提供されています。
- Linux に関する一般情報および資料については、URL `http://www.tldp.org/` の「***The Linux Documentation Project***」にアクセスしてください。オペレーティング・システムとその機能の使用に関する情報は、Linux についてのみ配布された man ページ・ヘルプおよび資料を参照してください。
- Linux に関する IBM 関連オファリングについての情報は、URL `http://www.ibm.com/linux/` の「***Linux at IBM***」ホーム・ページにアクセスしてください。
- 「*System V Application Binary Interface: PowerPC Processor Supplement*」は、汎用システム V ABI の補足であり、これには 32 ビット・モード PowerPC アーキテクチャー™ オペレーティングで作成された System V インプリメンテーション固有の情報が記載されています。
- 「*64-bit PowerPC ELF Application Binary Interface Supplement*」は、汎用システム V ABI の補足であり、これには 64 ビット・モード PowerPC アーキテクチャー・オペレーティングで作成された System V インプリメンテーション固有の情報が記載されています。

規格資料

本書で言及されているいくつかのフィーチャーの厳密な定義については、規格に関する以下の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 という略式名で呼んでいます。)
- 「*Federal (USA) Information Processing Standards Publication Fortran*」、FIPS PUB 69-1
- 「*ANSI/IEEE Standard for Binary Floating-Point Arithmetic*」、ANSI/IEEE Std 754-1985
- 「*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 という略式名で呼んでいます。)
- 「*Information technology - Programming Languages - Fortran - Floating-Point Exception Handling*」、ISO/IEC JTC1/SC22/WG5 N1379
- 「*Information technology - Programming Languages - Fortran - Enhanced Data Type Facilities*」、ISO/IEC JTC1/SC22/WG5 N1378

- 「*Military Standard Fortran DOD Supplement to ANSI X3.9-1978*」、MIL-STD-1753 (米国、米国国防総省標準規格) XL Fortran では、Fortran 90 および Fortran 95 の標準規格に追加された拡張機能のみをサポートしていることに注意してください。
- *OpenMP Fortran Application Program Interface, Version 2.0, (Nov 2000).* (本書では OpenMP Fortran API という略式名で呼んでいます。)

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・サポート

XL Fortran Advanced Editionバージョン 9.1 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 V9.1 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、および実行時ライブラリーは、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 プロセッサーは、PowerPC のタイプの 1 つです。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 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

「*XL Fortran ランゲージ・リファレンス*」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。 **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。 **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。

ソース・コードの適合性検査

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。

- 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)
- 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。

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。 **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。 **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。 **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**、および **fort77** 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/9.1/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/9.1/xlf.cfg** をカスタマイズすることができます。

次の事項については、以下の項を参照してください。
- 構成ファイルについては、 15 ページの『構成ファイルのカスタマイズ』を参照。
- 呼び出しコマンドについては、 23 ページの『XL Fortran プログラムのコンパイル』を参照。
- コンパイラー・オプションについては、 57 ページの『XL Fortran コンパイラー・オプションの概要』および 85 ページの『XL Fortran コンパイラー・オプションの詳細記述』を参照。
- コンパイラーの戻りコードについては、 362 ページの『XL Fortran メッセージに関する情報』を参照。

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、 70 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 371 ページの『XL Fortran コンパイラー・リストについて』を参照してください。

-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・サポート

プログラムに対して、**gdb**、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。

293 ページの『XL Fortran プログラムの最適化』では、上記の機能を順に説明しています。

文書とオンライン・ヘルプ

XL Fortran は、以下の形式の製品資料を提供します。

- Readme ファイル
- インストール可能な man ページ
- HTML ベースの情報システム
- Portable Document Format (PDF) 文書

以下の資料の配置またはアクセス場所は次のとおりです。

README ファイル
README ファイルは、**/opt/ibmcmp/xlf/9.1/doc/$LANG/**、およびインストール CD のルート・ディレクトリーにあります。

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。

HTML ベースの情報
HTML ファイルで構成された情報システムは、**/opt/ibmcmp/xlf/9.1/doc/$LANG/html** ディレクトリーに提供されます。HTML 情報にアクセスするには、フレーム対応のブラウザーで **index.htm** ファイルを開きます。

PDF 文書
PDF ファイルは **/opt/ibmcmp/xlf/9.1/doc/$LANG/pdf** ディレクトリーにあります。 PDF ファイルは、Xpdf などの PDF ビューアーからダウンロードすることができます。

LANG は言語およびロケーションのコードを表します。たとえば、**en_US** は米国英語用の言語およびロケーションのコードです。

文書を表示するには、システム上で X Window システムが動作している必要があります。さらに、フレーム対応の HTML ブラウザー (Mozilla バージョン 1.0 以降など) などの適切なビューアーと PDF ビューアー (Xpdf など) が必要です。

「ユーザーズ・ガイド」と「ランゲージ・リファレンス」の重要トピック間の相互参照はリンクされています。指定の文書を表示中、リンクをクリックすると、他の文書内にリンクされたトピックにアクセスすることができます。他の文書内のリンクされた情報にアクセスするために、表示中の文書を閉じる必要はありません。

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

- http://www.ibm.com/software/awdtools/fortran/xlfortran/support の製品サポート・サイト。

XL Fortran のセットアップとカスタマイズ

本節では、あらゆるユーザーに合わせて XL Fortran 設定をカスタマイズし、XL Fortran を使用するようにユーザー・アカウントをセットアップする方法を説明します。本節は、インストール手順すべてを網羅しているわけではないので、その詳細についてはインストール手順を扱っている資料を参照してください。

さらに本節は、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/9.1/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知らせる必要のある情報が入っています。
2. コンパイラーに添付されている「*Installation Guide*」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/9.1/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/9.1/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/9.1/lib** と **/opt/ibmcmp/xlf/9.1/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。
- **/opt/ibmcmp/xlf/9.1/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、再分散可能実行時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/9.1/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/9.1/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/9.1/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/9.1/doc/ja_JP/html には、HTML 形式の XL Fortran 資料 (日本語) が入っています。

また、**/etc/opt/ibmcmp/xlf/9.1/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/9.1** 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。

環境変数の原則

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国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: 39 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、実行時ライブラリー検索パスを設定する必要があります。実行時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、指定の実行時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、実行時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 90 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。例については、 322 ページの『スクラッチ・ファイルの命名』を参照してください。

XLFUNIT_unit: 暗黙に接続されるファイルの名前の指定

暗黙に接続されるファイル、または **FILE=** 指定子なしにオープンされるファイルの名前を指定するには、まず実行時オプションの **unit_vars=yes** を指定し、次に **XLFUNIT_***unit* という形式の名前が付いた 1 つ以上の環境変数をファイル名に設定します。例については、 322 ページの『明示的な名前に接続されていないファイルの命名』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/9.1/xlf.cfg** を提供します。

シングルユーザー・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/9.1/bin/xlf95 /home/lisa/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。引き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。引き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。引き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、 232 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, ...
```

構成ファイルの実例

構成ファイルの例を以下に示します:

```
xlf95:    use          = DEFLT
          libraries    = -lxlf90,-lxlopt,-lxlomp_ser,-lxl,-lxlfmath
          libraries_64 = -lxlf90,-lxlopt,-lxlomp_ser,-lxl,-lxlfmath
          gcc_libs     = -ldl,-lrt,-lpthread,-lm,-lc,-lgcc
          gcc_libs_64  = -ldl,-lrt,-lpthread,-lm,-lc,-lgcc
          options      = -qfree=f90

* Alias for standard Fortran compiler
f95:      use          = DEFLT
          libraries    = -lxlf90,-lxlopt,-lxlomp_ser,-lxl,-lxlfmath
          libraries_64 = -lxlf90,-lxlopt,-lxlomp_ser,-lxl,-lxlfmath
          gcc_libs     = -ldl,-lrt,-lpthread,-lm,-lc,-lgcc
          gcc_libs_64  = -ldl,-lrt,-lpthread,-lm,-lc,-lgcc
          options      = -qfree=f90
          fsuffix      = f95
```

```
* Fortran 90 compiler
xlf90:    use          = DEFLT
          libraries    = -lxlf90,-lxlopt,-lxlomp_ser,-lxl,-lxlfmath
          libraries_64 = -lxlf90,-lxlopt,-lxlomp_ser,-lxl,-lxlfmath
          gcc_libs     = -ldl,-lrt,-lpthread,-lm,-lc,-lgcc
          gcc_libs_64  = -ldl,-lrt,-lpthread,-lm,-lc,-lgcc
          options      = -qxlf90=noautodealloc:nosignedzero,-qfree=f90

* Alias for Fortran 90 compiler
f90:      use          = DEFLT
          libraries    = -lxlf90,-lxlopt,-lxlomp_ser,-lxl,-lxlfmath
          libraries_64 = -lxlf90,-lxlopt,-lxlomp_ser,-lxl,-lxlfmath
          gcc_libs     = -ldl,-lrt,-lpthread,-lm,-lc,-lgcc
          gcc_libs_64  = -ldl,-lrt,-lpthread,-lm,-lc,-lgcc
          options      = -qxlf90=noautodealloc:nosignedzero,-qfree=f90
          fsuffix      = f90

* Original Fortran compiler
xlf:      use          = DEFLT
          libraries    = -lxlf90,-lxlopt,-lxlomp_ser,-lxl,-lxlfmath
          libraries_64 = -lxlf90,-lxlopt,-lxlomp_ser,-lxl,-lxlfmath
          gcc_libs     = -ldl,-lrt,-lpthread,-lm,-lc,-lgcc
          gcc_libs_64  = -ldl,-lrt,-lpthread,-lm,-lc,-lgcc
          options      = -qnozerosize,-qsave,-qalias=intptr,
                         -qposition=appendold,-qxlf90=noautodealloc:nosignedzero,
                         -qxlf77=intarg:intxor:persistent:noleadzero:gedit77:
                          noblankpad:oldboz:softeof

* Alias for original Fortran compiler
f77:      use          = DEFLT
          libraries    = -lxlf90,-lxlopt,-lxlomp_ser,-lxl,-lxlfmath
          libraries_64 = -lxlf90,-lxlopt,-lxlomp_ser,-lxl,-lxlfmath
          gcc_libs     = -ldl,-lrt,-lpthread,-lm,-lc,-lgcc
          gcc_libs_64  = -ldl,-lrt,-lpthread,-lm,-lc,-lgcc
          options      = -qnozerosize,-qsave,-qalias=intptr,
                         -qposition=appendold,
                         -qxlf90=noautodealloc:nosignedzero,-qxlf77=intarg:
                          intxor:persistent:noleadzero:gedit77:noblankpad:
                          oldboz:softeof

* Alias for original Fortran compiler, used for XPG4 compliance
fort77:   use          = DEFLT
          libraries    = -lxlf90,-lxlopt,-lxlomp_ser,-lxl,-lxlfmath
          libraries_64 = -lxlf90,-lxlopt,-lxlomp_ser,-lxl,-lxlfmath
          gcc_libs     = -ldl,-lrt,-lpthread,-lm,-lc,-lgcc
          gcc_libs_64  = -ldl,-lrt,-lpthread,-lm,-lc,-lgcc
          options      = -qnozerosize,-qsave,-qalias=intptr,-qposition=appendold,
                         -qxlf90=noautodealloc:nosignedzero,-qxlf77=intarg:intxor:
                          persistent:noleadzero:gedit77:noblankpad:oldboz:softeof

* xlf with links to thread-safe components
xlf_r:    use             = DEFLT
          libraries       = -lxlf90_r,-lxlopt,-lxlomp_ser,-lxl,-lxlfmath
          libraries_64    = -lxlf90_r,-lxlopt,-lxlomp_ser,-lxl,-lxlfmath
          smplibraries    = -lxlf90_r,-lxlopt,-lxlsmp,-lxl,-lxlfmath
          smplibraries_64 = -lxlf90_r,-lxlopt,-lxlsmp,-lxl,-lxlfmath
          gcc_libs        = -ldl,-lrt,-lpthread,-lm,-lc,-lgcc
          gcc_libs_64     = -ldl,-lrt,-lpthread,-lm,-lc,-lgcc
          options         = -qthreaded,-qnozerosize,-qsave,-qalias=intptr,
                            -qposition=appendold,-qxlf90=noautodealloc:nosignedzero,
                            -qxlf77=intarg:intxor:persistent:noleadzero:gedit77:
                             noblankpad:oldboz:softeof

* xlf90 with links to thread-safe components
xlf90_r:  use             = DEFLT
          libraries       = -lxlf90_r,-lxlopt,-lxlomp_ser,-lxl,-lxlfmath
          libraries_64    = -lxlf90_r,-lxlopt,-lxlomp_ser,-lxl,-lxlfmath
```

```
          smplibraries    = -lxlf90_r,-lxlopt,-lxlsmp,-lxl,-lxlfmath
          smplibraries_64 = -lxlf90_r,-lxlopt,-lxlsmp,-lxl,-lxlfmath
          gcc_libs        = -ldl,-lrt,-lpthread,-lm,-lc,-lgcc
          gcc_libs_64     = -ldl,-lrt,-lpthread,-lm,-lc,-lgcc
          options         = -qxlf90=noautodealloc:nosignedzero,-qfree=f90,
                            -qthreaded

* xlf95 with links to thread-safe components
xlf95_r:  use             = DEFLT
          libraries       = -lxlf90_r,-lxlopt,-lxlomp_ser,-lxl,-lxlfmath
          libraries_64    = -lxlf90_r,-lxlopt,-lxlomp_ser,-lxl,-lxlfmath
          smplibraries    = -lxlf90_r,-lxlopt,-lxlsmp,-lxl,-lxlfmath
          smplibraries_64 = -lxlf90_r,-lxlopt,-lxlsmp,-lxl,-lxlfmath
          gcc_libs        = -ldl,-lrt,-lpthread,-lm,-lc,-lgcc
          gcc_libs_64     = -ldl,-lrt,-lpthread,-lm,-lc,-lgcc
          options         = -qfree=f90,-qthreaded


* Common definitions
DEFLT:    xlf             = /opt/ibmcmp/xlf/9.1/exe/xlfentry
          crt             = /usr/lib/crt1.o
          crtp            = /usr/lib/crti.o
          crte            = /usr/lib/crtn.o
          crtbegin        = /usr/lib/gcc-lib/powerpc-suse-linux/3.3.3/crtbegin.o
          crtend          = /usr/lib/gcc-lib/powerpc-suse-linux/3.3.3/crtend.o
          crtsavres       = /usr/lib/gcc-lib/powerpc-suse-linux/3.3.3/crtsavres.o
          mcrt            = /usr/lib/gcrt1.o
          gcrt            = /usr/lib/gcrt1.o
          crt_64          = /usr/lib64/crt1.o
          crtp_64         = /usr/lib64/crti.o
          crte_64         = /usr/lib64/crtn.o
          crtbegin_64     = /usr/lib/gcc-lib/powerpc-suse-linux/3.3.3/64/crtbegin.o
          crtend_64       = /usr/lib/gcc-lib/powerpc-suse-linux/3.3.3/64/crtend.o
          crtsavres_64    = /usr/lib/gcc-lib/powerpc-suse-linux/3.3.3/64/crtsavres.o
          mcrt_64         = /usr/lib64/gcrt1.o
          gcrt_64         = /usr/lib64/gcrt1.o
          include_32      = -I/opt/ibmcmp/xlf/9.1/include
          include_64      = -I/opt/ibmcmp/xlf/9.1/include64
          dis             = /opt/ibmcmp/xlf/9.1/exe/dis
          code            = /opt/ibmcmp/xlf/9.1/exe/xlfcode
          hot             = /opt/ibmcmp/xlf/9.1/exe/xlfhot
          ipa             = /opt/ibmcmp/xlf/9.1/exe/ipa
          bolt            = /opt/ibmcmp/xlf/9.1/exe/bolt
          defaultmsg      = /opt/ibmcmp/xlf/9.1/msg/en_US
          as              = /usr/bin/as
          as_64           = /usr/bin/as
          ld              = /usr/bin/ld
          ld_64           = /usr/bin/ld
          cppoptions      = -C
          cpp             = /opt/ibmcmp/xlf/9.1/exe/cpp
          dynlib          = -dynamic-linker,/lib/ld.so.1
          dynlib_64       = -dynamic-linker,/lib64/ld64.so.1
          libdirs         = -L/opt/ibmcmp/xlsmp/1.3/lib,-L/opt/ibmcmp/xlf/9.1/lib,
                            -R/opt/ibmcmp/xlsmp/1.3/../../lib,
                            -R/opt/ibmcmp/xlf/9.1/../../lib
          libdirs_64      = -L/opt/ibmcmp/xlsmp/1.3/lib64,-L/opt/ibmcmp/xlf/9.1/lib64,
                            -R/opt/ibmcmp/xlsmp/1.3/../../lib64,
                            -R/opt/ibmcmp/xlf/9.1/../../lib64
          gcc_path        = /usr
          gcc_path_64     = /usr
          gcc_libdirs     = -L/usr/lib/gcc-lib/powerpc-suse-linux/3.3.3,
                            -L/usr/powerpc-suse-linux/lib,
                            -L/usr/lib,-L/lib,
          gcc_libdirs_64  = -L/usr/lib/gcc-lib/powerpc-suse-linux/3.3.3/64,
                            -L/usr/lib/gcc-lib/powerpc-suse-linux/3.3.3,
                            -L/usr/powerpc-suse-linux/lib,
                            -L/usr/lib,-L/lib64,
```

```
                  -L/usr/lib64
  xlcmp_path     = /opt/ibmcmp/xlf/9.1
  bigdata        = -T/opt/ibmcmp/xlf/9.1/exe/elf32ppclinux.x
  bigdata_shr    = -T/opt/ibmcmp/xlf/9.1/exe/elf32ppclinux_shr.x
  options        = -qnoenablevmx
modes_configure  = 32_64
crt2 = NULL
crt2_64 = NULL
xlf_path = /opt/ibmcmp/xlf/9.1
```

XL Fortran は、ライブラリー **libxlf90_t.so** に加え、 **libxlf90_r.so** を提供します。ライブラリー **libxlf90_r.so** は部分的スレッド・サポート実行時ライブラリーである **libxlf90_t.so** のスーパーセットです。ファイル **xlf.cfg** は、**xlf90_r**、 **xlf95_r**、 および **xlf_r** コマンドの使用時に **libxlf90_r.so** に自動的にリンクするように設定されています。

関連情報: 93 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-9.1 | 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、「*IBM XL Fortran Advanced Edition V9.1 for Linux インストール・ガイド*」を参照してください。

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:

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、12 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. Fortran プログラムの書き方についてさらに学ぶには、「*XL Fortran ランゲージ・リファレンス*」を参照してください。

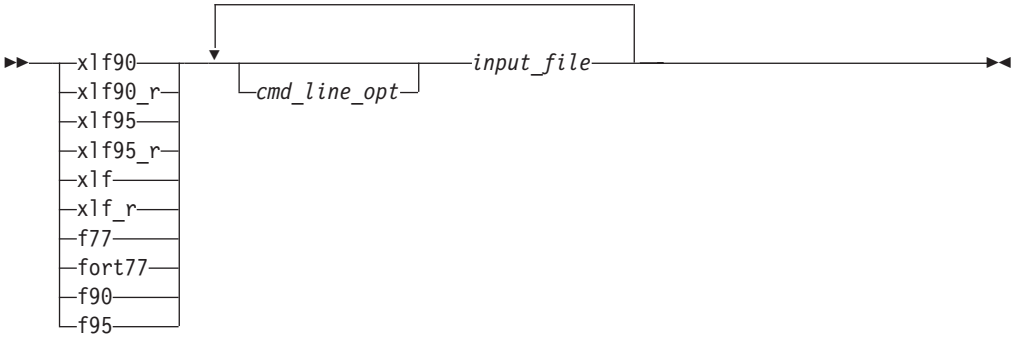
XL Fortran ソース・ファイルの編集

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ている場合、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。

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

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/9.1/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・ファイルのリンクおよびバインドを行います。

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 1. 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/9.1/bin	Fortran 90	libxlf90.so
xlf90_r	/opt/ibmcmp/xlf/9.1/bin	スレッド・セーフ Fortran 90	libxlf90_r.so
xlf95	/opt/ibmcmp/xlf/9.1/bin	Fortran 95	libxlf90.so
xlf95_r	/opt/ibmcmp/xlf/9.1/bin	スレッド・セーフ Fortran 95	libxlf90_r.so
xlf	/opt/ibmcmp/xlf/9.1/bin	FORTRAN 77	libxlf90.so
xlf_r	/opt/ibmcmp/xlf/9.1/bin	スレッド・セーフ FORTRAN 77	libxlf90_r.so
f77 あるいは **fort77**	/opt/ibmcmp/xlf/9.1/bin	FORTRAN 77	libxlf90.so
f90	/opt/ibmcmp/xlf/9.1/bin	Fortran 90	libxlf90.so
f95	/opt/ibmcmp/xlf/9.1/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、ライブラリー **libxlf90_t.so** に加え、 **libxlf90_r.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。ファイル **xlf.cfg** は、**xlf90_r**、 **xlf95_r**、 および **xlf_r** コマンドの使用時に、 **libxlf90_r.so** に自動的にリンクするようにセットアップされています。

libxlf90_t.so は、部分的スレッド・サポート実行時ライブラリーです。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、 I/O オペレーションを実行したり、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。スレッドの同期によるオーバーヘッドが **libxlf90_r.so** で生じないようにするには、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** を使用することができます。

マルチスレッドの実行可能プログラムを複数の Fortran スレッドとバインドする場合は、 ルーチンにリンクする目的で、**-lxlf90_t** または **-lxlf90** の代わりに **-lxlf90_r** をコマンド行に指定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、 **xlf90** および **xlf90_r** コマンドのセットと、 **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、**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、**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。完全コンパイルを必要とする場合は、次のコンパイラー・オプション (およびサブオプション) のいずれかを指定してコンパイルしてください。

```
-qnodirective -qnoescape -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"
```

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** コマンドと **xlf**、**xlf90_r** コマンドと **xlf90** コマンド、また **xlf95_r** コマンドと **xlf95** コマンドは、それぞれほとんど同じです。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

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

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 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。 **f90** および **f95** 呼び出しを使用するとき、**.f** はデフォルトでは許可されません。 **.f90** および **.f95** を使用してください。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: 33 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。

15 ページの『構成ファイルのカスタマイズ』および 232 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: 81 ページの『リンクを制御するオプション』および 36 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 (15 ページの『構成ファイルのカスタマイズ』および 232 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 (15 ページの『構成ファイルのカスタマイズ』および 232 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/9.1/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、 15 ページの『構成ファイルのカスタマイズ』および 93 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: 95 ページの『**-I** オプション』 と 188 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 つまたは複数の隠しファイルに格納されます。

関連情報: 196 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 103 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 90 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: 90 ページの『-c オプション』および 36 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、 103 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: 258 ページの『-S オプション』および 36 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、
103 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 つまたは複数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: 70 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、 188 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: 33 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 92 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: 196 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 85 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 つまたは複数の文字 (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
```

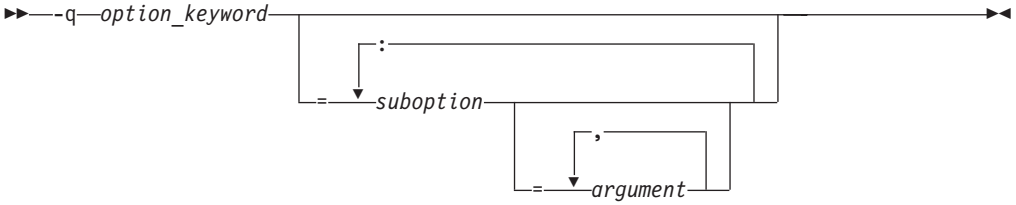
(例外がいくつかあります。たとえば、**-pg**。これは単一オプションで、**-p -g** と同じではありません。)

フラグの中には、引き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。最後に引き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、その後に続く引き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き数を分離するには等号を使用し、引き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** ディレクティブ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・ディレクティブによって、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・ディレクティブ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・ディレクティブ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・ディレクティブで指定するオプション設定は、ステートメント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・ディレクティブは、通常、コンパイル単位の最初のステートメント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、コマンドの引き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/9.1/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/9.1/xlf.cfg** の **xlf95** スタンザに以下の行

  ```
  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: 264 ページの『-W オプション』および 15 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/9.1/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/9.1/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 つまたは複数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、116 ページの『-qarch オプション』を参照してください。 100 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** オプションを指定すると行われません。 **#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 ランゲージ・リファレンス*」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: 92 ページの『-d オプション』、 259 ページの『-t オプション』、 264 ページの『-W オプション』、および 15 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** ステートメントの変更や、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
```

プリプロセスの問題の回避

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/9.1/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、38 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。

実行可能オブジェクト・ファイルのパス名とファイル名、実行時の引き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。 SMP 以外のプログラムの場合、通常必要とされる XL Fortran ライブラリーは **libxlf90.so** だけです。 SMP プログラムの場合、通常は少なくとも **libxlf90_r.so** および **libxlsmp.so** ライブラリーが必要です。 **libxlfpmt*.so** および **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だけです。

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: 36 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 がサポートする実行時ライブラリー

POSIX のスレッド・サポートを使用して接続された実行時ライブラリーが 2 種類あります。 **libxlf90_r.so** ライブラリーは、スレッド・セーフ・バージョンの Fortran 実行時ライブラリーです。 **libxlsmp.so** ライブラリーは、SMP 実行時ライブラリーです。

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/9.1/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
```

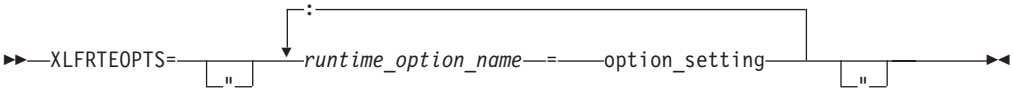
関連情報: 12 ページの『各国語サポートのための環境変数』を参照してください。

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・コールによって設定することができます。 2 つの環境変数 **XLFRTEOPTS** および **XLSMPOPTS** のどちらか を使用して、すべての 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** ステートメントが実行時エラー・メッセージを出す必要がある。
- **DEALLOCATE** ステートメントが実行時エラー・メッセージを出す必要がある。
- **MATMUL** プロシージャーのマルチスレッド・インプリメンテーションが実行される。

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

SETRTEOPTS プロシージャー (「*XL Fortran ランゲージ・リファレンス*」で定義されています) は、環境変数 **XLFRTEOPTS** と同じ名前値のペアを含んでいる単一ストリング引き数を受け入れます。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・コール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 の実行時ライブラリーが、I/O 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** ステートメントや **WRITE** ステートメント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** ステートメントは、I/O バッファーに必要以上のデータを読み込む場合があり、結果として次のデータを読み取るはずの、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 の実行時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** ステートメント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** ステートメントによるデータの書き込みも、ステートメント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 実行時ライブラリーは、接続されている各論理装置ごとに I/O バッファーを保持します。実行時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 実行時ライブラリーは、事前に接続されている各論理装置 (0、5、および 6) ごとに I/O 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 I/O バッファーはすべて保持します。実行時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 実行時ライブラリーは、どの論理装置にも I/O バッファーを保持しません。非同期 I/O を実行する Fortran プログラム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 ランゲージ・リファレンス*」の『*データ転送ステートメント*』を参照してください。
IOSTAT 値に関する詳細は、「*XL Fortran ランゲージ・リファレンス*」の『*条件および IOSTAT 値*』を参照してください。

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

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

cpu_time_type のサブオプションは、以下のとおりです。

usertime
プロセスのユーザー時間を戻します。

systime
プロセスのシステム時間を戻します。

alltime プロセスのユーザーおよびシステム時間の合計を戻します。

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

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

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

default_recl={64 | 32}
RECL= 指定子なしでオープンされた順次ファイル用のデフォルトのレコード・サイズを決定することができます。サブオプションは以下のとおりです。

64 デフォルトのレコード・サイズとして 64 ビット値を使用します。

32 デフォルトのレコード・サイズとして 32 ビット値を使用します。

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 ランゲージ・リファレンス*」の **OPEN** ステートメント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=** を持つ 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、これらのステートメントのいずれか

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 ドラフト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 ランゲージ・リファレンス*」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、実行時エラー・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** ステートメントで脱落している) 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、実行時のエラー・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、実行時のエラー・メッセージは作成されません。しかし、**langlvl=90std** を指定すると、実行時のエラー・メッセージが作成されます。

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

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

同じプログラム内の多重接続が許可されるのは、ディスク・ドライブなどのランダム・アクセス・デバイス上にあるファイルの場合だけです。次のような場合は、同じプログラム内の多重接続は許可されていません。
- 書き込み専用で接続されているファイル (**ACTION='WRITE'**)
- 非同期 I/O
- 順次アクセス装置 (パイプ、端末、ソケットなど) 上のファイル)

ファイルに損傷を与えないようにするために、以下の点に注意してください。
- 同じファイルに対する 2 度目の **OPEN** ステートメントおよび後続する **OPEN** ステートメントが許可されるのは読み取りの場合だけです。
- もともと入力と出力の両用でファイルがオープン (**ACTION='READWRITE'**) された場合、最初の **OPEN** ステートメント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** ステートメント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** ステートメントを使用して、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 ランゲージ・リファレンス*」の『*名前リストの形式設定*』を参照してください。

nlwidth=*record_width*

デフォルト時には、**NAMELIST** 書き込みステートメントは、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** 引き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** 引き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、指定したユニット番号へ関連付けたいファイルの名前を指定します。 例については、 322 ページの『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。例については、 322 ページの『明示的な名前に接続されていないファイルの命名』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}
I/O 操作、**RANDOM_SEED** 呼び出し、および **ALLOCATE** または **DEALLOCATE** ステートメントの実行中に、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。

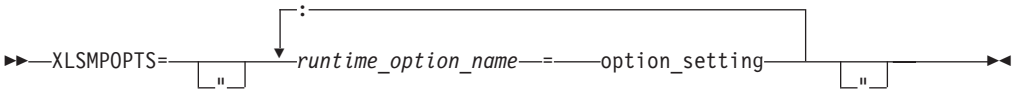
```
# Basic format
  XLFRTEOPTS=cnverr=yes:xrf_messages=no
  export XLFRTEOPTS

# 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.
```

XLSMPOPTS 環境変数

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。 **XLSMPOPTS** は、次の **bash** コマンド形式を使用して宣言できます。



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、**XLSMPOPTS** 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。

次の実行時オプションは、環境変数 **XLSMPOPTS** で指定することができます。

schedule
実行時にデフォルトとして使用されるスケジューリング・タイプとチャンク・サイズを選択します。この方法で指定されたスケジューリング・タイプは、コンパイル時にスケジューリング・タイプが指定されていないループにしか使用されません。

使用するスケジューリング・タイプとチャンク・サイズに応じて、さまざまな方法でスレッドに作業が割り当てられます。スケジューリング・タイプおよびそれらが作業の割り当てに与える影響について、以下に簡単に説明します。

dynamic *または* **guided**
実行時ライブラリーはスレッドの並列作業を「先着順実行」の原則に基づいて動的にスケジュールします。残りの作業の「チャンク」は、すべての作業が割り当てられるまで、使用可能なスレッドに割り当てられます。スリープ状態のスレッドには作業は割り当てられません。

static 作業のチャンクは「ラウンドロビン」方式でスレッドに割り当てられます。作業は、活動状態とスリープ状態の両方のスレッドに割り当てられます。システムは、作業に割り当てられるスリープ状態のスレッドを活動化して、割り当て作業を完了する必要があります。

affinity
実行時ライブラリーは、最初に反復を *number_of_threads* 個の区画に分割します。この区画に含まれる反復の数は次のとおりです。

```
CEILING(number_of_iterations / number_of_threads)
```

これらの区画は各スレッドに割り当てられます。そして、反復のチャンクにさらに分割されます。スレッドがスリープ状態の場合は、他の活動状態のスレッドが、割り当てられている作業区画を完了します。

チャンクをどの程度細分化するかによって、オーバーヘッドとロード・バランシングのバランスは異なってきます。このオプションの構文は **schedule=***suboption* です。 suboptions は次のように定義されます。

affinity[=*n***]** すでに説明したとおり、ループの反復は最初に区画に分割され、次いでその区画はスレッドに事前に割り当てられます。各区画は、*n* 回の反復を含むチャンクにさらに分割されます。 *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。

スレッドが使用可能になると、事前に割り当てられている区画から次のチャンクが選ばれます。その区画にチャンクがなくなると、別のスレッドに事前に割り当てられている区画から次に使用可能なチャンクが選ばれます。

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。

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* を指定していない場合は、*n* のデフォルト値は 1 回の反復になります。

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。

static[=*n*] ループの反復は、*n* 反復を含むチャンクに分割されます。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。この方式は、ブロック巡回スケジューリングとして知られています。 *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。

schedule を指定していない場合は、デフォルトが **schedule=static** に設定され、その結果ブロック・スケジューリングが行われます。

関連情報: 詳細については、「*XL Fortran* ランゲージ・リファレンス」の **SCHEDULE** ディレクティブに関する説明を参照してください。

並列実行オプション

3 つの並列実行オプション、 **parthds**、**usrthds**、および **stack** について次に説明します。

parthds=*num* **-qsmp** オプションを使ってコンパイルされたコードを並列実行するために使用するスレッドの数 (*num*) を指定します。デフォルト時には、この数はオンライン・プロセッサーの数と等しくなります。アプリケーションによっては、多くてもプロセッサーの最大数のスレッドしか使用できないものがあります。また、プロセッサーの数を超えるスレッドを使用するとパフォーマンス向上を達成できるアプリケーションもあります。

このオプションを使用すると、実行スレッドの数を完全に制御することができます。 **-qsmp**

を指定しない場合、*num* のデフォルト値は 1 です。 **-qsmp** を指定する場合は、マシン上のオンライン・プロセッサーの数になります。詳細については、「*XL Fortran ランゲージ・リファレンス*」の『**NUM_PARTHDS**』組み込み関数を参照してください。

usrthds=*num*

コードが明示的に作成されると予測されるスレッドの最大数 (*num*) を指定します (コードが明示的にスレッドを作成する場合)。 *num* のデフォルト値は 0 です。 詳細については、「*XL Fortran ランゲージ・リファレンス*」の『**NUM_PARTHDS**』組み込み関数を参照してください。

stack=*num*

スレッドのスタックが必要とするスペースの最大量のバイト数 (*num*) を指定します。 *num* のデフォルト値は 4194304 です。

stack=*num* を、受け入れ可能な上限におさまるように設定します。*num* は、32 ビット・モードでは最大 256 MB まで、64 ビット・モードではシステム・リソースによって決められている限界までにすることができます。上限を超えるアプリケーションでは、セグメンテーションの障害が発生する可能性があります。

パフォーマンスの調整オプション

スレッドが作業を完了し、新しい作業が残っていなければ、そのスレッドは「使用中待機」状態か「スリープ」状態に入ります。「使用中待機」状態では、スレッドは短いループで実行し続け、残っている新しい作業を探します。この状態は反応が速い一方、システムの総合的な使用効率に悪影響を与えます。スレッドがスリープ状態の場合は、別のスレッドがそのスリープ状態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は完全に中断されます。この状態では、システムの使用効率はよくなりますが、アプリケーションに余分なオーバーヘッドがかかります。

xlsmp 実行時ライブラリー・ルーチンは、作業を待機する方法として、「使用中待機」状態と「スリープ」状態の両方を使用します。 **spins**、**yields**、および **delays** オプションを使用して、このような状態を制御することができます。

使用中待機状態の作業を検索している間は、スレッドは作業キューを最高 *num* 回まで繰り返しスキャンします。ここで *num* は、**spins** オプションに指定した値です。指定されたスキャン中に作業が見つからない場合、スレッドは遅延ループを *num* 回実行して意図的にサイクルを浪費します。ここで *num* は、**delays** オプションに指定された値です。この遅延ループは、意味のない単一の反復から成っています。遅延ループにかかる実際の時間の長さはプロセッサーによって異なります。値 **spins** を超過してもまだ作業が見つからない場合は、スレッドは現行のタイム・スライス (プロセッサーがそのスレッドに割り当てた時間) を他のスレッドに譲渡します。スレッドはタイム・スライスを最高 *num* 回まで譲渡します。ここで、*num* は **yields** オプションに指定した数です。この値 *num* を超過すると、スレッドはスリープ状態になります。

要約すると、作業を検索するための順序付けられた方法は次のステップで構成されています。

1. 作業キューを **spins** 回までスキャンします。スキャン中に作業が見つからない場合は、**delays** 回ループしてから、新しい検索を開始します。
2. 作業が見つからなかった場合は、現行のタイム・スライスを譲渡します。
3. 上記のステップを **yields** 回繰り返します。
4. 作業が見つからなかった場合は、スリープ状態になります。

これらのオプションを指定する構文は次のとおりです。

spins[=*num***]** *num* は譲渡前のスピンの数です。 **spins** のデフォルト値は **100** です。

yields[=*num***]** *num* はスリープ前の譲渡の回数です。 **yields** のデフォルト値は **10** です。

delays[=*num***]** *num* は使用中待機状態中の遅延の数です。 **delays** のデフォルト値は 500 です。

spins と **yields** の特殊値はゼロです。ゼロを使えば、強制的に完全な使用中待機状態にすることができます。通常、専用システムのベンチマーク・テストでは、両方のオプションをゼロに設定します。しかし、それらを個別に設定すると、他の効果を持たせることができます。

たとえば、専用の 8 ウェイ SMP では、これらのオプションを次のように設定します。

```
parthds=8 : schedule=dynamic=10 : spins=0 : yields=0
```

この場合、CPU ごとに 1 スレッドという結果になります。各スレッドにはそれぞれ 10 回の反復から成るチャンクが割り当てられ、即時実行する作業がなければ使用中待機状態になります。

動的プロファイル作成を使用可能および制御するためのオプション

動的プロファイル作成を使用して、プログラムのループを並列化する際のコンパイラーの判断を再評価することができます。これには、**parthreshold**、**seqthreshold**、および **profilefreq** の 3 つのオプションを使用します。

parthreshold=*num*

その時間を下回った場合、各ループがシリアルに実行される時間をミリ秒で指定します。**parthreshold** を 0 に設定すると、コンパイラーで並列化されている各ループは、並列処理で実行されます。デフォルトの設定は、0.2 ミリ秒です。これは、1 つのループを並列に実行する時間が 0.2 ミリ秒以下の場合に、それをシリアルで処理するということです。

一般には、**parthreshold** を並列化のオーバーヘッドと等しくなるように設定します。並列化されたループの計算量が非常に少なく、並列化の設定においてこれらのループを実行するのにかかる時間が大部分を占める場合は、これらのループを順次実行した方がパフォーマンスはよくなります。

seqthreshold=*num*

動的プロファイラーによってあらかじめ直列化されたループが、その時間を上回った場合に並列ループに戻るようにするための時間を、ミリ秒で指定します。デフォルトの設定は、5 ミリ秒です。これは、1 つのループをシリアルに実行する時間が 5 ミリ秒を超える場合に、それを並列で処理するということです。

seqthreshold は、**parthreshold** とは逆の動作をします。

profilefreq=*num*

動的プロファイラーが、並列実行とシリアル実行のどちらが適切かを判別するために、ループに行く頻度を指定します。プログラムのループは、データに依存する可能性があります。動的プロファイル作成のパスを使ってシリアルに実行することを選択したループが、入力データが異なる場合はループの後続の処理で並列処理によって恩恵を受ける可能性があります。それで、これらのループを定期的に調べて、実行時に並列ループをシリアル化する判断を再評価しなければなりません。

このオプションで指定できる値は、0 から 32 の数字です。 **profilefreq** を 0 から 32 のいずれかに設定した場合、次の処理が行われます。

- **profilefreq** が 0 の場合、他の設定にかかわらず、すべてのプロファイル作成がオフになります。プロファイル作成の結果生じるオーバーヘッドは、なくなります。
- **profilefreq** が 1 の場合、コンパイラーが自動的に並列化することになるループは、それらのループが実行されるたびにモニターされます。
- **profilefreq** が 2 の場合、コンパイラーが自動的に並列化することになるループは、それらのループが 2 回実行されるたびに 1 回モニターされます。
- **profilefreq** が 2 以上 32 以下の場合、各ループは *n* 回実行されるたびにモニターされます。
- **profilefreq** が 33 以上の場合は、32 が指定されたものと見なされます。

動的プロファイル作成は、ユーザー指定の並列ループ (たとえば、**PARALLEL DO** ディレクティブを指定したループ) には適用できませんのでご注意ください。

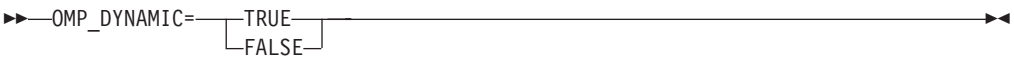
OpenMP 環境変数

以下の環境変数は、OpenMP 標準に含まれており、並列コードの実行を制御できます。

注: **XLSMPOPTS** 環境変数と OpenMP 環境変数を両方とも指定した場合、OpenMP 環境変数が優先されます。

OMP_DYNAMIC 環境変数

OMP_DYNAMIC 環境変数は、並列領域の実行に使用できるスレッドの数を動的に調整できるようにしたり、できないようにしたりします。この環境変数の構文は次のとおりです。



この環境変数を **TRUE** に設定すれば、実行時環境は並列領域を実行するときに使用できるスレッドの数を調整できるため、システム・リソースをより効率的に使用できるようになります。この環境変数を **FALSE** に設定した場合、動的な調整は行えません。

OMP_DYNAMIC のデフォルト値は **TRUE** です。そのため、決まった数のスレッドを正確にコードが実行する必要がある場合は、動的スレッド調整を使用不可にしなければなりません。

omp_set_dynamic サブルーチンは、 **OMP_DYNAMIC** 環境変数よりも優先されます。

OMP_NESTED 環境変数

OMP_NESTED 環境変数は、ネストされた並列処理を行えるようにしたり、行えないようにしたりします。この環境変数の構文は次のとおりです。

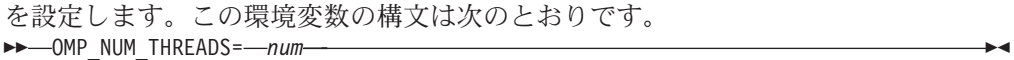
この環境変数を **TRUE** に設定した場合、ネストされた並列処理を行うことができます。つまり、実行時環境は追加のスレッドを展開して、ネストされた並列領域で使用するためのスレッド・チームを形成できます。この環境変数を **FALSE** に設定した場合、ネストされた並列処理は行えません。

OMP_NESTED のデフォルト値は **FALSE** です。

omp_set_nested サブルーチンは、 **OMP_NESTED** 環境変数よりも優先されます。

OMP_NUM_THREADS 環境変数

OMP_NUM_THREADS 環境変数は、プログラムが実行時に使用するスレッドの数を設定します。この環境変数の構文は次のとおりです。



num スレッド数の動的調整が可能な場合に、使用できるスレッドの最大数を指定

します。スレッド数の動的調整が不可能な場合、 **OMP_NUM_THREADS** の値は使用できるスレッドの実数になります。これは、正のスカラー整数でなければなりません。

プログラムが実行時に使用できるスレッドのデフォルト数は、マシン上のオンライン・プロセッサーの数です。

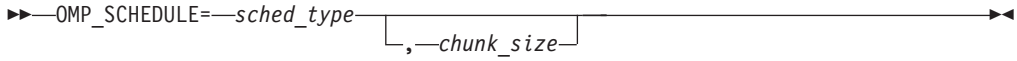
XLSMPOPTS 環境変数の **PARTHDS** サブオプションと **OMP_NUM_THREADS** 環境変数を両方とも使用してスレッドの数を指定した場合、**OMP_NUM_THREADS** 環境変数のほうが優先されます。 **omp_set_num_threads** サブルーチンは、 **OMP_NUM_THREADS** 環境変数よりも優先されます。

OMP_NUM_THREADS 環境変数を設定する方法について、以下の例で示します。

```
export OMP_NUM_THREADS=16
```

OMP_SCHEDULE 環境変数

OMP_SCHEDULE 環境変数は、スケジュール・タイプが **RUNTIME** の **PARALLEL DO** ディレクティブと作業共用 **DO** ディレクティブに適用されます。この環境変数の構文は次のとおりです。



sched_type
: **DYNAMIC**、**GUIDED**、または **STATIC** のいずれかです。

chunk_size
: これはチャンク・サイズを表す正のスカラー整数です。

この環境変数は、 **PARALLEL DO** ディレクティブおよび作業共用 **DO** ディレクティブのスケジュール・タイプが **RUNTIME** 以外の場合は無視されます。

コンパイル時にディレクティブによってスケジュール・タイプを指定しなかったか、あるいは実行時に **OMP_SCHEDULE** 環境変数、または **XLSMPOPTS** 環境変数の **SCHEDULE** オプションによってスケジュール・タイプを指定しなかった場合、デフォルトのスケジュール・タイプは **STATIC** となります。また、デフォルトのチャンク・サイズは、最初の N - 1 スレッドについては次のように設定されます。

```
chunk_size = ceiling(Iters/N)
```

N 番目のスレッドについては、次のように設定されます。ここで、N はスレッドの合計数、Iters は **DO** ループ内の反復の合計数です。

```
chunk_size = Iters - ((N - 1) * ceiling(Iters/N))
```

XLSMPOPTS 環境変数の **SCHEDULE** オプションと **OMP_SCHEDULE** 環境変数を両方とも指定した場合、**OMP_SCHEDULE** 環境変数が優先されます。

OMP_SCHEDULE 環境変数を設定する方法について、以下の例で示します。

```
export OMP_SCHEDULE="GUIDED,4"
export OMP_SCHEDULE="DYNAMIC"
```

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、 12 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/BPT トラップ」メッセージが送出されます。

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

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 **-qfltrap** 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: 89 ページの『-C オプション』、 154 ページの『-qfltrap オプション』、および 219 ページの『-qsigtrap オプション』を参照してください。

これらの例外の詳細については、 284 ページの『浮動小数点演算例外の検出とトラッピング』を参照してください。また、例外ハンドラーのリストについては、 286 ページの『浮動小数点状況および制御レジスターの制御』を参照してください。

XL Fortran コンパイラー・オプションに関する参照事項

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。
- 85 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・ディレクティブ **@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」節と概要表の「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: 26 ページの『XL Fortran 入力ファイル』および 60 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。

72 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 2. 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/9.1/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。	95
-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。	131

表 2. 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**	133
-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。	138
-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**、**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。	150

表 2. 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・コードが自由形式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。**	156
-qmbcs -qnombcs	MBCS NOMBCS	文字リテラル定数、ホレリス定数、H 編集記述子、文字列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**	185
-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**	260
-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。	232

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、コンパイルで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: 28 ページの『XL Fortran 出力ファイル』および 58 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 3. 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。	92

表 3. 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	103
-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。	188

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: 293 ページの『XL Fortran プログラムの最適化』を参照してください。

80 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 4. 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**	100
-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。	104
-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。	105

表 4. 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: xlf90**、**xlf90_r**、**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。	109
-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**	113
-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。 Y-HPC の場合、auto および ppc970 のみが有効です。そうでない場合、選択項目は com、ppc、ppcgr、ppc64、ppc64gr、ppc64grsq、rs64b、rs64c、pwr3、pwr4、pwr5、および ppc970 です。 **デフォルト:** Y-HPC の場合は **-qarch=ppc970** で、他のすべての場合は **-qarch=ppc64grsq**。	116
-qassert={ 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**	120

表 4. 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。	126
-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**	132
-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**	141
-qenablevmx -qnoenablevmx		コンパイラーに対して、VMX (Vector Multimedia eXtension) 命令を生成するように指示します。 **-qhot=simd** および **-qarch=ppc970** の各コンパイラー・オプションも指定する必要があります。 **デフォルト: -qnoenablevmx**	144
-qessl		Fortran 90 組み込みプロシージャーの代わりに 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**	146

表 4. 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
-qhot[=*suboptions*] -qnohot	HOT[=*suboptions*] NOHOT	**-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。 **-qhot** 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。 **デフォルト: -qnohot**	159
-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。	168
-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**	175
-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=2048**。 **-O3**、**-O4**、および **-O5** では、**-qmaxmem=-1**。	183
-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**	193

表 4. 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
-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。	196
-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**	206
-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**	216
-qshowpdf -qnoshowpdf		追加の呼び出しとブロック・カウント・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**	218
-qsmallstack[=*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。 **デフォルト: -qnosmallstack**	220
-qsmp[=*suboptions*] -qnosmp		**xlf_r**、**xlf90_r**、または **xlf95_r** とともに使用され、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**	221
-NS*bytes* -qSPILLsize=*bytes*	SPILLsize (*bytes*)	内部プログラムのストレージ域のサイズを指定します。 **デフォルト:** -NS512	99

表 4. 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
-qstrict -qnostrict	STRICT NOSTRICT	-O3、-O4、-O5、-qhot、および -qipa 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。	229
-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**	230
-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**	231
-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。	238

表 4. 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。 Y-HPC の場合、auto および ppc970 のみが有効です。他のすべての場合、auto、rs64b、rs64c、pwr3、pwr4、pwr5、または ppc970 の設定が有効です。 **デフォルト:** Y-HPC の場合は **-qtune=ppc970** で、他のすべての場合は **-qtune=pwr4** になります。	239
-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**	242
-qunwind -qnounwind	UNWIND NOUNWIND	プロシージャー呼び出し時に不揮発性レジスターの保管および復元を行うためのデフォルトの動作を指定します。 **デフォルト: -qunwind**	244
-qversion -qnoversion		呼び出し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**	245
-qzerosize -qnozerosize	ZEROSIZE NOZEROSIZE	サイズがゼロの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**、**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)。	257

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、 361 ページの『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。

- 86 ページの『-# オプション』、 262 ページの『-v オプション』、および 263 ページの『-V オプション』
- 109 ページの『-qalias オプション』
- 131 ページの『-qci オプション』
- 193 ページの『-qobject オプション』
- 212 ページの『-qreport オプション』
- 227 ページの『-qsource オプション』

表 5. 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**	89
-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**	91
-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**	94
-qflttrap [=*suboptions*] -qnoflttrap	FLTTRAP [(*suboptions*)] NOFLTTRAP	実行時に検出する浮動小数点演算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**	154

表 5. 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
-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。	157
-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**	158
-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。	163
-qlanglvl={ 77std | 90std | 90pure | 95std | 95pure | extended}	LANGLVL({ 77STD | 90STD | 90PURE | 95STD | 95PURE | 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**	176

表 5. 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**	214
-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。	219
-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。	237
-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。	247
-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**	254

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。

67 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 6. 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。	86
-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**	121
-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**	151
-qlist -qnolist	LIST 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**	180
-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**	181
-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:** 次のいずれかが指定された場合にリストが作成されます。 -qattr、 -qlist、 -qlistopt、 -qphsinfo、 -qreport、 -qsource、または -qxref。	190
-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**	200

表 6. 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
-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**	212
-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**	227
-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。	233
-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**	256
-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 つまたは複数の **.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。	258
-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。	262
-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。	263

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: 379 ページの『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。 283 ページの『他のシステムの浮動小数点結果の再現』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、 80 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。

156 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 7. 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**	122
-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	186
-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**。	129

表 7. 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
-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、実際の引き数として使用されるホレリス定数が、整数の実際の引き数として扱われることを指定します。 **デフォルト: -qnoctyplss**	134
-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**	137
-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**	143
-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**	145

表 7. 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** ステートメント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。	147
-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**	148
-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。	162
-qintlog -qnointlog	INTLOG NOINTLOG	式およびステートメント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**	165

表 7. 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=バイト	INTSIZE(バイト)	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**	166
-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**	182
-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自動でない変数参照ごとに、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**	186
-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。	189

表 7. 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	仮引き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**	191
-1 -qonetrip -qnoonetrip	ONETRIP NOONETRIP	**DO** ステートメントが実行される場合、反復回数が 0 であったとしても、コンパイルされた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**	87
-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**	203
-qposition= {appendold | appendunknown}	POSITION({APPENDOLD | APPENDUNKNOWN})	**POSITION=** 指定子を持たない **OPEN** ステートメント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** ステートメントの 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。	205

表 7. 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**	207
-qrealsize=*バイト*	REALSIZE(*バイト*)	**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**	208
-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。	215
-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**	217

表 7. 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
-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**	235
-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** ステートメントと同じ効果を持っています。 **デフォルト: -qnoundef**	261
-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ステートメントを識別します。 **デフォルト: -qnowarn64**	
-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 つまたは複数の文字として解釈されます。	248

表 7. 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** コマンドの場合はこの逆です。	249
-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。	252

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: 154 ページの『-qflttrap オプション』および 283 ページの『他のシステムの浮動小数点結果の再現』を参照してください。

表 8. 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。	152
-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**	161

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 9. 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	90
-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**	97

表 9. リンク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
-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	98
-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**。	202

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 10. コンパイラーの内部操作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
-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。	88
-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/9.1/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。	93
-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。	271
-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。	272

表 10. コンパイラーの内部操作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
-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。	259
-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)	264

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 11. 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。	211

XL Fortran コンパイラー・オプションの詳細記述

以下のアルファベット順のオプション・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。
- 個々の必須引き数は、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。
- オプションの引き数は、[ ] 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。
- 繰り返せる引き数の後には、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、渡すシステム引き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。 (**make** コマンドに精通している方々にとっては、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

262 ページの『-v オプション』と 263 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 ステートメントが実行される場合、反復回数が 0 であったとしても、コンパイルされた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** ステートメントや **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

-Bprefix

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。

引き数

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 つまたは複数の **-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/8.1/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/9.1/bin/xlf95 -tchIbdz -B/home/jim/xlf/9.1/exe/ test_suite.f
```

アップグレードが受け入れ基準を満たしたら、システム・アドミニストレーターは **/opt/ibmcmp/xlf/9.1** の以前のレベルにインストールします。

関連情報

259 ページの『-t オプション』、 93 ページの『-F オプション』、 15 ページの『構成ファイルのカスタマイズ』、および 21 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、 159 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。 257 ページの『-qzerosize オプション』を参照してください。

219 ページの『-qsigtrap オプション』および 285 ページの『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

103 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。

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

33 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

-Fconfig_file | *-Fconfig_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)。

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/9.1/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・スクリプトを書かずにコンパイラーの基本機能をカスタマイズできる簡単な別の方法は、 **/etc/opt/ibmcmp/xlf/9.1/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

15 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。

関連情報

369 ページの『Fortran 90 または Fortran 95 プログラムのデバッグ』および 9 ページの『シンボリック・デバッガー・サポート』を参照してください。

-qdbg は **-g** の長い形式です。

-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。

引き数

dir は有効なパス名 (たとえば **/home/dir**、**/tmp**、**./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/9.1/include** も検索します。

関連情報

188 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

156 ページの『-qfree オプション』および「*XL Fortran ランゲージ・リファレンス*」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/9.1/lib** または **/opt/ibmcmp/xlf/9.1/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 つまたは複数の **-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ありません。

関連情報

81 ページの『リンクを制御するオプション』および 36 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ありません。

関連情報

81 ページの『リンクを制御するオプション』および 36 ページの『XL Fortran プログラムのリンク』を参照してください。

-N オプション

構文

-NS*bytes*
SPILLSIZE(*bytes*)

内部プログラムのストレージ域のサイズを指定します。

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。

引き数

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。この形式は、現在は最適化を行わず、無視されます。過去のリリースでは、**-O** オプションと **-1** オプションの組み合わせと解釈されていました。この組み合わせは、意図しない結果が生じる場合があります。

-O2 コンパイルに必要な時間やストレージを過度に増やすことなく、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。

-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 ステートメントで **-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

229 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。

168 ページの『-qipa オプション』、 159 ページの『-qhot オプション』と 196 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。

293 ページの『XL Fortran 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技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。

引き数

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、通常はわずか数行のソース・ステートメントほど。) **-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・ディレクティブである、**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

168 ページの『-qipa オプション』および 308 ページの『サブプログラム呼び出しの最適化』を参照してください。

-q32 オプション

関連情報

271 ページの『-q32 オプション』を参照してください。

-q64 オプション

関連情報

272 ページの『-q64 オプション』を参照してください。

-qalias オプション

構文

```
-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。考慮しなければならない別名付けの方法については、293 ページの『XL Fortran プログラムの最適化』を参照してください。

引き数

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** ステートメント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、実引き数として同じデータ・オブジェクトが複数回渡されます。実引き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き数を関連付けます。仮引き数と関連付けられたオブジェクトの何らかの部分が、仮引き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。
- 仮引き数が、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、呼び出されたサブプログラムに仮引き数が実引き数として渡されなかった場合。
- 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。

該当する製品レベル

-qipa オプションが導入されても、**-qalias** は必要です。

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。

引き数

[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、自然の位置合わせ境界で保管されるコンポーネントを十分埋め込んで保管されます。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なら、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、 121 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き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。

引き数

注: Y-HPC の場合、アーキテクチャーに対して有効な選択項目は **-qarch=auto** および **-qarch=ppc970** です。デフォルトは **-qarch=ppc970** です。

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。

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 ビットになり、**-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、 269 ページの『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 になり、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、 269 ページの『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、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、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、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 ハードウェア・プラットフォーム上で実行できます。

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: **-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。

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き数を提供します。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**、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事である場合は、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 12. 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
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
ppc970	あり	あり	あり

関連情報

33 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』、 239 ページの『-qtune オプション』、および 126 ページの『-qcache オプション』を参照してください。

-qassert オプション

構文

-qassert={**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。

引き数

nodeps ループ送り依存性がないことを指定します。

itercnt 不明ループの反復カウント値を指定します。

関連情報

このようなアサーションを使用する際の背景情報および指示については、 302 ページの『ループ変換用のコスト・モデル』を参照してください。また、「*XL Fortran ランゲージ・リファレンス*」で **ASSERT** ディレクティブの説明も参照してください。

-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。

引き数

-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

70 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 375 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。

引き数

-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き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、 393 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。

208 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、最低でもレベル 2 の **-O** 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。

引き数

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。

現在、**-qcache** オプションが効果を持つのは、**-qhot** オプションも指定された場合だけです。

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

239 ページの『-qtune オプション』、 116 ページの『-qarch オプション』、および 159 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 ランゲージ・リファレンス*」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、 89 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** ステートメントのデバッグ、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 ランゲージ・リファレンス*」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文がある場合、**-qcompact** オプションを使用すると、アプリケーション全体のパフォーマンスに影響が出ます。このオプションの使用を、最適化による影響が出ないプログラム部分だけに制限することができます。

規則

-qcompact を有効にしても、その他の最適化オプションは依然として機能しています。コード・サイズは、最適化中に自動的に行われるコードの複製を制限することで縮小されます。

-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。

引き数

arg | **noarg**　　**-qctyplss** の動作を保存するサブオプション。さらに **arg** は、実際の引き数として使用されるホレリス定数が、実際の引き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、引き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、ホレリス定数が実整数引き数であるかのように渡されることを指定します。

```
@PROCESS CTYPLSS(ARG)
      INTEGER :: J

      J = 3HIBM
! These calls are to routines in other languages.
      CALL SUB(J)
      CALL SUB(3HIBM)   ! The Hollerith constant is passed as if
                        ! it were an integer actual argument
```

関連情報

「*XL Fortran ランゲージ・リファレンス*」の『ホレリス定数』と、 343 ページの『参照または値による引き数の引き渡し』を参照してください。

-qdbg オプション

構文

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

-qdbg は 94 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き数、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。

引き数

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 つまたは複数の *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)** ステートメントの後は、プログラムは自由形式ソースに戻ります。

関連情報

「*XL Fortran ランゲージ・リファレンス*」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 ランゲージ・リファレンス*」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 91 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

122 ページの『-qautodbl オプション』と 208 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**

コンパイラーに対して、VMX (Vector Multimedia eXtension) 命令を生成するように指示します。 **-qhot=simd** および **-qarch=ppc970** の各コンパイラー・オプションも指定する必要があります。

関連情報

116 ページの『-qarch オプション』 および 159 ページの『-qhot オプション』 を参照してください。

-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、341 ページの表 20 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** に依存しているため、リンク時にデフォルトとして **libxlf90_r.so** を使用する **xlf_r**、**xlf90_r**、または **xlf95_r** でコンパイルします。また、リンク時に直接リンカーまたは他のコマンドを使用する場合は、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** ステートメント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。

引き数

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、 147 ページの『-qextern オプション』、 105 ページの『-Q オプション』、および 219 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** コメント・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

156 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 ランゲージ・リファレンス*」の『*固定ソース形式*』を参照してください。

-qflag オプション

構文

```
-qflag=dvzlisting_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。

引き数

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文を示します。

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** です。そのため、重要な通知メッセージを見逃すことはありません。

関連情報

176 ページの『-qlanglvl オプション』および 361 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。

このオプションは、複数の個別のオプションに代わるものです。これは、**-qfold**、**-qmaf**、または新規コード用の関連オプションの代わりに使用する必要があります。

-qfloat 設定を変更する場合は、その前に 275 ページの『XL Fortran 浮動小数点処理』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。

引き数

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。このオプションによって、gcc コンパイル・コードとの互換性が保持されます。

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、 392 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。

-qfltrap 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演算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。

引き数

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。

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 -qfltrap=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

219 ページの『-qsigtrap オプション』を参照してください。

どのような場合にどのような方法で **-qfltrap** オプションを使用するかについての詳細な説明は、 284 ページの『浮動小数点演算例外の検出とトラッピング』を参照してください。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** コメント・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

150 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 ランゲージ・リファレンス*」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 ランゲージ・リファレンス*」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)。

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かった場合はソース・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

94 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。

引き数

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文を示します。

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。

-qhot オプション

構文

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

-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。 **-qhot** 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** を想定します。

ループ・アンロールの追加情報については、「*XL Fortran* ランゲージ・リファレンス」の『ループ最適化のためのディレクティブ』の節を参照してください。

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 個の整数
- 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せず、**-qarch=ppc970** を指定する必要があります。通常は、 **-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有利です。

ロング・ベクトル化: -qhot=vector: **vector** サブオプションは、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、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。

引き数

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。

simd | **nosimd**

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。

-qarch=ppc970 を指定する必要があります。

vector | **novector**

コンパイラーは、連続する配列エレメントに対してループで実行される特定の操作を、**libxlopt.a** ライブラリーにあるルーチンに対する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=novector 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、特定のコードをベクトル・ライブラリー・ルーチンに対する呼び出しに置き換えることはしません。 **-qhot=vector** オプションは、プログラムの結果の精度に影響を及ぼす場合がありますで、精度についての変更を受け入れられない場合は、**-qhot=novector** または **-qstrict** のいずれかを指定する必要があります。

関連情報

301 ページの『ループおよび配列言語の最適化』には、実行される変換がリストされています。

-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。

引き数

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

280 ページの『丸めモードの選択』、 100 ページの『-O オプション』、および 152 ページの『-qfloat オプション』を参照してください。

-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 以降だけでなく Fortran 95 にも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 ランゲージ・リファレンス*」の『*ポインター関連付け*』を参照してください。

-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、一般に、必要以上のものをゼロにリセットするため、その結果プログラムが遅くなることがあります。

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。

引き数

- *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

154 ページの『-qfltttrap オプション』、および「*XL Fortran* ランゲージ・リファレンス」の **AUTOMATIC** ディレクティブについての節を参照してください。

-qintlog オプション

構文

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

式およびステートメント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** ステートメント変数
- 割り当てられた **GOTO** 変数
- **DO** ループ・インデックス変数
- **DATA** ステートメント内の暗黙の **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

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=バイト
INTSIZE(バイト)
```

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** ステートメント。
- デフォルトの **INTEGER** 引き数、**LOGICAL** 引き数、戻り値などの授受を行う組み込み関数 (**INTRINSIC** ステートメント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。

byte で許可されているサイズは、以下のとおりです。

- 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・オプションを指定しても、**-qintsize** のデフォルト設定に影響は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 ステートメントを追加して、引き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、実引き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

208 ページの『-qrealsize オプション』および「*XL Fortran ランゲージ・リファレンス*」の『*型付き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コンパイルしなければなりません。)

   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。

引き数

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、入り口点 (実行可能プログラムの場合はメインプログラム、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。

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。

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。

inline=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。

inline=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。この引き数は、**inline=auto** がオンになっているときにのみ実施されます。

inline=*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。

inline=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。この引き数は、「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、プログラマーは、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、196 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き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き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関数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 13. 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。

表 13. 正規表現の構文 (続き)

式	説明
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。 .
[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

100 ページの『-O オプション』、 104 ページの『-p オプション』、および 105 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。

引き数

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。

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** ステートメントにフラグを付ける 計算型 **GOTO** ステートメントにフラグを付ける **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 に対する拡張機能
95std	**PAUSE** ステートメントにフラグを付ける **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 に対する拡張機能
extended	フラグを付けられるエラーはなし	

関連情報

151 ページの『-qflag オプション』、 158 ページの『-qhalt オプション』、および 214 ページの『-qsaa オプション』を参照してください。

39 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

関連情報

168 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

168 ページの『-qipa オプション』を参照してください。

-qlist オプション

構文

-qlist | **-qnolist**
LIST | **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。

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。

関連情報

70 ページの『リストとメッセージを制御するオプション』、 376 ページの『オブジェクト・セクション』、および 258 ページの『-S オプション』を参照してください。

-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。
- 任意の数値引き数 (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

70 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 372 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。

引き数

-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 2048 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

指定されたメモリーの容量が不十分で、コンパイラーが特定の最適化を算出できない場合は、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のみ効果があります。

-O2 を指定してコンパイルするときには、コンパイル時メッセージが限界を上げるように指示する場合にその指示に従うだけで十分です。**-O3** を指定してコンパイルするときには、マシンの実行によりストレージが不足するためにコンパイルが停止する場合に、限界の設定が必要な場合があります。この場合は 2048 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、ストレージを使い果たす場合があります。

関連情報

100 ページの『-O オプション』および 293 ページの『XL Fortran プログラムの最適化』を参照してください。

-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、文字列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自動でない変数参照ごとに、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、慎重に使用してください。

-qmixed オプション

構文

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

これは、 260 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、 95 ページの『-I オプション』を使用してください。

関連情報

28 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 ランゲージ・リファレンス*」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、モジュールとバージョン 9.1 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 以前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。

関連情報

モジュールは Fortran 90 または 95 の機能であり、「*XL Fortran* ランゲージ・リファレンス」の『モジュール』の節で説明されています。

333 ページの『XL Fortran 外部名の規則』 および 37 ページの『リンク中の命名競合の回避』 も参照してください。

-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

70 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**

仮引き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き数に 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き数 (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き数が初期化式である場合は NULL 文字が追加されます。

規則

このオプションは仮引き数の長さ (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き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き数にもそうでない引き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き数には影響を及ぼしません。このオプションは、入出力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

341 ページの『言語間での文字型の引き渡し』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、コード作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

70 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 376 ページの『オブジェクト・セクション』を参照してください。

391 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。

-qonetrip オプション

構文

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

これは、 87 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、 100 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。
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。

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なら、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/9.1/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・カウントを表示します。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 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 file1.o file2.o file3.o file4.o
```

関連情報

26 ページの『XL Fortran 入力ファイル』、 28 ページの『XL Fortran 出力ファイル』、 305 ページの『プロファイル指示フィードバック (PDF) の使用』、および 306 ページの『条件付き分岐の最適化』を参照してください。

-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

391 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。

引き数

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 man 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。

引き数

hexint | **nohexint**
: このオプションを指定する場合、型のない 16 進定数ストリングは、**INT** 組み込み関数へ実引き数として渡すとき、整数に変換されます。 **INT** へ実引き数として渡されない型のない 16 進定数ストリングは、影響を受けることはありません。

mod | **nomod**
: 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 つの引き数は、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、その引き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き数をヌル引き数として扱います。 このサブオプションは、引き数リストが空の場合は効果がありません。

  空の引き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、ヌル引き数が含まれます。

  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 ステートメントと同様の方法で動作する TYPE ステートメントは、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、関連する仮引き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き数である場合に、すべての型なし定数をデフォルト整数に変換します。型が非整数である型なし実引き数に関連付けられている仮引き数は、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き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 ランゲージ・リファレンス*」の **INT** および **MOD** 組み込み関数に関する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** ステートメント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。

規則

最初の I/O 操作がファイル・ポインターを移動させ、その操作が **WRITE** ステートメントまたは **PRINT** ステートメントであると、位置は **APPEND** になります。また、**BACKSPACE**、**ENDFILE**、**READ**、**REWIND** ステートメントであると、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** ステートメントは、**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** ステートメントは、**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** ステートメントは、**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

318 ページの『ファイルの位置決め』、および「*XL Fortran ランゲージ・リファレンス*」の **OPEN** ステートメントについての節を参照してください。

-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 ランゲージ・リファレンス*」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。トリガー定数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、 138 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 ランゲージ・リファレンス*」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=byte
REALSIZE(byte)
```

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。

引き数

byte 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、その引き数と戻り値が **-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・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

166 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。 122 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 ランゲージ・リファレンス*」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。

引き数

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だけです。

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

196 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

176 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/9.1/xlf.cfg** にデフォルト時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

211 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 ランゲージ・リファレンス*」の『変数のストレージ・クラス』を参照してください。

-qsaveopt 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/9.1/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/9.1/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 ランゲージ・リファレンス*」の『**SYSTEM_CLOCK**』を参照してください。

-qshowpdf オプション

構文

-qshowpdf | **-qnoshowpdf**

-qpdf1 とともに使用して、追加の呼び出しとブロック・ カウント・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・ カウントを収集します。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・ カウントを **._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、 196 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。

引き数

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。

関連情報

考えられる例外の原因については、 56 ページの『XL Fortran 実行時例外』で説明されています。 284 ページの『浮動小数点演算例外の検出とトラッピング』では、浮動小数点計算の結果生じる例外を扱う多くの方法を説明しています。 285 ページの『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[=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。

-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を持つ自動オブジェクトに影響を与えます。指定すると、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、自動変数はスタック上に割り振られます。

-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ならば、リンク時に **-qsmp** も一緒に指定する必要があります。

引き数

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文だけでなく、他の並列構文から (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文も含まれます。デフォルトでは、コンパイラーはネストされた並列構文を続けます。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 Fortran 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、実行時ライブラリーは、 **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 Fortran 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 Fortran API に準拠しない言語構文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 *200011* 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 ランゲージ・リファレンス*」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 ランゲージ・リファレンス*」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。
- コマンド行で **-qsmp** または **-qsmp=auto** をしていない場合、 **omp** サブオプションを指定すると **noauto** が暗黙指定されます。
- **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=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 Fortran 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、実行時ライブラリーは、 **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 Fortran API との整合性のない **CRITICAL** 構文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 Fortran 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ステートメント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

70 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 372 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。 99 ページの『-N オプション』を参照してください。

-qstrict オプション

構文

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

デフォルトにより **-O3**、**-O4**、**-O5**、**-qhot**、および **-qipa** 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**、**-O5**、**-qhot**、および **-qipa** 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

100 ページの『-O オプション』、 159 ページの『-qhot オプション』、および 152 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ドラフト 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。

引き数

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 a.f90 -qsuffix=f=f90:cpp=F90
```

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 .f90 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 .F90 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
```

引き数

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 (必要であればゼロを先行させます)。

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、 151 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** ステートメントに指定すると、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** ステートメントで宣言されます。

```
@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 ランゲージ・リファレンス*」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**)。

引き数

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

94 ページの『-g オプション』、 100 ページの『-O オプション』、 312 ページの『最適化したコードのデバッグ』、および 132 ページの『-qcompact オプション』を参照してください。

-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。通常、スレッド・セーフのコードを生成するには、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だけでは、プログラムがスレッド・セーフになる保証はありません。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、生成されたマシン命令を配置 (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。

引き数

注: Y-HPC の場合、インプリメンテーションに対して有効な選択項目は **-qtune=auto** および **-qtune=ppc970** のみです。デフォルトは **-qtune=ppc970** です。

そうでない場合、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
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
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

116 ページの『-qarch オプション』、 126 ページの『-qcache オプション』、および 33 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』を参照してください。

-qundef オプション

構文

-qundef | **-qnoundef**
UNDEF | **NOUNDEF**

-qundef は、 261 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・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。

引き数

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ると、コンパイラーは 1000 回の繰り返しを実行するのではなく、500 回だけ繰り返しを実行します。

```
!IBM* UNROLL(2)
      DO I = 1, 1000
         A(I) = I
      END DO
```

コンパイラーが前のループ・アンロールを選択すると、コンパイラーはそのループを変換して、次の例と本質的に同じになるようにします。

```
      DO I = 1, 1000, 2
         A(I) = I
         A(I+1) = I + 1
      END DO
```

関連情報

「*XL Fortran ランゲージ・リファレンス*」でループ・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**

301 ページの『ループおよび配列言語の最適化』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。

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**

呼び出し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

274 ページの『-qwarn64 オプション』を参照してください。

-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き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き数が 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き数が 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

275 ページの『XL Fortran 浮動小数点処理』、 154 ページの『-qflttrap オプション』、および 361 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。

-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。

引き数

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から 1 つまたは複数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。形式化出力ステートメント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き数を最も長い引き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、最初の引き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、すべての引き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

ENTRY ステートメントを持つサブプログラムの引き数のアドレスを静的ストレージに保管します。これはパフォーマンス向上のために変更された実施選択項目です。

softeof | nosofteof

ユニットが endfile レコードの後に位置付けられているときに、 **READ** 操作と **WRITE** 操作を実行できるようにします。ただし、その位置が **ENDFILE** ステートメント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。 **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。

引き数

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、パフォーマンス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 ランゲージ・リファレンス*」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」のいずれか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

91 ページの『-D オプション』と、「*XL Fortran ランゲージ・リファレンス*」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

70 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 375 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 つまたは複数の **.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

103 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、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

88 ページの『-B オプション』 (この節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/小文字混合でもかまいません。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なしに名前を受け入れることはできますが、それらを組み込み機能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。 187 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、暗黙のステートメントを許可する個々の有効範囲に含まれる **IMPLICIT NONE** ステートメント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、暗黙の型指定は許可されます。

関連情報

「*XL Fortran ランゲージ・リファレンス*」の『**IMPLICIT**』を参照してください。

これは、**-qundef** の短い形式です。 241 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き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

86 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、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。次に例を示します。

```
-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**) だけです。

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、その文字の前に円記号を置いてください。

例

32 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。

-W オプションに指定されたストリングにリテラルのコンマを組み込むには、**¥** を使用することができます。

次の例では、**¥** はリテラルのコンマを **-WF** ストリングに組み込みますが、4 つではなく、3 つの引き数が C プリプロセッサーに指定されます。

```
$ 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

151 ページの『-qflag オプション』の同義語です。**-qflag=e:e** を設定して、言語レベルのチェック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。

引き数

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

100 ページの『-O オプション』および 152 ページの『-qfloat オプション』を参照してください。

-y は 161 ページの『-qieee オプション』の短い形式です。

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・ポインターと 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。注意点として、静的であっても共用であっても、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ります。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。

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**、**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

本節で説明されているコンパイラー・オプションを使用して、以下を行うことができます。

- 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

前述のオプションはすでに 32 ビット環境で一部提供されていますが、そのオプションでは 64 ビット・アーキテクチャーに特有の新しい設定値が指定されています。本節では、こうした場合の新しい設定のみについて説明しています。ここにまとめたオプションは主に、64 ビット・プラットフォームの開発者を対象とします。

-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・ビット・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あります。
- **-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。

-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 **ppc64** 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 **ppc64** 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 **ppc64gr** にアップグレードします。)
  - **-qarch=ppc64**
  - **-qarch=ppc64gr**
  - **-qarch=ppc64grsq**
  - **-qarch=rs64b**
  - **-qarch=rs64c**
  - **-qarch=pwr3**
  - **-qarch=pwr4**
  - **-qarch=pwr5**
  - **-qarch=ppc970**
- **-q64** のデフォルトの **-qarch** 設定値は **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。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・モードで使用するときには、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、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
```

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 **ppc64** にアップグレードされます。

-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ステートメントが識別されます。

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。
  - 等価ステートメント内の整数ポインターの指定。

XL Fortran 浮動小数点処理

本節では、次のような、浮動小数点処理についてよくある疑問にお答えします。
- 予測可能な整合性のある結果を得る方法
- より速くより正確な結果を得る方法
- 例外条件を検出して、その回復を図る方法
- 浮動小数点の計算に使用できるコンパイラー・オプション

関連情報: 本節では、80 ページの『浮動小数点処理のためのオプション』においてと、特に 152 ページの『-qfloat オプション』において、グループにまとめられているコンパイラー・オプションを繰り返し参照します。 XL Fortran コンパイラーは、例外処理と IEEE 算術演算サポート用の 3 つの組み込みモジュールも提供して、移植性を容易にする IEEE モジュール準拠のコードを作成する助けとします。詳細については、「*XL Fortran ランゲージ・リファレンス*」の『*IEEE モジュールとサポート*』を参照してください。

浮動小数点の計算にコンパイラー・オプションを使用すると、浮動小数点計算の精度、パフォーマンス、そしておそらくは正確さに影響があります。ほとんどのプログラムを効率よくかつ正確に実行するように各オプションのデフォルト値は選択されていますが、思いどおりにアプリケーションを処理するには、デフォルト以外のオプションが必要になる場合もあります。それらのオプションを使用する前に、本節をお読みになることを強くお勧めします。

注: 本節の単精度、倍精度、および拡張精度の計算に関する説明はすべてデフォルト状態、すなわち **-qrealsize=4** が指定され、**-qautodbl** が指定されていない状態のことを述べています。これらの設定値を変更する場合、Fortran の **REAL** や **DOUBLE PRECISION** などのサイズを変更しても、単精度、倍精度、および拡張精度 (小文字) の用語はやはり、それぞれ 4 バイト、8 バイト、および 16 バイトのエンティティーを指すことを念頭に置いておいてください。

本節の情報は、PowerPC ファミリー・プロセッサーでの浮動小数点処理に関するものです。

IEEE 浮動小数点の概要

以下で、「*IEEE Standard for Floating-Point Arithmetic*」に関して要約し、それを特定のハードウェア・プラットフォームで XL Fortran に適用する方法について詳述します。ドラフトの Fortran 2003 IEEE Module と算術演算サポートについては、「*XL Fortran ランゲージ・リファレンス*」を参照してください。

IEEE を厳守するためのコンパイル方法

デフォルトでは、XL Fortran は IEEE 標準にほぼ従いますが、すべての規則に従うわけではありません。この標準を厳守するようコンパイルするには、次のようにします。
- コンパイラー・オプション **-qfloat=nomaf** を使います。

- プログラムが実行時に丸めモードを変更する場合は **-qfloat** サブオプションの間に **rrm** を入れます。
- データまたはプログラム・コードにシグナル方式 NaN の値 (NAN) が含まれている場合は、 **-qfloat** サブオプションの間に **nans** を入れてください。 (シグナル NaN は静止 NaN とは異なります。シグナル NaN は、プログラムまたはデータ内に明示的にコード化するか、あるいは **-qinitauto** コンパイラー・オプションを使用して作成する必要があります。)
- **-O3** でコンパイルする場合は、オプション **-qstrict** も入れてください。

IEEE 単精度値および倍精度値

XL Fortran は、単精度および倍精度の値を IEEE 形式でエンコードします。範囲および表示については、「*XL Fortran ランゲージ・リファレンス*」の『実数』を参照してください。

IEEE 拡張精度値

IEEE 標準は、強制ではありませんが、拡張精度値用の形式を提唱しています。 XL Fortran ではこの形式を使用しません。 279 ページの『拡張精度値』 は、XL Fortran が使用する形式についての説明です。

無限大と NaN

単精度の実数値の場合は、次のようになります。
- 正の無限大は、ビット・パターン X'7F80 0000' で表されます。
- 負の無限大はビット・パターン X'FF80 0000' で表されます。
- シグナル方式 NaN は、X'7F80 0001' と X'7FBF FFFF' の間、または X'FF80 0001' と X'FFBF FFFF' の間の任意のビット・パターンで表されます。
- 静止 NaN は、X'7FC0 0000' と X'7FFF FFFF' の間、または X'FFC0 0000' と X'FFFF FFFF' の間の任意のビット・パターンで表されます。

倍精度の実数値の場合は、次のようになります。
- 正の無限大は、ビット・パターン X'7FF00000 00000000' で表されます。
- 負の無限大は、ビット・パターン X'FFF00000 00000000' で表されます。
- シグナル方式 NaN は、X'7FF00000 00000001' と X'7FF7FFFF FFFFFFFF' の間、または X'FFF00000 00000001' と X'FFF7FFFF FFFFFFFF' の間の任意のビット・パターンで表されます。
- 静止 NaN は、X'7FF80000 00000000' と X'7FFFFFFF FFFFFFFF' の間、または X'FFF80000 00000000' と X'FFFFFFFF FFFFFFFF' の間の任意のビット・パターンで表されます。

これらの値は、Fortran の実定数とは対応しません。これらの値はすべて、ビット・パターンを直接エンコードすることによって、または **ieee_arithmetic** モジュールに提供されている **ieee_value** 関数を使用することによって生成できます。 **ieee_value** 関数は Fortran 2003 ドラフト標準で認められており、結果が移植可能であるため、この関数を使用することは、プログラミング技法として好ましいことです。ビット・パターンを直接エンコードすると、値によって異なるビット・パターンを使用しているマシンでは移植上の問題が生じることがあります。シグナル NaN 値以外はすべて、算術演算の結果として生じることがあります。

```
$ cat fp_values.f
real plus_inf, minus_inf, plus_nanq, minus_nanq, nans
real large

data plus_inf /z'7f800000'/
data minus_inf /z'ff800000'/
data plus_nanq /z'7fc00000'/
data minus_nanq /z'ffc00000'/
data nans /z'7f800001'/

print *, 'Special values:', plus_inf, minus_inf, plus_nanq, minus_nanq, nans

! They can also occur as the result of operations.
large = 10.0 ** 200
print *, 'Number too big for a REAL:', large * large
print *, 'Number divided by zero:', (-large) / 0.0
print *, 'Nonsensical results:', plus_inf - plus_inf, sqrt(-large)

! To find if something is a NaN, compare it to itself.
print *, 'Does a quiet NaN equal itself:', plus_nanq .eq. plus_nanq
print *, 'Does a signaling NaN equal itself:', nans .eq. nans
! Only for a NaN is this comparison false.

end
$ xlf95 -o fp_values fp_values.f
** _main   === End of Compilation 1 ===
1501-510  Compilation successful for file fp_values.f.
$ fp_values
 Special values: INF -INF NAN -NAN NAN
 Number too big for a REAL: INF
 Number divided by zero: -INF
 Nonsensical results: NAN NAN
 Does a quiet NaN equal itself: F
 Does a signaling NaN equal itself: F
```

例外処理モデル

IEEE 標準は、起こる可能性のあるいくつかの例外条件を定義しています。

OVERFLOW (オーバーフロー)
値の指数が大きすぎて表すことができません。

UNDERFLOW (アンダーフロー)
非ゼロ値が小さすぎるため、表現の精度が著しく低下します。値は、ゼロまたは非正規数としてのみ表すことができます。

ZERODIVIDE (ゼロ除算)
有限のゼロ以外の値がゼロで割られました。

INVALID (無効)
結果が定義されていない値で演算が実行されています。これには以下のものが含まれます。

- シグナル NaN 値での演算
- 無限大 - 無限大
- 0.0 * 無限大
- 0.0 / 0.0
- x が無限大か y がゼロであるときの mod(x,y) または ieee_rem(x,y) (またはその他の剰余関数)
- 負の数値の平方根

- 整数への浮動小数点数の変換 (変換値を正確に表せないとき)
- NaN 値を含む比較

INEXACT (不正確)
算出された値を正確に表すことはできないため、丸め誤差が生じた場合。(この例外は非常によくあります。)

XL Fortran では、これらの例外が発生したときに必ず検出されますが、デフォルトでは特別な処置は行われません。計算は続行され、通常、結果は NaN または無限大の値になります。例外が発生したら自動的に通知してほしい場合は、コンパイラー・オプションまたは組み込みサブプログラムの呼び出しを介して例外トラッピングをオンにします。ただし、例外ハンドラーで処理されるはずであった、次のようなさまざまな結果が生じます。

表 14. トラッピングを使用可能にした場合としない場合の IEEE 例外の結果

	Overflow	**Underflow**	**Zerodivide**	**Invalid**	**Inexact**
例外を使用可能にしない場合 (デフォルト)	INF	非正規数	INF	NaN	丸めた結果
例外を使用可能にした場合	バイアス指数を持つ正規化されない数	バイアス指数を持つ正規化されない数	結果なし	結果なし	丸めた結果

注: 別の結果が生じることがあるので、生成された例外が正しく処理されることを確認することは非常に重要です。この確認については、 284 ページの『浮動小数点演算例外の検出とトラッピング』を参照してください。

ハードウェア固有の浮動小数点の概要

単精度および倍精度の値

PowerPC 浮動小数点ハードウェアは、IEEE 単精度 (Fortran プログラムの **REAL(4)** と同等) か、または IEEE 倍精度 (Fortran プログラムの **REAL(8)** と同等) のどちらかで計算を実行します。

常に、次のことに配慮していなければなりません。

- 倍精度は単精度 (範囲は約 10**(-38) から 10**38 で、その精度は約 7 桁の 10 進数) よりも大きな範囲 (10**(-308) から 10**308) と高い精度 (約 15 桁の 10 進数) を提供します。
- シングルおよびダブルのオペランドを混合した計算は倍精度で実行されるため、単精度のオペランドを倍精度に変換する必要があります。その変換がパフォーマンスに影響を与えることはありません。
- 単精度に変換される倍精度値 (**SNGL** 組み込み関数を指定した場合や、倍精度の計算結果が単精度変数内に保管される場合など) では、丸め操作が必要です。丸め操作は、有効な IEEE 丸めモードに基づいて、正しい単精度の値を作成します。 この値は丸め誤差の結果、元の倍精度の値よりも精度が低くなる場合があります。倍精度値から単精度値への変換によって、作成するコードのパフォーマンスが低下することがあります。

- 大量の浮動小数点データを処理するプログラムは、 **REAL(8)** 変数ではなく **REAL(4)** 変数を使用した場合に、実行が早くなることがあります。 (**REAL(4)** 変数から提供される範囲と精度が許容できるものであることを確認する必要があります。) プログラムの実行が早くなるのは、データのサイズが小さくなれば、アプリケーションによってはパフォーマンス上の障害となるメモリー・トラフィックが少なくなるからです。

浮動小数点ハードウェアは、2 つの数を掛けて、その積に 3 番目の数を加算する特殊な一連の倍精度演算も行います。これらの組み合わせ乗加算 (**MAF**) 演算は、乗算または加算演算が単独で実行される場合と同じ速度で実行されます。 **MAF** 機能は、1 回 (2 回ではない) の丸め誤差で乗算および加算を実行するので、 IEEE 標準に拡張機能を提供します。 **MAF** 機能を使用すると、同等の別々の演算よりも速く、また精度も高くなります。

拡張精度値

XL Fortran 拡張精度は、IEEE 標準で提唱されている形式になっていません。IEEE 標準は、 (さらに範囲を大きくするため) 指数部のビットと (さらに精度を高めるため) 小数部のビットの両方を拡張する形式を提唱しています。

Fortran プログラムにおける **REAL(16)** と同等の XL Fortran 拡張精度がソフトウェアに組み込まれています。拡張精度は、倍精度と同じ範囲 (約 10**(-308) から 10**308) を提供しますが、精度は倍精度よりも高くなります (可変で、約 31 桁の 10 進数またはそれ以上)。ソフトウェアのサポートは、最も近い値への丸めモードだけに限定されています。拡張精度を使用するプログラムは、拡張精度計算の実行時にこの丸めモードが必ず有効になるようにする必要があります。丸めモードを制御するさまざまな方法については、 280 ページの『丸めモードの選択』を参照してください。

拡張精度の値を 16 進数、8 進数、2 進数、ホレリス定数として指定しているプログラムは、次の規則に従っている必要があります。

- 拡張精度の数字は、異なる絶対値を持ち、オーバーラップしない 2 つの倍精度の数字から構成されています。つまり、バイナリーの指数は、最低でも **REAL(8)** における小数部のビットの数だけ異なっています。高位倍精度値 (ストレージに入れられる最初の値) は、それよりも大きな絶対値を持っている必要があります。拡張精度の数値は、その 2 つの倍精度値の合計です。
- NaN または無限大の値の場合は、これらの値のうちの 1 つを高位倍精度値内でエンコードする必要があります。下位値は無効です。

XL Fortran 拡張精度値は 小数部に多数の想定ゼロを保持した大幅に異なる指数を持つ 2 つの値の和で、この形式は実際には可変精度を持つことになります。 (最小値は約 31 桁の 10 進数です。) 2 つの倍精度値の指数の絶対値が倍精度値の桁数よりも大きく異なっている場合は、精度はさらに高くなります。このエンコードを行うことにより、倍精度固有の範囲を超えずに、見かけ上倍精度より高い精度を必要とするアプリケーションを想定した処理系が効率よく実現できます。

注:

1. 式のコンパイル時フォールディングが原因の丸め誤差に関しては、このフォールディングによって、異なった結果が作成される頻度が他の精度の場合よりも拡張精度値の場合のほうが多いということを念頭に置いておいてください。

2. NaN および無限大などの特殊な数字は、拡張精度値に対して完全にはサポートされていません。算術演算は、これらの数字を必ずしも拡張精度で伝搬するとは限りません。
3. 拡張精度値の場合、XL Fortran で常に浮動小数点演算例外条件が検出されるわけではありません (284 ページの『浮動小数点演算例外の検出とトラッピング』を参照)。また、拡張精度を使用するプログラムで浮動小数点演算例外のトラッピングをオンにすると、例外条件が実際には発生していない場合でも、シグナルが生成されることがあります。

XL Fortran の浮動小数点計算の丸め方

XL Fortran での丸め操作を理解すれば、予測可能な整合性のある結果を得るのに役立ちます。また、スピードと正確度との妥協点を定めなければならない場合に、情報を基に決定を下すのにも役立ちます。

MAF 演算と、中間結果に使用される高い精度のために、XL Fortran プログラムから得られる浮動小数点計算は、一般に他の処理系の場合よりも正確になります。XL Fortran デフォルトでの追加の精度やパフォーマンスよりも、まったく同じ結果を得ることのほうが重要な場合は、 283 ページの『他のシステムの浮動小数点結果の再現』を参照してください。

丸めモードの選択

プログラムの丸めモードを変更するために、**fpsets** および **fpgets** ルーチンを呼び出すことができます。これらのルーチンは、インクルード・ファイル **/opt/ibmcmp/xlf/9.1/include/fpdt.h** および **fpdc.h** に定義されている **fpstat** という名前の論理値の配列を使用します。 **fpstat** 配列エレメントは、浮動小数点の状況レジスターおよび制御レジスターのビットに対応します。

浮動小数点の丸め制御の場合は、配列エレメント **fpstat(fprn1)** と **fpstat(fprn2)** が以下の表に記載されているとおりに設定されます。

表 15. fpsets および fpgets で使用する丸めモード・ビット

fpstat(fprn1)	**fpstat(fprn2)**	**丸めモードを使用可能にした場合**
.true.	.true.	- 無限大方向への丸め
.true.	.false.	+ 無限大方向への丸め
.false.	.true.	ゼロ方向への丸め
.false.	.false.	最も近い値への丸め

たとえば、次のようになります。

```
program fptest
  include 'fpdc.h'

  call fpgets( fpstat ) ! Get current register values.
  if ( (fpstat(fprn1) .eqv. .false.) .and. +
       (fpstat(fprn2) .eqv. .false.)) then
  print *, 'Before test: Rounding mode is towards nearest'
  print *, '             2.0 / 3.0 = ', 2.0 / 3.0
  print *, '            -2.0 / 3.0 = ', -2.0 / 3.0
  end if
```

```
        call fpgets( fpstat )   ! Get current register values.
        fpstat(fprn1) = .TRUE.  ! These 2 lines mean round towards
        fpstat(fprn2) = .FALSE. !   +infinity.
        call fpsets( fpstat )
        r = 2.0 / 3.0
        print *, 'Round towards +infinity:  2.0 / 3.0= ', r

        call fpgets( fpstat )   ! Get current register values.
        fpstat(fprn1) = .TRUE.  ! These 2 lines mean round towards
        fpstat(fprn2) = .TRUE.  !   -infinity.
        call fpsets( fpstat )
        r = -2.0 / 3.0
        print *, 'Round towards -infinity: -2.0 / 3.0= ', r
        end
! This block data program unit initializes the fpstat array, and so on.
        block data
        include 'fpdc.h'
        include 'fpdt.h'
      end
```

XL Fortran は、浮動小数点状況とプロセッサーの制御レジスターを直接制御するためのいくつかのプロシージャーを提供しています。これらのプロシージャーは、浮動小数点状況と制御レジスター (fpscr) を直接操作するインライン・マシン・インストラクションにマップされるため、**fpsets** および **fpgets** サブルーチンより効率的です。

XL Fortran は、**xlf_fp_util** モジュールで **get_round_mode()** および **set_round_mode()** プロシージャーを提供します。これらのプロシージャーはそれぞれ、現行の浮動小数点丸めモードの戻しと設定を行います。

たとえば、次のようになります。

```
      program fptest
        use, intrinsic :: xlf_fp_util
        integer(fpscr_kind) old_fpscr
        if ( get_round_mode() == fp_rnd_rn ) then
        print *, 'Before test: Rounding mode is towards nearest'
        print *, '             2.0 / 3.0 = ', 2.0 / 3.0
        print *, '            -2.0 / 3.0 = ', -2.0 / 3.0
        end if

        old_fpscr = set_round_mode( fp_rnd_rp )
        r = 2.0 / 3.0
        print *, 'Round towards +infinity:  2.0 / 3.0 = ', r

        old_fpscr = set_round_mode( fp_rnd_rm )
        r = -2.0 / 3.0
        print *, 'Round towards -infinity: -2.0 / 3.0 = ', r
      end
```

XL Fortran は、**ieee_arithmetic** モジュールで **ieee_get_rounding_mode()** および **ieee_set_rounding_mode()** プロシージャーを提供します。これらの移植可能プロシージャーはそれぞれ、現行の浮動小数点丸めモードの検索と設定を行います。

たとえば、次のようになります。

```
      program fptest
        use, intrinsic :: ieee_arithmetic
        type(ieee_round_type) current_mode
        call ieee_get_rounding_mode( current_mode )
        if ( current_mode == ieee_nearest ) then
        print *, 'Before test: Rounding mode is towards nearest'
```

```
  print *, '             2.0 / 3.0 = ', 2.0 / 3.0
  print *, '            -2.0 / 3.0 = ', -2.0 / 3.0
  end if

  call ieee_set_rounding_mode( ieee_up )
  r = 2.0 / 3.0
  print *, 'Round towards +infinity:  2.0 / 3.0 = ', r

  call ieee_set_rounding_mode( ieee_down )
  r = -2.0 / 3.0
  print *, 'Round towards -infinity: -2.0 / 3.0 = ', r
end
```

注:

1. 拡張精度の浮動小数点値は、最も近い値への丸めモード以外で使用してはなりません。
2. スレッド・セーフティーおよび再入可能性に関しては、インクルード・ファイル **/opt/ibmcmp/xlf/9.1/include/fpdc.h** の中に、トリガー定数 **IBMT** で保護された **THREADLOCAL** ディレクティブが入っています。呼び出しコマンド **xlf_r**、**xlf90_r**、および **xlf95_r** を実行すると、デフォルトで **-qthreaded** コンパイラー・オプションがオンになり、結果としてトリガー定数 **IBMT** が暗黙指定されます。スレッド・セーフティーの考慮されていないコードでファイル **/opt/ibmcmp/xlf/9.1/include/fpdc.h** を組み込む場合は、トリガー定数に **IBMT** を指定してはなりません。

丸め誤差の最小化

丸め誤差や、計算結果におけるその他の予期しないわずかな誤差を処理するいくつかの方法があります。以下の方法のうちの 1 つまたは複数の方法を考慮してみる必要があるでしょう。

- 全体にわたって丸めの量を最小化する方法。
- できるだけ多くの丸めを実行時まで遅らせる方法。
- 最も近い値への丸めモード以外のモードで丸めが実行される場合に、すべての 丸めが同じモードで実行されることを保証する方法。

全体にわたる丸めの最小化

丸め操作 (特にループの場合) によってコードのパフォーマンスは低下するので、計算の精度に対してよくない影響を与えることがあります。倍精度計算の一時結果を保管するときは、単精度変数ではなく倍精度を使うようにし、計算の最終結果が得られるまで丸め操作を遅らせるよう配慮してください。

実行時までの丸めの遅延

コンパイラーは、可能なら、コンパイル中に浮動小数点の式を評価します。したがって、その結果作成されたプログラムは、実行時の不必要な計算で実行スピードが遅くなることがありません。しかし、コンパイラーの評価による結果は、実行時の計算結果と正確に一致しない場合があります。これらの計算を実行時まで遅らせるには、**-qfloat** オプションの **nofold** サブオプションを指定してください。

それでも結果は同じにならない場合があります。たとえば、**DATA** および **PARAMETER** ステートメント内の計算は、やはりコンパイル時に実行されるからです。

fold/nofold が原因で結果に最も大きな差が出るのは、拡張精度計算を実行したり、**-O** オプションでコンパイルされたりする (またはこの両方) プログラムの場合です。

丸めモードでの整合性の確保

デフォルトで設定されている丸めモードを、最も近い値への丸めモードから変更することができます。 (例を参照してください。) これを行う場合は、プログラムのすべての 丸め操作で同じモードを使用するように注意しなければなりません。

- コンパイル時の計算がいつでも同じ丸めモードを使用するように、**-qieee** オプションで同等の設定を指定してください。
- 最も近い値への丸めモードの正しい作動を必要とするような最適化をコンパイラーが実行しないように、 **-qfloat** オプションの **rrm** サブオプションを指定してください。

無限大への丸めモードを一貫して使用した場合は、たとえば次のコマンドを使用して、 280 ページの『丸めモードの選択』の例のようなプログラムをコンパイルできます。

```
xlf95 -qieee=plus -qfloat=rrm changes_rounding_mode.f
```

他のシステムの浮動小数点結果の再現

異なる浮動小数点アーキテクチャー (乗加算命令は使用しない) を使用するシステム上のプログラムの倍精度結果を再現するには、**-qfloat** オプションの **nomaf** サブオプションを指定します。このサブオプションを使用すれば、コンパイラーが乗加算演算を生成しないようになります。その結果、正確度とパフォーマンスは低下しますが、倍精度算術演算に関する IEEE 標準がより厳守されます。

XL Fortran を実行しているシステムのものと異なる **REAL** 項目のデフォルト・サイズを持つプログラムの結果を再現するには、**-qrealsize** オプション (208ページ) を使用して、 XL Fortran でのコンパイル時にデフォルトの **REAL** サイズを変更します。

結果を再現したいシステムが、**DOUBLE PRECISION** 変数に割り当てられるデフォルトの実定数に対して完全な倍精度で保存される場合は、**-qdpc** または **-qrealsize** オプションを使用してください。

結果が他のシステムと整合性があることが重要な場合は、**-qfloat** オプションの設定に **norsqrt** および **nofold** を入れてください。 **-O3** を指定する場合は、**-qstrict** も入れてください。

浮動小数点パフォーマンスの最大化

第一の関心事がパフォーマンスであり、プログラムを比較的安全に保ちたいが、結果が他の場合に想定されるものと比べて多少異なっても (通常は他よりも正確) 構わないときは、**-O** オプションでプログラムを最適化して、**-qfloat=rsqrt:fltint** を指定してください。 以下の項で、それらのサブオプションの機能について説明します。

- **rsqrt** サブオプションは、平方根で割る除算を、ルートの逆数を掛ける乗算に置き換えます。それによって、演算は高速化されますが、まったく同じ結果が得られるとは限りません。

- **fltint** サブオプションは、オーバーフロー・チェックを減らすことによって、浮動小数点から整数への変換をスピードアップします。整数に変換される浮動小数点が、対応する整数型の範囲外にはないことを確認する必要があります。

浮動小数点演算例外の検出とトラッピング

始めに述べたとおり、浮動小数点算術演算に関する IEEE 標準は、回避または回復に特別な注意が必要な多数の例外 (エラー) 条件を規定しています。以下の項は、このような例外条件が存在する中でプログラムを安全に作動させ、その一方で、犠牲にするパフォーマンスを最小にとどめるのに役立つ内容になっています。

浮動小数点ハードウェアは、常に多数の浮動小数点演算の例外条件 (これは、 IEEE 標準で厳格に規定されています) を検出します。たとえば、オーバーフロー、アンダーフロー、ゼロ除算、無効、および不正確さなどです。

デフォルト時には、とられるアクションは状況フラグの設定だけです。プログラムは問題なく処理を続行します (それ以降の結果は予測どおりでない場合もあります)。例外がいつ発生するかを知りたい場合は、シグナルを生成するように、これらの例外条件の内の 1 つまたは複数を調整することができます。

シグナルによってハンドラー・ルーチンへの分岐が生じます。シグナルが生じたときに、ハンドラーはシグナルのタイプとプログラムの状態についての情報を受け取ります。ハンドラーはメモリー・ダンプの生成、例外が発生した場所のリストの表示、計算結果の変更、またはその他の指定された処理の実行を行うことができます。

XL Fortran コンパイラーは、浮動小数点例外条件の処理にオペレーティング・システム機能を使用します。これらの機能は、**SIGFPE** シグナルを生成することにより、浮動小数点例外の存在を示します。

浮動小数点演算例外をトラッピングするためのコンパイラー機能

XL Fortran の例外トラッピングをオンにするには、**-qflttrap** オプション、および **enable** を含むサブオプションの組み合わせを指定して、プログラムをコンパイルします。このオプションはトラップ操作を使用して浮動小数点例外を検出し、例外発生時に **SIGFPE** シグナルを生成します。

-qflttrap は、例外条件の名前に対応するサブオプションも持っています。たとえば、オーバーフローおよびアンダーフロー例外の処理のみに関心がある場合は、次のように指定することができます。

```
xlf95 -qflttrap=overflow:underflow:enable compute_pi.f
```

enable が必要なのは、メインプログラムをコンパイルするときだけですが、ただし、これは非常に重要であり、他のファイルに対して指定しても何の問題も発生させないので、**-qflttrap** を使用する場合は、常にこれを入れるようにしてください。

このアプローチの利点は、パフォーマンスの影響が比較的少なくなることです。パフォーマンスの影響をさらに少なくするために、**-qflttrap** オプションの **imprecise** サブオプションを入れることができます。これによって、プログラムがサブプログラムの始まりまたは終わりに達するまで、トラッピングが遅れます。

この方法の欠点は次のとおりです。

- **-qflttrap** でコンパイルされたコードで発生する例外のみをトラップします。これには、システム・ライブラリー・ルーチンは含まれません。
- **-qflttrap** の **imprecise** サブオプションを使用する場合は、失敗した計算結果の置換処理をハンドラーが実行することは、通常は不可能です。

注:

1. ご使用中のプログラムが、特定の演算に対して発生する浮動小数点演算の例外に依存している場合は、**nofold** が入っている **-qfloat** サブオプションも指定してください。そのようにしないと、コンパイラーは例外を発生させる計算を一定の NaN または無限大の値と置き換えるか、または、単精度演算におけるオーバーフローを除去します。
2. **-qflttrap** オプションのサブオプションによって、コードの変更を必要とした初期の手法は、 **fpsets** および **fpgets** プロシージャーへの呼び出しに置き換わります。適切な **-qflttrap** 設定を使用すれば、もうこれらの例外処理ハンドルを呼び出す必要はなくなります。

   重要: 浮動小数点演算の例外チェックを使用可能にする **fpsets** 呼び出しがコードに入っているときに、**-qflttrap** オプションを使用しないでプログラム全体をコンパイルすると、 278 ページの表 14 に説明してあるように、例外発生時にプログラムは予期しない結果を発生させます。

例外ハンドラーのインストール

XL Fortran または Linux の例外検出機能を使用するプログラムは、例外条件を検出すると、シグナルを生成します。これによって、プログラムによってどのハンドラーが指定されていても、その指定されているハンドラーへの分岐が発生します。

デフォルト時には、プログラムはコア・ファイル (これをデバッガーとともに使用して問題を突き止めることができます) を作成してから停止します。 **SIGTRAP** または **SIGFPE** シグナル・ハンドラーをインストールしたい場合は、**-qsigtrap** オプションを使用してください。これによって、トレースバックを作成する XL Fortran ハンドラーまたは自分が作成したハンドラーを使用できるようになります。

```
xlf95 -qflttrap=ov:und:en pi.f                              # Dump core on an exception
xlf95 -qflttrap=ov:und:en -qsigtrap pi.f                    # Uses the xl__trce handler
xlf95 -qflttrap=ov:und:en -qsigtrap=return_22_over_7 pi.f   # Uses any other handler
```

SIGNAL サブルーチン (**/opt/ibmcmp/xlf/9.1/include/fexcp.h** に定義されています) を呼び出すことによって、XL Fortran が提供するものでも、自分が作成したものでも、代替の例外ハンドラーをインストールすることができます。

```
INCLUDE 'fexcp.h'
CALL SIGNAL(SIGTRAP,handler_name)
CALL SIGNAL(SIGFPE,handler_name)
```

XL Fortran には次のような例外ハンドラーと関連ルーチンがあります。

xl__ieee
: トレースバックおよびシグナルの説明を作成し、失敗した計算にデフォルト IEEE 結果を提供することによって実行を継続します。このハンドラーを使用すると、プログラムは例外検出がオンになっていない場合と同じ結果を作成します。

xl__trce	トレースバックを作成し、プログラムを停止します。
xl__trcedump	トレースバックとコア・ファイルを作成し、プログラムを停止します。
xl__sigdump	呼び出された地点から出発するトレースバックを提供し、シグナルに関する情報を提供します。これは、ユーザー作成のシグナル・ハンドラー内からしか呼び出すことができません。このハンドラーはプログラムを停止しません。正常に継続するために、シグナル・ハンドラーはこのサブプログラムを呼び出した後に終結処理を行う必要があります。
xl__trbk	呼び出された地点から出発するトレースバックを提供します。このハンドラーは、**-qsigtrap** オプションで指定するのではなく、コードからサブルーチンとして呼び出します。パラメーターは必要ではありません。このハンドラーはプログラムを停止しません。

これらのハンドラー名のすべてに、プログラム内で宣言した名前と重複しないよう 2 つの下線が付いています。また、これらのルーチンはすべて **SIGTRAP** シグナルと **SIGFPE** シグナルの両方に対して機能します。

-g コンパイラー・オプションを使用すると、トレースバック・リスト内の行番号を知ることができます。ファイル **/opt/ibmcmp/xlf/9.1/include/fsignal.h** では、 **signal.h** システム・ヘッダー内の sigcontext 構造体に似ている Fortran 派生型が定義されています。この派生型にアクセスする Fortran シグナル・ハンドラーを作成できます。

関連情報: 289 ページの『例外処理のためのサンプル・プログラム』には、これらのシグナル・ハンドラーの使用法およびご自分で作成する方法を示すサンプル・プログラムがリストされています。詳細については、「*XL Fortran ランゲージ・リファレンス*」の『組み込みプロシージャー』の節にある『**SIGNAL**』を参照してください。

浮動小数点状況および制御レジスターの制御

-qflttrap サブオプションまたは **-qsigtrap** オプションができる前は、浮動小数点例外に対する処理を行うために、ソース・ファイルを変更して例外トラッピングをオンにしたり、シグナル・ハンドラーをインストールしなければならない場合がほとんどでした。現在でもそのようにできますが、新しいアプリケーションの場合は、このオプションの使用をお勧めします。

実行時に例外処理を停止するには、 **-qflttrap** オプションの **enable** サブオプションを指定しないでコンパイルします。

```
xlf95 -qflttrap compute_pi.f     # Check all exceptions, but do not trap.
xlf95 -qflttrap=ov compute_pi.f  # Check one type, but do not trap.
```

次に、プログラム内で **fpstats** 配列 (インクルード・ファイル **/opt/ibmcmp/xlf/9.1/include/fpdc.h** に定義されている) を操作して、**fpsets** サブルーチンを呼び出し、トラップを発生させる例外を指定します。

280 ページの『丸めモードの選択』にある **fpsets** と **fpgets** を使用したサンプル・プログラムを参照してください。

別の方法として、**xlf_fp_util** モジュールの中の **set_fpscr_flags()** サブルーチンを使用する方法があります。このサブルーチンを使用すると、**MASK** 引き数で指定する浮動小数点状況と制御レジスター・フラグを設定することができます。 **MASK** で指定していないフラグには影響がありません。 MASK は、型 **INTEGER(FPSCR_KIND)** でなければなりません。たとえば、次のようになります。

```
USE, INTRINSIC :: xlf_fp_util
INTEGER(FPSCR_KIND) SAVED_FPSCR
INTEGER(FP_MODE_KIND) FP_MODE

SAVED_FPSCR = get_fpscr()           ! Saves the current value of
                                    ! the fpscr register.

CALL set_fpscr_flags(TRP_DIV_BY_ZERO) ! Enables trapping of
! ...                                 ! divide-by-zero.
SAVED_FPSCR=set_fpscr(SAVED_FPSCR)    ! Restores fpscr register.
```

別の方法として、**ieee_exceptions** モジュールの中の **ieee_set_halting_mode** サブルーチンを使用する方法があります。この移植可能な ELEMENTAL サブルーチンを使用して、任意の **FPSCR** 例外フラグの停止 (トラッピング) 状況を設定することができます。たとえば、次のようになります。

```
USE, INTRINSIC :: ieee_exceptions
 TYPE(IEEE_STATUS_TYPE) SAVED_FPSCR
 CALL ieee_get_status(SAVED_FPSCR)   ! Saves the current value of the
                                     ! fpscr register

 CALL ieee_set_halting_mode(IEEE_DIVIDE_BY_ZERO, .TRUE.)  ! Enabled trapping
 ! ...                                                    ! of divide-by-zero.

 CALL IEEE_SET_STATUS(SAVED_FPSCR)  ! Restore fpscr register
```

xlf_fp_util プロシージャー

xlf_fp_util プロシージャーを使用すると、浮動小数点状況とプロセッサーの制御レジスター (fpscr) を直接照会したり制御することができます。これらのプロシージャーは、浮動小数点状況と制御レジスターを直接操作するインライン・マシン・インストラクションにマップされるため、**fpsets** および **fpgets** サブルーチンより効率的です。

組み込みモジュール **xlf_fp_util** には、これらのプロシージャーのインターフェースとデータ型定義、およびプロシージャーで必要な名前付き定数の定義が入っています。このモジュールでは、これらのプロシージャーのタイプを、リンク時ではなくコンパイル時に検査することができます。 **xlf_fp_util** モジュールには、以下のファイルが提供されています。

ファイル名	ファイル・タイプ	ロケーション
xlf_fp_util.mod	モジュール・シンボル・ファイル (32 ビット)	/opt/ibmcmp/xlf/9.1/include
	モジュール・シンボル・ファイル (64 ビット)	/opt/ibmcmp/xlf/9.1/include64

プロシージャーを使用するには、ソース・ファイルに **USE XLF_FP_UTIL** ステートメントを追加する必要があります。詳細については、「*XL Fortran ランゲージ・リファレンス*」の『**USE**』を参照してください。

-U オプションを指定してコンパイルする場合は、これらのプロシージャーの名前をすべて小文字でコーディングする必要があります。

xlf_fp_util プロシージャーのリストについては、「*XL Fortran ランゲージ・リファレンス*」の『サービス・プロシージャーおよびユーティリティー・プロシージャー』の節を参照してください。

fpgets および fpsets サブルーチン

サブルーチン **fpsets** および **fpgets** は、浮動小数点状況レジスターと制御レジスターの操作または照会を行う方法を提供します。オペレーティング・システム・ルーチンを直接呼び出す代わりに、**fpstat** (論理値の配列) 内で情報をやりとりします。次の表は、例外を処理する配列エレメントで、最もよく使用されるものを示しています。

表 16. fpsets および fpgets とともに使用するための例外ビット

可能にセットする配列エレメント	例外が発生したかどうかを検査する配列エレメント	.TRUE. の場合に示される例外
n/a	fpstat(fpfx)	浮動小数点演算例外の要約
n/a	fpstat(fpfex)	浮動小数点使用可能例外の要約
fpstat(fpve)	fpstat(fpvx)	浮動小数点無効演算例外の要約
fpstat(fpoe)	fpstat(fpox)	浮動小数点オーバーフロー例外
fpstat(fpue)	fpstat(fpux)	浮動小数点アンダーフロー例外
fpstat(fpze)	fpstat(fpzx)	Zero-divide (ゼロ除算) 例外
fpstat(fpxe)	fpstat(fpxx)	Inexact (不正確) 例外
fpstat(fpve)	fpstat(fpvxsnan)	浮動小数点無効演算例外 (シグナル NaN)
fpstat(fpve)	fpstat(fpvxisi)	浮動小数点無効演算例外 (INF-INF)
fpstat(fpve)	fpstat(fpvxidi)	浮動小数点無効演算例外 (INF/INF)
fpstat(fpve)	fpstat(fpvxzdz)	浮動小数点無効演算例外 (0/0)
fpstat(fpve)	fpstat(fpvximz)	浮動小数点無効演算例外 (INF*0)
fpstat(fpve)	fpstat(fpvxvc)	浮動小数点無効演算例外 (無効な比較)
n/a	fpstat(fpvxsoft)	浮動小数点無効演算例外 (ソフトウェア要求)、PowerPC のみ
n/a	fpstat(fpvxsqrt)	浮動小数点無効演算例外 (無効な平方根)、PowerPC のみ

表 16. fpsets および fpgets とともに使用するための例外ビット (続き)

可能にセットする配列エレメント	例外が発生したかどうかを検査する配列エレメント	.TRUE. の場合に示される例外
n/a	fpstat(fpvxcvi)	浮動小数点無効演算例外 (無効な整数変換)、PowerPC のみ

プログラム内の特定のポイントで特定の例外を明示的にチェックするには、**fpgets** を使用してから、 **fpstat** 内のエレメントが変更されたかどうかをテストします。いったん例外が発生すると、対応する例外ビット (上記の表の 2 番目の欄) は、明示的にリセットされるまで設定されています。ただし、**fpstat(fpfx)**、**fpstat(fpvx)**、**fpstat(fpfex)** は除きます。これらは、特定のビットがリセットされる場合のみリセットされます。

fpgets および **fpsets** サブルーチンを使用する利点 (**-qflttrap** オプションですべてを制御するのとは対照的) には、例外チェックの細分化の制御も含まれます。たとえば、プログラムの終了時に、プログラムの任意の場所で例外が発生したかどうかだけをテストしたい場合などです。

この方法の欠点は次のとおりです。
- ソース・コードを変更しなければなりません。
- これらのルーチンは、他のプラットフォーム上でよく使われるものと異なります。

たとえば、プログラムのある一定のセクションでのみ、浮動小数点オーバーフロー例外をトラップするには、 `fpstat(fpoe)` を `.TRUE.` に設定して、**fpsets** を呼び出します。例外が発生した後、対応する例外ビット `fpstat(fpox)` は、次のようにプログラムが実行されるまで `.TRUE.` に設定されます。

```
call fpgets(fpstat)
fpstat(fpox) = .FALSE.
call fpsets(fpstat)   ! resetting fpstat(fpox) to .FALSE.
```

例外処理のためのサンプル・プログラム

/opt/ibmcmp/xlf/9.1/samples/floating_point には、例外処理の異なる面を説明するために多くのサンプル・プログラムが入っています。

flttrap_handler.c および **flttrap_test.f**
: C 言語で作成された例外ハンドラーのサンプルと、それを使用する Fortran プログラムです。

xl__ieee.F および **xl__ieee.c**
: 例外を発生させる演算の特定の値を置き換える方法を示す、 Fortran と C 言語で書かれた例外ハンドラーです。このようなサポート・コードが使用される場合でも、 XL Fortran 例外処理のインプリメンテーションは、 IEEE 浮動小数点標準が提唱している例外処理環境を完全にはサポートしていません。

check_fpscr.f および **postmortem.f**
: **fpsets** プロシージャーと **fpgets** プロシージャーおよび **fpstats** 配列を使って作業する方法を示します。

fhandler.F
: サンプルの Fortran シグナル・ハンドラーを示し、**xl__sigdump** プロシージャーをデモします。

xl__trbk_test.f
: **xl__trbk** プロシージャーを使用してプログラムを停止せずにトレースバック・リストを生成する方法を示します。

サンプル・プログラムは、例示することだけを目的にしています。

特定の変数に対して例外を発生させるには

変数に「使用しない」というマークを付けるために、変数内にシグナル NaN と呼ばれる特殊値をエンコードします。それによって、計算でその変数が使用されるたびに、効力のない例外条件を発生させることができます。

この手法を使用する場合は、シグナル NaN が使用されるすべてのケースをプログラムが正しく検出するように、**-qfloat** オプションの **nans** サブオプションを使用し、対応する **SIGFPE** シグナルを生成するための方法 (前述) のいずれかを使用してください。

注:

1. 計算結果としてシグナル NaN が生成されることはなく、シグナル NaN は定数としてプログラムまたは入力データに明示的に取り入れられなければならないため、その中でシグナル NaN 値を故意に使用しない限り、この手法を使用する必要はありません。

浮動小数点演算例外のトラッピングによるパフォーマンスへの影響の最小化

浮動小数点演算例外条件を処理したいが、プログラムがあまり遅くならないようにしたい場合、以下にパフォーマンスへの影響を最小化するのに役立つ手法をいくつか示します。

- 絶対に発生しない条件または気にならない条件を自分で識別できる場合は、**-qfltrap** オプションを指定して、サブオプション **overflow**、**underflow**、**zerodivide**、**invalid**、および **inexact** のサブセットだけを使用することを考えてみてください。特に、**inexact** 例外は丸め誤差のたびに発生するので、パフォーマンスが重要である場合は、この例外をチェックしないでください。
- **-qfltrap** オプションを指定して **IMPRECISE** サブオプションを組み込み、コンパイラー・コマンドが以下のような形になるようにしてください。

  ```
  xlf90 -qfltrap=underflow:enable:imprecise does_underflows.f
  ```

  imprecise は、浮動小数点計算を実行するサブプログラムの入り口と出口でのみ、指定された例外のチェックをプログラムに実行させます。つまり、XL Fortran ではどのような例外も最終的には検出されます。ただし、例外が発生したおよその位置はわかりますが、正確な位置ではありません。

imprecise を付けないで **-qfltrap** を指定すると、浮動小数点操作には、そのつど後に例外チェックが続きます。 **-qfltrap** でコンパイルされていないルーチン (たとえばライブラリー・ルーチン) への呼び出し中にすべての例外が発生する場合、正確な位置を識別するのは難しいので、通常は **imprecise** を使用するのがよいでしょう。

nanq サブオプションを使用している場合、**enable** は効力を持たないことに注意してください。**nanq** は、**imprecise** が指定されている場合であっても、それぞれの浮動小数点数演算、浮動小数点値を戻すロード命令およびプロシージャーの後でトラッピング・コードを生成します。

XL Fortran プログラムの最適化

本節は、最適化の背景となる情報、XL Fortran の最適化機能の使用法に関する手引き、XL Fortran の最適化手法の詳細から構成されています。

単純なコンパイルとは、ソース・コードを実行可能または共用オブジェクトに翻訳または変換することを言います。最適化変換とは、実行時にアプリケーションの総合的パフォーマンスを向上させる変換方式です。 XL Fortran は、IBM ハードウェアに合わせて調整された最適化変換のポートフォリオを提供します。このような変換では、以下が可能になります。

- 重要な操作について実行する命令の数を削減します。
- PowerPC アーキテクチャーの使用が最適化されるように、生成されたコードを再構成します。
- メモリー・サブシステムの使用法を改善します。
- アーキテクチャーの能力を活用し、大容量の共用メモリー並列化を処理します。

コンパイラーは適応力の高い洗練されたプログラム分析および変換能力を持っているため、比較的少ない開発努力で大きなパフォーマンス改善が可能です。さらに、コンパイラーは OpenMP のような、ハイパフォーマンス・コードを作成できるプログラミング・モデルの使用を可能にします。

最適化は、製品リリースのビルドのような、アプリケーション開発サイクルの後のほうのフェーズを対象としています。可能であれば、コードを最適化する前に、最適化なしでコードをテストおよびデバッグしてください。

最適化は、コンパイラー・オプションとディレクティブで制御されます。ただし、コンパイラー・フレンドリーなプログラミング・イディオムは、オプションやディレクティブのようにパフォーマンスに有効である場合があります。手作業によるコードの最適化 (たとえば手動によるループ・アンロールなど) を過剰に行うのは、現在では不要であり、お勧めできません。異常な構文はコンパイラー (および他のプログラマー) を混乱させ、新しいマシンでのアプリケーションの最適化を困難なものにします。『***コンパイラー・フレンドリーなプログラミング***』に、いくつかのイディオムの提案と、良好な最適化が可能なコードを作成するためのプログラミング上のヒントが記載されています。

重要なことは、すべての最適化がすべてのアプリケーションにとって有効であるわけではないことです。デバッグ能力の削減に伴うコンパイル時間の増加と、コンパイラーによって行われる最適化の度合いとの間で、トレードオフを常に考慮しておく必要があります。

XL Fortran の最適化の考え方

XL Fortran の最適化は、*積極性*、つまり負うリスクの大きさに応じて特徴付けられます。最高の最適化レベルだけが以下の積極的な最適化を実行しますが、その場合でもリスクは、想定されるプログラムの小さなサブセットの中のごくわずかな結果の差にすぎません。

積極性の低い最適化は、最適化していない同等のプログラムの場合とまったく同じ結果を出すことを意図します。

- 例外がとにかく発生するということが確かでないと、例外を発生させるコードは移動されません。たとえば、以下のループを見てください。ループの個々の反復に対して結果が同じなので、プログラムはループの前に式 N/K を評価することができます。

```
      DO 10 J=1,N
        ...
        IF (K .NE. 0) M(J)=N/K
        ...
10    END
```

  しかし、K が 0 である場合、最適化していないプログラム内では何も起こらなくても、**N/K** の計算を行うと例外が発生するので、移動は行われません。
- IEEE 算術計算の規則は、それ以外の場合よりも厳密に守られています。[3] たとえば、IEEE 規則では **-0.0+0.0** は **0** でなければならず、この場合は **X+0** は **-X** に等しくなって、**X+0.0** は **X** にはなりません。
- 浮動小数点計算は連想とは見なしません。たとえば、XL Fortranプログラムがすでに **Y*Z** を計算していても、結果がまったく同じにならない場合があるので、**X*Y*Z** は左から右に評価されます。

最適化レベルが上がるにつれて、パフォーマンスを改善できる所でこれらの制限は認められます。

- 前の例の **N/K** のような計算および浮動小数点演算は、例外を発生させる可能性が少ないので、移動またはスケジュール変更することができます。
- IEEE への準拠は、不要と思われる規則に対しては強化されません。ゼロの記号は、上記の例のように、正しく保存されない場合があります。これは、間違った記号の付いたゼロに無限大を掛けて、最後に間違った記号の無限大になるような極端なケースでのみ問題となります。例外を発生させる浮動小数点演算は、例外を発生させないように移動、スケジュール変更、または処理することができます。
- 浮動小数点式は再共用することができ、その結果は異なる場合があります。

最高レベルの最適化を指定した場合、すでに説明したように、XL Fortran はリスクの可能性があってもスピードを要求していると見なします。結果としてリスクを発生させないで、できる限り高い最適化を望む場合は、別のコンパイラー・オプション **-qstrict** を追加する必要があります。

初期の XL ファミリーのコンパイラーは、最適化に対して保守的なアプローチを採用していたので、現実には起こり得ない極端なケースでも、最適化したプログラムを最適化していないプログラムとまったく同じように機能させようとしていました。たとえば、配列参照 **A(N)** は最適化されませんでした。**N** が非常に大きくて、アドレスが参照されるときにプログラムがセグメント化違反を発生させて、この動作が「保持」されるからです。それに比べれば、業界は一般的に保守的ではないアプローチを奨励しており、XL Fortran の最高の最適化レベルは、現在最適化してあるプログラムと最適化していないプログラムの間でまったく同じ実行を達成することよりもパフォーマンスを重視しています。

---

3. IEEE の規則への準拠を考えている場合は、**-qfloat** または **-qstrict** オプションのどちらかを指定することも必要です。

種々のレベルの **-O** オプションが、多種多様のプログラムに対して最適化技法を受け入れます。この技法を使ってパフォーマンスを改善することができます。特殊化された最適化オプション、たとえば **-qipa** と **-qhot** は、ある種のプログラムではパフォーマンスを改善できますが、他のプログラムではパフォーマンスを低下させます。したがって、所定のプログラムにそれらのオプションが適しているかどうかをテストする必要がある場合があります。

最適化のためのコンパイラー・オプションの概要

次の表では、パフォーマンスに最も影響を与えるコンパイラー・オプションを説明しています。異なる組み合わせのコンパイラー・オプションを試して、ご使用のアプリケーションに最も適切なオプションを確認することができます。

表 17. 最適化のためのコンパイラー・オプション

オプション	概説
-O0 または **-qnoopt**	コンパイラーは、非常に限定された最適化を行います。これは、デフォルトの最適化レベルです。アプリケーションの最適化を開始する前に、最適化レベル 0 でコンパイルと実行が正常に行われることを確認してください。
-qarch	コンパイラーは、指定したプロセッサー・アーキテクチャーに対する命令を生成します。このオプションを使用すると、コンパイラーは、指定したアーキテクチャーに存在し、他のアーキテクチャーには存在しないプロセッサー命令を利用することができます。
-qtune	コンパイラーは、指定したプロセッサー・アーキテクチャーのアプリケーションを最適化します。アプリケーションは、**-qarch** オプションで指定したすべてのプロセッサーで実行されますが、そのパフォーマンスは、 **-qtune** オプションで指定したプロセッサー用に調整されます。
-O2	コンパイラーは、基本の最適化を行います。このオプションは、コンパイル速度と実行時パフォーマンス間のバランスが取れています。
-O3	コンパイラーは、メモリーまたはコンパイル時間、あるいはその両方を大量に使用する最適化を含む、積極的な最適化を行います。
-qhot	コンパイラーは追加のループ最適化、自動ベクトル化を行い、オプションで配列埋め込みを実行します。このオプションは、数値の処理を含む化学アプリケーションで最も役立ちます。-04 より短いコンパイル時間、および -O3 より高いパフォーマンスを必要とする場合は、**-O3 -qhot** を試行してください。
-qcache	コンパイラーは、プロセッサーで指定のキャッシュ構成が行われていると想定します。このオプションは、アプリケーションを実行するすべてのプロセッサーでデフォルト以外の同じキャッシュ構成が行われている場合に、パフォーマンスを向上させます。
-qipa	コンパイラーは、プロシージャー間分析を実行して、アプリケーション全体を 1 つの単位として最適化します。このオプションは、頻繁に使用されるルーチンを大量に含むアプリケーションで最も役立ちます。多くの場合、このオプションによってコンパイル時間が大幅に長くなります。
-O4	これは、**-O3 -qipa -qhot -qarch=auto -qtune=auto -qcache=auto** に相当します。
-O5	これは、**-O4 -qipa=level=2** に相当します。

表 17. 最適化のためのコンパイラー・オプション (続き)

オプション	概説
-qpdf1 **-qpdf2**	コンパイラーは、プロファイル指示フィードバックを使用して、アプリケーションが異なるコード・セクションを実行する頻度の分析に基づいてアプリケーションを最適化します。

最適化レベルの選択

最適化を行うと、コンパイル時間は長くなりますが、通常は、実行時間は短くなります。 XL Fortran を使用すれば、最適化をコンパイル時に実行したいかどうかを選択することができます。デフォルトでは、コンパイラーは非常に限定された最適化を実行します (**-O0** または **-qnoopt**)。

コンパイラーの最適化を使用可能にするには、**-O** コンパイラー・オプションと、レベルを示す数字を指定してください。下の表は、それぞれの最適化レベルにおけるコンパイラーの動作を要約したものです。

最適化レベル

オプション **動作**

-qnoopt/-O0 コンパイルは高速、コードはデバッグ可能、プログラム・セマンティクスは保持される。

-O2 包括的な低レベルの最適化。部分的デバッグがサポートされる。(これは **-O** を指定するのと同じです。)

-O3 より広範囲な最適化。一部の精度がトレードオフされる。

-O4 および -O5
プロシージャー間の最適化。ループ最適化。自動マシン・チューニング。

最適化レベル -O2

最適化レベル **-O2** (**-O** と同じ) では、コンパイラーは包括的な低レベル最適化を実行します。これには以下の技法が含まれます。

- ユーザー変数のレジスターへのグローバル割り当て。*グラフ・カラーリング・レジスター割り振り* と呼ばれます。
- アドレッシング・モードの強度削減と効果的な使用。
- 冗長命令の除去。*共通副次式の除去* と呼ばれます。
- 結果が使用されない命令、または指定された制御フローによって到達できない命令の除去。*デッド・コード除去* と呼ばれます。
- 値の番号付け (代数的単純化)。
- インバリアント・コードのループからの移動。
- 定数式のコンパイル時間評価。*定数伝搬* とも呼ばれます。
- 制御フロー単純化。
- ターゲット・マシンの命令スケジューリング (再配列)。
- ループ・アンロールとソフトウェアのパイプライン化。

最適化レベル **-O2** での最小デバッグ情報 は、以下の動作から構成されます。

- 外部およびパラメーター・レジスターはプロシージャー境界で可視になります。これはプロシージャーの出入り口です。プロシージャーの入り口にブレークポイントを設定した場合に、これを見ることができます。ただし、**-Q** でインライン化された関数は、このような境界とその可視性を除去する場合があります。また、これはコンパイラーが非常に小さい関数をインライン化したときにも行われることがあります。
- **SNAPSHOT** ディレクティブは、レジスターをメモリーにフラッシュすることで、ストレージ可視性のための追加プログラム・ポイントを作成します。これにより、ローカルまたはグローバル変数の値、あるいはプログラム内のパラメーターの値を表示および変更することが可能になります。 **SNAPSHOT** の位置にブレークポイントを設定し、その特定ストレージ領域をデバッガーで参照することができます。
- **-qkeepparm** オプションは、パラメーターがスタック・トレースで可視になるように、パラメーターを強制的にプロシージャーへの入り口上のメモリーにします。

最適化レベル -O3

最適化レベル **-O3** では、コンパイラーは **-O2** よりもさらに幅広い最適化を実行します。最適化は、次のようにして拡大され掘り下げられます。
- より深い内部ループ・アンロール。
- ループ・スケジューリングの改善。
- 最適化有効範囲の拡大 (一般にプロシージャー全体に)。
- 最適化の特化 (すべてのプログラムに効果があるとは限らないもの)。
- 多くのコンパイル時間またはスペースを必要とする最適化。
- 暗黙のメモリー使用制限の除去 (**-qmaxmem=-1** 指定のコンパイルと同様)。
- **-qnostrict** の暗黙指定。これにより、浮動小数点計算と潜在的な例外が一部再配列されます。

-qnostrict の暗黙設定により、以下のような精度のトレードオフがコンパイラーによって行われます。
- 浮動小数点計算の再配列。
- 潜在的な例外 (ゼロ除算、オーバーフローなど) の再配列または除去。

-O3 最適化は以下を行います。
- コンパイル中により多くのマシンのリソースを要求します。
- コンパイルにより長い時間をかけます。
- プログラムのセマンティクスを多少変更します。

実行時パフォーマンスが非常に重要な要因であり、マシン・リソースが余分なコンパイル時の動作に適応できる場合は、**-O3** オプションを使用してください。

実行される正確な最適化は、次のような多数の要因によって決まります。
- プログラムを再配置できるかどうか、そして、依然として正しく実行できるかどうか
- 個々の最適化の相対的な利点
- マシン・アーキテクチャー

-O2 および -O3 を最大限に活用する

最適化レベル **-O2** と **-O3** を使用するための推奨アプローチは以下のとおりです。

- 可能であれば、**-O2** を使用する前に、最適化なしでコードをテストおよびデバッグしてください。
- コードがその言語標準に準拠していることを確認します。最適化プログラムは、そのコードが標準に準拠していることを想定し、信頼します。コードがわずかに標準から外れているだけで、最適化プログラムは正しいコード変換ができなくなります。

  サブルーチン・パラメーターが別名割り当て規則に準拠していることを確認してください。
- 独立した入出力プロセスと、独立した非同期割り込みプロセスによってデータ・オブジェクトをアクセスまたは操作するすべてのコードに **VOLATILE** のマークを付けます。たとえば、共用変数および共用変数へのポインターにアクセスするコードにマークを付けます。
- 可能な限り多くのコードを **-O2** でコンパイルします。
- **-O2** で問題が起こった場合は、**-qalias=nostd** オプションを使用する前に、別名割り当て規則の非標準使用についてコードを調べてください。
- 次に、**-O3** を可能な限り多くのコードに使用します。
- 問題またはパフォーマンス低下が発生した場合は、必要に応じて **-O3** とともに **-qstrict** または **-qcompact** の使用を検討してください。
- **-O3** で問題が解決しない場合は、ファイルのサブセットについては **-O2** に切り替えてください。ただし、**-qmaxmem=-1** または **-qnostrict**、あるいはその両方の使用を検討してください。
- **-O2** または **-O3** でコードが正しく機能する場合は、追加の最適化を行うことができる (特にループで) **-qhot** を追加して指定することを考慮してください。 **-O4** または **-O5** を試みる前に **-O3 -qhot** を試みることで、コードが改善され、**-O4** および **-O5** を使用する場合に **-qipa** が暗黙に持っている可能性のあるコンパイル時オーバーヘッドをなくすことができます。

-O4 および -O5 オプション

最適化レベル **-O4** と **-O5** は、自動的に他のいくつかの最適化オプションを活動化します。最適化レベル **-O4** には以下が含まれます。

- **-O3** のすべて
- **-qhot**
- **-qipa**
- **-qarch=auto**
- **-qtune=auto**
- **-qcache=auto**

最適化レベル **-O5** には以下が含まれます。

- **-O4** のすべて
- **-qipa=level=2**

-O5 をコンパイル・ステップで指定する場合は、リンク・ステップでも指定する必要があります。 **-qipa** オプションは厳密に言って最適化レベルではありませんが、これは最適化をすべてのプロシージャーに拡張します (プロシージャーが別のファイルにある場合も)。また、このオプションは他の最適化オプション、特に **-O** (任意

のレベルで) によって行われた最適化の効率を高めます。また、本質的にコンパイルの時間を長くする可能性があるので、すでにデバッグしてあって、使用する準備ができているアプリケーションを調整する場合などに使用できます。 Fortran と、IBM XL C/C+ コンパイラーでコンパイルした C または C++ コードの組み合わせがアプリケーションに含まれている場合、**-O5** オプションを指定してコードすべてをコンパイルおよびリンクして、追加の最適化を実行することができます。

ターゲット・マシンまたはターゲット・マシン・クラスの最適化

ターゲット・マシン・オプションは、指定のプロセッサーまたはアーキテクチャー・ファミリーでの最適な実行のためのコードを生成するように、コンパイラーに指示するオプションです。デフォルトでは、コンパイラーはサポートされるすべてのシステムで実行されるコードを生成しますが、指定のシステムではおそらく最適状態には及ばないものになります。 適切なターゲット・マシン・オプションを選択することで、ターゲット・プロセッサーの最も幅広い選択、指定ファミリー内のプロセッサーの範囲、または特定プロセッサーに合わせて、アプリケーションを最適化することができます。以下のコンパイラー・オプションが、ターゲット・マシンの個々の特徴に効果をもたらすように、最適化を制御します。

ターゲット・マシン・オプション

オプション	動作
-q32	32 ビット・アドレッシング・モデル (32 ビット実行モード) 用のコードを生成します。
-q64	64 ビット・アドレッシング・モデル (64 ビット実行モード) 用のコードを生成します。
-qarch	命令コードを生成する対象となる、プロセッサー・アーキテクチャーのファミリー、または特定アーキテクチャーを選択します。
-qtune	指定のプロセッサーでの実行に対してバイアス最適化を行います。ターゲットとして使用する命令セット・アーキテクチャーに関しては何も暗黙指定はしません。
-qcache	特定のキャッシュまたはメモリー形状を定義します。デフォルトは **-qtune** で設定されます。

事前定義最適化レベルを選択すると、上記の個々のオプションのデフォルト値が設定されます。

関連情報: 116 ページの『-qarch オプション』、 239 ページの『-qtune オプション』、および 126 ページの『-qcache オプション』を参照してください。

ターゲット・マシン・オプションを最大限に活用する

-qarch で、コードが適正に実行されることが期待されるマシンの最小ファミリーを指定してください。

- **-qarch=auto** は、コンパイル・マシン (または類似のマシン) でのみ使用可能な命令を利用できるコードを生成します。
- デフォルトは **-qarch=ppc** です。

- ハードウェアと互換性のない **-qarch** オプションを指定すると、プログラムは機能するように見えても、未定義の動作をする場合があります。コンパイラーはそのハードウェアで使用できない命令を発行する可能性があります。

-qtune で、最良のパフォーマンスを期待するマシンを指定してください。わからない場合は、指定の **-qarch** 設定での最適化に合わせて最良の調整を行う方法をコンパイラーに判断させるようにしてください。

-qcache オプションを使用する前に、**-qlist** を使用してリストのオプション・セクションを参照し、現行設定が満足のいくものであるかを確認してください。 **-qlistopt** オプションが指定されたときは、設定はリスト自体に示されます。システムが構成可能な L2 または L3 キャッシュ・オプションを持つ場合、あるいは実行モードがキャッシュの共用レベルの有効サイズを削減する場合、キャッシュ形状の変更は有効である可能性があります (たとえば、POWER4 でのチップあたり 2 コアの SMP 実行)。

-qcache を使用する場合は、それとともに **-qhot** または **-qsmp** を使用してください。

浮動小数点計算の最適化

浮動小数点計算を効率的に処理するための、特殊なコンパイラー・オプションがあります。デフォルトでは、コンパイラーはパフォーマンスを向上させるために IEEE 浮動小数点規則に対する違反とのトレードオフを行います。たとえば、乗算-加算命令がデフォルトで生成されます。これは、個別の乗算および加算命令よりも高速で、より正確な結果を出すためです。オーバーフローやゼロ除算のような浮動小数点例外は、デフォルトでマスクされます。このような例外をキャッチする必要がある場合は、例外のハードウェア・トラッピングを使用可能にするか、またはソフトウェア・ベースの検査を使用するかを選択できます。オプション **-qflttrap** は、ソフトウェア・ベースの検査を使用可能にします。 POWER4、POWER5、または PowerPC 970 プロセッサーでは、ハードウェア・トラッピングをお勧めします。

浮動小数点計算を扱うためのオプション

オプション	**説明**
-qfloat	浮動小数点計算の処理に対して正確な制御を提供します。
-qflttrap	IEEE 浮動小数点例外のソフトウェア検査を使用可能にします。検査を行う頻度が少なくてすむため、この技法はハードウェア検査より有効な場合があります。

コンパイラー・オプションの別の組み合わせを指定した場合、浮動小数点計算のパフォーマンスで考慮すべき点については、 283 ページの『浮動小数点パフォーマンスの最大化』および 290 ページの『浮動小数点演算例外のトラッピングによるパフォーマンスへの影響の最小化』を参照してください。

高位変換 (-qhot)

高位変換は、特にループおよび配列言語のパフォーマンスを向上させるための最適化です。最適化技法には、交換、フューズ、ループ・アンロール、一時配列生成の削減が含まれます。これらの最適化は以下を目標としています。

- キャッシュとルックアサイド・バッファーの効率的な使用により、メモリー・アクセスのコストを低減します。
- ハードウェアで提供されたデータ・プリフェッチ機能を効果的に利用することで、計算とメモリー・アクセスを並行化します。
- 補完的なリソース要件を持つ命令の使用を再配列および平衡化することで、プロセッサー・リソースの使用効率を向上させます。
- VMX 命令が使用可能で最適化プログラムで利点が確認できる場合は、これらの命令を利用して操作を並列化します。

-qhot=vector は、**-qhot** が指定されたときのデフォルトです。 **-qhot=vector** を指定したコンパイルは、ループのいくつかを変換し、標準バージョンではなく最適化バージョンの関数を活用します。最適化された関数は、逆数、平方根などの関数および演算を含む標準装備ライブラリーにあります。最適化バージョンは、精度とパフォーマンスに関するさまざまなトレードオフを行います。 **-qstrict** の使用は、**-qhot=novector** を暗黙指定します。

-qhot=simd (ショート・ベクトル化) 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、算術演算を並列で実行します。並列演算は、16 バイト・ベクトル・レジスターで行われます。この最適化は、倍精度の浮動小数点算術演算はサポートせず、**-qarch=ppc970** を指定する必要があります。

-qhot を最大限に活用する

すべてのコードに対して、**-qhot** を **-O3** とともに使用してみてください。 (コンパイラーは、**-qhot** について少なくとも **-O2** レベルを想定します。) これは、変換の機会がないときに中間的な効果を持たせるように設計されています。

- 許容できないほどコンパイル時間が長い場合 (複雑なループのネストで発生する可能性があります)、または **-qhot** の使用でパフォーマンスが低下する場合は、**-qhot=novector** を使用するか、**-qstrict** または **-qcompact** を **-qhot** とともに使用してみてください。
- 必要であれば、選択的に **-qhot** を非活動化してください。これにより、コードの一部が改善されます。

ループおよび配列言語の最適化

-qhot オプションは、ループ、配列言語およびメモリー管理のパフォーマンスを改善するために変換を行います。

- スカラーの置き換え、ループのブロック化、分配、融合、交換、反転、スキュー、アンロール
- 一時配列の生成の削減

このオプションは最低でもレベル 2 の **-O** が必要です。 **-C** オプションはこれを禁止します。

SMP ハードウェアがあれば、**-qsmp** オプションを指定して、ループを自動的に並列化できます。この最適化には、明示的にコード化された **DO** ループに加えて、配列言語用のコンパイラーで生成された **DO** ループ (**WHERE**、 **FORALL**、配列割り当てなど) が含まれています。コンパイラーは独立したループを並列化するにすぎません (個々の反復は、他の反復とは別個に計算できます)。コンパイラーがループの自動並列化を行わないケースとして、ループに I/O が入っている場合がありま

す。これは、予期しない結果につながるおそれがあるためです。この場合、ユーザーは、**PARALLEL DO** または作業共用 **DO** ディレクティブを使って、そのようなループを安全に並列化できるようコンパイラーに指示を与えることができます。ただし、I/O が以下の可能性のいずれかに適合していることが前提となります。

- 反復ごとに異なるレコードでの読み/書きが行われる直接アクセス I/O。
- 反復ごとに異なるユニットでの読み/書きが行われる順次 I/O。
- 反復ごとに **POS=** 指定子を使用してファイルの異なる部分の読み書きが行われるストリーム・アクセス I/O。
- 反復ごとに異なるユニットでの読み書きが行われるストリーム・アクセス I/O。

詳細については、「*XL Fortran ランゲージ・リファレンス*」にある **PARALLEL DO** または作業共用 **DO** ディレクティブの説明を参照してください。

-qhot および **-qsmp** オプション は、次のようなプログラムに使用できます。

- ループおよび構造的メモリー・アクセスが原因でパフォーマンスのボトルネックが発生するプログラム
- 配列言語 (配列操作で FORTRAN 77 ループと同様に最適化できる) を大量に含むプログラム

ループ変換用のコスト・モデル

-qhot オプションによって実行されるループ変換は、特定の変換の実行の標準的なループおよびコスト (使用するレジスターと持ち込まれる潜在的な遅延という意味で) の特性に関する一連の前提事項によって制御されます。

このコスト・モデルは、次のことを考慮に入れています。

- プロセッサーの使用可能なレジスターおよび機能ユニットの数
- システムのキャッシュ・メモリーの構成
- 各ループの反復回数
- 保守的な前提事項を行って正確な結果を確認する必要性

正確に情報 (たとえば、ループの反復回数) を判別できる場合は、コンパイラーはこの情報を使用してプログラムのその位置におけるコスト・モデルの正確性を向上させます。情報が判別できない場合は、コンパイラーはコスト・モデルのデフォルト時の前提事項に依存します。デフォルト時の前提事項を変更して、コンパイラーがループを最適化する方法に影響を及ぼすことができます。このとき、次のようなコンパイラー・オプションを指定します。

- **-qassert=nodeps** は、コンパイルされているファイル内のどのループにも 1 つの反復から次の反復へ継承される依存性がないことを表明します。この依存性をループ送り依存性と呼びます。 *n* 回目の反復中に実行される計算が他の反復中に計算される結果を必要としないことが断言できる場合は、コンパイラーには効率を上げるためのループの再配置が行いやすくなります。
- **-qassert=itercnt=***n* は、コンパイルされているファイル内の「通常の」ループが約 *n* 回繰り返されることを断言します。これを指定しない場合は、ループが約 1024 回繰り返されることを前提とします。コンパイラーはこの情報を使用して、反復回数の少ないループ内へ反復回数の多いループを配置するなどの変換の助けにします。

値を正確にすることは重要ではありません。また、ファイル内のすべてのループについて値が正確である必要はありません。次の場合は、この値はループに使用されません。
- コンパイラーが正確な反復回数を判別できる場合
- **ASSERT(ITERCNT(*n*))** ディレクティブが指定された場合

反復回数が多いループのスピードを上げるだけのループ変換もあります。プログラムにそのようなループが多く含まれている場合、または反復回数が多いループがホット・スポットおよびボトルネックとなっている場合は、 *n* に大きな値を指定してください。

プログラムにさまざまなループが入っている場合があるため (これらのオプションによってスピードが上がるもの、影響されないもの、遅くなるものがある)、どのループがどのオプションから最も益を得るかを判別し、いくつかのループを別のファイルに分割し、最も適したオプションのセットとディレクティブを指定してファイルをコンパイルすることができます。

ループ・アンロール

ループ・アンロールには、2 回、3 回またはそれ以上の反復作業を行うためにループ本体をアンロールし、それに比例して繰り返しのカウントを減らすことが関係しています。ループ・アンロールを行うことには、以下の利点があります。

- データ依存関係遅延が少なくなるか、まったくなくなる
- 連続したループ反復でロードと保管を行う必要がなくなる
- ループ・オーバーヘッドが少なくなる

ループをアンロールすると、新しいループ本体のコード・サイズが増えることになるので、レジスターの割り振りが増えてレジスター・スピルの原因となる場合があります。このため、アンロールを行ってもパフォーマンスが向上しないことがあります。

関連情報: 242 ページの『-qunroll オプション』を参照してください。

ハードウェア構成の説明

-qtune 設定によってプロセッサー内のレジスターと機能ユニットの数を決定します。たとえば、 ループをチューニングする場合、 **-qtune=pwr4** を設定すると、コンパイラーは、ほとんどの内側ループを深さが 2 になるまでアンロールして、余分の演算ユニットを利用するようにします。

-qcache 設定は、コンパイラーがループをブロックする際に使用するブロック化係数を決定します。使用可能なキャッシュ・メモリーが多いほど、ブロック化係数は大きくなります。

異なる形式の配列の効率

一般に、定数または調整可能な境界を持つ配列、大きさ引き継ぎ配列、pointee 配列に対する演算は、自動、想定形状、形状無指定配列、その他の配列よりも処理が少なくて済み、かつ、スピードも速いようです。

一時配列の使用の削減

プログラムが配列言語を使用していても、式の左辺の配列が右辺の配列と重複する配列の代入を決して実行しない場合は、オプション **-qalias=noaryovrlp** を指定すると、一時配列オブジェクトの使用が減るため、パフォーマンスを向上させることができます。

-qhot オプションも、多くの一時配列を削減することができます。

配列の埋め込み

XL Fortran 対応のチップ・ターゲットのキャッシュ・アーキテクチャーをインプリメントしたことが、2 の累乗となっている配列次元によるキャッシュの使用効率の低下を招く場合があります。

-qhot オプションの **arraypad** サブオプションを使用すると、コンパイラーは、配列処理をしているループの効率が向上する可能性があるところで配列の次元を増加させます。次元 (特に最初の次元) が 2 の乗数になっている大きな配列がある場合、あるいはキャッシュ・ミスまたはページ不在によって配列を処理しているプログラムが遅くなっている場合は、**-qhot** ではなく **-qhot=arraypad** または **-qhot=arraypad=***n* の指定を考慮する必要があります。

-qhot=arraypad が実行する埋め込みは内部的です。またソース・コード (**EQUIVALENCE** ステートメントで作成されるものなど) の中に該当するケースがないことを想定しており、その場合の記憶素子には埋め込みで分断された関係が入っています。また、配列の次元を手動で埋め込むことがプログラムの結果に影響しないと判断できる場合は、そのような埋め込みを行うこともできます。

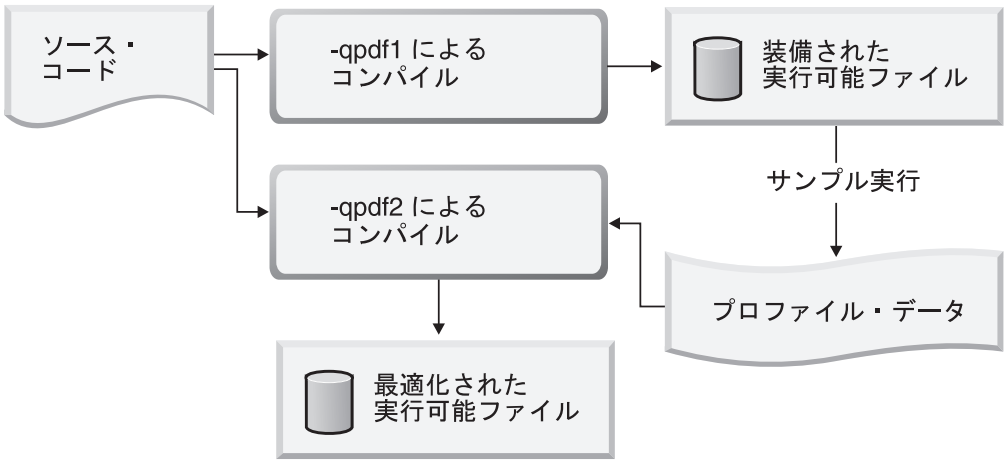
埋め込みが (特に多次元配列の場合に) 使用する追加ストレージは、プログラムが再び遅くなるか、あるいはストレージがなくなるところまでプログラムのオーバーヘッドを増加させる場合があります。詳細については、159 ページの『-qhot オプション』を参照してください。

プロファイル指示フィードバック (PDF)

プロファイル指示フィードバックは、一般的なプログラム動作のデータ特性を持つコンパイラーを提供する、2 段階のコンパイル・プロセスです。装備された実行可能ファイルは、任意の時間、さまざまなシナリオで実行され、副次作用としてプロファイル・データ・ファイルを作成します。プロファイル・データを使用した 2 回目のコンパイルは、最適化された実行可能ファイルを作成します。

PDF は、条件付エラー処理またはインスツルメンテーションをまれにしか実行しなかったコードで主に使用されます。この技法は、ファーム・プロファイル情報がない場合に中間的な効果を持っていますが、不十分な、または特性のないデータしか使用可能でない場合はお勧めできません。

下の図は PDF プロセスを示しています。



プロセスの 2 つのステージは、コンパイラー・オプション **-qpdf1** と **-qpdf2** で制御されます。ステージ 1 は最適化オプションの任意セットと **-qpdf1** を使用した通常のコンパイルで、任意の時間、さまざまなシナリオで実行できる実行可能ファイルまたは共用オブジェクトを作成します。ステージ 2 は **-qpdf1** の代わりに **-qpdf2** を使用すること以外は同じオプションを使用した再コンパイルで、その間コンパイラーは、パス偏向最適化の目的で以前に収集されたデータを消費します。

プロファイル指示フィードバック (PDF) の使用

アプリケーションが異なるコード・セクションを実行する頻度の分析に基づいて、以下のようにアプリケーションを最適化できます。

1. **-qpdf1** を使用してアプリケーションをコンパイルする。
2. 標準のデータ・セットを 1 つ以上使用して、アプリケーションを実行する。アプリケーションが存在する場合は、PDF ファイルにプロファイル情報が書き込まれます。
3. **-qpdf2** を使用してアプリケーションをコンパイルする。

最も頻繁に使用される関数を確認するには、次の作業を行います。

1. **-qpdf1 -qshowpdf** を使用してアプリケーションをコンパイルする。
2. 標準のデータ・セットを 1 つ以上使用して、アプリケーションを実行する。アプリケーションによって、PDF ファイルに詳細なプロファイル情報が書き込まれます。
3. **showpdf** ユーティリティーを使用して、PDF ファイルで情報を表示する。

次のようにして、PDF ファイルの生成をさらに制御できます。

1. **-qpdf1** を使用してアプリケーションをコンパイルする。
2. 標準のデータ・セットを 1 つ以上使用して、アプリケーションを実行する。これによって、現行ディレクトリーに PDF ファイルが作成されます。
3. アプリケーションを別のディレクトリーにコピーし、再実行する。これによって、2 番目のディレクトリーに PDF ファイルが作成されます。
4. 前のステップを必要なだけ繰り返す。

5. **mergepdf** ユーティリティーを使用して、複数の PDF ファイルを 1 つの PDF ファイルに結合する。たとえば、それぞれ 53%、32%、15% の時点で実行される使用パターンを表す 3 つの PDF ファイルを作成する場合は、次のコマンドを使用できます。

   ```
   mergepdf -r 53 path1  -r 32 path2  -r 15 path3
   ```

6. **-qpdf2** を使用してアプリケーションをコンパイルする。

PDF ディレクトリーの情報を消去するには、**cleanpdf** または **resetpdf** ユーティリティーを使用します。

条件付き分岐の最適化

-qpdf オプションは、条件付き分岐の周辺区域を微調整して、デフォルトの選択項目を最も可能性の高い実行パスに対応させるのに役立ちます。スローダウンを防ぐため、最も可能性の高い実行パスにある命令が、分岐の前に他の命令と並行して実行される場合もあります。

-qpdf オプションは余分なコンパイル・オーバーヘッドと代表的なデータを使用したサンプル実行を必要とするため、開発サイクルの終わり近くで使用してください。

関連情報: 196 ページの『-qpdf オプション』を参照してください。

プロシージャー間分析 (-qipa)

プロシージャー間分析 (IPA) を使用すると、コンパイラーは異なるファイルにまたがった最適化を行うことができ (全プログラム分析)、有効なパフォーマンス向上をもたらします。プロシージャー間分析はコンパイル・ステップでのみ、またはコンパイルおよびリンク・ステップで指定できます (**全プログラム** ・モード)。全プログラム・モードは、最適化の有効範囲をプログラム単位全体に拡張し、実行可能または共用オブジェクトにすることができます。全プログラム IPA 分析は、大きなプログラムのコンパイルまたはリンク時に、かなりのメモリー量と時間を消費します。

IPA は **-qipa** オプションによって使用可能になります。最も一般的に使用されるサブオプションの効果を以下に要約します。

一般に使用される **-qipa** サブオプション

サブオプション	**動作**
level=0	プログラム区分化と単純プロシージャー間最適化。これは以下から構成されます。 • 標準ライブラリーの自動認識。 • 静的にバインドされた変数およびプロシージャーのローカリゼーション。 • 呼び出し関係に応じたプロシージャーの区分化とレイアウト。*呼び出し類縁性* とも呼ばれます。 (相互に頻繁に呼び出すプロシージャーは、メモリー内で近くに置かれます。) • 一部の最適化の有効範囲の拡張 (特にレジスター割り振り)。

level=1 インライン化とグローバル・データ・マッピング。特に以下を行います。
- プロシージャーのインライン化。
- 参照類縁性に応じた静的データの区分化とレイアウト。 (ともに頻繁に参照されるデータは、メモリー内で近くに置かれます。)

-qipa が指定されたときは、これがデフォルトになります。

level=2 グローバル別名分析、特殊化、プロシージャー間データ・フロー。
- 全プログラム別名分析。このレベルには、ポインター間接参照と間接関数呼び出しの明確化と、関数呼び出しの副次作用に関する情報の改良が含まれます。
- 集約的プロシージャー内最適化。これは、値の番号付け、コード伝播および単純化、条件への、またはループからのコード動作、冗長度の除去のかたちをとります。
- プロシージャー間定数伝搬、不要コード除去、ポインター分析。
- プロシージャー特殊化 (クローン作成)。

inline=*inline-options*

インライン化の正確なユーザー制御を提供します。

fine_tuning **-qipa=** の他の値では、ライブラリー・コードの動作の指定、プログラム区分化の調整、ファイルからのコマンド読み取りなどの能力が提供されます。

-qipa を最大限に活用する

あらゆるものを **-qipa** でコンパイルする必要はありませんが、可能な限り多くのプログラムにこれを適用するようにしてください。以下にいくつかの提案を示します。

- makefile で最適化オプションを指定するときは必ず、リンクのためのコンパイラー・コマンド (**xlf**、**xlf90** など) を使用し、リンク・ステップですべてのコンパイラー・オプションを組み込みます。
- **-qipa** は、実行可能または共用オブジェクトを作成するときに機能しますが、常にメインおよび **-qipa** でエクスポートされた関数をコンパイルします。
- コンパイルとリンクを別個に行うときは、高速化のために **-qipa=noobject** をコンパイル・ステップで使用します。
- 十分なスペースが **/tmp** にあること (最低 200 MB) を確認するか、または TMPDIR 環境変数を使用して十分なフリー・スペースを持つ異なるディレクトリーを指定します。
- **level** サブオプションはスロットルです。リンク時間が長すぎる場合はこれを変えてみてください。 **-qipa=level=0** でのコンパイルは、余分なリンク時間をかけたくないときに非常に有効です。
- **-qlist** または **-qipa=list** でのコンパイルの後で、生成されたコードを見てください。インライン化された関数が少なすぎる、または多すぎる場合は、**-qipa=inline** または **-qipa=noinline** の使用を検討してください。特定の関数のインライン化を制御するには、**-Q+** および **-Q-** を使用してください。
- Fortran と、IBM XL C/C+ コンパイラーでコンパイルした C または C++ コードの組み合わせがアプリケーションに含まれている場合、**-qipa** オプションを指定してコードすべてをコンパイルすることで、より以上の最適化を達成することができます。

サブプログラム呼び出しの最適化

プログラムに多数のサブプログラム呼び出しがある場合、**-qipa=inline** オプションを使用して、インライン化をオンにすることができます。これにより、このような呼び出しのオーバーヘッドが削減されます。 **gprof** で **-p** または **-pg** オプションを使用して、最も頻繁に呼び出されるサブプログラムを判別し、コマンド行にその名前をリストすることを検討してください。

呼び出し側サブプログラムと呼び出し先サブプログラムが異なる有効範囲にある場合にインライン化を呼び出しに適用させるためには、**-qipa** オプションを組み込みます。

```
# Let the compiler decide (relatively cautiously) what to inline.
xlf95 -O3 -qipa=inline inline.f

# Encourage the compiler to inline particular subprograms.
xlf95 -O3 -qipa=inline=called_100_times,called_1000_times inline.f

# Explicity extend the inlining to calls across multiple files.
xlf95 -O3 -qipa=inline=called_100_times,called_1000_times -qipa inline.f
```

関連情報: 105 ページの『-Q オプション』および 168 ページの『-qipa オプション』を参照してください。

インライン化の正しいレベルの見つけ方

特定プログラムに対してインライン化の正しい量を知るには、ユーザー・サイドで行わなければならない作業がいくつかある場合があります。コンパイラーには、インライン化を多量に行うことを避けるために多数の安全機能および限界があります。そうでなければ、コンパイル中のストレージの制約が原因で、コンパイラーが実行する全体にわたる最適化が少なくなったり、その結果作成されたプログラムが大きくなりすぎて、キャッシュ・ミスおよびページ不在がより頻繁になるために実行スピードが遅くなります。ただし、これらの安全機能は、インライン化したいサブプログラムをコンパイラーがインライン化することを妨げることがあります。この事態が発生した場合、パフォーマンス面の恩恵を得るために、分析または作業のやり直し、またはその両方を行う必要があります。

一般的な規則として、最も頻繁に呼び出されるごく少ないサブプログラムを識別して、それらのサブプログラムだけをインライン化することを考えてください。

-qipa=inline でサブプログラムをインライン化できない一般的な条件は、次のとおりです。

- 呼び出し側プロシージャーと呼び出し先プロシージャーが異なる有効範囲にあります。この場合は、**-qipa** オプションを使用して異なるファイル間のインライン化を使用可能にすることができます。
- インライン化の結果として、ある一定の量だけコンパイラーがサブプログラムを拡張した後、そのサブプログラムから引き続き呼び出すと、インライン化されません。さらに、呼び出し先サブプログラムが **-qipa=inline** オプションで名前が指定されているかどうかによって、異なる制限があります。

  3 つのプロシージャーを例として考えてみましょう。**A** は呼び出し元で、**B** と **C** は自動インライン化のためのサイズの上限があるものとします。これらはすべて同じファイル内にあり、これは次のようにコンパイルされます。

```
xlf -qipa=inline=c file.f
```

-qipa=inline は、 **C** の呼び出しがインライン化されやすいことを意味します。**B** および **C** のサイズが 2 倍である場合、**B** の呼び出しはインライン化されず、**C** の呼び出しのいくつかはインライン化されます。

これらの制限は、**A** から **B**、または **A** から **C** への呼び出しがインライン化されるのを回避しますが、コンパイラーが処理 **A** を完了した後で、プロセスが再び最初からやり直されます。

- 引き数または戻り値の数、サイズ、型の違いなどのインターフェース・エラーによって、呼び出しがインライン化されるのを防止する場合もあります。呼び出されているプロシージャーのインターフェース・ブロックが正しく定義されていることを確認してください。
- 仮引き数または自動変数の実際の別名付け、あるいは、起こり得る別名付けは、プロシージャーがインライン化されるのを防止する場合があります。たとえば、次のような場合、インライン化が起こらない可能性があります。
  - オプション **-qalias=nostd** によって、呼び出すまたは呼び出されるプロシージャーが含まれているファイルをコンパイルする場合で、呼び出されるプロシージャーに対する引き数がいくつか存在する場合
  - 呼び出されるプロシージャーに対する引き数が約 31 よりも多い場合
  - 呼び出されるプロシージャー内の自動変数が **EQUIVALENCE** ステートメントに関係がある場合
  - 可変の同じ引き数が同一の呼び出しで複数回渡される場合、たとえば、**CALL SUB(X,Y,X)**
- 計算される **GO TO** ステートメント (このステートメントでは、対応するステートメント・ラベルが **ASSIGN** ステートメントでも使用されます) を使用するいくつかのプロシージャーをインライン化できない場合

インラインを制御するサイズの限界を変更するために、**-qipa=limit=***n* を使用することができます。この *n* は 0 から 9 です。値をより大きくすれば、さらにインライン化することができます。

リンク時の最適化で、C/C++ 関数を Fortran プログラムにインライン化する (およびその逆を行う) ことは可能です。 C/C++ コードは、IBM XL C/C++ コンパイラーを使用して、**-qipa** および XLF コンパイルで使用されたものと互換性のあるオプションを設定してコンパイルする必要があります。

共用メモリー並列処理 (-qsmp)

一部の IBM プロセッサーは、共用メモリー並列処理が可能です。この機能を活用するために必要なスレッド化されたコードを生成するには、 **-qsmp** を使用してコンパイルしてください。このオプションは、**-O2** 最適化レベルを暗黙指定します。サブオプションなしのオプションのデフォルト動作は、最適化を伴う自動並列化を行います。

一般に使用される **-qsmp** サブオプションが下の表に要約されています。

一般に使用される **-qsmp** サブオプション

サブオプション	動作
auto	ユーザー・アシスタンスなしで可能な並列コードの自動生成をコンパイラーに指示します。このオプションはまた、すべての SMP ディレクティブを認識します。
omp	明示的並列化を指定するために OpenMP Fortran API でのコンパイルを強制します。 **-qsmp=omp** は現在、**-qsmp=auto** とは非互換であることに注意してください。
opt	並列化とともに最適化をコンパイラーに指示します。最適化は、他の最適化オプションなしの **-O2 -qhot** と同等です。 **-qsmp** のデフォルト設定は **-qsmp=auto:noomp:opt** です。
suboptions	サブオプションの他の値では、スレッド・スケジューリング、ネストされた並列処理、ロッキングなどの制御が提供されます。

-qsmp を最大限に活用する

- OpenMP プログラムをコンパイルしていて、自動並列化を行いたくない場合に、**-qsmp=omp:noauto** を使用します。デフォルトでは、実行される並列化は明示的かつ自動的です。
- 自動並列化で **-qsmp** を使用する前に、最適化および **-qhot** を単一スレッド手法を使用してプログラムをテストしてください。
- **-qsmp** の使用時には、常に再入可能コンパイラー呼び出し (**xlf_r** などの **_r** コマンド呼び出し) を使用します。
- デフォルトでは、ランタイムは使用可能なすべてのプロセッサーを使用します。使用可能なプロセッサー数より少ないプロセッサーを使用したい場合でない限り、**XLSMPOPTS=PARTHDS** または **OMP_NUM_THREADS** 変数を設定しないでください。実行スレッド数を小さな値に、または 1 に設定して、デバッグを容易にしたい場合があります。
- 専用マシンまたはノードを使用している場合は、**SPINS** および **YIELDS** 変数 (**XLSMPOPTS** のサブオプション) を 0 に設定することを検討してください。これは、バリアなどの同期境界にまたがるスレッドのスケジューリングにオペレーティング・システムが介入することを妨げます。
- OpenMP プログラムをデバッグするときは、コンパイラーが作成するデバッグ情報をより正確なものにするために、**-qsmp=noopt** (**-O** なし) を使用してみてください。また、**SNAPSHOT** ディレクティブは、レジスターをメモリーにフラッシュすることで、ストレージ可視性のための追加プログラム・ポイントを作成することもできます。

その他のプログラム動作オプション

コンパイラー分析の精度は、メモリーを読み書き可能な命令によって大きく影響を受けます。別名割り当ては何かに別の名前を付けることに関係します。これは、ここではメモリーの参照です。メモリーの参照は、名前付きシンボルのように直接の場合と、またはポインターまたは仮引き数のように間接の場合があります。関数呼び出しも、メモリーを間接的に参照します。偽であるメモリーに対する見せ掛けの参照、つまりコンパイラーによって想定されたいくつかのロケーションを実際には参照しない参照は、コンパイラー分析に対する障害となります。

Fortran は、サブプログラムの実行中に仮引き数参照が他の仮引き数または外部的に可視であるシンボルをオーバーラップしない規則を定義します。

コンパイラーは洗練された分析を実行し、ポインター間接参照および呼び出しのために可能な別名のセットを定義しようとします。しかし、コンパイル時の制限された有効範囲と値の不在により、このような分析の有効性は抑制されます。最適化レベルを増加させること、特にプロシージャー間分析を適用すること (つまり **-qipa** でコンパイルすること) は、より良好な別名割り当てに寄与します。

言語別名割り当て規則に違反するプログラムは、上述のように、一般に最適化なし、または低レベルの最適化で正しく実行されますが、より高いレベルの最適化が試行されると障害が発生する可能性があります。これは、最適化が積極的なほど別名割り当て情報はより有効に利用されるため、わずかに誤ったプログラムのセマンティクスを露呈してしまうためです。

これらの問題に関連するオプションは **-qstrict** と **-qalias** です。これらの動作は、下の表に要約されています。

プログラム動作オプション

オプション	説明
-qstrict、**-qnostrict**	コンパイラーによる浮動小数点計算の再配列と、潜在的な除外命令を可能にします。潜在的な除外命令とは、誤った実行 (たとえば、浮動小数点オーバーフロー、メモリー・アクセス違反など) のために割り込みが発生する可能性のある命令のことです。デフォルトは **-qnoopt** では **-qstrict** と **-O2**、**-O3** では **-qnostrict**、 **-O4**、および **-O5** です。
-qalias	特定の変数が重複するストレージを参照しないようにコンパイラーが想定することを可能にします。これは、Fortran における仮引き数と配列割り当ての重複にフォーカスを置くものです。

その他のパフォーマンス・オプション

オプションは、最適化の特定の性質を制御するために指定するものです。これらはグループとして使用可能にされたり、より一般的な最適化オプションが使用可能にされたときはデフォルト値として指定されることがあります。

パフォーマンス最適化のために選択されるコンパイラー・オプション

オプション	説明
-qcompact	選択が必要なときに、実行時間の削減よりも最終コード・サイズの縮小を選択します。 **-O3** 以上の最適化を制限するために使用できます。
-qsmallstack	プログラムでのスタック・ストレージの使用を制限するようにコンパイラーに指示します。これにより、ヒープ使用が増加する場合があります。
-qunroll	ループ・アンロールを独立して制御します。 **-O3** 以上の最適化で暗黙的に活動化されます。
-qtbtable	トレースバック・テーブル情報の生成を制御します。 64 ビット・モードのみ。

-qunwind このコンパイル内のルーチンがアクティブである間は、スタックがアンワインド可能であることをコンパイラーに通知します。言い換えると、コンパイラーは、アプリケーションがプログラム・スタックのアンワインド・メカニズムに頼る可能性があること、または頼ることを通知されます。

-qnounwind このコンパイル内のルーチンがアクティブである間、スタックがアンワインドされないことをコンパイラーに通知します。
-qnounwind オプションは、最適化プロローグ調整を使用可能にします。これにより、不揮発性レジスターの保管数と復元数が削減されます。

最適化したコードのデバッグ

最適化プログラムをデバッグすると、特殊な問題が発生します。最適化は演算の順序を変更し、コードを追加または削除し、作成したコードを元のソース・ステートメントと関連付けるのが難しくなるようなその他の変換を行う場合があります。たとえば、最適化プログラムは、変数の保管をすべて除去し、それをレジスターにのみ保持します。ほとんどのデバッガーは、これに従うことができず、その変数が決して更新されないように見えます。

最初にプログラムをデバッグしてから、最適化オプションで再コンパイルし、最適化したプログラムをテストしてからプログラムを製品化することをお勧めします。最適化したコードが期待どおりの結果を作成しない場合は、別のデバッグ・セッションで特定の最適化の問題を分離してください。

下の表は、最適化されたコードの開発中に役立つ、特化された情報を提供するオプションを示しています。

診断オプション

オプション	動作
-qlist	オブジェクト・リストを出力するようにコンパイラーに指示します。オブジェクト・リストには、生成された命令、トレースバック・テーブル、テキスト定数の 16 進および疑似アセンブリー表記が含まれています。
-qreport	実行したループ変換と、プログラムが並列化された方法についての報告を作成するようにコンパイラーに指示します。このオプションは、**-qhot** または **-qsmp** が指定されると使用可能になります。
-qinitauto	すべての自動変数を指定の値に初期化するコードを出力するように、コンパイラーに指示します。
-qipa=list	IPA 最適化の情報を提供するオブジェクト・リストを出力するように、コンパイラーに指示します。

最適化されたプログラムでの異なる結果

最適化されたプログラムが最適化されていないプログラムと異なる結果を発生する理由を以下に示します。

- プログラムに無効なコードが入っていると、最適化コードに障害が発生する場合があります。たとえば、呼び出されたプロシージャーの共通ブロックにもある実引き数をプログラムが渡したり、2 つ以上の仮引き数が同じ実引き数と関連付けられていると、障害が発生する場合があります。
- 最適化を行わずに動作したプログラムが、最適化を使用してコンパイルすると失敗する場合は、相互参照リストと、初期化前に使用される変数に対するプログラムの実行の流れをチェックしてください。 **-qinitauto=*hex_value*** オプションでコンパイルして、一貫して誤った結果を起こしてみてください。たとえば、**-qinitauto=FF** を使用すると、**REAL** および **COMPLEX** 変数の初期値は「負の非数字」(-NAN) になります。これらの変数で演算を行っても、結果は NAN 値になります。他のビット・パターン (***hex_value***) は異なる結果を生み、何が起きているかに関してさらなる手掛かりを提供します。 (初期化されていない変数を持つプログラム、コンパイラーが行うデフォルトでの解釈のために、最適化をせずにコンパイルすると、正しく機能しているかのように見える場合がありますが、最適化を行ってコンパイルすると、障害が発生する場合があります。同様に、プログラムは最適化を行ったときに正しく実行されているように見えることがありますが、より低レベルの最適化において、または異なる環境においては障害が発生する場合があります。)
- ストレージ内の値の検査に依存するデバッグ手法は、十分に注意して使用してください。共通式の評価が、コンパイラーにより削除または移動されている可能性があります。いくつかの変数がレジスターに割り当てられている場合がありますし、ストレージにはまったくない場合もあります。

関連情報: 94 ページの『-g オプション』、 163 ページの『-qinitauto オプション』、および 361 ページの『問題判別とデバッグ』を参照してください。

コンパイラー・フレンドリーなプログラミング

コンパイラー・フレンドリーなプログラミング・イディオムは、オプションやディレクティブのようにパフォーマンスに有効である場合があります。以下にいくつかの提案を示します。

一般

- 可能であれば、**xlf90** または **xlf95** のようなコマンド呼び出しを使用して、標準の準拠性とコード移植性を高めてください。これが不可能な場合は、**-qnosave** オプションを使用して、すべてのローカル変数を自動化してください。これにより、さらに最適化の機会がもたらされます。
- モジュールを使用して、関連するサブルーチンおよび関数をグループ化してください。
- カスタム・インプリメンテーションまたは汎用ライブラリーではなく、高度にチューニングされた **MASS** および **ESSL** ライブラリー を使用することを検討してください。

手作業によるチューニング

- 手作業によるコードの最適化を過度に行ってはいけません。異常な構文はコンパイラー (および他のプログラマー) を混乱させ、新しいマシンでのアプリケーションの最適化を困難なものにします。

- インライン化による小さな関数の手作業によるチューニングを限定的に行ってください。
- プログラムを数多くの小さな関数に過度に分割すると、呼び出しのオーバーヘッドの処理にプログラムが費やす時間の比率が増えるため、これは避けてください。小さな関数を数多く使用することを選択する場合は、**-qipa** を使用することを検討してください。

変数

- グローバル変数およびポインターの不要な使用は避けてください。これらをループ内で使用するときは、ループの前にローカル変数にロードし、ループの後で保管し戻してください。
- **INTENT** ステートメントを使用して、パラメーターの使用を記述してください。

ストレージの節約

- スカラーには、レジスター・サイズの整数 (INTEGER(4) または INTEGER(8) データ型) を使用してください。
- 計算に適した、最小の浮動小数点精度を使用してください。
- 新しいコードを作成するときは、グローバル・ストレージの共通ブロックではなくモジュール変数を使用してください。
- **CONTAINS** ステートメントは、スレッド・ローカル・ストレージを共用するときにのみ使用してください。

ポインター

- すべての言語別名割り当て規則に従ってください。 **-qalias=nostd** の使用は避けるようにしてください。
- **ALLOCATABLE** 配列と **POINTER** 変数は、動的割り振りを必要とする状況に限って使用するようにしてください。

配列

- ループ指標変数および境界には、可能であれば常にローカル変数を使用してください。
- 配列指標式は、可能な限りシンプルにしてください。指標付けが間接であることを必要とする場合は、**PERMUTATION** ディレクティブの使用を検討してください。
- 配列割り当てまたは **WHERE** ステートメントの使用時には、**-qlist** または **-qreport** で生成されたコードには特に注意してください。パフォーマンスが不適当なものである場合は、-qhot の使用を検討するか、または配列言語をループ形式で書き直してください。

XL Fortran 入出力のインプリメンテーションの詳細

本節では、Linux ファイル・システムに対する XL Fortran サポートについて (拡張機能、プラットフォーム固有の詳細を通して) 説明します。

関連情報: 205 ページの『-qposition オプション』および 335 ページの『混合言語の入出力』を参照してください。

ファイル形式のインプリメンテーション

XL Fortran は、以下のような方法でファイルをインプリメントします。

不定様式順次アクセスファイル

レコードの長さを含む整数が各レコードの前後に入れられます。整数の長さは 4 バイトです (32 ビット・アプリケーションの場合)。 **uwidth** 実行時オプションを 32 (デフォルト) に設定する場合は 4 バイト長で、64 ビット・アプリケーション用に **uwidth** を 64 に設定する場合は 8 バイト長です。

定様式順次アクセスファイル

XL Fortran プログラムは、定様式順次ファイルを読み取るとき、改行文字 (X'0A') が現れるたびにそれをレコード分離文字として使用して、ファイルをいくつかのレコードに分割します。

出力時には、入出力システムは各レコードの終わりに改行文字を書き込みます。プログラム自体も改行文字を書き込むことができますが、この方法はお勧めできません。単一レコードが書き込まれるように見えても、読み取りまたはバックスペースされるときには複数のレコードとして処理されるからです。

直接アクセス・ファイル

XL Fortran は、長さが XL Fortran ファイルのレコード長の倍数であるファイルで直接アクセス・ファイルをシミュレートします。ユーザーは **OPEN** ステートメントに、直接アクセス・ファイルのレコード長 (**RECL**) を指定する必要があります。 XL Fortran はこのレコードを使用して、それぞれのレコードを区別します。

たとえば、レコード長が 100 バイトの直接アクセス・ファイルの 3 番目のレコードは、Linux ファイルの単一レコードの 201 番目のバイトから始まり、300 番目のバイトで終わります。

直接アクセス・ファイルのレコードの長さがレコードに書き込みたいデータの量よりも多い場合、XL Fortran はレコードの右側をブランク (X'20') で埋め込みます。

ストリーム・アクセス不定様式ファイル:

不定様式ストリーム・ファイルは、ファイル記憶単位の集まりとして表示されます。 XL Fortran では、ファイル記憶単位は 1 バイトです。

不定様式ストリーム・アクセス用に接続されたファイルには、次のような特性があります。

- 最初のファイル記憶単位の位置は 1 です。後続のファイル記憶単位はそれぞれ、先行する記憶単位より 1 大きい位置を持ちます。
- 位置付けが可能なファイルの場合、ファイル記憶単位は、その位置の順序で読み取りまたは書き込みが行われる必要はありません。ファイルの作成以降にファイル記憶単位が書き込まれ、接続用の READ ステートメントが許可されている場合、装置に接続されている間は、どのようなファイル記憶単位もファイルから読み取ることができます。

ストリーム・アクセス定様式ファイル:

定様式ストリーム・アクセス用に接続されたレコード・ファイルには、次のような特性があります。

- ファイル記憶単位の一部がレコード・マーカーを表しています。レコード・マーカーとは 改行文字 (X'0A') です。
- ファイルはストリーム構造体に加えて、レコード構造体を持ちます。
- レコード構造体は、ファイルに保管されたレコード・マーカーから推測されます。
- レコードは、XL Fortran で許されている内部制限までの長さを持つことができます (403 ページの『付録 D. XL Fortran 内部制限』を参照してください。)
- ファイルの終わりには、レコード・マーカーがある場合とない場合があります。ファイルの終わりにレコード・マーカーがない場合、最終レコードは不完全ですが、空ではありません。

定様式ストリーム・アクセス用に接続されたファイルには、次のような特性があります。

- 最初のファイル記憶単位の位置は 1 です。後続のファイル記憶単位はそれぞれ、先行する記憶単位より大きい位置を持ちます。不定様式ストリーム・アクセスとは異なり、連続するファイル記憶単位の位置は常に連続的なものになります。
- 定様式ストリーム・アクセス用に接続されるファイルの位置は、**INQUIRE** ステートメントの **POS=** 指定子で決定することができます。
- 位置付けが可能なファイルの場合、ファイル位置は以前に **INQUIRE** ステートメントの **POS=** 指定子で指定された値に設定することができます。

ファイル名

ファイル名は、相対ファイル名 (たとえば **file**、**dir/file**、**../file**) としても、絶対ファイル名 (たとえば **/file**、**/dir/file**) としても指定できます。 I/O ステートメントに相対パス名だけを指定する場合でも、ファイル名 (絶対パス名) の最大長は 4095 文字です。パスなしのファイル名の最大長は 255 文字です。

以下のような場所に、有効なファイル名を指定する必要があります。

- **OPEN** および **INQUIRE** ステートメントの **FILE=** 指定子
- **INCLUDE** 行

関連情報: 位置が環境変数に依存しているファイルを指定する場合は、**GETENV** 組み込みプロシージャーを使用してその環境変数の値を検索することができます。

```
character(100) home, name
call getenv('HOME', value=home)
! Now home = $HOME + blank padding.
! Construct the complete path name and open the file.
name=trim(home) // '/remainder/of/path'
open (unit=10, file=name)
...
end
```

事前接続ファイルおよび暗黙接続ファイル

装置 0、5、6 はプログラムの実行前に標準エラー、標準入力、標準出力にそれぞれ事前接続されています。

オープンされていない装置上で **ENDFILE**、**PRINT**、**READ**、 **REWIND**、**WRITE** ステートメントなどが実行されるときに、上記以外のすべての装置を暗黙接続することができます。装置 *n* は **fort.***n* という名前のファイルに接続されます。これらのファイルは存在する必要はなく、ユーザーがそれらの装置を使用しなければ、XL Fortran はそれらのファイルを作成しません。

注: 装置 0 は標準エラーに対して事前接続されているので、**CLOSE**、**ENDFILE**、**BACKSPACE**、**REWIND**、直接またはストリーム入出力などのステートメントでこの装置を使用することができません。 **BLANK=**、**DELIM=**、**PAD=** などの指定子の値を変更する目的でのみ、**OPEN** ステートメントでこの装置を使用することができます。

装置 5 と 6 (そして *) も、**CLOSE** ステートメントの後の I/O ステートメントによって暗黙接続することができます。

```
      WRITE (6,10) "This message goes to stdout."
      CLOSE (6)
      WRITE (6,10) "This message goes in the file fort.6."
      PRINT *, "Output to * now also goes in fort.6."
10    FORMAT (A)
      END
```

暗黙接続されたファイルの **FORM=** 指定子は、装置上で **READ**、 **WRITE**、**PRINT** ステートメントが実行される前は、値 **FORMATTED** を持っています。このようなファイル上では最初のステートメントが、そのポイントの **FORM=** 指定子を決定します。つまり、ステートメントの形式制御が形式指示、リスト指示、名前リストのいずれかの場合は **FORMATTED** で、ステートメントが不定様式の場合は **UNFORMATTED** です。

また、事前接続ファイルは、デフォルト指定子として **FORM='FORMATTED'**、**STATUS='OLD'**、**ACTION='READWRITE'** を持っています。

事前接続ファイルまたは暗黙接続ファイルのその他の特性は、 **OPEN** ステートメントのデフォルト指定子です。これらのファイルは、常に順次アクセスを使用します。

XL Fortran で、**fort.***n* ファイルの代わりに独自のファイルを使用したい場合は、**OPEN** ステートメントでその装置に対して独自のファイルを指定することもできま

すし、アプリケーションを実行する前にシンボリック・リンクを作成することもできます。下の例では、**myfile** と **fort.10** の間にシンボリック・リンクを作成します。

```
ln -s myfile fort.10
```

事前接続されているファイル **fort.10** を入出力のために使用するアプリケーションを実行すると、ファイル **myfile** が代わりに使用されます。ファイル **fort.10** は存在しますが、シンボリック・リンクとしてのみ存在します。次のコマンドはシンボリック・リンクを除去しますが、**myfile** の存在には影響を与えません。

```
rm fort.10
```

ファイルの位置決め

表 18. POSITION= 指定子を指定しないでファイルがオープンされたときのファイル・ポインターの位置

-qposition サブオプション	**暗黙 OPEN**		**明示 OPEN**					
			STATUS = 'NEW'		**STATUS = 'OLD'**		**STATUS = 'UNKNOWN'**	
	ファイルが存在する場合	**ファイルが存在しない場合**	**ファイルが存在する場合**	**ファイルが存在しない場合**	**ファイルが存在する場合**	**ファイルが存在しない場合**	**ファイルが存在する場合**	**ファイルが存在しない場合**
オプションが指定されていない	Start	Start	Error	Start	Start	Error	Start	Start
appendold	Start	Start	Error	Start	End	Error	Start	Start
appendunknown	Start	Start	Error	Start	Start	Error	End	Start
appendold および appendunknown	Start	Start	Error	Start	End	Error	End	Start

I/O のリダイレクト

XL Fortran プログラムへの入出力は、コマンド行でリダイレクト演算子を使用してリダイレクトすることができます。この演算子の指定および使用方法は、どのシェルを実行しているかによって異なります。以下に **bash** の例を挙げます。

```
$ cat redirect.f
        write (6,*) 'This goes to standard output'
        write (0,*) 'This goes to standard error'
        read (5,*) i
        print *,i
        end
$ xlf95 redirect.f
** _main    === End of Compilation 1 ===
1501-510  Compilation successful for file redirect.f.
$ # No redirection. Input comes from the terminal. Output goes to
$ # the screen.
$ a.out
 This goes to standard output
 This goes to standard error
4
 4
$ # Create an input file.
$ echo >stdin 2
$ # Redirect each standard I/O stream.
$ a.out >stdout 2>stderr <stdin
$ cat stdout
 This goes to standard output
 2
$ cat stderr
 This goes to standard error
```

リダイレクトの詳細については、man ページを参照してください。

パイプ、スペシャル・ファイル、リンクとの XLF I/O 対話方法

通常のオペレーティング・システム・ファイルとブロック特殊ファイルは、順次アクセス方式、直接アクセス方式、またはストリーム・アクセス方式でアクセスすることができます。

疑似装置、パイプ、キャラクター特殊ファイルは、順次アクセス方式、または **POS=** 指定子を使用しないストリーム・アクセス方式でのみアクセスが可能です。

ファイルがリンクされると、それらのファイル名を相互に交換して使用することができます。

```
OPEN (4, FILE="file1")
OPEN (4, FILE="link_to_file1", PAD="NO") ! Modify connection
```

パイプに対する **REWIND** または **APPEND** として **POSITION=** 指定子を指定しないでください。

パイプに **ACTION='READWRITE'** と指定しないでください。

疑似装置またはキャラクター特殊ファイルであるファイル上では、**BACKSPACE** ステートメントは使用しないでください。

疑似装置またはパイプになるファイルでは、**REWIND** ステートメントは使用しないでください。

デフォルトのレコード長

疑似装置、パイプ、キャラクター特殊ファイルなどが、**RECL=** 指定子を指定しない定様式または不定様式順次アクセス、あるいは定様式ストリーム・アクセスのために接続されている場合、デフォルトのレコード長は 2 147 483 647 ではなく 32 768 で、これはランダム・アクセス装置に接続されている順次アクセス・ファイルのデフォルトです。 (**default_recl** 実行時オプションを参照してください。)

標準出力に長いレコードを書き込むプログラムに対応するため、定様式ファイルに対するデフォルトの最大レコード長が増やされている場合もあります。ユニットを端末に接続して定様式ファイルの順次アクセスを行う場合は、明示的な **RECL=** 修飾子が **OPEN** ステートメント内にないと、プログラムは通常のデフォルト値である 32,768 バイトではなく、2,147,483,646 (2**31-2) バイトの最大レコード長を使用します。最大レコード長が増大する場合は、定様式 I/O に 1 つの制限があります。すなわち、**T** または **TL** 編集記述子を使用する **WRITE** ステートメントは 32 768 バイトより多く書き込んではなりません。なぜならユニットの内部バッファーが 32 768 バイトごとにフラッシュされ、**T** または **TL** 編集記述子はこの境界を超えて戻ることができなくなるからです。

ファイル許可

ファイルは、そのファイルで実行する対応の操作に応じて、適切な許可 (読み取り許可、書き込み許可、またはその両方) を取得する必要があります。

ファイルの作成時におけるデフォルト許可 (**umask** 設定が 000 の場合) は、ユーザー、グループ、オープンその他に対して、読み取りと書き込みの両方です。個々の許可ビットは、プログラムの実行前に **umask** 設定を変更することによって、オフにすることができます。

エラー・メッセージと回復処置の選択

デフォルトでは、ステートメントが **ERR=** または **IOSTAT=** 指定子を持っていない場合でも、多数の種類のエラーを検出した後、XLF コンパイル済みプログラムは処理を続行します。このプログラムは、欠陥データその他の問題から正常に回復できるように処置をいくつか実行します。

エラーを検出するプログラムの動作を制御するには、プログラムを実行する前に **XLFRTEOPTS** 環境変数を設定します。この環境変数については、39 ページの『実行時オプションの設定』で説明されています。

- エラーを検出した場合、回復処置を実行しないでプログラムを停止させるには、**XLFRTEOPTS** 設定で **err_recovery=no** を入れてください。
- エラー検出時のメッセージを作成してプログラムを停止させるには、**xrf_messages=no** を入れてください。
- 実行時に XL Fortran 拡張を Fortran 90 に許可しない場合は、**langlvl=90std** を入れてください。実行時に XL Fortran 拡張を Fortran 95 に許可しない場合は、**langlvl=95std** を入れてください。実行時に XL Fortran 拡張を Fortran 2003 の動作で許可しないようにする、**langlvl=2003std** を入れてください。 **-qlanglvl** コン

パイラー・オプションと組み合わせてこの設定を行うと、あるプログラムを別のプラットフォームに移植する準備をしているときに拡張機能を見つけるのに役立ちます。

たとえば、次のようになります。

```
# Switch defaults for some run-time settings.
XLFRTEOPTS="err_recovery=no:cnverr=no"
export XLFRTEOPTS
```

環境変数の設定とは無関係に、プログラムを常に同じように機能させたい場合、またはプログラムのさまざまな部分での動作を変更したい場合は、**SETRTEOPTS** プロシージャーを呼び出すことができます。

```
PROGRAM RTEOPTS
USE XLFUTILITY
CALL SETRTEOPTS("err_recovery=no") ! Change setting.
... some I/O statements ...
CALL SETRTEOPTS("err_recovery=yes") ! Change it back.
... some more I/O statements ...
END
```

ユーザーは環境変数 **XLFRTEOPTS** を使用してこれらの設定を変更できるため、プログラムの動作に望みどおりの影響を与える実行時オプションの設定には、必ず **SETRTEOPTS** を使用してください。

I/O バッファーのフラッシュ

プログラムが予期せずに終了した場合にデータが失われないようにするために、**FLUSH** ステートメントまたは **flush_** サブルーチンを使用して、バッファーに入れたデータをファイルに書き込むことができます。(移植性を高めるためには、**FLUSH** ステートメントが推奨されます。) 次の例は、**flush_** サブルーチンの使用法を示しています。

```
USE XLFUTILITY
INTEGER, PARAMETER :: UNIT=10

DO I=1,1000000
    WRITE (10,*) I
    CALL MIGHT_CRASH
! If the program ends in the middle of the loop, some data
! may be lost.
END DO

DO I=1,1000000
    WRITE (10,*) I
    CALL FLUSH_(UNIT)
    CALL MIGHT_CRASH
! If the program ends in the middle of the loop, all data written
! up to that point will be safely in the file.
END DO

END
```

関連情報: 335 ページの『混合言語の入出力』 および「*XL Fortran ランゲージ・リファレンス*」の『**FLUSH**』ステートメントを参照してください。

入出力ファイルの位置と名前の選択

入出力ファイルのデフォルトの位置と名前をオーバーライドする場合は、ソース・コードに変更を加えずに次の方法を使うことができます。

明示的な名前に接続されていないファイルの命名

通常なら **fort.***unit* という形式の名前が付くファイルへ特定の名前を指定するには、まず実行時オプションの **unit_vars** を設定し、次に、個々のスクラッチ・ファイルごとに **XLFUNIT_***unit* という形式の名前が付いた環境変数を設定する必要があります。 Fortran プログラム内の装置番号と、ファイル・システム内のパス名とが関連付けられます。

たとえば、Fortran プログラムに次のステートメントが入っているとします。

```
OPEN (UNIT=1, FORM='FORMATTED', ACCESS='SEQUENTIAL', RECL=1024)
...
OPEN (UNIT=12, FORM='UNFORMATTED', ACCESS='DIRECT', RECL=131072)
...
OPEN (UNIT=123, FORM='UNFORMATTED', ACCESS='SEQUENTIAL', RECL=997)
```

```
XLFRTEOPTS="unit_vars=yes"     # Allow overriding default names.
XLFUNIT_1="/tmp/molecules.dat" # Use this named file.
XLFUNIT_12="../data/scratch"   # Relative to current directory.
XLFUNIT_123="/home/user/data"  # Somewhere besides /tmp.
export XLFRTEOPTS XLFUNIT_1 XLFUNIT_12 XLFUNIT_123
```

注:

1. **XLFUNIT_***number* 変数名は大文字でなければならず、*number* に先行ゼロが付いていてはなりません。
2. **unit_vars=yes** は、他に設定した実行時オプションによっては、**XLFRTEOPTS** 変数の値の一部にすぎない場合もあります。 **XLFRTEOPTS** 値の一部になり得る他のオプションについては、39 ページの『実行時オプションの設定』を参照してください。
3. **unit_vars** 実行時オプションが **no** に設定されているか、未定義である場合、あるいは、プログラムの実行時に適当な **XLFUNIT_***number* 変数が設定されていない場合、プログラムはファイルにデフォルトの名前 (**fort.***unit*) を使用し、ファイルを現行ディレクトリーに入れます。

スクラッチ・ファイルの命名

すべてのスクラッチ・ファイルを特定のディレクトリーへ入れるには、**TMPDIR** 環境変数にそのディレクトリーの名前を設定します。そのようにした場合、プログラムはそのディレクトリーに入っているスクラッチ・ファイルをオープンします。これは、**/tmp** ディレクトリーがスクラッチ・ファイルを入れるには小さすぎる場合などに行う必要があります。

スクラッチ・ファイルに特定の名前を指定するには、以下のことを行ってください。

1. 実行時オプションの **scratch_vars** を設定します。
2. 個々のスクラッチ・ファイルごとに **XLFSCRATCH_***unit* という形式の名前が付いた環境変数を設定します。

Fortran プログラム内の装置番号と、ファイル・システム内のパス名とが関連付けられます。その場合、スクラッチ・ファイルの位置は **TMPDIR** 変数の影響を受けません。

たとえば、Fortran プログラムに次のステートメントが入っているとします。

```
      OPEN (UNIT=1, STATUS='SCRATCH', &
            FORM='FORMATTED', ACCESS='SEQUENTIAL', RECL=1024)
      ...
      OPEN (UNIT=12, STATUS='SCRATCH', &
            FORM='UNFORMATTED', ACCESS='DIRECT', RECL=131072)
      ...
      OPEN (UNIT=123, STATUS='SCRATCH', &
            FORM='UNFORMATTED', ACCESS='SEQUENTIAL', RECL=997)
```

```
XLFRTEOPTS="scratch_vars=yes"     # Turn on scratch file naming.
XLFSCRATCH_1="/tmp/molecules.dat" # Use this named file.
XLFSCRATCH_12="../data/scratch"   # Relative to current directory.
XLFSCRATCH_123="/home/user/data"  # Somewhere besides /tmp.
export XLFRTEOPTS XLFSCRATCH_1 XLFSCRATCH_12 XLFSCRATCH_123
```

注:

1. **XLFSCRATCH_***number* 変数名は大文字でなければならず、*number* に先行ゼロが付いている必要があります。
2. **scratch_vars=yes** は、他に設定した実行時オプションによっては、**XLFRTEOPTS** 変数の値の一部にすぎない場合もあります。 **XLFRTEOPTS** 値の一部になり得る他のオプションについては、39 ページの『実行時オプションの設定』を参照してください。
3. **scratch_vars** 実行時オプションが **no** に設定されているか、未定義である場合、あるいはプログラムの実行時に適当な **XLFSCRATCH_***number* 変数が設定されていない場合、プログラムはスクラッチ・ファイルに固有の名前を選択し、そのファイルを **TMPDIR** 変数で指定されたディレクトリーか、 **TMPDIR** 変数が指定されていない場合は **/tmp** ディレクトリーに入れます。

非同期 I/O

大量に上るデータの I/O を実行する科学プログラムでは、速度と効率を高めるために非同期 I/O が必要とされる場合があります。同期 I/O では、I/O 操作が完了するまでアプリケーションの実行がブロックされます。非同期 I/O では、I/O 操作をバックグラウンドで実行しながら、アプリケーションによる処理を続行できます。処理と I/O 操作を並行して行う能力を使用する場合はアプリケーションを修正できます。独立した装置に常駐する複数のファイルに対して、複数の非同期 I/O 操作を同時に実行することもできます。この機能を使用するために必要な構文および言語エレメントに関する詳細な説明については、「*XL Fortran ランゲージ・リファレンス*」で以下のトピックを参照してください。

- **INQUIRE** ステートメント
- **READ** ステートメント
- **WAIT** ステートメント
- **WRITE** ステートメント

非同期データ転送操作の実行

非同期データ転送操作を実行すると、以下のステップを指定順序で実行した場合と同様の効果が得られ、ステップ (6) から (9) が (可能であれば) 非同期で発生します。

1. データ転送の方法を判別します。
2. 装置を識別します。
3. 形式がある場合はその形式を明確化します。
4. エラー条件、ファイルの終わり条件、またはレコードの終わり条件が発生したかどうかを判別します。
5. データ転送ステートメント中の **IOSTAT=** 指定子で指定した変数が定義されるようにします。
6. データ転送に先立ってファイルを位置決めします。
7. 入出力リスト (もしあれば) で指定されたファイルとエンティティーとの間でデータを転送します。
8. エラー条件、ファイルの終わり条件、またはレコードの終わり条件が発生したかどうかを判別します。
9. データ転送後にファイルを位置決めします。
10. **WAIT** ステートメント中の **IOSTAT=** および **SIZE=** 指定子で指定された変数があれば、その変数を定義するようにします。

使用法

Fortran で非同期データ転送を開始するには、Fortran の非同期 **READ** および **WRITE** ステートメントを使用します。実際のデータ転送が完了したかどうかに関係なく、非同期 I/O ステートメント後も実行が継続されます。

WAIT ステートメントを使えば、先に開始された非同期 I/O ステートメントとプログラムを同期化することができます。 **WAIT** ステートメントには 2 つの形式があります。

1. **DONE=** 指定子のない **WAIT** ステートメントでは以下のように、対応する非同期 I/O ステートメントが完了するまで、 **WAIT** ステートメントが実行を一時停止します。

```
integer idvar
integer, dimension(1000):: a
....
READ(unit_number,ID=idvar) a
....
WAIT(ID=idvar)
....
```

2. **DONE=** 指定子がある **WAIT** ステートメントでは以下のように、**WAIT** ステートメントが非同期 I/O ステートメントの完了状況を戻します。

```
integer idvar
logical done
integer, dimension(1000):: a
....
READ(unit_number,ID=idvar) a
....
WAIT(ID=idvar, DONE=done)
....
```

   DONE= 指定子で指定した変数は、対応する非同期 I/O ステートメントが完了している場合は、真に設定されます。それ以外の場合は、偽に設定されます。

実際のデータ転送は、以下の場合に起こると思われます。

- 非同期 **READ** または **WRITE** ステートメントの間
- 対応する **WAIT** ステートメント実行前の任意の時刻

- 対応する **WAIT** ステートメントの間

非同期 I/O の性質上、要求の実際の完了時刻は予測できません。

Fortran 非同期 **READ** および **WRITE** ステートメントは、**ID=** 指定子を使って指定します。非同期 **READ** または **WRITE** ステートメントによる **ID=** 指定子の値セットは、対応する **WAIT** ステートメント中の **ID=** 指定子と同じでなければなりません。関連した非同期 I/O ステートメントが完了するまでこの値を保持しておく責任はプログラマーにあります。

以下のプログラムでは、有効な非同期 **WRITE** ステートメントを示します。

```
        program sample0
        integer, dimension(1000):: a
        integer idvar
        a = (/(i,i=1,1000)/)
        WRITE(10,ID=idvar) a
        WAIT(ID=idvar)
        end
```

XL Fortran では、関連する **WAIT** ステートメントの前の非同期 I/O 識別子の値が破棄されるため、以下のプログラムは無効です。

```
        program sample1
        integer, dimension(1000):: a
        integer idvar
        a = (/(i,i=1,1000)/)
        WRITE(10,ID=idvar) a
        idvar = 999     ! Valid id is destroyed.
        WAIT(ID=idvar)
        end
```

非同期 I/O を使用するアプリケーションは一般に、I/O 操作を並列処理してパフォーマンスを向上させます。以下に簡単な例を示します。

```
        program sample2
        integer  (kind=4), parameter :: isize=1000000, icol=5
        integer  (kind=4) :: i, j, k
        integer  (kind=4), dimension(icol) :: handle
        integer  (kind=4), dimension(isize,icol), static :: a, a1

!
!       Opens the file for both synchronous and asynchronous I/O.
!
        open(20,form="unformatted",access="direct", &
           status="scratch", recl=isize*4,asynch="yes")

!
!       This loop overlaps the initialization of a(:,j) with
!       asynchronous write statements.
!
!       NOTE: The array is written out one column at a time.
!             Since the arrays in Fortran are arranged in column
!             major order, each WRITE statement writes out a
!             contiguous block of the array.
!
        do 200 j = 1, icol
           a(:,j) = (/ (i*j,i=1,isize) /)
           write(20, id=handle(j), rec=j) a(:,j)
200     end do

!
!       Wait for all writes to complete before reading.
!
```

```
        do 300 j = 1, icol
           wait(id=handle(j))
300     end do

!
!       Reads in the first record.
!
        read(20, id=handle(1), rec=1) a1(:,1)

        do 400 j = 2, icol
           k = j - 1
!
!          Waits for a previously initiated read to complete.
!
           wait(id=handle(k))
!
!          Initiates the next read immediately.
!
           read(20, id=handle(j), rec=j) a1(:,j)
!
!          While the next read is going on, we do some processing here.
!
           do 350 i = 1, isize
              if (a(i,k) .ne. a1(i,k)) then
                 print *, "(",i,",",k,") &
                 &  expected ", a(i,k), " got ", a1(i,k)
              end if
350        end do
400     end do

!
!       Finish the last record.
!
        wait(id=handle(icol))

        do 450 i = 1, isize
           if (a(i,icol) .ne. a1(i,icol)) then
              print *, "(",i,",",icol,") &
              &  expected ", a(i,icol), " got ", a1(i,icol)
           end if
450     end do

        close(20)
        end
```

パフォーマンス

非同期 I/O の利点を最大に生かすには、非同期 I/O を大量の連続データ項目に対して実行することをお勧めします。

大量の小さな項目に対して非同期 I/O を実行することも可能ですが、パフォーマンス自体は悪化します。これは、非同期 I/O で各項目を保守するために余分の処理オーバーヘッドが必要とされるという状況に起因します。非同期 I/O を大量の小さな項目に対して実行することは控えたほうがよいでしょう。以下に例をいくつか示します。

1. WRITE(unit_number, ID=idvar) a1(1:100000000:2)
2. WRITE(unit_number, ID=idvar) (a2(i,j),j=1,100000000)

不定様式順次ファイルに非同期 I/O を実行する場合、各レコードには異なる長さがあり、それらの長さはレコードそのものと一緒に格納されるので、効率は劣るものになります。非同期 I/O の利点を最大に引き出すには、できれば不定様式直接アクセスまたは不定様式ストリーム・アクセスを使用することをお勧めします。

コンパイラーで生成する一時 I/O 項目

状況によっては、コンパイラーで一時変数を生成して、I/O 項目式の結果を保持する必要がある場合もあります。そのような場合、I/O ステートメントでどのようなモードが指定されていても、一時変数に対しては非同期 I/O が実行されます。以下に、その事例を示します。

1. **READ** で、入力項目にベクトル添え字を持つ配列が現れる場合
   a.
   ```
   integer a(5), b(3)

   b = (/1,3,5/)
   read(99, id=i) a(b)
   ```
   b.
   ```
   real a(10)
   read(99,id=i) a((/1,3,5/))
   ```
2. **WRITE** で、出力項目が定数の式、または特定の派生型の定数である場合
   a.
   ```
   write(99,id=i) 1000
   ```
   b.
   ```
   integer a
   parameter(a=1000)

   write(99,id=i) a
   ```

c.
```
type mytype
integer a
integer b
end type mytype

write(99,id=i) mytype(4,5)
```

3. **WRITE** で、出力項目が一時変数である場合

a.
```
write(99,id=i) 99+100
```

b.
```
write(99,id=i) a+b
```

c.
```
external ff
real(8) ff

write(99,id=i) ff()
```

4. **WRITE** で、出力項目が配列コンストラクターである式の場合

```
write(99,id=i) (/1,2,3,4,5/)
```

5. **WRITE** で、出力項目がスカラー型配列である場合

```
integer a(5),b(5)
write(99,id=i) a+b
```

エラー処理

非同期データ転送では、データ転送ステートメントの実行中または後続のデータ転送中にエラーまたはファイルの終わり条件が発生する場合があります。プログラムの終了時にこれらの条件が発生しない場合、プログラマーは、データ転送または **WAIT** ステートメントの突き合わせで **ERR=**、**END=**、および **IOSTAT=** 指定子を使ってそれらの条件を検出できます。

IOSTAT= 指定子も **ERR=** 指定子も入っていない I/O ステートメントの実行中、またはその後続データ転送中にエラー条件が発生した場合、プログラムの実行が終了します。回復可能エラーの場合は、**IOSTAT=** および **ERR=** 指定子が存在しなければ、**err_recovery** 実行時オプションが **no** に設定されている場合、プログラムは終了します。 **err_recovery** 実行時オプションが **yes** に設定されていれば、回復処置が発生し、プログラムは続行します。

非同期データ転送ステートメントが原因で以下の状況が発生する場合、**ID=** 値が定義されていないので、**WAIT** ステートメントは許可されません。

- **ERR=** または **END=** で指定されたラベルへの分岐
- ゼロ以外の値に設定される **IOSTAT=** 指定子

XL Fortran スレッド・セーフ I/O ライブラリー

XL Fortran スレッド・セーフ I/O ライブラリー **libxlf90_r.so** では、Fortran I/O ステートメントの並列実行がサポートされます。並列ループで I/O ステートメントが入っている、または異なるスレッドからマルチスレッドを作成すると同時に I/O ステートメントを実行するプログラムでは、このライブラリーを使用する必要があります。つまり、Fortran I/O を並列実行する場合、期待どおりの結果を得るには、アプリケーションをこのライブラリーとリンクしなければなりません。

I/O 操作の同期化

並列実行中には、マルチスレッドで I/O 操作が同じファイル上で同時に実行される場合があります。操作が同期化されていなければ、これらの I/O 操作の結果は切り捨てられるか結合される、またはその両方が当てはまる場合があり、アプリケーシ

ョンは間違った結果を作成し、壊れる場合さえあります。 XL Fortran スレッド・セーフ I/O ライブラリーは並列アプリケーションの I/O 操作を同期化します。同期化は I/O ライブラリー内で実行され、アプリケーション・プログラムに対して透過的です。同期化の目的は、個々の I/O 操作の整合性と正確さを保証することにあります。ただし、スレッド・セーフ I/O ライブラリーは、スレッドが I/O ステートメントを実行する順序を制御しません。したがって、並列 I/O 操作では、読み取ったり書き出したりするレコードの順序を予測することはできません。詳細については、『並列 I/O の問題』を参照してください。

外部ファイル: 外部ファイルの場合、同期化は装置単位で実行されます。 XL Fortran スレッド・セーフ I/O ライブラリーでは必ず、特定の論理装置にアクセスするスレッドが 1 つだけになり、いくつかのスレッドが互いに干渉することがなくなります。スレッドがある装置上で I/O 操作を実行しているときに同じ装置上で I/O 操作を実行しようとする別のスレッドは、最初のスレッドが操作を終了するまで待機しなければなりません。したがって、同じ装置上のマルチスレッドによる I/O ステートメントの実行は逐次化されます。しかし、スレッド・セーフ I/O ライブラリーによって、スレッドが別の論理装置上で並列操作を行わなくなるということはありません。つまり、異なる論理装置への並列アクセスは実質的に逐次化されません。

同期化における I/O の機能性: XL Fortran スレッド・セーフ I/O ライブラリーは、論理装置へのアクセスを同期化するために内部ロックを設定します。これによって、Fortran プログラムで実行される I/O 操作に機能面での影響が及ぶことはありません。また、Fortran I/O ステートメントの操作性に付加的な制限が課されることもありません。ただし、非同期的に起動されるシグナル・ハンドラーで I/O ステートメントを使用する場合はこの限りではありません。詳細については、331 ページの『シグナル・ハンドラーでの I/O ステートメントの使用』を参照してください。

Fortran の規格では、関数参照によって別の I/O ステートメントが実行される場合、I/O ステートメントのどの場所でも、式の中に関数参照を入れることが禁止されています。この制限は引き続き XL Fortran スレッド・セーフ I/O ライブラリーに適用されます。

並列 I/O の問題

並列スレッドが I/O 操作を実行する順序は予測できません。 XL Fortran スレッド・セーフ I/O ライブラリーでは、順序付けが制御されず、どのようなスレッドでも特定の論理装置上の I/O ステートメントを実行し、そのロックを取得するものであれば、まずそのスレッドが操作を行うことになります。したがって、並列 I/O は、少なくとも以下のいずれかが真の場合にのみ使用できます。

- 各スレッドは、異なる直接アクセス・ファイル中の事前決定したレコードに対して I/O を実行します。
- 各スレッドは、ストリーム・アクセス・ファイルの異なる部分で I/O を実行します。異なる I/O ステートメントがファイルの同じ、または重複した領域を使用することはできません。
- 結果は、レコードの書き込みまたは読み取りの順序には左右されません。
- 各スレッドは、異なるファイル上で I/O を実行します。

これらの事例では、I/O 操作の結果はスレッドが実行される順序に依存していません。ただし、複数のスレッドから同じ論理装置に並列アクセスを行うと、I/O ライ

ブラリーよる逐次化が行われるため、パフォーマンスは思ったほど向上しない場合があります。これらの事例を、以下に例で示します。

- 各スレッドは、直接アクセス・ファイル中の事前決定したレコードに対して I/O を実行します。

```
do i = 1, 10
  write(4, '(i4)', rec = i) a(i)
enddo
```

- 各スレッドは、ストリーム・アクセス・ファイルの異なる部分で I/O を実行します。異なる I/O ステートメントがファイルの同じ、または重複した領域を使用することはできません。

```
do i = 1, 9
  write(4, '(i4)', pos = 1 + 5 * (i - 1)) a(i)
  ! We use 5 above because i4 takes 4 file storage
  ! units + 1 file storage unit for the record marker.
enddo
```

- スレッドはファイルに接続された論理装置の状況を共用するため、各スレッドが異なるファイル上で実行される場合は、論理装置の状況の検索または更新のいずれかを行うために、そのスレッドでは、依然として論理装置のロックを取得する必要があります。ただし、スレッド・セーフ I/O ライブラリーを使用すると、スレッドは、論理装置と I/O リスト項目間でデータ転送を並列に実行することができます。アプリケーションの並列領域に大量の小さな I/O 要求がある場合は、ロックの競合のために、期待したパフォーマンスを得ることができない可能性があります。次の例を考慮します。

```
      program example

      use omp_lib

      integer, parameter :: num_of_threads = 4, max = 5000000
      character*10 file_name
      integer i, file_unit, thread_id
      integer, dimension(max, 2 * num_of_threads) :: aa

      call omp_set_num_threads(num_of_threads)

!$omp parallel private(file_name, thread_id, file_unit, i) shared(aa)

      thread_id = omp_get_thread_num()
      file_name = 'file_'
      file_name(6:6) = char(ichar('0') + thread_id)
      file_unit = 10 + thread_id

      open(file_unit, file = file_name, status = 'old', action = 'read')

      do i = 1, max
         read(file_unit, *) aa(i, thread_id * 2 + 1), aa(i, thread_id * 2 + 2)
      end do

      close(file_unit)

!$omp end parallel
      end
```

I/O ライブラリーは、データ転送の並列実行中に、論理装置の状況の検索と更新を同期化します。 I/O ライブラリーが提供する並列性を最大にするには、各 I/O 要求のデータ転送のサイズを増やすことをお勧めします。そのためには、DO ループを次のように書き直す必要があります。

```
read(file_unit, *) a(:, thread_id * 2 + 1 : thread_id * 2 + 2)

do i = 1, max
   ! Do something for each element of array 'aa'.
end do
```

- 結果は、レコードの書き込みまたは読み取りの順序には左右されません。

```
real a(100)
do i = 1, 10
  read(4) a(i)
enddo
call qsort_(a)
```

- 各スレッドは、異なる直接アクセス、順次アクセス、またはストリーム・アクセス論理装置上で I/O を実行します。

```
do i = 11, 20
  write(i, '(i4)') a(i - 10)
enddo
```

同じ順次アクセス・ファイルでマルチスレッドによる書き込みまたは読み取りを実行する場合、あるいは **POS=** 指定子を使用せずに同じストリーム・アクセス・ファイルで書き込みまたは読み取りを実行する場合、レコードの書き込みまたは読み取りの順序は、スレッドがそのファイルで I/O ステートメントを実行する順序に依存しています。前に述べたとおり、この順序は予測できません。したがって、アプリケーションの結果で、レコードが定順位でつながっており、書き込みや読み取りを任意に行えないと判断される場合、その結果は誤ったものになる可能性があります。このループが並列化されると、数字が逐次実行の結果と同様に 1 から 500 までの順序で出力されることはなくなります。

```
do i = 1, 500
  print *, i
enddo
```

その順序で厳密に配列された数字に依存するアプリケーションは、正確に作動しなくなります。

XL Fortran の実行時オプション **multconn=yes** を指定すると、同じファイルを同時に複数の論理装置に接続できます。そのような接続は読み取り (**ACCESS='READ'**) にのみ設定できるため、同じファイルに接続された論理装置へのマルチスレッドによるアクセスの結果は予測可能になります。

シグナル・ハンドラーでの I/O ステートメントの使用

POSIX シグナル・モデルには基本的に 2 種類のシグナル、つまり*同期生成シグナル* と*非同期生成シグナル* があります。非マップ式メモリー、保護メモリー、または不良メモリー (**SIGSEGV** または **SIGBUS**) への参照、浮動小数点例外 (**SIGFPE**)、トラップ命令の実行 (**SIGTRAP**)、または無許可命令の実行 (**SIGILL**) など、スレッドのコードを実行して生成されるシグナルを、同期生成されたシグナルと言います。シグナルはプロセス外のイベント、たとえば **SIGINT**、**SIGHUP**、**SIGQUIT**、**SIGIO** といったイベントによっても生成されます。そのようなイベントは割り込みと呼ばれます。割り込みで生成されるシグナルを、非同期生成されたシグナルといいます。

XL Fortran スレッド・セーフ I/O ライブラリーは、非同期シグナル・アンセーフです。つまり、非同期生成シグナルであるために入力されるシグナル・ハンドラーでは XL Fortran I/O ステートメントを使用できません。 I/O ステートメントに割り

込むシグナル・ハンドラーから XL Fortran I/O ステートメントが呼び出される場合、システムの動作は未定義です。ただし、同期シグナルのシグナル・ハンドラーで I/O ステートメントを使用することは問題ありません。

アプリケーションでシグナル・ハンドラーの同期入力はないことが保証される場合もあります。たとえば、特定の部分の認知コードの実行を除いて、シグナルをマスクするアプリケーションがあります。そのような状況では、シグナルが I/O ステートメントや、非同期シグナル・アンセーフ機能に割り込むことはないことが知られています。したがって、非同期シグナル・ハンドラーでは引き続き Fortran I/O ステートメントを使用できます。

非同期シグナルをさらに簡単かつ安全に処理する方法は、すべてのスレッドでシグナルをブロックし、それらのスレッドを 1 つ以上の別個のスレッドで明示的に待機 (**sigwait()** を使用) することです。この方法による利点は、**handler** スレッドが、Fortran I/O ステートメントに加えて、他の非同期シグナル・アンセーフ・ルーチンを使用できることです。

非同期スレッドの取り消し

スレッドで非同期スレッドの取り消しを使用可能にすると、取り消し要求がある場合はその要求がただちに処理されます。 XL Fortran スレッド・セーフ I/O ライブラリーは、非同期スレッドの取り消しに対してセーフではありません。スレッドが XL Fortran スレッド・セーフ I/O ライブラリーにある間にそのスレッドが非同期に取り消される場合、システムの動作は未定義です。

言語間呼び出し

本節では、Fortran プログラムから言語間呼び出し (つまり Fortran 以外の言語で作成されたルーチンの呼び出し) を実行することについての詳細を説明しています。本章では、該当するすべての言語の構文にユーザーが精通していることを前提としています。

XL Fortran 外部名の規則

混合言語プログラムの作成をサポートするため、グローバル・エンティティーの名前をリンカーが解決できる外部名に変換するときには、XL Fortran は以下のような整合性のある一連の規則に従います。

- 下線 (_) およびドル記号 ($) は、名前の任意のか所に有効な文字として使用できます。

  下線で始まる名前はライブラリー・ルーチン名として予約されているため、下線は Fortran 外部名の最初の文字として使用しないでください。

  Fortran の関数名と Fortran 以外の言語の関数名の競合を避けるため、 **-qextname** オプションを指定して Fortran プログラムをコンパイルできます。このオプションは、Fortran 名の最後に下線を追加します。その後、Fortran から呼び出したい Fortran 以外の言語のプロシージャーの最後の文字として下線を使用してください。
- 名前の長さは、250 文字まで可能です。
- プログラム名とシンボル名は、デフォルトではすべて小文字であると解釈されます。 Fortran 以外の言語のコードを新しく作成している場合は、Fortran からのプロシージャー呼び出しを単純化するため、すべて小文字のプロシージャー名を使用してください。

  名前に大文字と小文字の両方を使用したい場合は、**-U** オプションまたは **@PROCESS MIXED** ディレクティブを使用することができます。

```
@process mixed
       external C_Func    ! With MIXED, we can call C_Func, not just c_func.
       integer aBc, ABC   ! With MIXED, these are different variables.
       common /xYz/ aBc   ! The same applies to the common block names.
       common /XYZ/ ABC   ! xYz and XYZ are external names that are
                          ! visible during linking.
       end
```

- モジュール・プロシージャーの名前は、__ (2 つの下線)、モジュール名、_IMOD_ (組み込みモジュールの場合) または _NMOD_ (組み込みモジュール以外の場合)、およびモジュール・プロシージャーの名前を連結することによって構成されます。たとえば、モジュール MYMOD のモジュール・プロシージャー MYPROC には、外部名 __mymod_NMOD_myproc があります。

  注: シンボリック・デバッガーその他のツールでは、モジュール・プロシージャーが入っている XL Fortran プログラムのデバッグ時にこの命名体系を考慮に入れる必要があります。例えば、一部のデバッガーではプログラム名および

シンボル名については小文字にデフォルト設定されます。モジュール・プロシージャーで XL Fortran プログラムをデバッグするとき、この動作は大/小文字混合に変更する必要があります。

- XL コンパイラーは、外部入り口点の名前として `main` を使用するコードを生成します。以下の場合以外には、外部名として `main` を使用しないでください。
  - Fortran プログラムまたはローカル変数名。 (この制限によって、`main` は、外部関数、外部サブルーチン、ブロック・データ・プログラム単位、または共通ブロックの名前として使用できません。そのようなオブジェクトの参照には、ユーザーの **main** ではなく、コンパイラー生成の **main** が使用されます。)
  - C プログラムのトップレベルの main 関数の名前。
- プログラムのリンク時に発生する可能性のある、その他の潜在的な命名競合がいくつかあります。これらを回避するには、 37 ページの『リンク中の命名競合の回避』を参照してください。

別のシステムからアプリケーションを移植していて、このような命名競合をアプリケーションが検出する場合は、 148 ページの『-qextname オプション』を使用しなければならないこともあります。

混合言語の入出力

パフォーマンスを向上させるために、XL Fortran 実行時ライブラリーには専用のバッファーとそのバッファーの専用の処理方法があります。つまり、混合言語プログラムは、異なる言語から同一ファイル上で I/O 操作を自由に混合することはできないということです。そのような場合にデータ保全性を保つためには、以下のようにします。

- ファイルの位置が重要ではない場合は、プログラムの Fortran 内部でファイルをオープンして明示的にクローズしてから、別の言語で作成されているサブプログラムから、そのファイルに対して I/O 操作を行ってください。
- Fortran でファイルをオープンし、そのオープンしたファイルを別の言語から操作するには、**flush_** プロシージャーを呼び出してそのファイル用のバッファーを保管してから、**getfd** プロシージャーを使用して対応するファイル記述子を見つけて、それを Fortran 以外の言語のサブプログラムに渡してください。 **flush_** プロシージャーを呼び出す代わりに、 **buffering** 実行時オプションを使用して、 I/O 操作でのバッファリングを使用不能にすることができます。
  buffering=disable_preconn を指定すると、 XL Fortran は事前接続された装置のバッファリングを使用不能にします。 **buffering=disable_all** を指定すると、 XL Fortran はすべての論理装置のバッファリングを使用不能にします。

  注: **flush_** を呼び出してファイルのバッファーをフラッシュした後、Fortran 以外の処理の完了時にファイルをクローズする以外は、プログラムの Fortran 部分からそのファイルに何も行わないでください。
- **WRITE** ステートメントを含んでいる XL Fortran サブプログラムが Fortran 以外のメインプログラムから呼び出された場合は、データ・ファイルを明示的に **CLOSE** するか、または XL Fortran サブプログラムの **flush_** サブルーチンを使用して、必ずバッファーがフラッシュされるようにしてください。あるいは、**buffering** 実行時オプションを使用して、 I/O 操作でのバッファリングを使用不能にすることができます。

関連情報: **FLUSH_** および **GETFD** プロシージャーの詳細は、「*XL Fortran* ランゲージ・リファレンス」の『サービス・プロシージャーおよびユーティ

リティー・プロシージャー』の節を参照してください。 **buffering** 実行時オプションの詳細については、 39 ページの『実行時オプションの設定』を参照してください。

Fortran と C++ の混在

本節の多くの情報は、Fortranおよび C (これらの言語はデータ型と命名体系が似ている) に適用されます。しかし、Fortran と C++ を同じプログラムに混合させるためには、間接のレベルを余分に追加し、C++ の wrapper 関数で言語間呼び出しを渡す必要があります。

C++ コンパイラーはいくつかの C++ オブジェクトの名前をマングル処理してしまうため、C++ コンパイラーを使用して最後のプログラムをリンクし、 XL Fortran ライブラリー・ディレクトリーおよびライブラリーのために **-L** および **-l** オプションを組み込む必要があります。

```
program main

integer idim,idim1

idim = 35
idim1= 45

write(6,*) 'Inside Fortran calling first C function'
call cfun(idim)
write(6,*) 'Inside Fortran calling second C function'
call cfun1(idim1)
write(6,*) 'Exiting the Fortran program'
end
```

図 1. C++ を呼び出す Fortran のメイン・プログラム (main1.f)

```
 #include <stdio.h>
 #include "cplus.h"

extern "C" void cfun(int *idim);
extern "C" void cfun1(int *idim1);

 void cfun(int *idim){
   printf("%%Inside C function before creating C++ Object¥n");
   int i = *idim;
   junk<int>* jj= new junk<int>(10,30);
   jj->store(idim);
   jj->print();
   printf("%%Inside C function after creating C++ Object¥n");
   delete jj;
   return;
 }

 void cfun1(int *idim1) {
   printf("%%Inside C function cfun1 before creating C++ Object¥n");
   int i = *idim1;
   temp<double> *tmp = new temp<double>(40, 50.54);
   tmp->print();
   printf("%%Inside C function after creating C++ temp object¥n");
   delete tmp;
   return;
 }
```

図 2. C++ を呼び出すための C++ Wrapper 関数 (cfun.C)

```
#include <iostream.h>

template<class T> class junk {

  private:
   int inter;
   T   templ_mem;
   T   stor_val;

  public:
   junk(int i,T j): inter(i),templ_mem(j)
                   {cout <<"***Inside C++ constructor" << endl;}

    ~junk()          {cout <<"***Inside C++ Destructor"  << endl;}

   void store(T *val){ stor_val = *val;}

   void print(void) {cout << inter << "\t" << templ_mem ;
                     cout <<"¥t" << stor_val << endl; }};

template<class T> class temp {

  private:
    int internal;
    T temp_var;

  public:
    temp(int i, T j): internal(i),temp_var(j)
                     {cout <<"***Inside C++ temp Constructor" <<endl;}

     ~temp()          {cout <<"***Inside C++ temp destructor"  <<endl;}

    void print(void) {cout << internal << "¥t" << temp_var << endl;}};
```

図 3. Fortran から呼び出される C++ コード (cplus.h)

このプログラムをコンパイルし、**xlC** または**g++** コマンドとリンクして実行したときの出力は次のようになります。

```
 Inside Fortran calling first C function
%Inside C function before creating C++ Object
***Inside C++ constructor
10      30      35
%Inside C function after creating C++ Object
***Inside C++ Destructor
 Inside Fortran calling second C function
%Inside C function cfun1 before creating C++ Object
***Inside C++ temp Constructor
40      50.54
%Inside C function after creating C++ temp object
***Inside C++ temp destructor
 Exiting the Fortran program
```

C 関数の呼び出しを機能させる方法

引き数をサブプログラム呼び出しへ渡す場合、通常の Fortran の規則では、引き数のアドレスを渡すことになっています。 C 関数の多くは、引き数がアドレスとしてでなく値として渡されることを予期しています。そのような引き数については、以下のように C の呼び出しの中で引き数を **%VAL** (*引き数*) として指定してください。

```
MEMBLK = MALLOC(1024)    ! Wrong, passes the address of the constant
MEMBLK = MALLOC(N)       ! Wrong, passes the address of the variable

MEMBLK = MALLOC(%VAL(1024)) ! Right, passes the value 1024
MEMBLK = MALLOC(%VAL(N))    ! Right, passes the value of the variable
```

詳細については、 343 ページの『参照または値による引き数の引き渡し』と、「*XL Fortran ランゲージ・リファレンス*」の『*%VAL および %REF*』を参照してください。

言語から別の言語にデータを渡す

次の表は、XL Fortran および C 言語で使用可能なデータ型を示しています。本節では、C プログラムへの参照によって Fortran 引き数がどのように渡されるかを説明します。 **BIND(C)** 属性および **ISO_C_BINDING** モジュール・サポートなどの Fortran 2003 ドラフト標準の相互運用可能フィーチャーを使用するには、「*XL Fortran ランゲージ・リファレンス*」を参照してください。

言語間での引き数の引き渡し

表 19. Fortran および C において対応するデータ型： C のルーチンは、Fortran を呼び出す時、この表にリストされている型に対するポインターとして引き数を渡す必要があります。

XL Fortran データ型	**C データ型**
INTEGER(1), BYTE	signed char
INTEGER(2)	signed short
INTEGER(4)	signed int
INTEGER(8)	signed long long
REAL, REAL(4)	float
REAL(8), DOUBLE PRECISION	double
REAL(16)	long double (注 1 を参照)
COMPLEX, COMPLEX(8)	_Complex float (注 2 を参照)
COMPLEX(16), DOUBLE COMPLEX	_Complex double (注 2 を参照)
COMPLEX(32)	_Complex long double (注 1 および 2 を参照)
LOGICAL(1)	unsigned char
LOGICAL(2)	unsigned short
LOGICAL(4)	unsigned int
LOGICAL(8)	unsigned long long
CHARACTER	char
CHARACTER(n)	char[n]
Integer POINTER	void *
Array	array
シーケンス派生型	構造体 (C パック構造体付き)

注:

1. 128 ビット長倍精度をサポートする C コンパイラーが必要です。 XL C は、Linux 上では **-qldbl128** を正式にサポートしません。使用には注意が必要です。
2. C99 準拠の C コンパイラーが必要です。

注:

1. 言語間通信では、 343 ページの『参照または値による引き数の引き渡し』で定義された **%VAL** および **%REF** 組み込み関数を使用しなければならないことがあります。
2. プロトタイプ化されていない C 関数を呼び出す時に、C プログラムは浮動小数点値を倍精度へ、および短精度整数値へと自動的に変換します。 XL Fortran は、値によって引き渡された **REAL(4)** の数量に対して変換を実行しないので、関数プロトタイプで宣言していない C 関数の引き数として **REAL(4)** 値と **INTEGER(2)** 値を渡さないでください。
3. Fortran 派生型および C の構造体が、互換データ型であるためには、サブオブジェクトの数、データ型、および長さが一致している必要があります。

関連情報: ディレクトリー **/opt/ibmcmp/xlf/9.1/samples** の 1 つまたは複数のサンプル・プログラムは、Fortran から C への呼び出し方法を説明しています。

XL Fortran によって提供される Fortran 2003 ドラフト標準の相互運用可能フィーチャーを使用するには、「*XL Fortran ランゲージ・リファレンス*」の『*言語相互運用可能フィーチャー*』を参照してください。

言語間でのグローバル変数の引き渡し

Fortran プログラム内から C データ構造体にアクセスする場合、あるいは C プログラム内から共通ブロックにアクセスする場合は、次のようにします。

1. C 構造体メンバーの 1 対 1 マッピングを提供する名前付き共通ブロックを作成します。名前なし共通ブロックがある場合は、名前付き共通ブロックに変更します。その共通ブロックには、C 構造体の名前を付けます。
2. C 構造体をグローバル変数として宣言します。宣言は、関数の外に置くか、**extern** 修飾子を持つ関数内に置きます。
3. C ソース・ファイルをコンパイルしてパック構造体を取得します。

```
program cstruct
real(8) a,d
integer b,c
.
.
common /mystuff/ a,b,c,d
.
.
end
```

```
struct mystuff {
       double a;
       int b,c;
       double d;
};

main() {

}
```

名前付き共通ブロックを特に必要としない場合は、C 構造体と同じ 1 対 1 マッピングを指定してシーケンス派生型を作成し、C 関数に引き数として渡すことができます。 C ソース・ファイルをコンパイルしてパック構造体を取得する必要があります。

THREADLOCAL と宣言された共通ブロックは、コンパイラー生成コードにより動的に割り振られる、スレッド固有のデータ域です。静的ブロックは **THREADLOCAL** 共通ブロック用に予約済みのままですが、コンパイラーおよびコンパイラーの実行時環境が制御情報用にそれを使用します。 **THREADLOCAL** 共

通ブロックを Fortran と C プロシージャーとの間で共用する必要がある場合、 C ソースには **THREADLOCAL** 共通ブロックのインプリメンテーションを知らせておく必要があります。詳細については、「*XL Fortran* ランゲージ・リファレンス」の『ディレクティブ』の節にある『**THREADLOCAL**』共通ブロック、および 385 ページの『付録 A. サンプルの Fortran プログラム』を参照してください。

THREADPRIVATE と宣言されている共通ブロックへのアクセスは、**THREADPRIVATE** として宣言されている C グローバル変数を使用して行うことができます。

言語間での文字型の引き渡し

言語間呼び出しの難しい面の 1 つは、言語間で文字ストリングを引き渡すことです。これが困難なのは、以下に示すように、複数の異なる言語がそのようなエンティティーを表す方法が異なるためです。

- Fortran の唯一の文字型は **CHARACTER** で、これは一連の連続バイトとして、1 バイトにつき 1 文字ずつ保管されます。長さはエンティティーの一部として保管されません。その代わり、エンティティーが引き数として渡される時に宣言された引き数リストの終わりに追加引き数として値によって渡されます。
- C のストリングは、**char** 型の配列として保管されます。ヌル文字はストリングの終了を示します。

  注: コンパイラーが特定の文字引き数へ NULL 文字を自動的に追加させたい場合は、 191 ページの『-qnullterm オプション』が使用できます。

混合言語プログラムとして両方の部分を作成している場合、C ルーチンに追加の Fortran 長さ引き数を処理させるようにすることもできますし、 **%REF** 関数を使用してストリングを渡すことによってこの追加引き数を抑止することもできます。**%REF** (通常はあらかじめ存在している C ルーチン用) を使用すると、 C ルーチンに渡される個々のストリングの終わりにヌルを連結して、どこでストリングが終わるかを示す必要があります。

```
! Initialize a character string to pass to C.
               character*6 message1 /'Hello¥0'/
! Initialize a character string as usual, and append the null later.
               character*5 message2 /'world'/

! Pass both strings to a C function that takes 2 (char *) arguments.
               call cfunc(%ref(message1), %ref(message2 // '¥0'))
               end
```

C 言語の使用との互換性を得るために、XL Fortran ストリングで以下のエスケープ・シーケンスをエンコードすることができます。

表 20. 文字ストリングのエスケープ・シーケンス

エスケープ	意味
¥b	バックスペース
¥f	用紙送り
¥n	改行
¥r	改行
¥t	タブ

表 20. 文字ストリングのエスケープ・シーケンス (続き)

エスケープ	意味
¥0	ヌル
\'	アポストロフィ (ストリングは終了しません)
\"	二重引用符 (ストリングは終了しません)
\\	円記号
¥*x*	*x*。ここで *x* は任意の文字 (円記号は無視されます)

ストリング内で円記号をエスケープ・シーケンスとして解釈させたくない場合は、**-qnoescape** オプションを指定してコンパイルすることができます。

言語間での配列の引き渡し

Fortran は、配列エレメントを昇順で記憶単位に保管します。 C は、行順に配列エレメントを保管します。 Fortran 配列指標は 1 から開始しますが、C 配列指標は 0 から開始します。

以下の例は、Fortran および C で A(3,2) によって宣言された 2 次元の配列がどのように保管されるかを示しています。

表 21. Fortran および C の対応する配列レイアウト： Fortran の配列参照 `A(X,Y,Z)` は、C では `a[Z-1][Y-1][X-1]` で表現します。 C では個々のスカラー配列エレメントは値によって渡しますが、配列は参照によって渡すことに注意してください。

	Fortran エレメント名	**C エレメント名**
一番低い記憶単位	A(1,1)	A[0][0]
	A(2,1)	A[0][1]
	A(3,1)	A[1][0]
	A(1,2)	A[1][1]
	A(2,2)	A[2][0]
一番高い記憶単位	A(3,2)	A[2][1]

Fortran 配列のすべてまたは一部を他の言語に渡すには、 Fortran 90 または Fortran 95 配列表記を使用することができます。

```
REAL, DIMENSION(4,8) :: A, B(10)

! Pass an entire 4 x 8 array.
CALL CFUNC( A )
! Pass only the upper-left quadrant of the array.
CALL CFUNC( A(1:2,1:4) )
! Pass an array consisting of every third element of A.
CALL CFUNC( A(1:4:3,1:8) )
! Pass a 1-dimensional array consisting of elements 1, 2, and 4 of B.
CALL CFUNC( B( (/1,2,4/) ) )
```

必要な場合は、Fortran プログラムは一時配列を作成して、すべてのエレメントを連続するストレージにコピーします。あらゆる場合に、C ルーチンは配列の列順レイアウトを考慮に入れる必要があります。

配列セクションまたは不連続配列は、対応する仮引き数が想定形状配列またはポインターとして宣言される場所に明示インターフェースが存在しないと、連続する一

時アドレスとして渡されます。配列引き数で Fortran 以外のプロシージャーを呼び出す時に、配列記述子 (言語間呼び出しに対してサポートされていない) の作成を回避するために、Fortran 以外のプロシージャーに明示インターフェースを与えたり、対応する仮引き数をインターフェースの想定形状配列またはポインターとして宣言しないでください。

```
! This explicit interface must be changed before the C function
! can be called.
INTERFACE
  FUNCTION CFUNC (ARRAY, PTR1, PTR2)
    INTEGER, DIMENSION (:) :: ARRAY          ! Change this : to *.
    INTEGER, POINTER, DIMENSION (:) :: PTR1  ! Change this : to *
                                             ! and remove the POINTER
                                             ! attribute.
    REAL, POINTER :: PTR2                    ! Remove this POINTER
                                             ! attribute or change to TARGET.
  END FUNCTION
END INTERFACE
```

言語間のポインターの引き渡し

整数 **POINTER** は、常に pointee オブジェクトのアドレスを表し、次のように、必ず値によって渡す必要があります。

```
CALL CFUNC(%VAL(INTPTR))
```

Fortran 90 **POINTER** は、言語間でやりとりすることはできますが、これは呼び出されたプロシージャーに対して明示インターフェースが存在しない場合にのみ、あるいは、明示インターフェース内の引き数が **POINTER** 属性または想定形状宣言子を持っていない場合に限られます。 **POINTER** 属性を除去したり、**TARGET** に変更したり、さらに形状無指定配列の配列宣言子があれば、形状明示型または大きさ引き継ぎに変更することができます。

XL Fortran では参照によって呼び出すという規則があるので、他言語からのスカラー値であっても、値そのものではなく値のアドレスとして渡す必要があります。たとえば、整数値 `x` を Fortran へ渡す C 関数は `&x` を渡す必要があります。さらに、ポインター値 `p` を整数 **POINTER** として使用できるように Fortran に渡す C 関数はその値を `void **p` として宣言する必要があります。 C の配列は例外で、`&` 演算子なしで Fortran に渡すことができます。

参照または値による引き数の引き渡し

Fortran 以外の言語で書かれたサブプログラム (たとえば、ユーザー作成 C プログラム、オペレーティング・システム・ルーチンなど) を呼び出すためには、Fortran が使用するデフォルトの方式とは異なる方式で実引き数を渡さなければならない場合があります。 C ルーチン (**libc.so** などのようなシステム・ライブラリー内の C ルーチンも含む) は、引き数を参照によって渡すのではなく、値によって渡す必要があります。 (C は、個々のスカラー配列エレメントを値によって渡しますが、配列は参照によって渡します。)

デフォルトの引き渡し方法は、**CALL** ステートメントまたは関数参照の引き数リスト内で組み込み関数 **%VAL** および **%REF** を使用して変更することができます。これらの組み込み関数を、Fortran プロシージャー参照の引き数リストまたは選択戻り指定子で使用することはできません。

%REF 参照によって引き数を渡します (つまり、呼び出されたサブプログラムは引

き数のアドレスを受け取ります)。これは、文字ストリングに対して余分な長さ引き数の抑止も行うということを除き、Fortran のデフォルト呼び出し方式と同じです。

%VAL

値によって引き数を渡します (つまり、呼び出されたサブプログラムは、実引き数と同じ値を持つ引き数を受け取りますが、この引き数に対して加えられた変更は、実引き数には影響しません)。

この組み込み関数は、型が **CHARACTER(1)** 式、**BYTE** 式、論理式、整数式、実数式、複素数式、またはシーケンス派生型のいずれかである実引き数で使用することができます。派生型のオブジェクトに、ポインター、配列、または長さが 1 バイトよりも長い文字構造体コンポーネントを入れることはできません。

%VAL は、長さが 1 バイトよりも長い配列エンティティー、プロシージャー名、文字式である実引き数で使用することはできません。

%VAL を使用すると、XL Fortran は実引き数を 32 ビットまたは 64 ビットの中間値として渡します。

32 ビット・モード

実引き数が、次のいずれかである場合:

- 32 ビットより短い整数値または論理値の場合は、符号付きの 32 ビット値に拡張されます。
- 32 ビットよりも長い整数値または論理値の場合は、2 つの 32 ビット中間値として渡されます。
- 実数式または複素数式の場合は、複数の 64 ビット中間値として渡されます。
- シーケンス派生型の場合は、複数の 32 ビット中間値として渡されます。

バイト名付き定数および変数は、**INTEGER(1)** であるかのように渡されます。実引き数が **CHARACTER(1)** の場合は、**-qctyplss** コンパイラー・オプションを指定したかどうかに関係なく、32 ビット値になるまでコンパイラーが左側にゼロを埋め込みます。

64 ビット・モード

実引き数が、次のいずれかである場合:

- 64 ビットより短い整数値または論理値の場合は、符号付きの 64 ビット値に拡張されます。
- 実数式または複素数式の場合は、複数の 64 ビット中間値として渡されます。
- シーケンス派生型の場合は、複数の 64 ビット中間値として渡されます。

バイト名付き定数および変数は、**INTEGER(1)** であるかのように渡されます。実引き数が **CHARACTER(1)** の場合は、**-qctyplss** コンパイラー・オプションを指定したかどうかに関係なく、64 ビット値になるまでコンパイラーが左側にゼロを埋め込みます。

-qautodbl コンパイラー・オプションを指定した場合、埋め込まれたストレージ・スペースは、派生型のオブジェクト以外には渡されません。

```
EXTERNAL FUNC
COMPLEX XVAR
IVARB=6

CALL RIGHT2(%REF(FUNC))       ! procedure name passed by reference
CALL RIGHT3(%VAL(XVAR))       ! complex argument passed by value
CALL TPROG(%VAL(IVARB))       ! integer argument passed by value
END
```

%VAL および %REF 用の明示インターフェース

Fortran 以外のプロシージャーに対して明示インターフェースを以下のよう指定して、個々の引き数リスト内の **%VAL** および **%REF** への呼び出しのコーディングを回避することができます。

```
INTERFACE
    FUNCTION C_FUNC(%VAL(A),%VAL(B)) ! Now you can code "c_func(a,b)"
        INTEGER A,B                  ! instead of
    END FUNCTION C_FUNC              ! "c_func(%val(a),%val(b))".
END INTERFACE
```

gcc への複合値の引き渡し

Fortran と Gnu C++ との間での複合値の引き渡しは、**-qfloat=[no]complexgcc** サブオプションに何が指定されているかによって異なります。 **-qfloat=complexgcc** が指定されている場合、複素数を渡すか、または戻すときにコンパイラーは Linux 規則を使用します。 **-qfloat=nocomplexgcc** がデフォルトです。

32 ビット・モードの **-qfloat=complexgcc** では、コンパイラーは **COMPLEX** *8 値を 2 つの汎用レジスター (GPR) に入れて渡し、**COMPLEX** *16 値を 4 つの GPR に入れて渡します。 64 ビット・モードでは、**COMPLEX** *8 値は 1 つの GPR に入れて渡され、**COMPLEX** *16 値は 2 つの GPR に入れて渡されます。**-qfloat=nocomplexgcc** の場合、**COMPLEX** *8 値と **COMPLEX** *16 値が 2 つの浮動小数点レジスター (FPR) に入れて渡されます。 **-qfloat=complexgcc** と

-qfloat=nocomplexgcc では、**COMPLEX** *32 値は常に 4 つの FPR に入れて渡されます (**gcc** が **COMPLEX***32 をサポートしないため)。

32 ビット・モードの **-qfloat=complexgcc** では、**COMPLEX** *8 値は GPR3 から GPR4 に戻され、**COMPLEX** *16 は GPR3 から GPR6 に戻されます。 64 ビット・モードでは、**COMPLEX** *8 値は GPR3 に戻され、**COMPLEX** *16 値は GPR3 から GPR4 に戻されます。 **-qfloat=nocomplexgcc** の場合、**COMPLEX** *8 値と **COMPLEX** *16 値が FPR1-FPR2 に戻されます。 **-qfloat=complexgcc** と **-qfloat=nocomplexgcc** では、**COMPLEX** *32 は常に FPR1-FPR4 に戻されます。

Fortran 関数からの値の戻り

XL Fortran は、Fortran 以外のプロシージャーからのある種の呼び出しをサポートしていません。 Fortran 関数がポインター、配列、または不定長の文字を戻す場合は、 Fortran 以外からその関数を呼び出さないでください。

以下のような関数を間接的に呼び出すことはできます。

```
SUBROUTINE MAT2(A,B,C)    ! You can call this subroutine from C, and the
                          ! result is stored in C.
INTEGER, DIMENSION(10,10) :: A,B,C
C = ARRAY_FUNC(A,B)       ! But you could not call ARRAY_FUNC directly.
END
```

OPTIONAL 属性を持つ引き数

オプションの引き数を参照によって引き渡す場合、引き数が存在しなければ、引き数リスト内のアドレスはゼロです。

オプションの引き数を値によって引き渡す場合、引き数が存在しなければ値はゼロです。コンパイラーは追加のレジスター引き数を使用して、その値を通常のゼロ値と区別します。レジスターが値 1 を持っている場合は、オプションの引き数が存在します。値 0 を持っている場合は、オプションの引き数は存在しません。

関連情報: 356 ページの『引き数リスト内の引き数の順序』を参照してください。

INTENT 属性を持つ引き数

現在では、**INTENT** 属性を持つ引き数を宣言しても、プロシージャーに対するリンケージ規約は変更されません。しかし、この規則は将来変更される可能性があるので、Fortran 以外のプロシージャーから **INTENT(IN)** 引き数を持つ Fortran プロシージャーへの呼び出しはお勧めできません。

型のエンコードと検査

実行時エラーは見つけにくく、多くの場合、プロシージャー・インターフェースの不一致またはデータ定義の矛盾が原因です。 そのため、これらの問題のできるだけ多くをコンパイル時またはリンク時に見つけることが得策です。

アセンブラー・レベルのサブルーチンのリンケージ規約

サブルーチン・リンケージ規約は、サブルーチンの入り口と出口でのマシンの状態を指定し、同じ言語または異なる言語で別個にコンパイルされるルーチンのリンクを許可します。 「*System V Application Binary Interface: PowerPC Processor Supplement*」および「*64-bit PowerPC ELF Application Binary Interface Supplement*」に記載されているサブルーチン・リンケージおよびシステム呼び出しに関する情報は、この項目に関する基本的な説明になっています。完全な詳細を知るために、これらの情報を参照してください。本節は、混合言語 Fortran およびアセンブラー・プログラムを作成したり、アセンブラー・レベルでデバッグするのに必要な情報を要約したもので、この種の基礎知識は持っておかなければなりません。

システム・リンケージ規約は、多数の浮動小数点レジスター (FPR) と汎用レジスター (GPR) を最大限に利用して、サブルーチンの入り口と出口でのレジスターの保管と復元を最小化して、レジスター内の引き数を渡します。このリンケージ規約は、引き数の引き渡しおよび戻り値が FPR か GPR、またはその両方に入ることが許可されます。

次の表は、浮動小数点レジスターと、それらの機能をリストしたものです。浮動小数点レジスターは、倍精度です (64 ビット)。

表 22. 呼び出し間の浮動小数点レジスターの使用法

レジスター	呼び出し間での保存	使用法
0	なし	
1	なし	FP パラメーター 1、関数戻り 1
2	なし	FP パラメーター 2、関数戻り 2
3	なし	FP パラメーター 3、関数戻り複素数 *32
4	なし	FP パラメーター 4、関数戻り複素数 *32
. . .	. . .	. . .
8	なし	FP パラメーター 8
9-13	なし	
14-31	あり	ローカル変数

次の表は、汎用レジスターと、それらの機能をリストしたものです。

表 23. 呼び出し間の汎用レジスターの使用法

レジスター	呼び出し間での保存	使用法
0	なし	
1	あり	スタック・ポインター
2	あり	システム予約済み。
3	なし	引き数リストの 1 番目のワード、戻り値 1
4	なし	引き数リストの 2 番目のワード、戻り値 2
5	なし	引き数リストの 3 番目のワード

表 23. 呼び出し間の汎用レジスターの使用法 (続き)

レジスター	呼び出し間での保存	使用法
. . .	. . .	. . .
10	なし	引き数リストの 8 番目のワード
11-12	なし	
13	あり	SDA ポインター
14-30	なし	ローカル変数
31	あり	ローカル変数または「環境ポインター」
レジスターが保持すると示されていないものは、呼び出し中に内容を変更することができます。呼び出し元は責任を持って、後で値が必要になるレジスターを保管する責任があります。逆に言えば、レジスターが保持されると考えられる場合は、呼び出し先が呼び出し間で内容を保持する責任があり、呼び出し元は特別な処置を行う必要はありません。		

次の表は、特殊目的のためのレジスター規約をリストしたものです。

表 24. 呼び出し間の特殊目的レジスターの使用法

レジスター	呼び出し間での保存
条件レジスター ビット 0-7 (CR0,CR1) ビット 8-22 (CR2,CR3,CR4) ビット 23-31 (CR5,CR6,CR7)	 なし あり なし
リンク・レジスター	なし
カウント・レジスター	なし
XER レジスター	なし
FPSCR レジスター	なし

スタック

スタックは、ローカル・ストレージ、レジスター保管域、パラメーター・リスト、呼び出しのチェーン・データを保持するのに使用されるストレージの一部です。スタックは高位アドレスから低位アドレスに向かって広がります。スタック・ポインター・レジスター (レジスター 1) は、スタックの現在の「最上位」を示すのに使用されます。

スタック・フレームは、1 つのプロシージャーで使用されるスタックの部分です。入力パラメーターは、現在のスタック・フレームの一部と見なされます。ある意味では、個々の出力引き数は、呼び出し元のスタック・フレームと呼び出し先のスタック・フレームの両方に属しています。どちらの場合にも、スタック・フレーム・サイズは呼び出し元のスタック・ポインターと呼び出し先のスタック・ポインターとの違いとして最適化を図って定義されます。

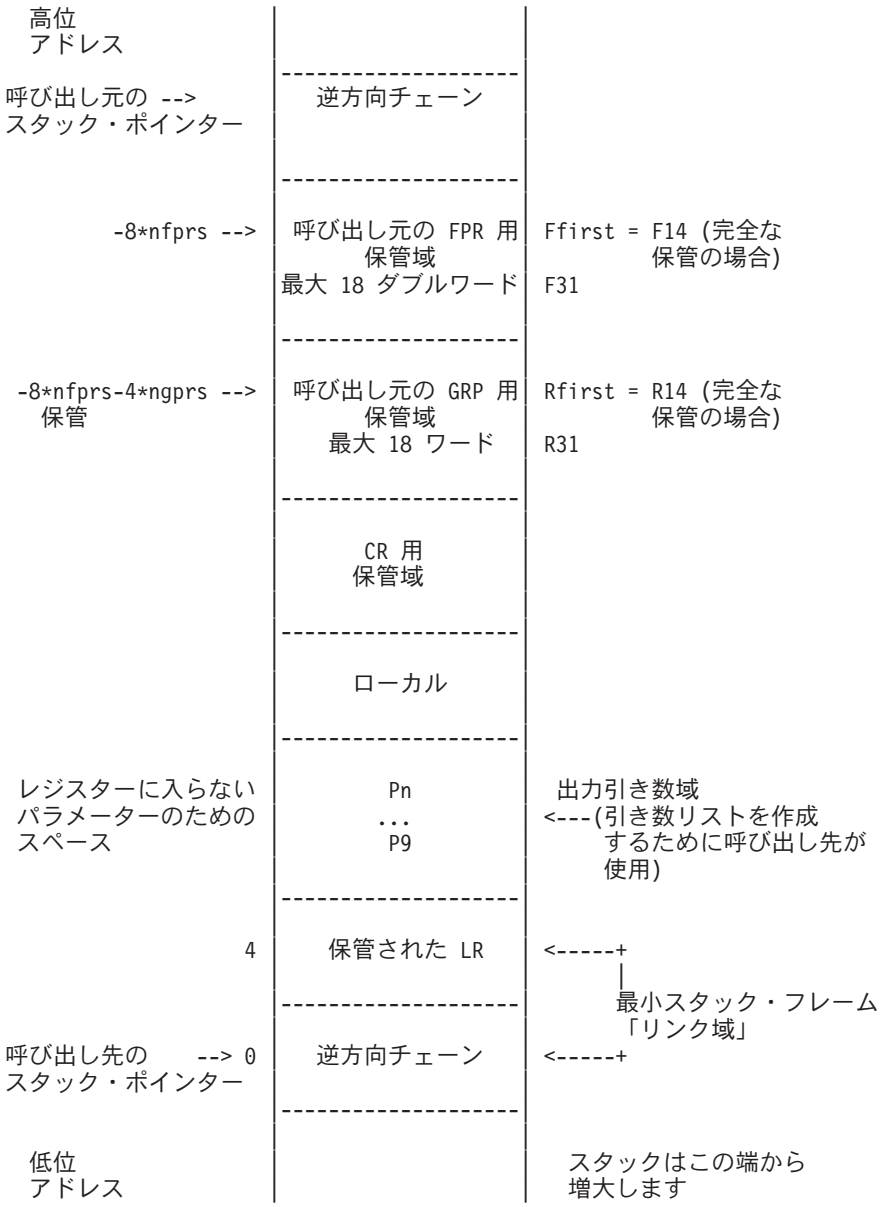
以下の図は、32 ビットおよび 64 ビット環境での典型的なスタック・フレームのストレージ・マップを示しています。

これらの図では、現行ルーチンは他の関数を呼び出せるようにするスタック・フレームを獲得しています。ルーチンが呼び出しを行わず、ローカル変数または一時変

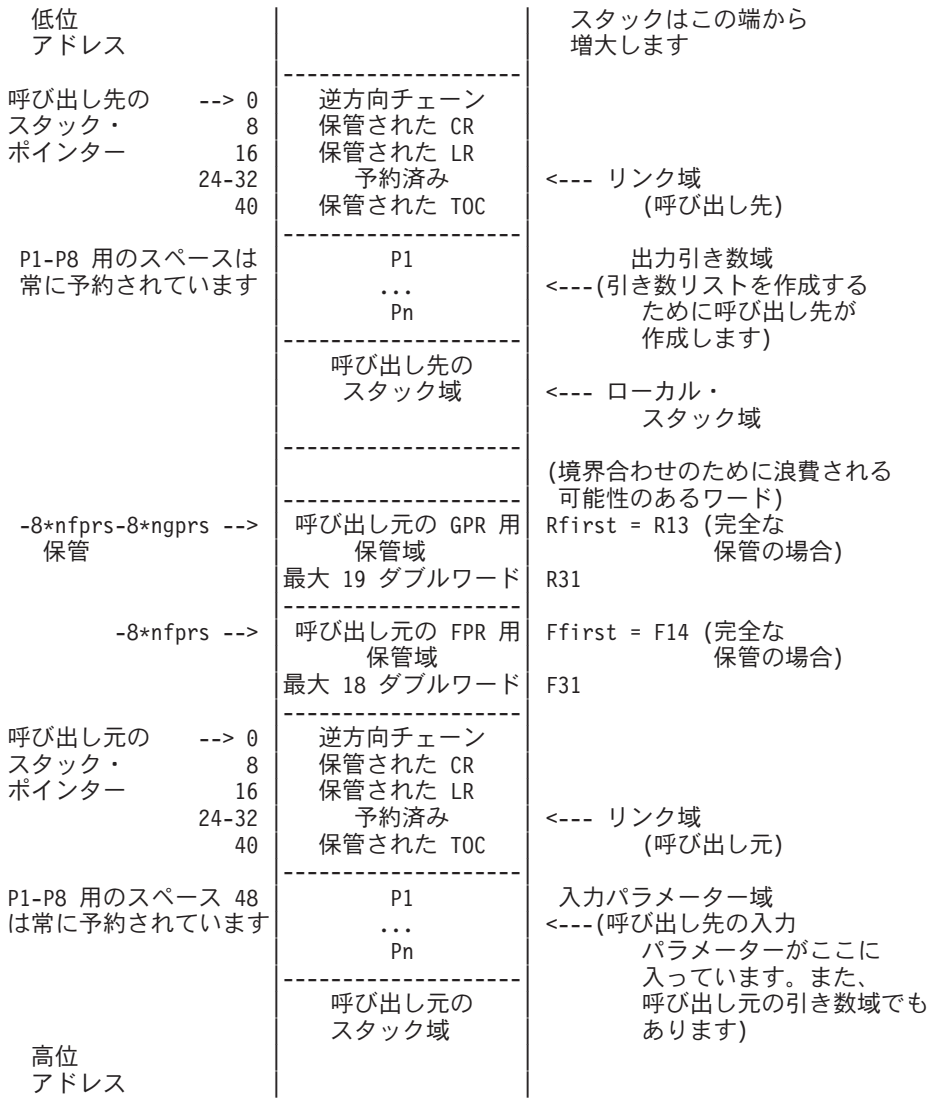
数が存在しない場合で、不揮発性レジスターを保管する必要がなければ、関数がスタック・フレームを割り振る必要はありません。必要であれば、呼び出し元のスタック・フレームの先頭にあるレジスター保管域をその後も使用することができます。

スタック・フレームは、ダブルワード 境界に位置合わせされます。

32 ビット環境の実行時スタック



64 ビット環境の実行時スタック



リンク域および最小スタック・フレーム

32 ビット環境では、リンク域は 2 ワードで構成され、プロシージャーへの入り口の呼び出し先のスタック・ポインターからオフセット 0 にあります。最初のワードには、呼び出し元の逆方向チェーン (前のスタック・フレームへのポインター) が含まれています。 2 番目のワードは、必要な場合に呼び出し元がリンク・レジスター (LR) を保管するロケーションです。

64 ビット環境では、この区域は 6 個のダブルワードで構成され、プロシージャーへの入り口の呼び出し元のスタック・ポインターからオフセット 0 にあります。最初のダブルワードには、呼び出し元の逆方向チェーン (スタック・ポインター) が含まれています。 2 番目のダブルワードは、必要な場合に呼び出し先が条件レジスター (CR) を保管するロケーションです。 3 番目のダブルワードは、必要な場合に呼び出し先の prolog コードがリンク・レジスターを保管するロケーションです。 4 番目のダブルワードは、C **SETJMP** および **LONGJMP** 処理用に予約されていて、

5 番目のダブルワードは将来の使用に備えて予約されています。最後のダブルワード (ダブルワード 6) は、他のオブジェクト・モジュール (たとえば共用ライブラリー) 内のルーチンを呼び出す時に使用されるグローバル・リンケージ・ルーチン用に予約されています。

入力パラメーター域

32 ビット環境では、レジスターに入らない入力パラメーターは、出力引き数域 (P9... Pn) に入ります。

入力パラメーター域は、64 ビット環境では、呼び出し先の入力パラメーターのレジスター・イメージを表すために、呼び出し側プログラムによって予約されるストレージの連続部分です。入力パラメーター域は、ダブルワード境界に合わせられ、呼び出し元のリンク域の直後のスタックに入れられます。この区域のサイズは、最低でも 8 ダブルワードあります。 8 ダブルワードを超えたパラメーターが予期される場合は、入力スタック・ポインターからの正方向オフセット 112 から始まるレジスター・イメージとして保管されます。

最初の 8 ダブルワードは、呼び出し点でレジスターに現れるだけで、スタックには現れません。残りのワードはスタックに常に入っていて、レジスターに入れることもできます。

レジスター保管域

32 ビット環境では、レジスター保管域は、呼び出し先のプログラムで使用されるすべての不揮発性 FPR および GPR を保管するために必要なスペースを提供します。FPR は呼び出し元の最小スタック・フレームの隣に保管されます。 GPR は FPR の下 (低位アドレス) に保管されます。

64 ビット環境では、レジスター保管域は、ダブルワード境界に合わせられます。呼び出し先のプログラムで使用されるすべての不揮発性 FPR および GPR を保管するのに必要なスペースを提供します。 FPR はリンク域の隣りに保管されます。 GPR は FPR の下 (低位アドレス) に保管されます。呼び出された関数は、新しいスタック・フレームを割り振る必要がない場合でも、ここにレジスターを保管することができます。システム定義のスタック・フロアには、以下のような可能な最大の保管域が含まれています。

```
32-bit platforms:  18*8 for FPRs + 18*4 for GPRs
64-bit platforms:  18*8 for FPRs + 19*8 for GPRs
```

呼び出し先が行わなければならないことは、実際に使用する不揮発性レジスターの保管だけです。

ローカル・スタック域

ローカル・スタック域は、ローカル変数および一時変数用に呼び出し先のプロシージャーが割り振るスペースです。

出力パラメーター域

32 ビット環境では、レジスターに入らない入力パラメーターは、出力引き数域 (P9... Pn) に入ります。

8 ワードを超えて渡される場合は、現行スタック・ポインターからのオフセット 8 から始まる拡張リストが作成されます。

最初の 8 ワードは、呼び出し点でレジスターに現れるだけで、スタックには現れません。残りのワードはスタックに常に入っていて、レジスターに入れることもできます。

64 ビット環境では、出力パラメーター域 (P1...Pn) には、このスタック・フレームを所有しているプロシージャーが呼び出すすべてのプロシージャーの最大のパラメーター・リストを保持できるだけの十分な大きさが必要です。この域の長さは、引き数リストの長さまたは存在とは無関係に、最低でも 8 ダブルワードあります。 8 ダブルワードを超えて渡される場合は、現行スタック・ポインターからのオフセット 112 から始まる拡張リストが作成されます。

最初の 8 ダブルワードは、呼び出し点でレジスターに現れるだけで、スタックには現れません。残りのダブルワードはスタックに常に入っていて、レジスターに入れることもできます。

引き数の引き渡しに関するリンケージ規約

システム・リンケージ規約は、使用可能な多数のレジスターを利用します。リンケージ規約では、引き数を GPR と FPR の両方に入れて渡します。 2 つの固定リスト R3-R10 および FP1-FP13 は、引き数の引き渡しに使用可能な GPR および FPR を指定します。

使用可能な引き数 GPR および FPR よりも多くの引き数ワードがある場合は、残りのワードはスタックのストレージに入れて渡されます。ストレージ内の値は、レジスター内の場合と同一です。

64 ビット環境では、パラメーター域のサイズは、スタック・フレームに関連付けられているプロシージャーからの呼び出しステートメントで渡されるすべての引き数を入れられるだけの大きさがあります。特定の呼び出しのための引き数がすべてストレージに実際に現れるわけではありませんが、各引き数が 1 つ以上のワードを占有し、この区域内にリストを作成すると考えるとわかりやすくなります。

参照による呼び出し (Fortran ではデフォルトの場合のように) では、引き数のアドレスはレジスターに入れて渡されます。以下の情報は、**%VAL** が使用される時の C または Fortran の場合と同じように、値による呼び出しのことを述べています。リスト内に現れるようにするために、引き数は浮動小数点値または非浮動小数点値として分類されます。

32 ビット環境の場合

- 個々の **INTEGER(8)** および **LOGICAL(8)** 引き数には、2 つのワードが必要です。
- 組み込み型のその他の非浮動小数点スカラー引き数またはプロシージャー/関数ポインターは、1 ワードを必要とし、GPR に現れるのとまったく同じようにそのワードに現れます。これは、言語のセマンティクスが指定されている場合、符号付きまたは符号なしで拡張され、ワード境界に合わせられます。
- 個々の単精度 (**REAL(4)**) 値は 1 ワードを占有します。個々の倍精度 (**REAL(8)**) 値はリスト内で 2 つの連続ワードを占有します。個々の倍精度 (**REAL(16)**) 値はリスト内で 2 つの連続ワードを占有します。
- **COMPLEX** 値は、同じ kind 型付きパラメーターを持つ **REAL** 値の 2 倍のワードを占有します。
- Fortran および C では、構造体値は「参照による値」で渡されます。つまり、コンパイラーは構造体のコピーのアドレスを実際に渡します。

64 ビット環境の場合

- すべての非浮動小数点の値には、ダブルワード境界の合わせられた 1 つのダブルワードが必要です。
- 個々の単精度 (**REAL(4)**) 値と個々の倍精度 (**REAL(8)**) 値はリスト内で 1 つのダブルワードを占有します。個々の拡張精度 (**REAL(16)**) 値はリスト内で 2 つの連続ダブルワードを占有します。
- **COMPLEX** 値は、同じ kind 型付きパラメーターを持つ **REAL** 値の 2 倍のダブルワードを占有します。
- Fortran および C では、構造体はストレージ内にあるとおりに連続ワードに現れて (ストレージ内のどこにあっても) すべての適切な境界合わせの要件を満たします。構造体はダブルワードに境界を合わせられ、`(sizeof(struct X)+7)/8` ダブルワードを占有します (終わりに埋め込みされます)。 1 ダブルワードよりも小さい構造体は、そのダブルワードまたはレジスター内で左寄せされます。 1 ワードよりも大きい構造体は複数のレジスターを占有することができ、一部はストレージに一部はレジスターに入れられて渡される場合があります。
- Pascal レコードなどにあるその他の集合値は、「参照による値」で渡されます。つまり、コンパイラーは実際にそれらのアドレスを渡して、呼び出されたプログラム内にコピーが作成されるようにします。
- プロシージャーまたは関数ポインターは、ルーチンの関数記述子を指し示すポインターとして渡されます。その最初のワードには、入り口ポインターのアドレスが含まれています。 (詳細は、 357 ページの『関数を指し示すポインター』を参照)

引き数の引き渡し規則 (値による)

以下の図から、次のような規則を理解できます。

- 32 ビット環境では、呼び出される関数の引き数は GPR および FPR で渡されます。 8 つまでのワードは GPR3-GPR10 で渡され、8 つまでの浮動小数点引き数は FPR1-FPR8 で渡されます。それより少ない引き数が渡される場合、不要なレジスターはロードされません。渡された引き数がレジスターに入らない場合、レジスターに入らなかった引き数を保持するためのスペースのみがスタック・フレーム内で割り振られます。
- 64 ビット環境では、呼び出されたプロシージャーがストレージの連続部分としてパラメーター・リストを扱う場合 (たとえば、パラメーターのアドレスが C で使用される場合)、パラメーター・レジスターは、パラメーター・レジスター用に予約されている、スタック内のスペースに保管されます。
- レジスター・イメージは、スタック上に保管されます。
- 64 ビット環境では、引き数域 ($P_1...P_n$) には、最大のパラメーター・リストを保持できるだけの十分な大きさが必要です。

関数への呼び出しの例を次に挙げます。

```
f(%val(l1), %val(l2), %val(l3), %val(l4), %val(l5), %val(l6), %val(l7),
        %val(d1), %val(f1), %val(c1), %val(d2), %val(s1), %val(cx2))
```

上記の意味は次のとおりです。

l integer(4) (フルワードの整数) を示します。
d real(8) (倍精度) を示します。
f real(4) (実数) を示します。
s integer(2) (ハーフワード整数) を示します。
c character (1 文字) を示します。
cx complex(8) (倍精度複素数) を示します。

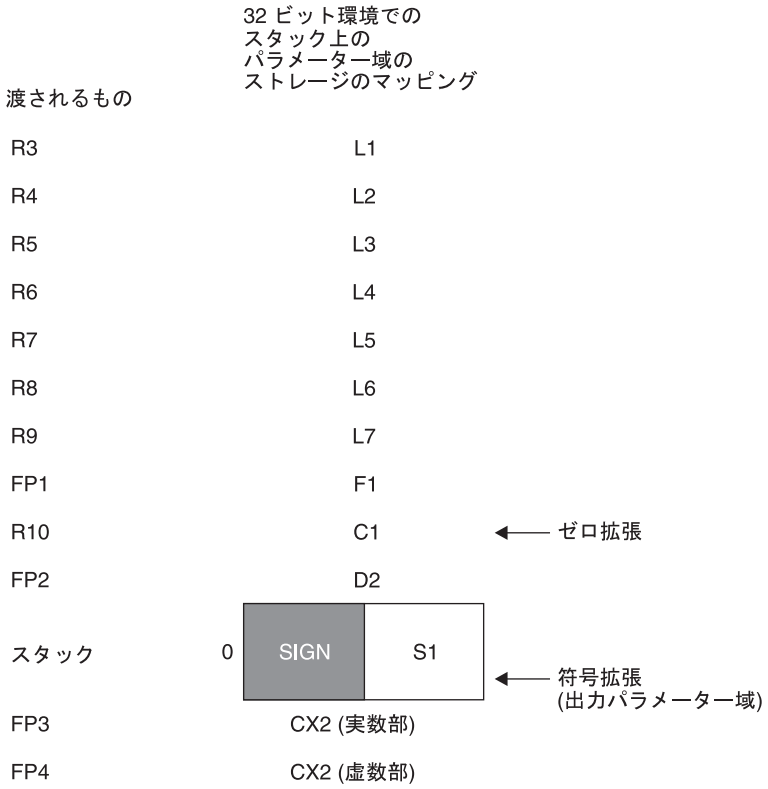


図 4. 32 ビット環境でのスタック上のパラメーター域のストレージのマッピング

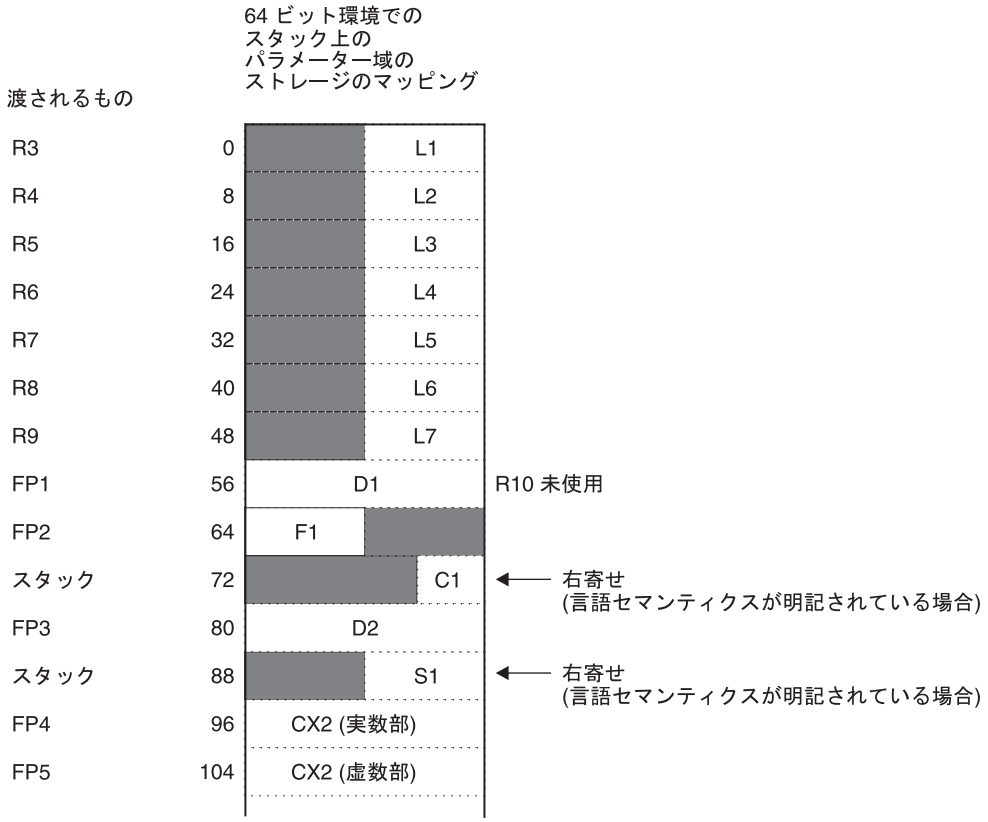


図 5. 64 ビット環境でのスタック上のパラメーター域のストレージのマッピング

引き数リスト内の引き数の順序

引き数リストは以下の順序で作成します。同一の中黒内の項目は、呼び出しで引き数キーワードが使用されてもされなくても、プロシージャー宣言の場合と同じ順序で現れます。

- 実引き数[4] のすべてのアドレスまたは値 (もしくはその両方)
- 値によって渡されるオプションの引き数の「現在の」標識
- ストリング用の引き数の長さ[4]

関数呼び出しのリンケージ規約

64 ビット・モードでは、1 つのルーチンに、2 つのシンボルが関連付けられています。それらは関数記述子 (*name*) と入り口点 (*.name*) です。ルーチンが呼び出されると、プログラムはその入り口点に直接分岐します。適切なレジスターへのパラメーター (存在する場合) のロードを除くと、コンパイラーは関数への呼び出しを以下の 2 つの命令に文字列の形で展開します。

```
BL    .foo                # Branch to foo
ORI R0,R0,0x0000          # Special NOP
```

4. Fortran-Fortran 呼び出しの間、このリストに別の項目がある場合もありますが、このセクションの呼び出し規則に従っている Fortran 以外のプロシージャーには見えません。

リンカーは、**BL** 命令を検出すると、以下の 2 つの内のいずれかを行います。

1. `foo` が (同一オブジェクト・モジュール外に) インポートされると、リンカーは `.foo` への **BL** を `foo` の `.glink` (グローバル・リンケージ・ルーチン) への **BL** に変更して、`.glink` をオブジェクト・モジュールに挿入します。また、**NOP** 命令 (`ORI R0,R0,0x0000`) が **BL** 命令の直後にあると、リンカーは **LOAD** 命令を **NOP** 命令 `L R2, 20(R1)` に置き換えます。
2. `foo` が呼び出し元と同じオブジェクト・モジュールにバインドされていて、**LOAD** 命令 (32 ビットの場合は `L R2,20(R1)`、64 ビットの場合は `L R2,40(R1)`) または `ORI R0,R0,0` が **BL** 命令の直後にあると、リンカーは **LOAD** 命令を **NOP** (`ORI R0,R0,0`) に置き換えます。

注: エクスポートの場合は常に、リンカーはプロシージャーの記述子をオブジェクト・モジュールに挿入します。

関数を指し示すポインター

64 ビット・モードでは、関数ポインターは、値の範囲がプロシージャー名にまで及んでいるデータ型です。このタイプの変数は、C や Fortran などのいくつかのプログラム言語に見られます。 Fortran で、**EXTERNAL** ステートメントにある仮引き数は関数ポインターです。 Fortran は、呼び出しステートメントのターゲット、またはこのようなステートメントの実引き数の文脈で、関数ポインターの使用をサポートします。

関数ポインターは、関数記述子のアドレスである 1 フルワードです。関数記述子は、3 ワードのオブジェクトです。最初のワードはプロシージャーの入り口点のアドレスを含んでいます。 2 番目のワードはプロシージャーがバインドされるオブジェクト・モジュールの TOC のアドレスを持っています。 3 番目のワードは Fortran 以外の言語用の環境ポインターです。関数記述子は、1 つの入り口点につき 1 つしかありません。関数が外部関数である場合は、関数記述子は自分が識別するのと同じオブジェクト・モジュールにバインドされます。記述子は外部名を持っていて、この外部名は関数名と同じですが、それを一意に識別する異なるストレージ・クラスを持っています。この記述子名が、すべてのインポートまたはエクスポート操作で使用されます。

関数値

関数は型に従って値を戻します。

- 32 ビットのモードでは、kind 1、2、および 4 の **INTEGER** と **LOGICAL** は R3 に戻されます (符号/ゼロ拡張)。
- 64 ビットのモードでは、kind 1、2、および 4 の **INTEGER** と **LOGICAL** は R3 に戻されます (右寄せ)。
- 64 ビットのモードでは、kind 8 の **INTEGER** と **LOGICAL** は R3 に戻されます。
- **REAL** *4 または *8 は FP1 に戻されます。 **REAL***16 は FP1 と FP2 に戻されます。
- **COMPLEX***8 または *16 は FP1 と FP2 に戻されます。 **COMPLEX** *32 は FP1 から FP4 に戻されます。

- 32 ビット・モードでは、 **-qfloat=complexgcc** が指定されると、 **COMPLEX***8 は R3 から R4 に戻され、 **COMPLEX***16 は R3 から R6 に戻されます。 64 ビット・モードでは、**COMPLEX***8 は R3 に戻され、**COMPLEX***16 は R3 から R4 に戻されます。
- 文字ストリングは、呼び出し元によって割り振られたバッファーに戻されます。このバッファーのアドレスと長さは、隠しパラメーターとして R3 と R4 に入れて渡されます。最初の明示的なパラメーター・ワードは R5 に入れられ、それ以降のパラメーターはすべて次のワードに移動されます。
- 構造体は、呼び出し元によって割り振られたバッファーに戻されます。アドレスは R3 に入れて渡され、長さの指定はありません。最初の明示的なパラメーターは R4 に入れられます。

スタック・フロア

64 ビット・モードでは、スタック・フロアとは、スタックがそれより下に広がることができないシステム定義のアドレスです。システム内のすべてのプログラムは、スタック・フロアより下にあるスタック・セグメント内の位置へのアクセスを回避する必要があります。

すべてのプログラムは、スタックに関連した他のシステム不変量を保守する必要があります。

- スタック・フロアよりも下のアドレスからは、データが保管されたり、アクセスされません。
- スタック・ポインターは常に有効です。スタック・フレーム・サイズが 32,767 バイトよりも大きい場合は、その値が必ず 1 つの命令で変更されるように十分注意してください。このステップは、シグナル・ハンドラーがスタック・データをオーバーレイしたり、誤ってスタック・セグメントをオーバーフローしているように見えるタイミング・ウィンドウが生じないようにするものです。

スタック・オーバーフロー

リンケージ規約では、オーバーフローのための明示的なインライン・チェックは必要ありません。オペレーティング・システムは、記憶保護機構を使用して、スタック・セグメントの終わりを超える保管を検出します。

プロローグとエピローグ

プロシージャーへの入り口では、以下のステップの中のいくつか、または全部を行わなければならない場合があります。

1. リンク・レジスターを保管します。
2. CR ビット 8 から 23 (CR2、CR3、CR4、CR5) のいずれかを使用する場合、CR を保管します。
3. このプロシージャーによって使用された不揮発性 FPR を FPR 保管域に保管します。
4. このプロシージャーによって使用されたすべての不揮発性 GPR を GPR 保管域に保管します。

5. 逆方向チェーンを保管し、スタック・ポインターをスタック・フレームのサイズ分だけ減らします。スタック・オーバーフローが発生した場合は、逆方向チェーンの保管が行われるとすぐにわかることに注意してください。

プロシージャーからの出口では、以下のステップのいずれか、または全部を実行しなければならない場合があります。

1. 保管したすべての GPR を復元します。
2. スタック・ポインターを入り口で持っていた値に復元します。
3. 必要に応じて、リンク・レジスターを復元します。
4. 必要に応じて、CR のビット 8-23 に復元します。
5. FPR を保管した場合、これらを復元します。
6. 呼び出し元に戻ります。

トレースバック

64 ビット・モードでは、コンパイラーはトレースバック・メカニズムをサポートしています。このメカニズムは、シンボリック・デバッガーが呼び出しを解決し、スタックを戻すために必要です。個々のオブジェクト・モジュールは、コードの終わりのテキスト・セグメントにトレースバック・テーブルを持っています。このテーブルには、スタック・フレーム情報やレジスター情報だけではなく、オブジェクト・モジュールのタイプなどのオブジェクト・モジュールに関する情報も入っています。

関連情報: 237 ページの『-qtbtable オプション』を使用して、トレースバック・テーブルを小さくするか、完全に除去することができます。

C を使用した THREADLOCAL 共通ブロックと ILC

Fortran **THREADLOCAL** 共通ブロックは、 POSIX pthreads ライブラリーで定義された、スレッド固有のデータ機能を使用してインプリメントされます。スレッド固有のデータ域についての詳細は、スレッド・プログラミングに関する の資料を参照してください。

内部的には、スレッド固有の共通ブロックのためのストレージは、 Fortran 実行時ライブラリーによって動的に割り振られます。この Fortran 実行時ライブラリーは、共通ブロックについての情報を保持する制御構造体を保守します。この制御域は、共通ブロックの名前と同じ名前が付いた外部構造体のことです。

たとえば、以下のように Fortran で共通ブロックを宣言した場合、

```
        common /myblock/ i
!ibm*   threadlocal /myblock/
```

Fortran コンパイラーによって作成されるのは、スレッド固有の共通ブロックについての制御情報を含む、*myblock* という外部構造体 (または共通域) です。

この制御構造体は以下のようなレイアウトになり、C でも同じようにコーディングされます。

```
typedef struct {
   pthread_key_t key;
   int flags;
   void *unused_1;
   int unused_2;
} FORT_LOCAL_COMMON;
extern FORT_LOCAL_COMMON myblock;
```

「key」フィールドは、スレッド・ローカル・データ域を記述する、固有の ID です。それぞれのスレッド・ローカルの共通ブロックには、独自のキーがあります。「flags」フィールドは、共通ブロックのためにキーが取得されているかどうかを示します。 C 関数内では、スレッド・ローカルの共通域のスレッド固有アドレスを取得するために、**pthread_getspecific** への呼び出しで、制御ブロックのこの 「key」を使用する必要があります。

例

```
! Example 1:  "fort_sub" is invoked by multiple threads.  This is an invalid example
! because "fort_sub" and "another_sub" both declare /block/ to be THREADLOCAL.
! They intend to share the common block, but they are executed by different threads.

SUBROUTINE fort_sub()
  COMMON /block/ j
  INTEGER :: j
  !IBM* THREADLOCAL /block/        ! Each thread executing fort_sub
                                   ! obtains its own copy of /block/.
  INTEGER a(10)

  ...
  !IBM* INDEPENDENT
  DO index = 1,10
    CALL another_sub(a(i))
  END DO
  ...

END SUBROUTINE fort_sub

SUBROUTINE another_sub(aa)        ! Multiple threads are used to execute another_sub.
  INTEGER aa
  COMMON /block/ j                ! Each thread obtains a new copy of the
  INTEGER :: j                    ! common block:  /block/.
  !IBM* THREADLOCAL /block/
  ...
  aa = j                          ! The value of 'j' is undefined.
END SUBROUTINE another_sub
```

詳細については、「*XL Fortran ランゲージ・リファレンス*」の『**THREADLOCAL**』ディレクティブを参照してください。

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。
- 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。
- 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。

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、 233 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

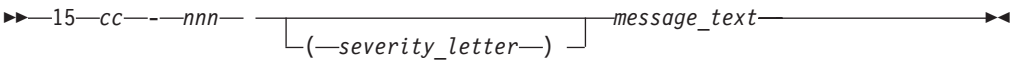
XLF コンパイル・プログラムが異常終了した場合は、オペレーティング・システムへの戻りコードは 1 です。

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* ステートメント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 コード作成または最適化メッセージを示します。

01 XL Fortran 共通メッセージを示します。

11-20 Fortran 特定のメッセージを示します。

25 XL Fortran アプリケーション・プログラムからの実行時メッセージを示します。

85 ループ変換メッセージを示します。

86 プロシージャー間分析 (IPA) メッセージを示します。

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: 12 ページの『各国語サポートのための環境変数』および 39 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、実行時エラー・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/9.1/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、ハードの限界もソフトの限界もこれらの **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/9.1/lib** と **/opt/ibmcmp/xlf/9.1/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、 14 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、12 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: I/O 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 72 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 379 ページの『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、 15 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 1 つまたは複数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、39 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、56 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 72 ページの

72 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 379 ページの『XL Fortran へのプログラムの移植』を参照してください。

引き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き数によって、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 ランゲージ・リファレンス*」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 ランゲージ・リファレンス*」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 I/O エラーに対して、メッセージ番号とメッセージ・テキストが端末 (標準エラー) に送出されます。入出力ステートメントで **IOMSG** を指定した場合は、エラーが検出されると、エラー・メッセージ・テキストが **IOMSG** 変数に割り当てられます。エラーが検出されない場合、**IOMSG** 変数の内容は変更されません。このカタログがシステムにインストールされていないと、メッセージ番号だけが表示されます。 39 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.
  X = 0               ! The program continues.
END IF
```

例外ハンドラーに関する指示については、285 ページの『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、56 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

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: 67 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』を参照してください。

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: 70 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 - 1514-092: (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
  - 仮引き数について、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

タイプは、次のうちの 1 つです。
- バイト
- 文字
- 複素数
- 派生型
- 整数
- 論理
- 実数

-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、作成されたマシン命令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。

関連情報: サンプル・プログラムが、 385 ページの『付録 A. サンプルの Fortran プログラム』に記載されています。

XL Fortran へのプログラムの移植

XL Fortran は、本来は他のコンピューター・システムまたはコンパイラー用に作成されたプログラムを、XL Fortran で容易に再コンパイルして実行するための、多くの機能を提供しています。

移植プロセスの概要

通常のプログラムの移植プロセスは、以下のようになります。

1. 元のプログラムで使用されている移植不能な言語拡張機能またはサブルーチンを識別します。これらのどれを XL Fortran がサポートするかを、以下によってチェックします。
   - 言語拡張機能は、「*XL Fortran ランゲージ・リファレンス*」に示されています。
   - XL Fortran コンパイラー・オプションの指定が必要な拡張機能もあります。これらのオプションは 73 ページの表 7 のリストを参照してください。
2. XL Fortran がサポートしていない移植不能な機能の場合は、それらを削除するか迂回するようにソース・ファイルを変更します。
3. インストール・システム固有の機能に対しても同じことを行います。たとえば、プログラムが浮動小数点値の表示に依存していたり、システム固有のファイル名を使用している場合は、変更しなければならないことがあります。
4. XL Fortran を使用してプログラムをコンパイルします。コンパイル・エラーが発生する場合は、プログラムが正常にコンパイルされるまで、修正、再コンパイル、および追加のエラーの修正を繰り返します。
5. XLF コンパイル済みプログラムを実行して、その出力を他のシステムからの出力と比較します。結果がかなり異なる場合は、変更しなければならないインストール・システム固有の機能がさらにある可能性があります。結果がわずかに異なるだけの場合 (たとえば、XL Fortran を使用した結果、精度の桁数が異なる、または小数点以下の最下位桁の数字が異なる) は、その違いがさらに調査が必要なほど重大であるかどうかを判断してください。これらの違いは修正できる可能性があります。

プログラムを XL Fortran に移植する前に、以降の項のヒントを読み、 XL Fortran がどのような互換機能を提供しているかを前もって頭に入れておいてください。

ディレクティブの移植性

XL Fortran は、他の Fortran 製品で使用できる多くのディレクティブをサポートしています。これによって、製品間での移植が簡単に実行できます。 XL Fortran で、デフォルト値以外の *trigger_constants* がコードに含まれている場合、 **-qdirective** コンパイラー・オプションを使用して、それらを指定することができます。 たとえば、`xx.f` ファイルにある CRAY コードを移植する場合、以下のコマンドを使用して CRAY *trigger_constant* を追加することができます。

```
xlf95 xx.f -qdirective=mic¥$
```

固定ソース形式コードに関しては、XL Fortran では、ディレクティブの *trigger_head* 部分の **!** 値だけでなく、*trigger_head* 値 **C**、**c**、および ***** もサポートされています。

詳細については、 138 ページの『-qdirective オプション』を参照してください。

XL Fortran は多くのプログラミング用語を同義語としてサポートしています。これにより、他の Fortran 製品からのコードの移植が容易になります。サポートされている用語は、以下の表に示されているようにコンテキストに依存しています。

表 25. PARALLEL DO 文節およびその XL Fortran 同義語

PARALLEL DO 文節	**XL Fortran 同義語**
LASTLOCAL	LASTPRIVATE
LOCAL	PRIVATE
MP_SCHEDTYPE および CHUNK	SCHEDULE
SAVELAST	LASTPRIVATE
SHARE	SHARED
NEW	PRIVATE

表 26. PARALLEL DO スケジューリング・タイプおよびその XL Fortran 同義語

スケジューリング・タイプ	**XL Fortran 同義語**
GSS	GUIDED
INTERLEAVE	STATIC(1)
INTERLEAVED	STATIC(1)
INTERLEAVE(n)	STATIC(n)
INTERLEAVED(n)	STATIC(n)
SIMPLE	STATIC

表 27. PARALLEL SECTIONS 文節およびその XL Fortran 同義語

PARALLEL SECTIONS 文節	**XL Fortran 同義語**
LOCAL	PRIVATE
SHARE	SHARED
NEW	PRIVATE

NEW

NEW ディレクティブを使用して、**PARALLEL DO** ループまたは **PARALLEL SECTIONS** 構文でローカルでなければならない変数を指定します。このディレクティブは、**PARALLEL DO** ディレクティブおよび **PARALLEL SECTIONS** ディレクティブの **PRIVATE** 文節と同じ機能を果たします。

背景情報

NEW ディレクティブは、**-qsmp** コンパイラー・オプションが指定された場合のみ有効です。

構文

NEW ディレクティブは、**PARALLEL DO** ディレクティブまたは **PARALLEL SECTIONS** ディレクティブの直後にする必要があります。

NEW ディレクティブを指定する場合、対応する **PARALLEL DO** または **PARALLEL SECTIONS** ディレクティブを文節なしで指定する必要があります。

PARALLEL DO ディレクティブの後に **NEW** ディレクティブが続く場合、 **NEW** ディレクティブの次の最初の非コメント行 (他のディレクティブは含まない) は **DO** ループにする必要があります。 この行は、無限の **DO** または **DO WHILE** ループにすることはできません。

NEW ディレクティブの *named_variable_list* にある変数名には、**PARALLEL DO** ディレクティブの **PRIVATE** 文節、または **PARALLEL SECTIONS** ディレクティブの **PRIVATE** 文節にある変数名と同じ制約事項が適用されます。「*XL Fortran ランゲージ・リファレンス*」の **PARALLEL DO** ディレクティブと **PARALLEL SECTIONS** 構文についての節を参照してください。

例

```
INTEGER A(10), C(10)
REAL B(10)
INTEGER FUNC(100)
!SMP$ PARALLEL DO
!SMP$ NEW I, TMP
        DO I = 1, 10
          TMP = A(I) + COS(B(I))
          C(I) = TMP + FUNC(I)
        END DO
```

XL Fortran がサポートしている共通の業界用拡張機能

XL Fortran は、普及している他のコンパイラーと同じく、多くの FORTRAN 77 拡張機能が使用可能です。それには、次のものが含まれます。

拡張機能	参照する「*XL Fortran ランゲージ・リファレンス*」の項
型なし定数	*型なしリテラル定数*
len 型の長さ指定子	*データ型*
BYTE データ型	**BYTE**
長い変数名	*名前*
小文字	*名前*
整数値と論理値の混在 (**-qintlog** オプションを使用)	*式の計算*
文字カウント **Q** 編集記述子 (**-qqcount** オプションを使用)	*Q (文字カウント) 編集*
レジスター内の設定ビットをカウントしてデータ・オブジェクト・パリティーを判別するための組み込み機能	**POPCNT**, **POPPAR**
64 ビットのデータ型 (**INTEGER(8)**、**REAL(8)**、**COMPLEX(8)**、 **LOGICAL(8)**)。デフォルトの 64 ビット型 (**-qintsize** および **-qrealsize** オプションを使用) のサポートを含みます。	*整数 実数 複素数 論理*
CRAY および Sun のコンパイラーでサポートしているものと同様の整数 **POINTER** (XL Fortran 整数ポインターの算術演算は 1 バイト増分を単位としているのに対し、 CRAY コンピューターでは 8 バイト増分を単位としています。 CRAY コンピューターから移植されたプログラムを正しく機能させるためには、ポインターの増分および減分に 8 を乗算することが必要です。)	**POINTER(整数)**
条件付きベクトル組み合わせ (CVMGx) 組み込み関数	*CVMGx (TSOURCE, FSOURCE, MASK)*
日付および時刻サービスとユーティリティー機能 (rtc、irtc、jdate、clock_、timef、date)	*サービス・プロシージャーおよびユーティリティー・プロシージャー*
STRUCTURE、**UNION**、および **MAP** 構文	*構造体コンポーネント*、*Union* および *Map*

ステートメント内でのデータ型の混在

-qctyplss オプションを指定すると、型なし定数を使用する位置と同じ位置で文字定数式を使用することができます。 **-qintlog** オプションを指定すると、論理式を使用できる場所で整数式を使用することができます (その逆も可能です)。 kind 型付きパラメーターは、**-qintlog** がオンの場合でも論理定数と置き換えることができず、**-qctyplss** がオンの場合でも文字定数と置き換えることができません。また、型なし定数にすることもできません。

日付および時刻ルーチン

dtime、**etime**、**jdate** などの日付および時刻ルーチンは、Fortran サブルーチンとしてアクセス可能です。

その他の libc ルーチン

libc ライブラリーからのその他の一般的なルーチン (たとえば、 **flush**、**getenv**、**system**) が多数あり、これらも Fortran サブルーチンとしてアクセス可能です。

データ型のデフォルト・サイズの変更

ワード・サイズの大きいまたは小さいマシンからの移植の場合は、**-qintsize** オプションを指定すると整数値および論理値のデフォルト・サイズを指定することができ、**-qrealsize** オプションを指定すると実数および複素数のコンポーネントのデフォルト・サイズを指定することができます。

ユーザーのプロシージャーと XL Fortran 組み込みプロシージャー間の名前の競合

XL Fortran 組み込みプロシージャーと同じ名前のプロシージャーがある場合は、プログラムは組み込みプロシージャーを呼び出します。 (この状態は、多数の新しい Fortran 90 および Fortran 95 組み込みプロシージャーを追加した場合に、さらに発生する可能性が高くなります。)

それでもユーザー自身のプロシージャーを呼び出したい場合は、名前が競合するプロシージャーに明示的なインターフェース、**EXTERNAL** ステートメント、または **PROCEDURE** ステートメントを追加するか、あるいはコンパイル時に **-qextern** オプションを使用してください。

その他のシステムからの結果の再現

XL Fortran は、浮動小数点結果を他の IEEE システムからの結果と矛盾させないために、**-qfloat** オプションによる設定をサポートしています。 283 ページの『他のシステムの浮動小数点結果の再現』を参照してください。

非標準拡張機能の検出

XL Fortran は、さまざまな言語標準に関するいくつかの拡張機能をサポートしています。これらの拡張機能の多くは一般的に使用されているため、プログラムを別のシステムに移植する際には、すべてのコンパイラーがそれらの機能を備えているわけではないことを覚えておいてください。移植作業を始める前に、**-qlanglvl** オプションを使用して、XL Fortran プログラム内のそのような拡張部分を見つけてください。

```
$ # -qnoobject stops the compiler after parsing all the source,
$ # giving a fast way to check for errors.
$ # Look for anything above the base F77 standard.
$ xlf  -qnoobject -qlanglvl=77std f77prog.f
  ...
$ # Look for anything above the F90 standard.
$ xlf90 -qnoobject -qlanglvl=90std use_in_2000.f
  ...
$ # Look for anything above the F95 standard.
$ xlf95 -qnoobject -qlanglvl=95std use_in_2000.f
  ...
```

関連情報: 176 ページの『-qlanglvl オプション』および 203 ページの『-qport オプション』を参照してください。

付録 A. サンプルの Fortran プログラム

以下のプログラムは、XL Fortran のコーディング例です。これらのサンプルのいくつかでは、多くのユーザーにとっては新しい SMP プログラミングのさまざまな局面を示しています。 SMP プログラミングを初めて行う場合、これらのサンプルを検討して、SMP のコーディング・スタイルをよく理解することが大切です。ソースのキー・エリアを内部で文書化して、理解しやすいようになっています。

最初のプログラムをコンパイルして実行すると、コンパイラーが正しくインストールされており、ユーザー ID が Fortran プログラムを実行できるように設定されていることを確かめられます。

例 1 - XL Fortran ソース・ファイル

```
      PROGRAM CALCULATE
!
! Program to calculate the sum of up to n values of x**3
! where negative values are ignored.
!
      IMPLICIT NONE
      INTEGER I,N
      REAL SUM,X,Y
      READ(*,*) N
      SUM=0
      DO I=1,N
         READ(*,*) X
         IF (X.GE.0) THEN
            Y=X**3
            SUM=SUM+Y
         END IF
      END DO
      WRITE(*,*) 'This is the sum of the positive cubes:',SUM
      END
```

実行結果

このプログラムの実行結果を次に示します。

```
$ a.out
5
37
22
-4
19
6
 This is the sum of the positive cubes:  68376.00000
```

例 2 - 有効な C ルーチン・ソース・ファイル

```
/*
 * ********************************************************************
 * This is a main function that creates threads to execute the Fortran
 * test subroutines.
 * ********************************************************************
 */
#include <pthread.h>
#include <stdio.h>
#include <errno.h>


extern char *optarg;
extern int optind;

static char *prog_name;

#define MAX_NUM_THREADS 100

void *f_mt_exec(void *);
void f_pre_mt_exec(void);
void f_post_mt_exec(int *);

void
usage(void)
{
    fprintf(stderr, "Usage: %s -t number_of_threads.\n", prog_name);
    exit(-1);
}

main(int argc, char *argv[])
{
    int i, c, rc;
    int num_of_threads, n[MAX_NUM_THREADS];
    char *num_of_threads_p;
    pthread_attr_t attr;
    pthread_t tid[MAX_NUM_THREADS];

    prog_name = argv[0];
    while ((c = getopt(argc, argv, "t")) != EOF)
    {
        switch (c)
        {
        case 't':
            break;

        default:
            usage();
            break;
        }
    }

    argc -= optind;
    argv += optind;
    if (argc < 1)
    {
        usage();
    }

    num_of_threads_p = argv[0];
    if ((num_of_threads = atoi(num_of_threads_p)) == 0)
    {
        fprintf(stderr,
         "%s: Invalid number of threads to be created <%s>¥n", prog_name,
                num_of_threads_p);
        exit(1);
```

```
    }
    else if (num_of_threads > MAX_NUM_THREADS)
    {
        fprintf(stderr,
                "%s: Cannot create more than 100 threads.¥n", prog_name);
        exit(1);
    }
    pthread_attr_init(&attr);
    pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

    /* ****************************************************************
     * Execute the Fortran subroutine that prepares for multi-threaded
     * execution.
     * ****************************************************************
     */
    f_pre_mt_exec();

    for (i = 0; i < num_of_threads; i++)
    {
        n[i] = i;
        rc = pthread_create(&tid[i], &attr, f_mt_exec, (void *)&n[i]);
        if (rc != 0)
        {
            fprintf(stderr, "Failed to create thread %d.¥n", i);

            exit(1);
        }
    }
    /* The attribute is no longer needed after threads are created. */
    pthread_attr_destroy(&attr);

    for (i = 0; i < num_of_threads; i++)
    {
        rc = pthread_join(tid[i], NULL);
        if (rc != 0)
        {
            fprintf(stderr, "Failed to join thread %d. ¥n", i);

        }
    }
    /*
     * Execute the Fortran subroutine that does the check after
     * multi-threaded execution.
     */
    f_post_mt_exec(&num_of_threads);

    exit(0);
}

! ************************************************************************
! This test case tests the writing list-directed to a single external
! file by many threads.
! ************************************************************************

        subroutine f_pre_mt_exec()
        integer array(1000)
        common /x/ array

        do i = 1, 1000
          array(i) = i
        end do

        open(10, file="fun10.out", form="formatted", status="replace")
        end

        subroutine f_post_mt_exec(number_of_threads)
        integer array(1000), array1(1000)
        common /x/ array
```

```
close(10)
open(10, file="fun10.out", form="formatted")
do j = 1, number_of_threads
  read(10, *) array1

  do i = 1, 1000
    if (array1(i) /= array(i)) then
      print *, "Result is wrong."
      stop
    endif
  end do
end do
close(10, status="delete")
print *, "Normal ending."
end

subroutine f_mt_exec(thread_number)
integer thread_number
integer array(1000)
common /x/ array

write(10, *) array
end
```

例 3 - 有効な Fortran SMP ソース・ファイル

```
!*********************************************************************
!* This example uses a PARALLEL construct and a DO construct     *
!* to calculate the value of pi.                                 *
!*********************************************************************
      program compute_pi
      integer n, i
      real*8 w, x, pi, f, a
      f(a) = 4.d0 /(1.d0 + a*a)  !! function to integrate

      pi = 0.0d0
!$OMP PARALLEL private(x, w, n), shared(pi)
      n = 10000                      !! number of intervals
      w = 1.0d0/n                    !! calculate the interval size
!$OMP DO reduction(+: pi)
      do i = 1, n
         x = w * (i - 0.5d0)
         pi = pi + f(x)
      enddo
!$OMP END DO
!$OMP END PARALLEL
      print *, "Computed pi = ", pi
     end
```

例 4 - 無効な Fortran SMP ソース

```
!*********************************************************************
!* In this example, fort_sub is invoked by multiple threads.     *
!*                                                               *
!* This example is not valid because                             *
!*  fort_sub and another_sub both declare /block/ to be          *
!*  THREADLOCAL. They intend to share the common block, but      *
!*  they are executed via different threads.                     *
!*                                                               *
!* To "fix" this problem, one of the following approaches can    *
!* be taken:                                                     *
!*  (1) The code for another_sub should be brought into the loop.*
!*  (2) "j" should be passed as an argument to another_sub, and  *
!*       the declaration for /block/ should be removed from      *
```

```
!*       another_sub.                                                *
!*  (3) The loop should be marked as "do not parallelize" by     *
!*      using the directive "!SMP$ PARALLEL DO  IF(.FALSE.)".    *
!*****************************************************************

subroutine fort_sub()

  common /block/ j
  integer :: j
  !IBM* THREADLOCAL /block/         ! Each thread executing fort_sub
                                    ! obtains its own copy of /block/.
  integer a(10)

  ...
  !IBM* INDEPENDENT
  do index = 1,10
    call another_sub(a(i))
  enddo
  ...

end subroutine fort_sub

subroutine another_sub(aa)          ! Multiple threads are used to
  integer aa                        ! execute another_sub.
  common /block/ j                  ! Each thread obtains a new copy
  integer :: j                      ! of the common block /block/.
  !IBM* THREADLOCAL /block/

  aa = j                            ! The value of "j" is undefined.
end subroutine another_sub
```

Pthreads ライブラリー・モジュールを使用したプログラミング例

```
!*********************************************************************
!* Example 5 : Create a thread with Round_Robin scheduling policy.*
!* For simplicity, we do not show any codes for error checking,   *
!* which would be necessary in a real program.                    *
!*********************************************************************
        use, intrinsic::f_pthread
        integer(4) ret_val
        type(f_pthread_attr_t) attr
        type(f_pthread_t)      thr

        ret_val = f_pthread_attr_init(attr)
        ret_val = f_pthread_attr_setschedpolicy(attr, SCHED_RR)
        ret_val = f_pthread_attr_setinheritsched(attr, PTHREAD_EXPLICIT_SCHED)
        ret_val = f_pthread_create(thr, attr, FLAG_DEFAULT, ent, integer_arg)
        ret_val = f_pthread_attr_destroy(attr)
        ......
```

pthread 属性オブジェクトを処理するには、まず pthread 属性オブジェクトを作成して初期化する必要があります。適切なインターフェースを呼び出して、属性オブジェクトを処理する必要があります。 **f_pthread_attr_setschedpolicy** を呼び出すと、Round_Robin にスケジューリング・ポリシー属性が設定されます。この属性は、作成元スレッドのスケジューリング特性を継承している新規作成のスレッドには影響を与えないことに注意してください。これらのスレッドに関しては、明示的に **f_pthread_attr_setinheritsched** を呼び出して、継承しているデフォルトの属性をオーバーライドします。コードの残りの部分は、理解しやすいものになっています。

```
!*********************************************************************
!* Example 6 : Thread safety                                      *
!* In this example, we show that thread safety can be achieved    *
!* by using the push-pop cleanup stack for each thread. We        *
```

```
!* assume that the thread is in deferred cancellability-enabled  *
!* state.  This means that any thread-cancel requests will be    *
!* put on hold until a cancellation point is encountered.        *
!* Note that f_pthread_cond_wait provides a                      *
!* cancellation point.                                           *
!*****************************************************************
        use, intrinsic::f_pthread
        integer(4) ret_val
        type(f_pthread_mutex_t) mutex
        type(f_pthread_cond_t) cond
        pointer(p, byte)
        ! Initialize mutex and condition variables before using them.
        ! For global variables this should be done in a module, so that they
        ! can be used by all threads. If they are local, other threads
        ! will not see them. Furthermore, they must be managed carefully
        ! (for example, destroy them before returning, to avoid dangling and
        ! undefined objects).
        mutex = PTHREAD_MUTEX_INITIALIZER
        cond  = PTHREAD_COND_INITIALIZER

        ......
        ! Doing something

        ......

        ! This thread needs to allocate some memory area used to
        ! synchronize with other threads. However, when it waits on a
        ! condition variable, this thread may be canceled by another
        ! thread. The allocated memory may be lost if no measures are
        ! taken in advance. This will cause memory leakage.

        ret_val = f_pthread_mutex_lock(mutex)
        p = malloc(%val(4096))

        ! Check condition. If it is not true, wait for it.
        ! This should be a loop.

        ! Since memory has been allocated, cleanup must be registered
        ! for safety during condition waiting.

        ret_val = f_pthread_cleanup_push(mycleanup, FLAG_DEFAULT, p)
        ret_val = f_pthread_cond_wait(cond, mutex)

        ! If this thread returns from condition waiting, the cleanup
        ! should be de-registered.

        call f_pthread_cleanup_pop(0)      ! not execute
        ret_val = f_pthread_mutex_unlock(mutex)

        ! This thread will take care of p for the rest of its life.
        ......


        ! mycleanup looks like:

        subroutine mycleanup(passed_in)
            pointer(passed_in, byte)
            external free

            call free(%val(passed_in))
        end subroutine mycleanup
```

付録 B. 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 実行時環境

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 実行時環境の機能は、次のように分類されます。
- Fortran I/O 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 実行時環境は、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 実行時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 実行時環境には上位互換性があります。あるレベルの実行時環境とオペレーティング・システムでコンパイル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実行時環境とオペレーティング・システムが必要となります。

実行時環境の外部名

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 152 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、変数を仮引き数として渡す場合、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記憶スペースが続きます。埋め込みは **BYTE**、**INTEGER**、**LOGICAL** およびプロモートされていない **REAL** と **COMPLEX** オブジェクトに適用されます。これら 2 つのプロモートされていないオブジェクトは、プロモートされた項目と記憶スペースを共用することがあります。安全のため、**POINTER**、**TARGET**、実引き数と仮引き数、**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。

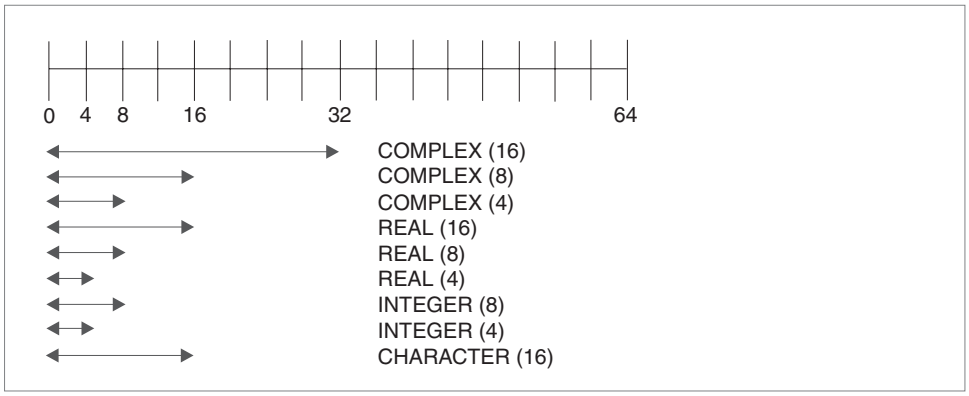


図 6. -qautodbl オプションを指定しなかった場合のストレージの関係

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
```

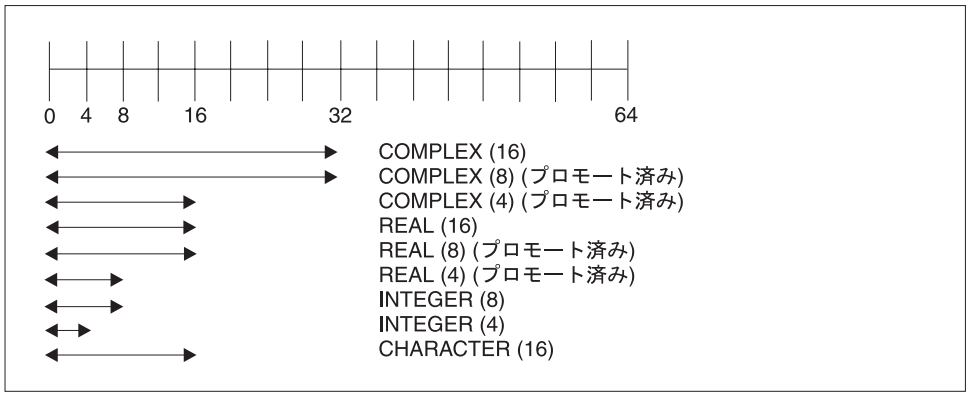


図 7. -qautodbl=dbl を指定した場合のストレージの関係

```
@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
```

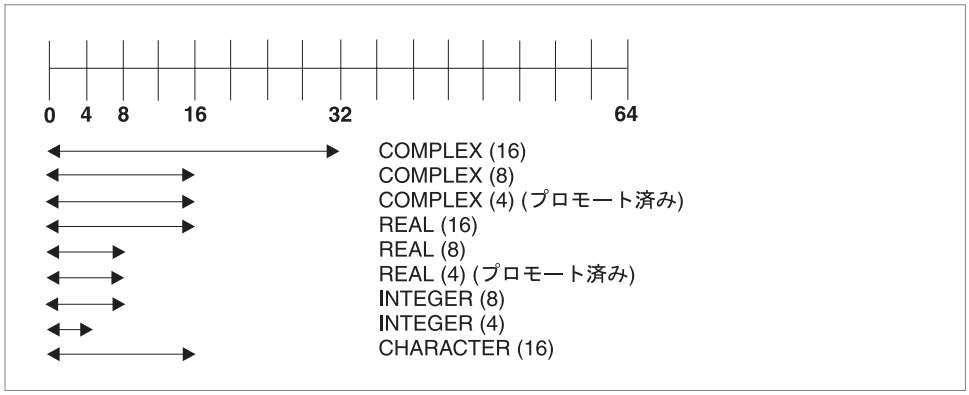


図 8. -qautobl=dbl4 を指定した場合のストレージの関係

```
@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
```

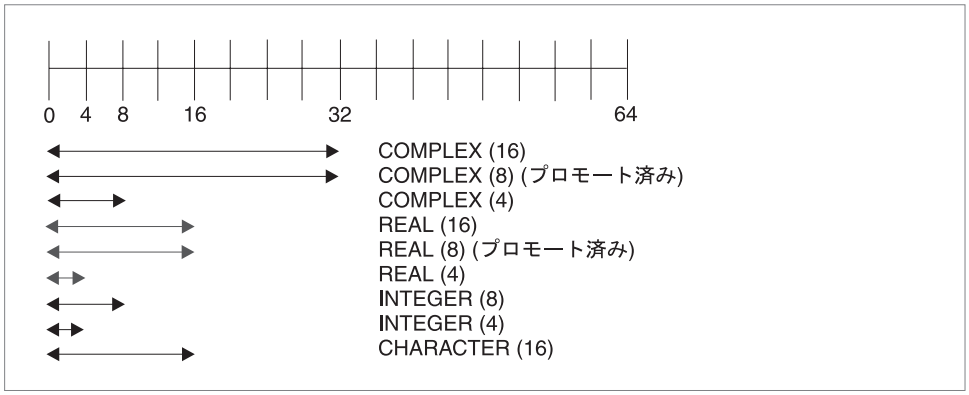


図 9. -qautodbl=dbl8 を指定した場合のストレージの関係

```
@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
```

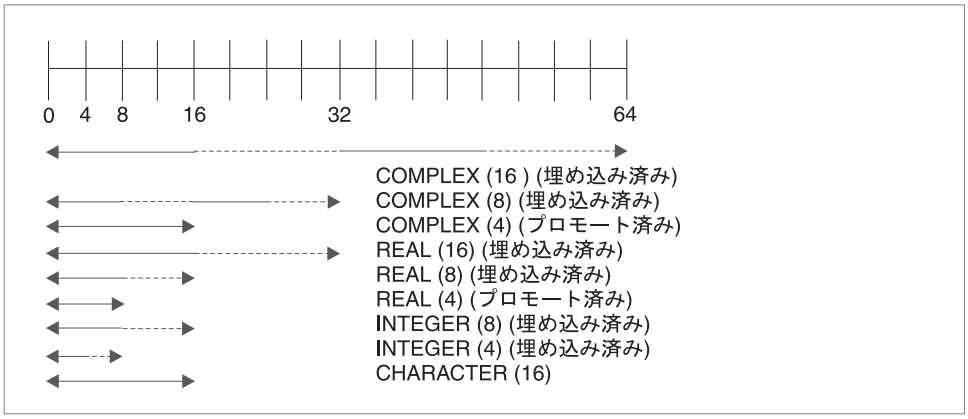


図 10. -qautodbl=dblpad4 を指定した場合のストレージの関係

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
```

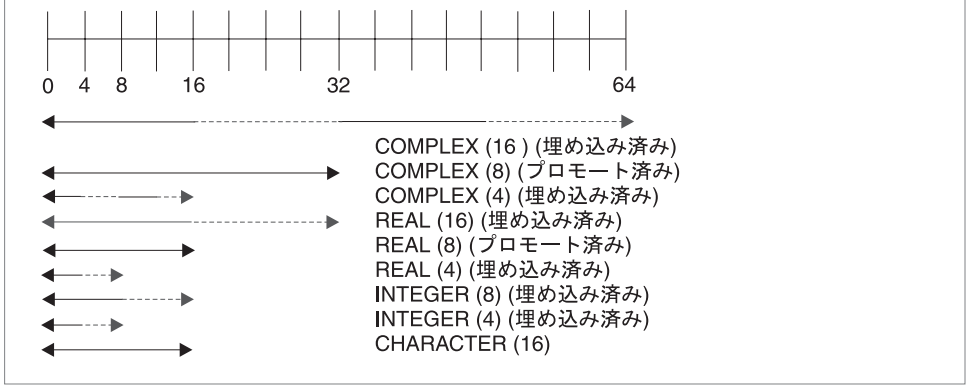


図 11. -qautodbl=dblpad8 を指定した場合のストレージの関係

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
```

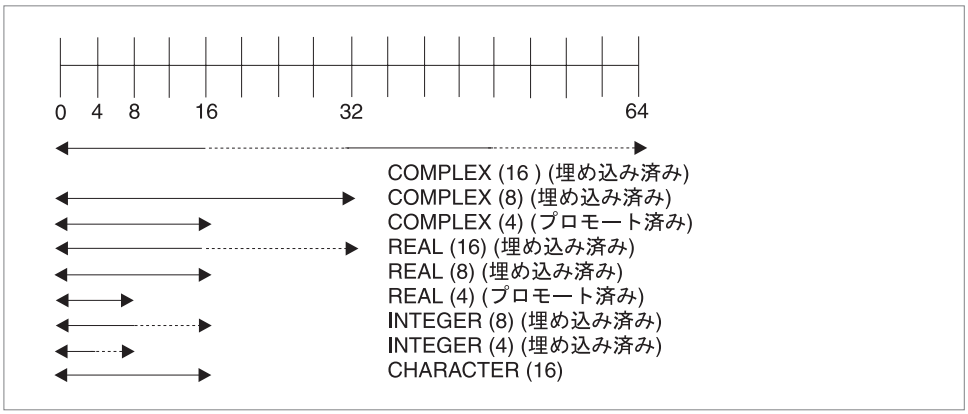


図 12. -qautodbl=dblpad を指定した場合のストレージの関係

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 C. Mathematical Acceleration Subsystem (MASS) の使用

XL Fortran Advanced Edition V9.1 for Linux は、該当する標準の数学ライブラリー関数で改善されたパフォーマンスを提供する、調整された算術組み込み関数のライブラリー・セットである Mathematical Acceleration Subsystem (MASS) を提供します。 MASS 関数と標準の数学ライブラリー関数では、正確度と例外処理は同一ではない可能性があります。

Linux の MASS ライブラリーは、『ベクトル・ライブラリーの使用』で説明されているベクトル関数のライブラリーで構成されます。 402 ページの『MASS でのプログラムのコンパイルとリンク』には、MASS ライブラリーを使用するプログラムのコンパイル方法とリンク方法が記載されています。

ベクトル・ライブラリーの使用

Linux の MASS ライブラリーは、AIX 用の MASS ライブラリーのサブセットです。 Linux では、32 ビット・オブジェクトと 64 ビット・オブジェクトは単一のライブラリーで混在しない場合があるため、2 つのバージョンの MASS ライブラリー **libmassvp4.a** (32 ビット) および **libmassvp4_64.a** (64 ビット) が提供されます。

ベクトル・ライブラリーに含まれている単精度関数と倍精度関数は、表 28 で要約されています。関数のインターフェース宣言を指定するには、ソース・ファイルに **massv.include** を入れます。

表 28. MASS ベクトル・ライブラリー関数

倍精度関数	単精度関数	引き数	説明
vrec	vsrec	(y,x,n)	i=1,..,n の場合に、y(i) を x(i) の逆数に設定します。
vrsqrt	vsrsqrt	(y,x,n)	i=1,..,n の場合に、y(i) を x(i) の平方根の逆数に設定します。
vsqrt	vssqrt	(y,x,n)	i=1,..,n の場合に、y(i) を x(i) の平方根に設定します。

以下の例は、一部の MASS 倍精度ベクトル関数のインターフェース宣言を示しています。

```
interface

subroutine vsqrt (y, x, n)
  real*8 y(*), x(*)
  integer n        ! Sets y(i) to the square root of x(i), for i=1,..,n
end subroutine vsqrt

subroutine vrsqrt (y, x, n)
  real*8 y(*), x(*)
  integer n        ! Sets y(i) to the reciprocal of the square root of x(i),
```

```
                  ! for i=1,..,n
end subroutine vrsqrt

end interface
```

以下の例は、一部の MASS 単精度ベクトル関数のインターフェース宣言を示しています。

```
interface

subroutine vssqrt (y, x, n)
  real*4 y(*), x(*)
  integer n       ! Sets y(i) to the square root of x(i), for i=1,..,n
end subroutine vssqrt

subroutine vsrsqrt (y, x, n)
  real*4 y(*), x(*)
  integer n       ! Sets y(i) to the reciprocal of the square root of x(i),
                  ! for i=1,..,n
end subroutine vsrsqrt

end interface
```

MASS ベクトル関数の一貫性

MASS ベクトル・ライブラリーのすべての関数は、ベクトル内での位置とベクトル長に関係なく、指定された入力値が常に同じ結果を生成するという意味で、一貫性があります。

MASS でのプログラムのコンパイルとリンク

MASS ライブラリーのルーチンを呼び出すアプリケーションをコンパイルするには、**-l** リンカー・オプションで **massvp4** (32 ビット) または **massvp4_64** (64 ビット) を指定します。たとえば、MASS ライブラリーがデフォルト・ディレクトリーにインストールされている場合は、以下のいずれかをを指定できます。

```
xlf progf.f -o progf -lmassvp4
xlf progf.f -o progf -lmassvp4_64 -q64
```

MASS 関数は、最も近い値への丸めモードで、浮動小数点例外のトラッピングを使用不可にして実行する必要があります。(これらは、デフォルトのコンパイル設定です。)

付録 D. 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
文字形式フィールドの最大幅	(2**31)-1
形式仕様の最大長	(2**31)-1
ホレリス定数および文字定数編集記述子の最大長	(2**31)-1
固定ソース形式ステートメントの最大長	6,700
自由ソース形式ステートメントの最大長	6,700
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数とレコード長	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数とレコード長	レコード番号は最大 (2**63)-1 で、レコード長は最大 (2**63)-1 バイトです。 ただし、不定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最大レコード長は (2**31)-1 バイトです。
配列次元の許容境界範囲	配列次元の境界は、 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。
外部装置の許容数	0 から (2**31)-1 **2**
数値形式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 つのステートメント (最大サイズ 6 700 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 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 Fortran Language Application Program Interface* バージョン 2.0 (November 2000) の仕様から抜粋されたものです。 Copyright 1997-2000 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

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重大な不正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適合し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** ステートメントと似ているが、**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・オブジェクトの型と長さの仕様。明示的な仕様が指定されていない場合、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** のいずれかのステートメントから、対応する **END** ステートメントまでの、インターフェース・ブロック内のステートメントの順序列。

インターフェース・ブロック (interface block). **INTERFACE** ステートメントから、対応する **END INTERFACE** ステートメントまでのステートメントの順序列。

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ステートメント。オブジェクトには、初期値を割り当てることができる。

仮引き数 (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**、ステートメント関数のいずれかのステートメント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.        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) 引き数キーワードは、仮引き数のための名前を指定する。

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). ステートメント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). **SELECT CASE**、**DO**、**IF**、**WHERE** のいずれかのステートメントで始まり、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). ステートメント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・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** ステートメント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). 分割できない固有の解釈を持つ文字の順序列。

シグナル通知 NaN (signalling 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、ステートメント関数のいずれかの特性を記述するステートメントで、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。

実引き数 (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** ステートメントで指定される。

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ステートメント。このステートメントは、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 (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。ステートメントには大きく分けて、実行可能と実行不能の 2 つのクラスがある。

ステートメント関数 (statement function). 後ろに仮引き数のリストが続く名前。これは、組み込み式または派生型の式と等価であり、プログラム全体にわたってこれらの式の代わりに使用することができる。

ステートメント・ラベル (statement label). 1 ～ 5 桁の番号。ステートメント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** ステートメント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). 1 つ以上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). ステートメントの連続的な順次呼び出しを変更するのに使用されるステートメント。制御ステートメントは、条件ステートメント (**IF** など) の場合と、命令ステートメント (**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** ステートメント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** ステートメント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). ある仮引き数が実引き数と関連しており、かつ、この実引き数が呼び出しプロシージャーに存在する仮引き数である場合、または呼び出しプロシージャーの仮引き数でない場合、この仮引き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** ステートメント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** の各ステートメント。

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 ステートメントで使用する名前のリストを指定する NAMELIST ステートメント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). 入力または出力ステートメント内の変数のリスト。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). ステートメントの一部であり、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。

引き数 (argument). 関数やサブルーチンへ引き渡される式。*実引き数 (actual argument)*、*仮引き数 (dummy argument)* も参照。

引き数関連付け (argument association). プロシージャー起動時の実引き数と仮引き数の関係。

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。 たとえば、入力イベントはユーザーによって制御され、プログラムは入力イベントを後で読み取ることができる。

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** ステートメントが含まれていると、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** ステートメント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。

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). 複数の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で宣言された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). ステートメント (たとえば CALL ステートメント) のために指定される引き数で、サブルーチンが RETURN ステートメント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き数。

[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. スレッドが堅固な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** ステートメントで指定される変数。 **DO** の範囲内にある 1 つ以上のステートメント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** ステートメント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 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** ステートメント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]

[マ行]

[ヤ行]

[ラ行]

[ワ行]

[数字]

A

B

C

M

N

O

P

Q

Y

Z

[特殊文字]

プログラム番号: 5724-K76

SC88-9699-01

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