

XL C/C++ Enterprise Edition for AIX



XL C/C++ スタートアップ・ガイド

バージョン 7.0

ご注意

本書の情報およびそれによってサポートされる製品を使用する前に、81 ページの『特記事項』に記載する一般情報をお読みください。

本書は、AIX® 用 XL C/C++ Enterprise Edition のバージョン 7.0.0 (プロダクト番号 5724-I11)、および新しい版で特に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。

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： SC09-7889-00
XL C/C++ Enterprise Edition for AIX
Getting Started with XL C/C++
Version 7.0

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 2004.7

この文書では、平成明朝体™W3、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™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

© Copyright International Business Machines Corporation 2004. All rights reserved.

© Copyright IBM Japan 2004

目次

本書について	v
強調表示の規則	v
構文図の読み方	v

XL C/C++ 概要	1
コマンド行 C および C++ コンパイラー	1
ライブラリー	1
標準 C++ ライブラリー	2
IBM Mathematics Acceleration Subsystem ライブラリー	2
IBM 分散デバッガー	2
その他のツールとユーティリティー	3
各国語サポート	4
資料とオンライン・ヘルプ	4

バージョン 7 の新機能	7
パフォーマンスおよび最適化	7
マシン・アーキテクチャーとハードウェア	7
POWER5 プロセッサのための新規組み込み関数	7
新規の XL C/C++ プラグマ	9
新しい最適化ユーティリティー	10
IBM Mathematics Accelerated Subsystem (MASS) ライブラリー	10
SMP スレッド・バインディング	11
業界標準の準拠	11
使用上の利便性	14
新規の XL C/C++ オプション	15

コンパイル環境のカスタマイズ	17
環境変数	17
パスのシンボリック・リンクの作成	18
構成ファイル	18

コンパイル・プロセスの制御	19
コンパイラーの呼び出し	19
オブジェクト・モデル	20
入出力ファイルの種類	21
デフォルトの動作	21

コンパイラー・オプションについて	23
コンパイラー・メッセージ	23
戻りコード	24
コンパイラー・メッセージ・フォーマット	24
gxc および gxc++ を使用した GNU C および C++ コンパイラー・オプションの再利用	25
gxc および gxc++ 構文	26
GNU C および C++ から XL C/C++ へのオプション・マッピング	27
オプション・マッピングの構成	30
オプションの要約: C コンパイラー	32
基本変換	34

特別な処理および制御	35
リンクおよびライブラリー関連オプション	36
オプションの要約: C++ コンパイラー	36

最適化について	37
最適化のための選択可能コンパイラー・オプション	38
プラグマの最適化入門	40

移植の考慮事項	41
言語に組み込まれた移植性の問題	41
コンパイル時エラーの診断	43
32 ビットおよび 64 ビット・アプリケーションの開発	44
AIX の 32 ビットおよび 64 ビット開発環境	45
AIX のオブジェクトとライブラリー	48
AIX における共用オブジェクトと共用ライブラリーの相違点	48
AIX における共用オブジェクトと静的オブジェクトの相違点	49
リンク時とロード時	50
リンク時エラーの診断	50
実行時エラーの診断	52
共用メモリーの並列処理	53
OpenMP ディレクティブ	54
AIX のスレッド	55
GNU C および C++ の移植性に関連したフィーチャー	64
関数属性	65
変数属性	66
型属性	66
GNU C および C++ アサーション	67
GNU C および C++ 関連のその他の拡張子	67

付録 A. 言語サポート	69
ISO/IEC 国際規格との互換性	69
ISO/IEC 14882:2003(E) 国際規格の互換性	69
ISO/IEC 9899:1990 国際規格の互換性	69
ISO/IEC 9899:1999 国際規格サポート	69
拡張言語レベル・サポート	72

付録 B. OpenMP 準拠とサポート	73
OpenMP ディレクティブ	74
OpenMP データ・スコープ属性文節	75
OpenMP ライブラリー関数	76
OpenMP 環境変数	78
OpenMP インプリメンテーション定義動作	79
OpenMP プログラムの調整	80

特記事項	81
プログラミング・インターフェース情報	83
商標	83

業界標準	83
----------------	----

本書について

XL C/C++ Enterprise Edition for AIX は、PowerPC アーキテクチャーで実行される AIX オペレーティング・システム用の最適化標準コマンド行コンパイラーです。このコンパイラーは、拡張 C および C++ プログラム言語で 32 ビット・アプリケーションおよび 64 ビット・アプリケーションを作成および保守するためのプロフェッショナル・プログラミング・ツールです。

本書 では XL C/C++ コンパイラーについて紹介します。説明する内容は、さまざまなコンパイラー呼び出しと、コンパイル環境をカスタマイズしてコンパイル・プロセスを制御する方法についてです。本書では、コンパイラーが実行できる変換のタイプ、入出力可能なファイル・タイプ、コンパイラー・オプションのカテゴリー別の要約、および既存アプリケーションの移植時の考慮事項も紹介します。また、本書では、アプリケーションのパフォーマンスを最適化する方法も簡単に紹介します。コンパイラーの機能を最適化すると、PowerPC プロセッサの多層アーキテクチャーを活用できます。

AIX および Linux[®] は、補完的なオペレーティング・システムです。XL C/C++ で開発したアプリケーション用に作成した makefile は、Linux プラットフォームへの移植で再使用するために簡単に適応させることができます。本書は、XL C/C++ を使用してプログラムを開発および保守したり、コンパイル、リンク、および実行時にパフォーマンスを改善する際に役立ちます。

本書は、読者が C および C++ プログラム言語、AIX オペレーティング・システム、および ksh シェルについての知識をもっていることを前提としています。

強調表示の規則

太字	コマンド、キーワード、およびシステムによって名前が事前定義されているその他の項目を識別します。
イタリック	その実際の名前または値がプログラマーによって提供されるパラメーターを識別します。イタリック は、新規用語を最初に言及する際にも使用されます。
例	特定のデータ値の例、ユーザーに表示されるテキストに類似したテキストの例、プログラム・コードの部分、システムからのメッセージ、またはユーザーが実際に入力すべき情報の例などを識別します。

これらの例は、言語の使用方法を説明するもので、実行時間の最小化、ストレージの節約、エラーのチェックを行うためのものではありません。これらの例では、言語構成の使用についてのすべては説明しません。例の中には、コードの一部だけを示し、コードを追加しないとコンパイルできないものもあります。

構文図の読み方

- 構文図は、左から右、上から下に、線のパスに従って読んでください。

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

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

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

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

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

注: 次の図で、statement は、C または C++ コマンド、ディレクティブ、またはステートメントを表しています。

- 必須項目は、水平線（メインパス）上に記述されます。

▶▶—statement—required_item—▶▶

- オプション項目は、メインパスの下に記述されます。

▶▶—statement—
 └optional_item┘—▶▶

- 2 つ以上の項目から選択可能な場合は、スタック内に垂直に記述されます。

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

▶▶—statement—
 └required_choice1┘
 └required_choice2┘—▶▶

いずれか 1 つの項目の選択がオプションの場合は、スタック全体がメインパスの下に記述されます。

▶▶—statement—
 └optional_choice1┘
 └optional_choice2┘—▶▶

デフォルト項目は、メインパスの上に記述されます。

▶▶—statement—
 └default_item┘
 └alternate_item┘—▶▶

- メインパスの線の上の左に戻る矢印は、繰り返し可能な項目を示します。

▶▶—statement—
 └repeatable_item┘—▶▶

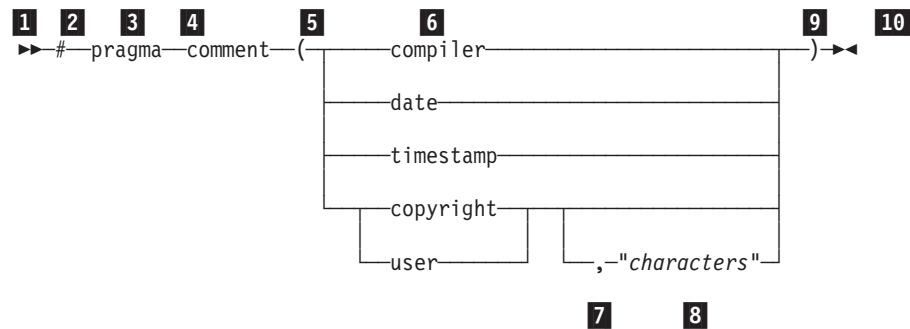
スタックの上の繰り返し矢印は、スタック内の項目から複数の項目を選択するか、1 つの項目を繰り返し選択できることを示しています。

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

変数は、イタリック体の小文字で記述されます (例えば、*identifier*)。変数は、ユーザー提供の名前または値を表します。

- 構文図に、句読記号、小括弧、算術演算子、または、ほかの同様な記号が示されている場合は、構文の一部としてこれらの文字を入力する必要があります。

次の構文図の例では、**#pragma comment** ディレクティブの構文を示しています。**#pragma** ディレクティブの詳細については、「*XL C/C++ ランゲージ・リファレンス*」を参照してください。



- 1 構文図の始まりを示します。
- 2 記号 # を最初に記述します。
- 3 キーワード pragma は、記号 # の次に記述されます。
- 4 プラグマの名前 comment は、キーワード pragma の次に記述します。
- 5 左括弧が必要です。
- 6 コメントの型を、表示されている compiler、date、timestamp、copyright、または user のうちいずれか 1 つだけ入力します。
- 7 コンマが、コメントの型 copyright または user とオプションの文字ストリングの間に必要です。
- 8 文字ストリングをコンマの次に記述します。文字ストリングは、二重引用符で囲みます。
- 9 右小括弧は必須です。
- 10 これが、構文図の終わりを示します。

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

```

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

XL C/C++ 概要

IBM XL C/C++ Enterprise Edition V7.0 は、AIX オペレーティング・システム用の最適化標準コマンド行コンパイラーおよび PowerPC アーキテクチャーです。このコンパイラーは、拡張 C および C++ プログラム言語で 32 ビット・アプリケーションおよび 64 ビット・アプリケーションを作成および保守するためのプロフェッショナル・プログラミング・ツールです。このコンパイラーは、パフォーマンスとクロスプラットフォームの移植性に重点を置いて進化し、他のプラットフォームでのリリースから獲得した柔軟性、フィーチャー、および改良を含む、完成したテクノロジーを表しています。

この製品は、IBM VisualAge C++ Professional for AIX バージョン 7.0 の後継リリースです。IBM は VisualAge C++ を XL C/C++ に商標変更しました。

XL C/C++ には、コンパイラー自体に加え、プログラムを効果的に作成するために役立つ、ライブラリー、ユーティリティー、およびツールが同梱されています。コンパイラーの資料には、コンパイラー呼び出しとすべてのコマンド行ユーティリティーに関する、検索可能なヘルプ・システム、PDF 資料、およびマニュアル・ページが含まれています。

コマンド行 C および C++ コンパイラー

XL C/C++ では、基本コンパイラー呼び出しコマンドを選択でき、さまざまなバージョン・レベルの C および C++ 言語をサポートしています。各呼び出しコマンドは、言語レベルのコンパイラー・サブオプション、その他の関連言語フィーチャーのオプション、および関連する定義済みマクロを自動的に設定します。多くの場合、C ソース・ファイルをコンパイルするときは **xlc** コマンドを使用し、C++ ソース・ファイルをコンパイルするとき、または C と C++ の両方のソース・ファイルが存在するときは **xlc** コマンドを使用します。






さまざまな基本コマンドがあり、特殊環境およびファイル・システムの要件をサポートしています。各種コマンドは、基本コマンドにサフィックスを付加することによって形成されます。例えば、サフィックス **_r** が付いたコマンドを使用すると、コンパイラーが、再入可能バージョンとスレッド・セーフ・バージョンの関数とライブラリーを使用するようになります。

また、**gxlc** および **gxlc++** ユーティリティーは特殊なコンパイラー呼び出しです。

ライブラリー

XL C/C++ では、以下のライブラリーを出荷しています。

- SMP ランタイム・ライブラリーは、明示的な並列処理と自動化された並列処理の両方をサポートします。
- 調整された数学組み込み関数から成る、32 ビット・モードと 64 ビット・モード用の IBM Mathematics Acceleration Subsystem (MASS) ライブラリー。

- メモリー・デバッグ・ランタイムは、メモリー・リークを診断するために使用されます。
-  **標準 C++ ライブラリー** (標準 C ライブラリーと標準テンプレート・ライブラリーを含む) は、標準 C++ に準拠するコードを作成するために使用できます。
-  **C++ ランタイム・ライブラリー**には、コンパイラーが必要とするサポート・ルーチンが含まれています。
-  **USL Complex Mathematics Class Library** には、複素数を操作するためのクラスが含まれています。このライブラリーは、古いアプリケーションで使用するために提供されています。新しいアプリケーションでは、標準 C++ ライブラリーを使用する必要があります。
-  **UNIX System Laboratories (USL) I/O Stream Class Library** には、C++ の入出力機能のストリーム・クラスが含まれています。このライブラリーは、古いアプリケーションで使用するために提供されています。新しいアプリケーションでは、標準 C++ ライブラリーを使用する必要があります。
-  **デマングラー・ライブラリー**は、C++ コンパイラーが作成したリンケージ名をデマングリングするためのルーチンとクラスを提供します。

標準 C++ ライブラリー

XL C/C++ Enterprise Edition for AIX は、標準 C++ ライブラリーに準拠するインプリメンテーションである、変更されたバージョンの Dinkum C++ Library を出荷します。標準 C++ ライブラリーは、標準テンプレート・ライブラリー (STL) を構成する 13 のヘッダーを含む、51 のヘッダーから成ります。さらに、標準 C++ ライブラリーは、標準 C ライブラリーの 18 のヘッダーと連動して機能します。これらのヘッダーの関数は、入出力などの必要不可欠なサービスを実行します。また、頻繁に使用される操作の効果的なインプリメンテーションを提供します。

関連資料

- 「*Standard C++ Library Reference*」の『C++ Library Overview』

IBM Mathematics Acceleration Subsystem ライブラリー

XL C/C++ では、バージョン 7 以降、調整された数学組み込み関数から成る、32 ビットと 64 ビット用の IBM Mathematics Acceleration Subsystem (MASS) ライブラリーを出荷しています。MASS ライブラリーはスレッド・セーフで、対応する libm ルーチンに対して改良されたパフォーマンスを提供します。さらに、MASS ライブラリーは、コード変更を必要とすることなく、使用できます。

IBM 分散デバッガー

XL C/C++ は、ご使用のプログラムのエラーを検出および診断できるクライアント/サーバー・アプリケーションである、IBM 分散デバッガーとともに出荷されます。クライアント/サーバーの設計により、ネットワーク接続でアクセス可能なシステムで実行されているプログラム、およびご使用のワークステーション上にあるプログラムの両方をデバッグすることができます。

デバッグ・エンジン の別名でも知られるデバッガー・サーバーは、デバッグしたいプログラムが実行されているシステムと同じシステムで実行されます。このシステムは、ご使用のワークステーション、またはネットワーク経由でアクセス可能なシステムです。ワークステーションで実行されているプログラムをデバッグする場合は、ローカル・デバッグ を実行します。ネットワーク接続でアクセス可能なシステムで実行されているプログラムをデバッグする場合は、リモート・デバッグ を実行します。

分散デバッガー・クライアントは、プログラムの実行を制御するためにデバッグ・エンジンが使用するコマンドをユーザーが発行することができる、グラフィカル・ユーザー・インターフェースです。例えば、ブレークポイントの設定、コードのステップスルー、および変数の内容の検査を実行できます。分散デバッガーのユーザー・インターフェースを使用すると、さまざまな言語で作成された複数のアプリケーションを、単一のデバッガー・セッションからデバッグすることができます。デバッグする各プログラムは、別個のプログラム・ページに表示されます。表示される情報の種類は、接続先のデバッグ・エンジンによって異なります。

分散デバッガーは、デフォルトでは `/usr/idebug` ディレクトリーにインストールされます。分散デバッガーを開始するには、コマンド行で `idebug` と入力します。

その他のツールとユーティリティー

CreateExportList コマンド

指定のオブジェクト・ファイルのセットで見つかったすべてのグローバル・シンボルのリストを含むファイルを作成します。

c++filt Name Demangling Utility

XL C/C++ が C++ プログラムをコンパイルする場合、すべての関数名および他の ID を、タイプ情報およびスコープ情報を含めてエンコードします。このエンコード・プロセスを、マングリングといいます。このユーティリティーは、マングルされた名前を、元のソース・コード名に変換します。

linkxlc コマンド

C++ の `.o` ファイルと `.a` ファイルとをリンクします。このコマンドは、XL C/C++ コンパイラーをインストールせずに、システムでリンク操作を行うために使用されます。

makeC++SharedLib コマンド

XL C/C++ コンパイラーがインストールされていないシステムで C++ 共有ライブラリーを作成できるようにします。

cleanpdf コマンド

PDFDIR ディレクトリーを管理するために使用されるプロファイル指示フィードバックに関連するコマンド。指定されたディレクトリー、PDFDIR ディレクトリー、または現行ディレクトリーからすべてのプロファイル情報を除去します。

mergepdf コマンド

プロファイル指示フィードバック (PDF) に関連するコマンドで、2 つ以上の PDF レコードを単一のレコードに結合するときに、それぞれのレコード

の重要性に応じて重みを付けることができます。これらの PDF レコードは、同一の実行可能ファイルから派生しなければなりません。

resetpdf コマンド

resetpdf コマンドの現行の振る舞いは cleanpdf コマンドと同じで、他のプラットフォームでの以前のリリースとの互換性を保持しています。

showpdf コマンド

プロファイル指示フィードバック・トレーニング実行 (オプション -qpdf1 および -qshowpdf を指定したコンパイル) で実行されたすべてのプロシージャの呼び出しおよびブロック数を表示するコマンド。

gxlc および gxlc++ ユーティリティ

GNU C または GNU C++ 呼び出しコマンドを対応する **xlC** または **xlC** コマンドに変換し、XL C/C++ コンパイラーを呼び出す、呼び出しメソッド。このユーティリティは、GNU コンパイラーでビルドされた既存アプリケーションで使用した makefile に対する変更数を最小限にすることと、XL C/C++ への移行を容易にすることを目的としています。

関連資料

- 「XL C/C++ プログラミング・ガイド」の『CreateExportList コマンド (CreateExportList Command)』
- 「XL C/C++ プログラミング・ガイド」の『c++filt Name Demangling Utility』
- 「XL C/C++ プログラミング・ガイド」の『linkxlc コマンド (linkxlc Command)』
- 「XL C/C++ プログラミング・ガイド」の『makeC++SharedLib コマンド (makeC++SharedLib Command)』

各国語サポート

XL C/C++ は、Unicode 規格、マルチバイト文字、UTF-16 および UTF-32 ストリング・リテラル、複数のロードされたロケール、および双方向性をサポートします。これらのフィーチャーにより、国際化対応アプリケーションを容易に作成できるようになります。

関連資料

- 「XL C/C++ ランゲージ・リファレンス」の『ユニコード規格』
- 「XL C/C++ コンパイラー・リファレンス」の『各国語サポート』

資料とオンライン・ヘルプ

XL C/C++ Enterprise Edition for AIX では、以下の形式で製品資料を提供しています。

- README ファイル。
- インストール可能なマニュアル・ページ。
- 検索可能な HTML ベースのヘルプ・システム
- PDF 文書。

これらの項目は以下の場所にあるか、以下の場所でアクセスできます。

README ファイル	README ファイルは、/usr/vacpp/ およびインストール CD のルート・ディレクトリにあります。
マニュアル・ページ	マニュアル・ページは、コンパイラ呼び出しおよび製品に提供されているすべてのコマンド行ユーティリティのために提供されています。
HTML ベースのヘルプ・システム	インフォメーション・センターと呼ばれる検索可能なヘルプ・システムは、HTML ファイルで構成されています。ヘルプ・システムは、プライベート・イントラネットにインストールするか、製品 Web サイトでオンラインで表示することができます。
PDF 文書	PDF ファイルは、/usr/vacpp/doc/\$LANG/pdf ディレクトリにあります。Adobe Acrobat Reader で表示および印刷ができます。Adobe Acrobat Reader をインストールしていない場合には、 http://www.adobe.com からダウンロードすることができます。

XL C/C++ PDF 文書の全ライブラリーは、以下のファイルで構成されています。

install.pdf

「XL C/C++ インストール・ガイド」には、コンパイラのインストール、マニュアル・ページの使用可能化、および検索可能な HTML のヘルプ・システムの設定に関する説明が含まれています。

getstart.pdf

「XL C/C++ スタートアップ・ガイド」には、XL C/C++ コンポーネントの概要、新フィーチャーの説明、コンパイル環境およびプロセスのカスタマイズに関する手引き情報、カテゴリ別のコンパイラ・オプションの概要表、パフォーマンス最適化およびチューニングに対する概要、およびアプリケーションの AIX プラットフォームへの移植に関する一般情報が含まれています。

language.pdf

「XL C/C++ ランゲージ・リファレンス」には、GNU C および g++ で元々開発されたアプリケーションの移植に関するインプリメンテーション定義の拡張機能などの、C および C++ プログラム言語の IBM インプリメンテーションに関する情報が記載されています。

compiler.pdf

「XL C/C++ コンパイラ・リファレンス」には、並列処理での使用も含め、コンパイラ・オプション、プラグマ、マクロ、および組み込み関数についての情報が記載されています。

stdlib.pdf

「*Standard C++ Library Reference*」には、XL C/C++ とともに出荷される標準テンプレート・ライブラリーを含む、標準 C++ ライブラリーに関する詳細情報が記載されています。

proguide.pdf

「XL C/C++ プログラミング・ガイド」には、他の資料で説明されていない、XL C/C++ を使用したプログラミングについての情報が記載されています。

legacy.pdf

「*C/C++ Legacy Class Libraries Reference*」には、前のリリースのコンパイ

ラーとの互換性のために XL C/C++ Enterprise Edition とともに出荷される、USL Complex Mathematics Class Library および USL I/O Stream Library に関する情報が記載されています。

debug.pdf

「*IBM Distributed Debugger*」には、分散デバッガー・ツールの資料が含まれています。

追加情報へのアクセス

XL C/C++ の最新情報については、製品資料および以下の URL のサポート・ページをご覧ください。さらに、IBM Technical Support Organization が作成した IBM Redbooks には、実際の経験からの現実的なシナリオに基づいた技術情報が含まれています。

- インフォメーション・センター (<http://www.ibm.com/software/awdtools/vacpp/library>)
- 製品サポート・サイト (<http://www.ibm.com/software/awdtools/ccompilers>)
- IBM Redbooks (<http://www.redbooks.ibm.com>)

XL C/C++ でのアプリケーション開発に役立つと思われる Redbooks としては、以下が挙げられます。

- *AIX 5L ポーティング・ガイド*、SG88-6705-00。
- *AIX における C および C++ アプリケーションの開発および移植*、SG88-6713-00。
- *POWER4 Processor Introduction and Tuning Guide*、SG24-7041-00。
- *Scientific Applications in RS/6000 SP Environments*、SG24-5611-00。
- *Understanding IBM eServer pSeries Performance and Sizing*、SG24-4810-01。

バージョン 7 の新機能

XL C/C++ Enterprise Edition for AIX の新規フィーチャーおよび機能拡張は、以下の 3 つのカテゴリに分類することができます。パフォーマンスおよび最適化、業界標準への準拠、および使用上の利便性です。

パフォーマンスおよび最適化

新フィーチャーと機能拡張の多くは、最適化とパフォーマンス・チューニングに分類されるものです。

マシン・アーキテクチャーとハードウェア

オプション **-qarch** および **-qtune** の改良

コンパイラー・オプション **-qarch** を使用すると、指定されたマシン・アーキテクチャーに生成される特定の命令を制御することができます。また、オプション **-qtune** を使用すると、命令、スケジューリング、および他の最適化を調整し、指定されたハードウェアのパフォーマンスを向上することができます。これらのオプションを共に使用すると、指定されたアーキテクチャーに最良のパフォーマンスを与えるアプリケーション・コードを生成できます。この 2 つのオプションを組み合わせてうまく使用することが、IBM プロセッサーおよびハードウェアを最大限に活用する鍵となります。本リリースでは、これらのオプションの組み合わせを機能拡張することにより、POWER5 および PowerPC 970 ハードウェア・プラットフォームがサポートされ、また使いやすさも向上しました。

-qarch で特定のアーキテクチャーを指定した場合、デフォルトの **-qtune** サブオプションを使用してコンパイルすると、そのアーキテクチャーに最適なパフォーマンスを与えるコードを生成します。現在、オプション **-qarch** には、アーキテクチャーのグループを指定することができます。 **-qtune=auto** を使用してコンパイルすると、指定したグループ内の全アーキテクチャーで稼働するコードを生成しますが、命令シーケンスはコンパイルするマシンのアーキテクチャーで最適なパフォーマンスを行うものになります。

POWER5 プロセッサーのための新規組み込み関数

以下の組み込み関数は、すべての PowerPC システムで使用可能です。POWER5 システムでは、これらの関数は、POWER5 命令を使用して POWER5 ハードウェアを利用します。サポートされているすべての組み込み関数は、「XL C/C++ コンパイラー・リファレンス」に詳しく説明されています。

すべての PowerPC システムのための新規組み込み関数

関数	説明
<code>int __popcnt4(unsigned int);</code>	32 ビット整数のビット・セット (=1) の数を返します。
<code>int __popcnt8(unsigned long long);</code>	64 ビット整数のビット・セット (=1) の数を返します。
<code>int __poppar4(unsigned int);</code>	ビットの奇数が 32 ビットの整数に設定されている場合、1 を返します。その他の場合は、0 を返します。

すべての PowerPC システムのための新規組み込み関数

関数	説明
int __poppar8 (unsigned long long);	ビットの奇数が 64 ビットの整数に設定されている場合は、1 を返します。その他の場合は、0 を返します。
unsigned long __mfspr(const int);	指定の特殊目的のレジスターに値を返します。
void __mtspr(const int, unsigned long);	const int で指定した特殊目的のレジスターを設定します。
unsigned long __mfmsr();	マシン状態レジスターを返します。
void __mtmsr(unsigned long);	マシン状態レジスターを設定します。

以下の組み込み関数は、POWER5 プロセッサでのみ使用可能です。

関数	説明
double __fre(double);	浮動小数点逆数演算の結果を返します。結果は、1/x の倍精度の推定値です。
float __frsqrtes(float);	逆数平方根演算の結果を返します。結果は、x の平方根の逆数の単精度推定値です。
unsigned long __popcntb (unsigned long);	ソース・オペランドの各バイトで 1 ビットをカウントし、そのカウントを対応する結果のバイトに入れます。
void __protected_unlimited_stream_set_go(unsigned int direction, const void* addr, unsigned int ID);	ID ID を使用する無制限の長さの protected ストリームを確立します。ストリーム ID の範囲は、0 から 15 である必要があります。ストリームは addr でキャッシュ・ラインから開始します。ストリームは、direction で指定したように、インクリメンタル・メモリー・アドレスまたはデクリメンタル・メモリー・アドレスのいずれかから取り出します。インクリメンタル・メモリー・アドレス (すなわち順方向) の場合 direction の値は 1 で、デクリメンタル・メモリー・アドレスの場合 direction の値は 3 です。ストリームは、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。(PowerPC 970 および POWER5 で使用可能。)
void __protected_stream_set(unsigned int direction, const void* addr, unsigned int ID);	ID ID を使用する限られた長さの protected ストリームを確立します。ストリームは addr でキャッシュ・ラインから開始し、その後、direction で指定したように、インクリメンタル・メモリー・アドレスまたはデクリメンタル・メモリー・アドレスのいずれかから取り出します。ストリームは、ハードウェアで検出されるストリームに置き換えられないように保護されます。
void __protected_stream_count(unsigned int unit_cnt, unsigned int ID);	ID によって識別する限られた長さの protected ストリームにキャッシュ・ライン数を設定します。キャッシュ・ライン数は、パラメーター unit_cnt によって指定し、0 から 1023 の範囲にする必要があります。
void __protected_stream_go();	すべての限られた長さの protected ストリームの事前取り出しを開始します。
void __protected_stream_stop(unsigned int ID);	ID によって識別する protected ストリームの事前取り出しを停止します。

関数	説明
<code>void __protected_stream_stop_all();</code>	すべての <code>protected</code> ストリームの事前取り出しを停止します。

浮動小数点除法のための新規組み込み関数

このリリースには、浮動小数点除法用の 4 つの新規組み込み関数が組み込まれました。浮動小数点除法アルゴリズムのソフトウェア・インプリメンテーションは、PowerPC アーキテクチャーを利用しており、また、ベクトル・コンテキストでの使用時に、対応するハードウェア命令よりもかなり高速になる可能性があります。これらの新規の組み込み関数は、POWER5 を含む、すべての PowerPC プロセッサでサポートされます。

ハードウェア除法命令は、ソース・プログラムに浮動小数点除法がコード化されているが、コンパイラーがより高速と考える程度に応じて、ハードウェアまたはソフトウェアの除法コードの間での選択を行う場合、デフォルトで取得されます。ユーザーは新規の組み込み関数を使用することで、明示的にソフトウェア・アルゴリズムを呼び出すことができます。これらのルーチンを呼び出す際、デフォルトの丸めモード（最近似値に丸め）が有効でなければなりません。

浮動小数点除法のための組み込み関数

関数	説明
<code>double __swdiv_nochk(double, double);</code>	double 型の浮動小数点除法。範囲検査なし。引き数の制約事項。無限大に等しい分子、または、無限大、ゼロ、または正規化解除に等しい分母は許可されません。
<code>double __swdiv(double, double);</code>	double 型の浮動小数点除法。引き数の制限はなし。
<code>float __swdivs_nochk(float, float);</code>	float 型の浮動小数点除法。範囲検査なし。引き数の制約事項。無限大に等しい分子、または、無限大、ゼロ、または正規化解除に等しい分母は許可されません。
<code>float __swdivs(float, float);</code>	double 型の浮動小数点除法。引き数の制限はなし。

新規の XL C/C++ プラグマ

プラグマ・ディレクティブは「XL C/C++ コンパイラー・リファレンス」に詳しく説明されています。

プラグマ	説明
<code>#pragma unrollandfuse</code>	ネストされた for ループを最適化するプラグマ。コンパイラーに命令して、外部ループの本体部分、つまりループ・ネスト自体を複製し、複製したものを単一のループ・ネストに展開します。
<code>#pragma stream_unroll</code>	for ループに含まれるストリームを複数のストリームに分割します。大規模な反復数および小規模なストリーム数を持つループを対象としています。

プラグマ	説明
#pragma block_loop	コンパイラーに命令して、ループ・ネストの特定の for ループのためにブロック化ループを作成します。ブロック化ループとは、ループの反復部分をパーツまたはブロックに分割したものです。ブロック化ループ という外部ループを追加で作成し、このループの各ブロックで元のループを実行します。
#pragma loopid	for ループにスコープ固有の ID でマークを付けます。ID は #pragma block_loop および他のプログラムによって使用され、そのループでの変換を制御し、オプション -qreport の使用によってループ変換に関する情報を提供します。ID は、ブロック化ループを識別するためにも使用できます。
#pragma disjoint	C++ インプリメンテーションの追加。
#pragma unroll の拡張	ループの展開とは、ループを完了するのに必要な反復数を減らすために、ループの本体部分を複製することです。 #pragma アンロール・ディレクティブを指定すると、このディレクティブの直後に記述されている for ループが展開可能であることをコンパイラーに示します。このプラグマの機能が拡張され、最内部および最外部の両方の for ループに適用できるようになりました。ただし、拡張された #pragma 機能では、代替エントリー・ポイントを持つ for ループへの適用はまだ含まれていません。

新しい最適化ユーティリティー

このリリースには、プロファイル指示フィードバック (PDF) コンパイル・プロセスに関連した 2 つの新しいユーティリティーが含まれています。プロファイル指示フィードバックを使用することにより、コンパイラーは、その実行可能ファイルを幾つかの異なるシナリオで実行した結果に基づいて、実行可能ファイルを最適化できます。計測する実行可能ファイルをいずれかのシナリオで実行すると、副次的に PDF レコードも作成されます。このレコードは、そのプログラムの典型的な動作を定義するために照合するデータとなります。

showpdf コマンドでは、プロファイル指示フィードバック・トレーニング実行として実行したすべてのプロシーチャーの呼び出しおよびブロック数を表示できます。このユーティリティーでは、オプション -qpdf1 と -qshowpdf を指定してコンパイルする必要があります。

mergepdf コマンドでは、ユーザーが 2 つ以上の PDF レコードの関連する重要な部分を指定し、1 つのレコードに組み合わせることができます。これにより、ユーザーはより高い実行カウント (より長い実行時間) のトレーニング実行を補正し、そのトレーニング実行だけがプロファイル・データを決定付けてしまわないようにすることができます。

IBM Mathematics Accelerated Subsystem (MASS) ライブラリ

XL C/C++ では、バージョン 7.0 以降、調整された数学組み込み関数から成る IBM Mathematical Accelerated Subsystem (MASS) ライブラリを出荷しています。MASS

スカラー・ライブラリーである `libmass.a` には、AIX システム・ライブラリー `libm.a` で頻繁に使用される数学組み込み関数の上級セットが含まれています。MASS ベクター・ライブラリー `libmassv.a`、`libmassvp3.a`、および `libmassvp4.a` には、Fortran または C アプリケーションのいずれかで使用できる、調整されたスレッド・セーフ組み込み関数が含まれています。一般的なベクター・ライブラリーである `libmassv.a` には、IBM pSeries および RS/6000 ファミリーのすべてのコンピュータで実行されるベクター関数が含まれており、`libmassvp3.a` と `libmassvp4.a` には、それぞれ POWER3 および POWER4 プロセッサ用に特別に調整された `libmassv.a` 関数のサブセットが含まれています。

SMP スレッド・バインディング

共用メモリ並列化 (SMP) は、オペレーティング・システムによってカーネル・スレッドで実行するようにスケジュールされるユーザー・スレッドを作成することで、インプリメントされます。一部のワークロードの場合、プロセッサに対してスレッドをバインディングすると、スレッド・マイグレーションのコストが回避されるので、パフォーマンスが改善される可能性があります。現在、SMP ランタイムによって作成されたスレッドは、特定のプロセッサにはバインドされず、スレッドのスケジューリングは AIX オペレーティング・システムが対処します。SMP スレッド・バインディング機能により、SMP ランタイムに動的にリンクされたプログラムは、スレッドを、ユーザーの指定のとおり、プロセッサにバインディングさせることができます。

SMP スレッド・バインディングでは、`XLSPMPOPT` 環境変数で設定される、2 つの部分からなるオプションを利用します。ユーザーは、バインドする最初のスレッドの CPU ID と、現行プロセッサから次のスレッドに進める (ストライドする) プロセッサの数を指定します。

開始 CPU ID を指定できることは、同じマシンで複数の OpenMP プログラムが実行されているときに有利です。どの OMP プログラムも、そのスレッドを、CPU 0 から開始してバインドした場合、不均衡が起こります。その場合、システムの最初のいくつかの CPU はロードされますが、残りの CPU は全く使用されません。ユーザーが 2 のストライドを指定できるようにすることで、非 HPC システムは、CPU 0 での開始を想定して、偶数の CPU ID にそのスレッドをバインドさせることができます。偶数の CPU へのバインディングにより、フル L2 キャッシュおよび帯域幅に対するスレッドが提供されます。

プロセッサ・バインディングを使用するプログラムは、動的論理区画 (DLPAR) 対応にならなければなりません。DLPAR の認識について詳しくは、AIX システムの資料の一部である、「*General Programming Concepts: Writing and Debugging Programs*」を参照してください。

業界標準の準拠

このセクションでは、さまざまな業界標準に準拠させるために、XL C/C++ によってインプリメントされる新規フィーチャーについて説明します。

ISO/IEC 14882:2003(E) Programming languages -- C++

XL C/C++ は、改訂された国際的な C++ 標準である ISO/IEC 14882:2003(E) Programming languages -- C++ に準拠しています。

XL C++ では、バージョン 7 以降、「*Draft Technical Report on Standard Library Extensions*」(TR1) に準拠した、順序なし連想コンテナのサポートが追加されています。標準 C++ ライブラリーの拡張として追加されたハッシュ関数および新規のハッシュ・ベースのコンテナは、以下のとおりです。

ヘッダー・ファイル	追加
標準ヘッダー <functional>	関数テンプレート <code>std::tr1::hash</code>
新規ヘッダー <unordered_set>	コンテナ <code>std::tr1::unordered_set</code> コンテナ <code>std::tr1::unordered_multiset</code>
新規ヘッダー <unordered_map>	コンテナ <code>std::tr1::unordered_map</code> コンテナ <code>std::tr1::unordered_multimap</code>

TR1 ライブラリーは、ネストされたネーム・スペース `std::tr1` で宣言されます。新規 XL C++ TR1 ライブラリー・コンポーネントは、マクロ `__IBMCPP_TR1__` を定義することによって使用可能になります。

C、C++、および Fortran に対する OpenMP API V2.0 サポート

OpenMP アプリケーション・プログラム・インターフェース (API) は、移植可能で拡張が容易なプログラミング・モデルであり、マルチプラットフォームでメモリーを共用する並列アプリケーションを C、C++、および Fortran で開発するための標準インターフェースを提供します。OpenMP API の仕様は、OpenMP 組織という、IBM を含む主なコンピューター・ハードウェアおよびソフトウェア・ベンダーの集まりによって定義されます。XL C/C++ Enterprise Edition for AIX は OpenMP 仕様 2.0 に準拠しています。コンパイラーは、以下の OpenMP V2.0 エLEMENTのセマンティクスを認識し、保持します。

- `#pragma omp` ディレクティブ内の複数文節のコンマ区切り文字。
- `num_threads` 文節。
- `copyprivate` 文節。
- `threadprivate` 静的ブロック・スコープ変数。
- C99 可変長配列のサポート。
- `private` 変数の冗長な宣言。
- タイミング・ルーチン `omp_get_wtime` および `omp_get_wtick`。

拡張ユニコードおよび NLS サポート

C 標準委員会の最近のレポートでは、C コンパイラーで C99 を拡張し、新規データ型を追加して UTF-16 および UTF-32 リテラルをサポートすることを推奨しています。また、C++ コンパイラーでも、C との互換性のために、これらの新規データ型をサポートすることを推奨しています。




さらに本リリースでは、アプリケーションが AIX V5.2 オペレーティング・システムで稼働している場合、C++ ランタイムでこのシステムの機能を使用して複数のロケールをロードすることができるようになりました。

Boost ライブラリーに対するサポート

XL C++ コンパイラーは、1.30.2 Boost ライブラリーでの高レベルの互換性を実現します。これらのライブラリーは再使用可能セット、標準化に適切な Open Source C++ ライブラリーを提供するために作成されました。詳しくは、Boost Web サイト <http://www.boost.org> を参照してください。

GNU C および C++ に関連する言語拡張機能

C99 に対する GNU C 拡張および標準 C++ に対する GNU C++ 拡張は、業界標準ではありません。しかし、独自のものではなく、ある程度一般的となっているオープン・ソース・コミュニティの言語フィーチャーです。XL C/C++ は、GNU C および C++ 拡張のサブセットをインプリメントしています。本リリースでは、以下の GNU C フィーチャーに対するサポートが追加されています。

フィーチャー	注釈
ラベルを値として使用	計算済み goto 文を含む。このフィーチャーは現在、GNU C インプリメンテーションと完全に互換性があります。
型属性	属性 <code>aligned</code> および <code>packed</code> 。 
関数属性	属性 <code>format</code> 、 <code>format_arg</code> 、 <code>always_inline</code> 、 <code>noinline</code> 。
代替キーワード	<code>__extension__</code> のインプリメンテーションを内部的に変更。
ネストされた関数	 C のみサポート。
共用体型へのキャスト	 C のみサポート。
引き数の変数番号を含むマクロ	<code>__VA_ARGS__</code> 引き数が指定されていない場合、 <code>__VA_ARGS__</code> の代わりに <code>ID</code> を使用し、末尾のコンマを除去する。
C の式オペランドを使用する gcc インライン・アセンブラー命令	部分的なサポートのみ。
GNU C 複素数型	C++ サポートが追加されました。
GNU C 16 進浮動小数点定数	C++ サポートが追加されました。
C99 複合リテラル	C++ サポートが追加されました。
長さゼロの配列	C++ サポートが追加されました。
可変長配列	C++ サポートが追加されました。

サード・パーティーの C++ ランタイム・ライブラリーへの対応

C++ コンパイラーは、アプリケーションがコア・ランゲージのみをサポートするように C++ アプリケーションをコンパイルすることができ、それによりそのアプリケーションをサード・パーティー・ベンダーの C++ ランタイム・ライブラリーとリンクすることができます。以下のアーカイブ・ファイルにより、このような機能が可能になっています。

lib*C*core.a	例外処理、RTTI、静的初期化、new および delete 演算子を含みます。以下のライブラリーは、いずれも含みません。 Input/Output、Localization、STL Containers、Iterators、Algorithms、Numerics、Strings。
libCcore.a	C++ ランタイム・ライブラリー libC.a のコア・ランゲージ版。
libC128core.a	libC128.a のコア・ランゲージ版。
libhCcore.a	libhC.a のコア・ランゲージ版

これらのアーカイブの使用を助けるために、以下の呼び出しコマンドが追加されました。

xlCcore xlC128core	xlCcore_r xlC128core_r	xlCcore_r7 xlC128core_r7
-----------------------	---------------------------	-----------------------------

使用上の利便性

新規 C++ コンパイラー呼び出し

コンパイラー呼び出し **xlC++** が、すべてのサポートされているプラットフォームで移植可能になりました。この呼び出しは、すべてのプラットフォームで使用可能な呼び出し **xlC** と同等であり、使用を推奨されています。ただし、**xlC** も完全にサポートされています。

資料

XL C/C++ は、検索可能な HTML ファイルから成るインフォメーション・センターとともに出荷されます。新規ヘルプ・システムの検索エンジンでは、前のリリースに比べ、各検索におけるヒットの妥当性が向上しています。インフォメーション・センターには、イントラネットにインストールして、ブラウザーで http://server_name:5312/help/index.jsp を指定することによってアクセスすることができます。また、製品のヘルプ・システムは、オンライン (<http://www.ibm.com/software/awdtools/vacpp/library>) でも表示可能です。

バージョン 7 以降、各コンパイラー呼び出しコマンドと各コマンド行ユーティリティーに関するマニュアル・ページが提供されています。コンパイラー呼び出しに対するマニュアル・ページは、前のリリースで提供されていたテキストのヘルプ・ファイルを置き換えるものです。

テンプレート・レジストリー機能拡張

C++ コンパイラーは、テンプレートのインスタンス化の登録による、テンプレートのインスタンス化のバッチ処理スキームを使用します。本リリースでは、このコンパイラーにより、作成されるテンプレート・レジストリー・ファイルにバージョン管理情報を追加しています。この情報は、コンパイラーによって内部的に使用され、どのバージョンのテンプレート・レジストリー・ファイル・フォーマットを使用する必要があるかを追跡します。コンパイラー・オプション `-qtemplateregistry` はデフォルトで使用可能に変更されています。

新規の XL C/C++ オプション

新規および変更されたコンパイラー・オプションが、「XL C/C++ コンパイラー・リファレンス」で詳しく説明されています。

オプション	説明および注釈
-qasm=gcc	C の式オペランドによるアセンブラー命令に対する部分的サポートを使用可能にします。 asm キーワードおよびその代替スペルを認識し、gcc 構文およびキーワードのセマンティクスを使用するようコンパイラーに命令します。デフォルトは、 -qnoasm です。
-qasm_as	asm ディレクティブでコードを処理するために、代替アセンブラー・プログラムを呼び出すのに使用するパスおよびフラグを指定します。このオプションは、コンパイラー構成ファイルで定義される as コマンドのデフォルト設定をオーバーライドします。
-qdirectstorage	ライトスルー使用可能またはキャッシュ禁止ストレージが、一定のコンパイル単位で参照される可能性があることを表明します。このオプションの意図は、PowerPC アーキテクチャーで使用可能な異なるストレージ管理属性のために、予期しない振る舞いを行わないようにすることです。デフォルトは、 -qnodirectstorage です。
-qkeepparm	パフォーマンスを改善するために、異なるメモリー・ロケーションに移動させる代わりに、レジスターに渡される関数のパラメーターがスタックに確実に保管されるようにします。デフォルトは、 -qnokeepparm です。
-qnoprefetch	自動的にソフトウェア事前取り出し命令を挿入しないようにコンパイラーに命令します。このオプションを使用すると、ユーザーは、事前取り出しという最適化の機能をオフにすることができます。デフォルトは、 -qprefetch です。
-qnotrigraph	どの言語レベルを指定しているかに関わらず、3 文字表記を解釈しないようにコンパイラーに命令します。AIX では、デフォルトは -qtrigraph です。
-qroptr	読み取り専用ポインターが .data セクションから .text セクションに移動できるようにコンパイラーに命令します。 .text セクションは常に読み取り専用で、ローダーまたはアプリケーションによって決して変更されません。アプリケーションの定数ポインターが .data セクションから .text セクションに移動することができ、複数のプロセスで共有される場合、メモリーの使用量が減少することがあります。 -qroptr を指定して作成されたコードは、共用ライブラリーでは有効ではありません。デフォルトは -qnoroptr で、読み取り専用ポインターは .data セクションから移動しません。
-qsaveopt	ソース・ファイルを対応するオブジェクト・ファイルにコンパイルするコマンド行オプションを保管するようにコンパイラーに命令します。このオプションは、コンパイルで .o ファイルが生成されない場合は効果がありません。デフォルトは、 -qnosaveopt です。
-qshowpdf	-qpdf1 を指定した場合、コンパイラーは、コンパイルされたアプリケーションに追加のプロファイル情報を挿入し、アプリケーションの全プロシージャーに対する呼び出しおよびブロック数を収集します。コンパイルされたアプリケーションを実行すると、呼び出しおよびブロック数をファイル ._pdf に記録します。 ._pdf の内容は showpdf ユーティリティーで検索することができます。デフォルトは、 -qnoshowpdf です。

オプション	説明および注釈
-qsourcetype	入力ファイル名の解釈を制御します。デフォルトの振る舞いは、ソース・ファイルのプログラミング言語がそのファイル名のサフィックスによって示すものです。デフォルトは、 -qsourcetype です。
-qweaksymbol	インライン関数に、外部結合、 <code>#pragma weak</code> で <code>weak</code> と指定した ID、または <code>__attribute__((weak))</code> で <code>weak</code> と指定した関数をもつ、弱いシンボルを生成するようにコンパイラーに命令します。また、このオプションを指定すると、外部インライン関数を含む C++ プログラムをコンパイルする際、重複するシンボルのリンカー・メッセージ警告が抑制されます。デフォルトは、 -qnoweaksymbol です。
-qutf	ユニコード・エンコード形式に 16 および 32 ベースのストリング・リテラルを提供する UTF リテラル構文の認識を使用可能にします。
-qflltrap=nanq	NaNQ (Not a Number Quiet) をキャッチするために、コードに追加の命令を生成するようにコンパイラーに命令します。このオプションは、有効な演算によって作成されたものを含む、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 NaNQ を検出することを目的としています。
-qipa=infrequentlabel	標準的なプログラム実行中にまれに呼び出されることがあるラベルのリストを指定します。これらのラベルの呼び出しで最適化を行う対象を限定することによって、コンパイラーでプログラムのその他の部分をより高速に処理できる場合があります。このオプションは、ユーザー定義のラベルにのみ適用されます。
-qlanglvl=newexcp	-qlanglvl=newexcp サブオプションは、割り振り関数がストレージの割り振りに失敗したことでスローされる例外のデータ型を決定します。このサブオプションが有効な場合、ストレージの割り振りに失敗すると、空でない <code>throw</code> 式で宣言された割り振り関数が、クラス <code>std::bad_alloc</code> または <code>std::bad_alloc</code> から派生したクラスの例外をスローします。この振る舞いは C++ 規格に準拠しています。このサブオプションが明示的に指定されていない場合、例外仕様の有無に関係なく、ストレージの割り振りに失敗した割り振り関数はヌル・ポインターを戻します。
-qshowinc サブオプション	-qsource と共に使用し、選択的にユーザーにプログラム・ソース・リストでヘッダー・ファイルまたはシステム・ヘッダー・ファイルを表示します。デフォルトは、 -qnoshowinc です。新規サブオプションにより、ヘッダー・ファイルが含まれるものに対してさらに特定の制御が可能です。個々のサブオプションは、 <code>all</code> 、 <code>usr</code> (ユーザー・ヘッダー・ファイルを含む)、 <code>nouser</code> (ユーザー・ヘッダー・ファイルを除く)、 <code>sys</code> (システム・ヘッダー・ファイルを含む)、および <code>nosys</code> (システム・ヘッダー・ファイルを除く) です。区切り文字としてコロンの使用すると、複数のサブオプションを指定することができます。

コンパイル環境のカスタマイズ

このセクションでは、ファイル、およびライブラリーを含むディレクトリーの検索パスを指定するために XL C/C++ が使用する機構について説明します。その機構とは、環境変数、シンボリック・リンク、および構成ファイルです。

環境変数

コンパイル環境の一部に、ライブラリーやインクルード・ファイルなどの特殊ファイルの検索パスがあります。コンパイラーでは次のシステム変数が使用されます。

LIBPATH 動的にロードされたライブラリーのディレクトリー・パスを指定します。他のシステムの **LD_LIBRARY_PATH** 変数と関連があります。リンケージ・エディターとシステム・ローダーに影響します。

環境で **LIBPATH** を設定すると、**ld** コマンドによって値が読み取られて使用されます。直接的に、またはコンパイラー呼び出しのいずれかの一部としてリンク操作が行われると、従属ライブラリーを検索するために **LIBPATH** の値が使用され、変数の内容は、その結果得られるモジュールに保管されます。**-L** コマンド行オプションがない場合は、**LIBPATH** の内容は、デフォルトのライブラリー検索パスの前に付加されます。

コマンド行で **-L** オプションが使用されると、リンク時に **LIBPATH** が無視されます。1 つ以上の **-L** オプションによってコマンド行で指定されたパスは、現れる順序で連結されます。デフォルトのライブラリー検索の前に複合パス指定が付加され、構成されたモジュールのローダー・セクションに保管されます。

LIBPATH は、動的ロードが使用されるかどうかに応じて、実行時にシステム・ローダーによって異なる方法で使用されます。

LIBPATH 環境変数は実行時に検査され、プログラムの特権がプログラムを呼び出したユーザーの特権と異なる場合に、クリアされます。代替の有効なユーザーまたはグループ ID で実行するように設計されているプロセスは、アプリケーション内に組み込まれている、信頼できるロケーションの従属モジュールのみを検出する必要があります。

MANPATH 製品マニュアル・ページのディレクトリー・パスを指定します。

NLSPATH 各国語サポート・ライブラリーのディレクトリー・パスを指定します。

PATH コンパイラーの実行可能ファイルのディレクトリー・パスを指定します。

PDFDIR プロファイル・データ・ファイルを作成するディレクトリーを指定します。デフォルト値は設定されず、コンパイラーはプロファイ

ル・データ・ファイルを現在の作業ディレクトリーに入れます。プロファイル指示フィードバックの場合は、この変数を絶対パスに設定することをお勧めします。

TMPDIR

一時ファイルを作成するディレクトリーを指定します。デフォルトのロケーションは、高レベルの最適化には不適切である場合があります。高レベルの最適化の場合は、一時ファイルがディスク・スペースを大量に消費する可能性があるためです。

パスのシンボリック・リンクの作成

XL C/C++ のコマンド行インターフェースは、`/usr/bin` に自動的にインストールされません。絶対パスを指定せずにコンパイラーを呼び出すには、以下のステップのうちの 1 つを行なってください。

- `/usr/vacpp/bin` および `/usr/vac/bin` に含まれる特定のドライバーのための `/usr/bin` へのシンボリック・リンクを作成します。
- `PATH` 環境変数に `/usr/vac/bin` および `/usr/vacpp/bin` を追加します。

構成ファイル

構成ファイルは、コンパイラーを実行するたびに読み取られるオプションを指定するプレーン・テキスト・ファイルです。構成ファイルの名前は、`.cfg` ファイル名拡張子で終わります。

-F オプションを指定してコンパイラーを起動し、完全修飾ファイル名を指定することで、特定の構成ファイルを使用するようコンパイラーに指示することができます。

コンパイラー・オプション **-I** *directory_name* を使用すると、構成ファイル内の検索パスへディレクトリーを追加できます。構成ファイル自体は、制御するディレクトリー・パスを設定するために内部的に **-I** オプションを使用します。コンパイラーは、コマンド行の **-I** オプションによって指定されるディレクトリーを検索する前に、構成ファイル内の **-I** によって指定されるディレクトリーを検索します。

詳細については、「*XL C/C++ コンパイラー・リファレンス*」を参照してください。

コンパイル・プロセスの制御

コンパイル・プロセス全体は、以下の 3 つのフェーズから構成されています。プリプロセス、オブジェクト・コードへの変換、およびリンクです。デフォルトでは、コンパイラ呼び出しコマンドは、コンパイル・プロセスのすべてのフェーズを呼び出し、ソース・コードからプログラムを変換して、出力として実行可能ファイルを作成します。コンパイラを呼び出す際、入出力ファイルのファイル名を指定すると、コンパイラは、入出力ファイルのファイル名サフィックス (拡張子) から開始および終了フェーズを判断します。

また、適切なコンパイラ・オプションを使用すると、どのコンパイル・フェーズでも特定のタイプの出力ファイルを作成することができます。例えば、**xlc** または **xlc** コマンドを **-E** または **-P** オプションを指定して呼び出すと、入力ファイルに対してプリプロセス・フェーズだけが実行されます。コンパイラ呼び出しにより、コンパイラ、アセンブラ、またはリンカーを呼び出すかどうかを入力ファイル名の拡張子から判断します。

関連資料

- 「XL C/C++ コンパイラ・リファレンス」の『-qphsinfo コンパイラ・オプション』

コンパイラの呼び出し

XL C/C++ では基本コンパイラ呼び出しコマンドを選択でき、さまざまなバージョン・レベルの C および C++ 言語をサポートしています。各呼び出しコマンドは、言語レベルのコンパイラ・サブオプション、その他の関連言語フィーチャーのオプション、および関連する定義済みマクロを自動的に設定します。多くの場合、C ソース・ファイルをコンパイルするときは **xlc** コマンドを使用し、C++ ソース・ファイルをコンパイルするとき、または C と C++ の両方のソース・ファイルが存在するときは **xlc** コマンドを使用します。

基本呼び出しコマンド

xlc	cc	xlc++core
xlc++	c89	xlCcore
xlC	c99	

xlc++ 呼び出しは **xlC** 呼び出しと同等です。

基本コマンドにはさまざまなバリエーションがあり、特殊な環境やファイル・システムに適應できます。コマンドの各バリエーションは、基本コマンドにサフィックスを付け足すことで形成されます。

サフィックス	説明
_r _r4 _r7	スレッド・セーフなアプリケーションのコンパイルに使用されます。「再入可能コンパイラー呼び出し」とも呼ばれます。
128 128_r 128_r4 128_r7	long double 型の長さを 64 ビットから 128 ビットに増やし、128 ビット・バージョンの C および C++ ランタイムとリンクします。基本呼び出し c89 または c99 に付加する場合、サフィックスの前に下線を付加します (例えば、 c99_128)。

また、`gxc` および `gxc++` ユーティリティーは特殊なコンパイラー呼び出しです。

オブジェクト・モデル

オブジェクト・モデルは、C++ オブジェクトおよびヘルパー・データ構造がどのようにメモリに配置されているかを記述します。また、ネーム・マングリングにも影響します。オブジェクト・モデル `compat` および `ibm` は、AIX プラットフォームでサポートされています。これらは、コンパイラーが仮想関数テーブル、仮想基底クラスのサポートのタイプ、および使用されるネーム・マングリング体系を配置する方法が異なります。`compat` オブジェクト・モデルは、同じモデルでコンパイルされた他のモジュール、または以前のバージョンのコンパイラーでコンパイルされた他のモジュールと互換性のあるランタイム・モジュールを作成します。`ibm` オブジェクト・モデルは、特にソースが多くの仮想基底クラスとともにクラス階層を含む場合、パフォーマンスを改良することができます。このオブジェクト・モデルは、比較的小さな派生クラスを作成し、仮想関数テーブルへのアクセスが早くなります。

入出力ファイルの種類

コンパイラーは、ファイル名拡張子を使用して、適切なコンパイル・フェーズを判別し、関連するツールを呼び出します。

コンパイラーは、入力として次のタイプのファイルを受け入れます。

受け入れられる入力ファイル・タイプ

ファイル・タイプの説明	ファイル名拡張子	例
C および C++ ソース・ファイル	C 言語ソース・ファイルの場合は .c (小文字の c) C++ ソース・ファイルの場合は .C (大文字の c)、.cc、.cp、.cpp、.cxx、.c++	<i>file_name.c</i> <i>file_name.C</i> 、 <i>file_name.cc</i> 、 <i>file_name.cpp</i> 、 <i>file_name.cxx</i> 、 <i>file_name.c++</i>
プリプロセスされたソース・ファイル	.i	<i>file_name.i</i>
オブジェクト・ファイル	.o	hello.o
アセンブラー・ファイル	.s	check.s
アーカイブ・ファイル	.a	v1r5.a
ロード可能なモジュールまたは共用ライブラリー・ファイル	.so	my_shrplib.so
IPA 制御ファイル (-qipa= <i>file_name</i>)	<i>file_name</i> の命名規則は強制されない。	ipa.ct1

コンパイラーを呼び出すときに、次のタイプの出力ファイルを指定できます。

出力ファイルの種類

ファイル・タイプの説明	例
実行可能ファイル	デフォルトでは、a.out
オブジェクト・ファイル	<i>file_name.o</i>
ロード可能なモジュールまたは共用ファイル	<i>file_name.so</i>
アセンブラー・ファイル	<i>file_name.s</i>
プリプロセスされたファイル	<i>file_name.i</i>
リスト・ファイル	<i>file_name.lst</i>
ターゲット・ファイル	<i>file_name.u</i>

関連資料

- ・「XL C/C++ コンパイラー・リファレンス」の『-qsourcetype』コンパイラー・オプション

デフォルトの動作

オプションを指定せずにコンパイラーを呼び出すと、コンパイラーの動作は、次のデフォルト設定によって制御されます。

- ・ 構成ファイルに指定されたオプションを読み取り、呼び出そうとする。
- ・ デフォルトの位置合わせ -qalign=power を使用して構造体の位置合わせをする。
- ・ 最適化されていない a.out という名前の実行可能ファイルを現行ディレクトリーに生成する。

詳細については、「*XL C/C++* コンパイラー・リファレンス」を参照してください。

コンパイラー・オプションについて

コンパイラー・オプションは、さまざまな関数を実行します。例えば、コンパイラー特性の設定、生成されるオブジェクト・コードの記述、出力される診断メッセージの制御、および一部のプリプロセッサ関数の実行などです。コンパイラー・オプションは、コマンド行、構成ファイル内、ソース・コード内、またはこれらの技法の組み合わせで指定できます。明示的に設定されないほとんどのオプションが、デフォルト設定を受け入れます。

複数のコンパイラー・オプションを指定した場合、オプションの矛盾や非互換が起きる可能性があります。このような矛盾を一貫性のある方法で解決するために、これ以外の順位が指定されていない限り、コンパイラーは次の優先順位を適用します。

1. ソース・ファイルのオーバーライド
2. コマンド行のオーバーライド
3. 構成ファイルのオーバーライド
4. デフォルト設定

一般に、複数のコマンド行オプションでは、最後に指定されたものが優先されます。

注: **-I** コンパイラー・オプションは特殊なケースです。コンパイラーは、コマンド行で **-I** を使用して指定されたディレクトリーを検索する前に、vac.cfg ファイル内の **-I** で指定されたディレクトリーを検索します。これらのオプションは、先に認識されたものが優先されるのではなく、後から認識されたものが優先されます。

関連資料

- 詳細については、「[XL C/C++ コンパイラー・リファレンス](#)」を参照してください。


コンパイラー・メッセージ

XL C/C++ は、診断メッセージを 5 つのレベルに分類します。各重大度レベルは、コンパイラーの応答に関連付けられています。すべてのエラーでコンパイルが停止されるわけではありません。次の表に、重大度レベルに割り当てられた省略形と、関連付けられているコンパイラーの応答を示します。

重大度レベルとコンパイラーの応答

文字	重大度	コンパイラーの応答
I	通知	コンパイルは継続します。このメッセージは、コンパイルと途中で検出された条件を報告します。
W	警告	コンパイルは継続します。このメッセージは、有効であり、意図したものではない条件を報告します。

重大度レベルとコンパイラーの応答

文字	重大度	コンパイラーの応答
 C E	エラー	コンパイルは継続し、オブジェクト・コードが生成されます。コンパイラーが訂正することのできるエラー条件が存在しますが、プログラムは期待される結果を作成できない可能性があります。
S	重大エラー	コンパイルは継続しますが、オブジェクト・コードは生成されません。コンパイラーが訂正できないエラー条件が存在します。
U	回復不能エラー	コンパイラーは停止します。回復不能エラーが見つかりました。メッセージがリソースの限界 (例えばファイル・システムまたはページング・スペースがいっぱいになった) を示している場合は、リソースを追加して再コンパイルしてください。異なるコンパイラー・オプションが必要であると示された場合は、それらを使用して再コンパイルしてください。メッセージが内部コンパイラー・エラーを示す場合は、そのメッセージを IBM 技術員に報告してください。

コンパイラーのデフォルトの動作は、オプション **-qnoinfo** または **-qinfo=noall** でコンパイルすることです。 **-qinfo** のサブオプションは、特定のカテゴリの情報診断を指定する機能を提供します。例えば、**-qinfo=por** は、移植性の問題に関連するメッセージへの出力を制限します。

注: C では、サブオプションなしで指定されるオプション **-qinfo** は **-qinfo=all** に相当します。C++ では、サブオプションなしで指定される **-qinfo** は **-qinfo=all:noppt** に相当します。

戻りコード

コンパイルの終了時に、コンパイラーは、以下の任意の条件のもとで戻りコードをゼロに設定します。

- メッセージが発行されない。
- 診断されたすべてのエラーの中の最高重大度レベルが、**-qhalt** コンパイラー・オプションの設定値よりも小さく、かつエラーの数が **-qmaxerr** コンパイラー・オプションで設定した限界値に達していない。
- **-qhaltormsg** コンパイラー・オプションで指定されたメッセージが発行されない。

それ以外の場合、コンパイラーは「XL C/C++ コンパイラー・リファレンス」に記載されている戻りコードの 1 つを設定します。

コンパイラー・メッセージ・フォーマット

デフォルトでは、診断メッセージは次の形式になります。

"file", line line_number.column_number: 15cc-nnn (severity) message_text.

ここで、15 はコンパイラー製品 ID、cc は、メッセージを発行したコンパイラー・コンポーネントを示す 2 桁のコード、nnn はメッセージ番号、さらに、severity は重大度レベルの文字です。cc に指定できる値は、以下のとおりです。

- 00 メッセージを生成または最適化するコード
- 01 コンパイラー・サービス・メッセージ
- 05 C コンパイラーに固有のメッセージ
- 06 C コンパイラーに固有のメッセージ
- 40 C++ コンパイラーに固有のメッセージ
- 86 プロシージャーク間分析 (IPA) に固有のメッセージ

この形式は、**-qnosrcmsg** オプションを使用可能にしてコンパイルする場合と同じです。診断メッセージとともにソース行を表示する代替メッセージ・フォーマットにするには、**-qsrcmsg** オプションでのコンパイルを試行してください。このオプションを使用可能にすると、エラーがあるとコンパイラーが判断したソース行、その下 (2 行目) にそのソース行の特定の箇所を指し示す行 (可能な場合)、および診断メッセージを、標準エラーに出力するようにコンパイラーに指示します。

注: メッセージは、他のプログラムの入力として使用するものではありません。メッセージのフォーマットおよび内容は、プログラミング・インターフェースにするためのものではなく、リリース毎に変更する可能性があります。

gxlc および gxlc++ を使用した GNU C および C++ コンパイラー・オプションの再利用

各 gxlc および gxlc++ ユーティリティーは、GNU C または C++ コンパイラー・オプションを受け取り、それを同等の XL C/C++ オプションに変換します。両方のユーティリティーは、XL C/C++ オプションを使用して **xlc** または **xlc** 呼び出しコマンドを作成し、それを使用して XL C/C++ を呼び出します。これらのユーティリティーは、GNU C および C++ で以前に開発されたアプリケーション用に作成された Make ファイルを簡単に再利用するために提供されています。ただし、XL C/C++ の機能を十分に活用するためには、XL C/C++ 呼び出しコマンドおよびその関連オプションを使用することをお勧めします。

gxlc および gxlc++ のアクションは、構成ファイル `gxlc.cfg` によって制御されます。XL C または XL C++ に対応する GNU C および C++ オプションは、このファイルに示されています。すべての GNU オプションに対応する XL C/C++ オプションがあるわけではありません。gxlc および gxlc++ は、変換されなかった入力オプションについて警告を戻します。

gxlc および gxlc++ オプション・マッピングは変更可能です。gxlc および gxlc++ 構成ファイルへの追加または編集の詳細については、30 ページの『オプション・マッピングの構成』を参照してください。

例

Hello World プログラムの C バージョンをコンパイルするために `gcc -ansi` オプションを使用するには、以下を使用することができます。

```
gxlc -ansi hello.c
```

これは、以下のように変換されます。

```
xlc -F:c89 hello.c
```

変換後、このコマンドを使用して XL C コンパイラーを呼び出します。

gxlc および gxlcpp 戻りコード

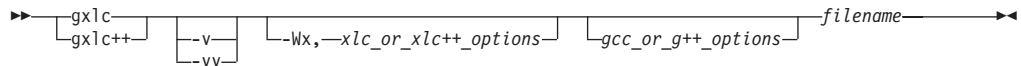
他の呼び出しコマンドのように、gxlc および gxlcpp はリスト、コンパイルに関する診断メッセージ、GNU オプションの変換の失敗に関する警告、および戻りコードなどの、出力を戻します。gxlc または gxlcpp が正常にコンパイラを呼び出すことができない場合、戻りコードを以下のいずれかの値に設定します。

40 gcc または g++ オプション・エラーまたは回復不能エラーが検出されました。

255 プロセスの実行中にエラーが検出されました。

gxlc および gxlcpp 構文

以下の図は gxlc および gxlcpp 構文を示しています。



以下、説明です:

filename

コンパイルするファイルの名前です。

-v XL C/C++ を呼び出すのに使用されるコマンドを確認することができます。gxlc または gxlcpp は、作成した XL C/C++ 呼び出しコマンドを、コンパイラの呼び出しに使用する前に表示します。

-vv シミュレーションを実行することができます。gxlc または gxlcpp は、作成した XL C/C++ 呼び出しコマンドを表示しますが、コンパイラを呼び出しません。

-Wx, xlc_or_xlcpp_options


指定した XL C/C++ オプションを xlc または xlc++ 呼び出しコマンドに直接送信します。gxlc または gxlcpp は、作成している XL C/C++ 呼び出しに、指定したオプションを変換せずに追加します。このオプションを既知の XL C/C++ オプションと共に使用し、ユーティリティのパフォーマンスを改善します。複数の *xlc_or_xlcpp_options* がコンマ区切り文字を使用します。

gcc_or_g++_options

xlc または xlc++ オプションに変換する gcc または g++ オプションです。ユーティリティは、変換できないオプションに対しては警告を発行します。現在 gxlc および gxlcpp によって認識される gcc および g++ オプションは、構成ファイル *gxlcpp.cfg* にリストされています。複数の *gcc_or_g++_options* は、スペース文字で区切られています。

GNU C および C++ から XL C/C++ へのオプション・マッピング













以下のテーブルは、gxlc および gxlcpp が受け取り、変換する GNU C および C++ オプションをリストしています。リストにない GNU オプションをこれらのユーティリティへの入力として指定した場合、そのオプションは無視されるか、またはエラーを生成します。GNU オプションにマイナス記号がある場合、マイナス記号も gxlc および gxlcpp によって認識され、変換されます。

GNU C および C++ オプション	XL C/C++ オプション
-###	-#
-ansi	-F:c89
-B	-B
-C	-C
-c	-c
-Dmacro[=defn]	-Dmacro[=defn]
-E	-E
-e	-e
-fdollars-in-identifiers	-qddollar
 -fdump-class-hierarchy	-qdump_class_hierarchy
 -fexceptions	-qeh
 -ffor-scope	-qlanglvl=ansifor
 -fno-for-scope	-qlanglvl=noansifor
-ffunction-sections	-qfuncsect
-finline	-qinline
-finline-functions	-qinline
 -finline-limit=n	-qinline=limit=n
 -fkeep-inline-functions	-qkeepinlines
 -fno-gnu-keywords	-qnokeyword=typeof
 -fno-operator-names	-qnokeyword=and -qnokeyword=bitand -qnokeyword=bitor -qnokeyword=compl -qnokeyword=not -qnokeyword=or -qnokeyword=xor
-fpascal-strings	-qmacpstr
-fPIC	-qpik=large
-fpic	-qpik=small
 -frtti	-qrtti
-fshort-enums	-qenum=small
-fsigned-bitfields	-qbitfields=signed
-fsigned-char	-qchars=signed
-fstrict-aliasing	-qalias=ansi
-fsyntax-only	-qsyntaxonly
-funroll-all-loops	-qunroll=yes
-funroll-loops	-qunroll=yes
-funsigned-bitfields	-qbitfields=unsigned
-funsigned-char	-qchars=unsigned
-fwritable-strings	-qnoro

オプション・マップ: GNU C および C++ から XL C/C++

GNU C および C++ オプション	XL C/C++ オプション
-g	-g
-g3	-g
-ggdb	-g
-gxcoff	-g
-I <i>dir</i>	-I <i>dir</i>
-L <i>dir</i>	-L <i>dir</i>
-l <i>library</i>	-l <i>library</i>
-M	-M
-MD	-M
-maix32	-q32
-maix64	-q64
-mcpu=403	-qarch=403
-mcpu=601	-qarch=601
-mcpu=602	-qarch=602
-mcpu=603	-qarch=603
-mcpu=604	-qarch=604
-mcpu=common	-qarch=com
-mcpu=power	-qarch=pwr
-mcpu=power2	-qarch=pwr2
-mcpu=powerpc	-qarch=ppc
-mcpu=powerpc64	-qarch=ppc64
-mcpu=rs64a	-qarch=rs64a
-mno-fused-madd	-qfloat=nomaf
-mfused-madd	-qfloat=maf
-mlong-double-64	-qnoqlonglong
-mlong-double-128	-qlonglong
-mpower	-qarch=pwr
-mpower2	-qarch=pwr2
-mpowerpc	-qarch=ppc
-mpowerpc-gfxopt	-qarch=pwrgr
-mpowerpc64	-qarch=ppc64
-mtune=403	-qtune=403
-mtune=601	-qtune=601
-mtune=602	-qtune=602
-mtune=603	-qtune=603
-mtune=604	-qtune=604
-mtune=common	-qtune=com
-mtune=power	-qtune=pwr
-mtune=power2	-qtune=pwr2
-mtune=powerpc	-qtune=ppc

オプション・マップ: GNU C および C++ から XL C/C++

GNU C および C++ オプション	XL C/C++ オプション
-mtune=powerpc64	-qtune=ppc64
-mtune=rs64a	-qtune=rs64a
-nodefaultlibs	-qnolib
-nostartfiles	-qnocrt
-nostdinc	-qnostdinc
-nostdlib	-qnolib -qnocrt
-O	-O
-O0	-qnoopt
-O1	-O
-O2	-O2
-O3	-O3
-Os	-O2 -qcompact
-o	-o
-p	-p
-pg	-pg
-r	-r
-S	-S
-s	-s
 -std=c89	-F:c89
 -std=iso9899:1990	-F:c89
 -std=iso9899:199409	-F:c89
 -std=c99	-F:c99
 -std=c9x	-F:c99
 -std=iso9899:1999	-F:c99
 -std=iso9899:199x	-F:c99
 -std=gnu89	-qlanglvl=extc89
 -std=gnu99	-qlanglvl=extc99
 -std=gnu9x	-qlanglvl=extc99
 -std=c++98	-qlanglvl=strict98
 -std=gnu++98	-qlanglvl=extended
-time	-qphsinfo
-trigraphs	-qtrigraph
-Umacro	-Umacro
-u	-u
-Wformat	-qformat

オプション・マップ: GNU C および C++ から XL C/C++

GNU C および C++ オプション	XL C/C++ オプション
-Wuninitialized	-qinfo=ini
-Wunreachable-code	-qinfo=eff
-Wa, <i>option</i>	-Wa, <i>option</i>
-Wl, <i>option</i>	-Wl, <i>option</i>
-Wp, <i>option</i>	-Wp, <i>option</i>
-w	-w
-x assembler	-qsourcecetype=assembler
-x c	-qsourcecetype=c
-x c++	-qsourcecetype=c++
-x none	-qsourcecetype=default
-Z	-Z

その他の GNU オプションはすべて無視され、通知メッセージを発行します。

オプション・マッピングの構成

gxlc および gxlc++ ユーティリティーは、構成ファイル gxlc.cfg を使用して、GNU C および GNU C++ オプションを XL C/C++ オプションに変換します。gxlc.cfg の各項目は、ユーティリティーが GNU C または GNU C++ オプションを XL C/C++ オプションにマップする方法およびその処理方法を記述しています。

項目は、処理命令用のフラグのストリング、GNU C オプションのストリング、および XL C/C++ オプションのストリングから構成されています。3 つのフィールドは、空白で分離する必要があります。項目に最初の 2 つのフィールドのみが含まれ、XL C/C++ オプションのストリングが省略されている場合、2 番目のフィールドの GNU C オプションは、gxlc で認識はされますが無視されます。

文字を使用して、構成ファイルにコメントを挿入することができます。コメントは、独立した行、または項目の最後に記述することができます。

gxlc.cfg の項目には、以下の構文を使用します。

```
abcd    "gcc_or_g++_option"    "xlc_or_xlc++_option"
```

以下、説明です:

- a プレフィックスとして no- を追加し、オプションを無効にします。値は y (yes)、または n (no) のいずれかです。例えば、フラグが y に設定されると、finline は fno-inline として使用不可となり、項目は以下のようになります。

```
ynn*          "-finline"          "-qinline"
```

-fno-inline の場合、gxlc はそれを -qnoinline に変換します。

- b XL C/C++ オプションに関連した値があることをユーティリティーに知らせます。値は y (yes)、または n (no) のいずれかです。例えば、-finline-limit=n は -qinline=limit=n にマップします。この場合、フラグは y に設定され、項目は以下のようになります。

nyn* "-finline-limit" "-qinline=limit"

このように指定すると、gxc および gxc++ はこれらのオプションに値があることを予期します。

c オプションの処理を制御します。値は以下のようになります。

- n。gcc-option フィールドにリストされているオプションを処理するようにユーティリティに指示します。
- i。gcc-option フィールドにリストされているオプションを無視するようにユーティリティに指示します。gxc および gxc++ は、オプションが無視されたというメッセージを生成し、指定されたオプションの処理を継続します。
- e。gcc-option フィールドにリストされているオプションが検出されると、処理を停止するようにユーティリティに指示します。gxc および gxc++ はエラー・メッセージも生成します。

例えば、gcc オプション `-I-` はサポートされていないため、gxc および gxc++ で無視する必要があります。この場合、フラグを `i` に設定します。項目は以下のようになります。

nni* "-I-"

gxc および gxc++ がこのオプションを入力として検出すると、それを処理せず、警告を生成します。

d コンパイラーのタイプに基づいて、gxc および gxc++ にオプションを含めるか、または無視させます。値は以下のようになります。

- c。C のオプションのみを変換するように gxc および gxc++ に指示します。
- x。C++ のオプションのみを変換するように gxc および gxc++ に指示します。
- *。C および C++ のオプションを変換するように gxc および gxc++ に指示します。

例えば、`-fwritable-strings` は、両方のコンパイラーによってサポートされ、`-qnor` にマップします。項目は以下のとおりです。

nnn* "-fwritable-strings" "-qnor"

"gcc_or_g++_option"

GNU C バージョン 3.3 によってサポートされる gcc または g++ オプションを示す文字列です。このフィールドは必須で、二重引用符で囲む必要があります。

"xlc_or_xlc++_option"

XL C/C++ オプションを示す文字列です。このフィールドはオプションで、指定する場合は二重引用符で囲む必要があります。空のままにした場合、gxc および gxc++ はその項目の *gcc_or_g++_option* を無視します。

また、オプションの範囲をマップする項目を作成することが可能です。これは、アスタリスク (*) をワイルドカードとして使用することによって行うことができます。

す。例えば、gcc -D オプションには、ユーザー定義の名前が必須で、オプションの値を取ることができます。これは、以下の一連のオプションを持つことができます。

```
-DCOUNT1=100
-DCOUNT2=200
-DCOUNT3=300
-DCOUNT4=400
```

このオプションのバージョン毎に項目を作成する代わりに、以下のように単一の項目を作成します。

```
nnn*          "-D*"          "-D*"
```

ここで、アスタリスクは、-D オプションの後に続く任意のストリングに置き換えられます。

逆に、アスタリスクを使用して、指定したオプションの範囲を除外することができます。例えば、gxc または gxc++ ですべての -std オプションを無視する場合、項目は以下のようになります。

```
nni*          "-std*"
```

アスタリスクをオプション定義で使用する場合、オプション・フラグ *a* および *b* はこれらの項目に適用できません。

GNU C または GNU C++ オプションで文字 % を使用すると、そのオプションに関連するパラメーターがあることを示します。これは、gxc または gxc++ でオプションを無視し、関連するパラメーターも間違いなく無視するために使用されます。例えば、-include オプションはサポートされていません。このオプションにはパラメーターがあります。アプリケーションでは、オプションとパラメーターの両方を無視する必要があります。この場合、項目は以下のようになります。

```
nni*          "-include %"
```

関連資料

- GNU Compiler Collection のオンライン資料は <http://gcc.gnu.org/onlinedocs/> にあります。

オプションの要約: C コンパイラー

この章の付録では、タイプ毎にグループ化して、C コンパイラー・オプションの要約を示します。上位のグループには、オプションのサブグループが含まれています。ソース・コードの基本変換用のサブグループに加え、あるサブグループは、特殊なデバッグ情報の追加などの特別なコード処理または制御用のオプションを構成します。別のサブグループは、リンカーおよびライブラリー検索パスの制御に関係しています。パフォーマンスおよび最適化に関連するオプションについては、37 ページの『最適化について』の最後に要約されています。オプションの説明、完全なオプション構文、および各オプションの使用方法については、「*XL C/C++ コンパイラー・リファレンス*」を参照してください。

基本変換

このグループのオプションは、ソース・コードの基本変換に広く適用することができます。コンパイラ・オプションのサブグループは、一般に以下に関係しています。

- 規格への準拠。
- コンパイル・モードまたはコンパイラ・ドライバーの制御。
- コード生成用ソース・コードの操作。
- 特殊な診断の生成。
- コンパイルされたコードの操作。

ソース・コードの基本変換に関連するオプション

規格への準拠	コンパイル・モードまたはコンパイラ・ドライバの制御
-qgenproto, -qnogenproto -qlanglvl -qlibansi, -qnoibansi	-# -q32 -q64 -F -qpath -qproto, -qnoproto -qsourcetype
ソース・コードの生成	
-qalloca -qasm, -qnoasm -qasm_as -qattr, -qnoattr -B -C -qcpluscmt, -qnocpluscmt -D -qdbcs, -qnodbcs -qdigraph, -qnodigraph -qdirectstorage, -qnodirectstorage -E -qfuncsect, -qnofuncsect -qignprag -M -ma	-qmacpstr, -qnomacpstr -qmakedep -qmbcs, -qnombcs -P -qpascal, -qnopascal -qroptrs, -qnoroptrs -qsmallstack, -qnosmallstack -qsyntaxonly -t -qtabsize -qtrigraph, -qnotrigraph -U -qutf, -qnoutf -W -qweaksymbol, -qnweaksymbol
診断	コンパイルされたコード
-qflag -qinfo, -qnoinfo -qmaxerr, -qnmaxerr -qphsinfo, -qnphsinfo -qprint, -qnoprint -qshowinc, -qnoshowinc -qsource, -qnosource -qsrcmsg, -qnosrcmsg -qsuppress, -qnosuppress -V -v -w -qwarn64, -qnwarn64 -qxcall, -qnoxcall	-qbitfields -c -qchars -qdataimported -qdatalocal -qdollar, -qnodollar -qexpfile -o -qprocimported -qproclcal -qprocunknown -S -qstatsym, -qnstatsym -qtbtable -qupconv, -qnoupconv

特別な処理および制御

このグループのオプションは、変換処理のきめ細かい制御を提供し、基本変換オプションより一般に適用されません。コンパイラ・オプションのこのグループ内のトピックは、一般に以下に関係しています。

- データ位置合わせ。

- コンパイル・モードまたはコンパイラー・ドライバーの制御。
- コード生成用ソース・コードの操作。
- 特殊な診断の生成。
- コンパイルされたコードの操作。

特別な処理、微調整、およびデバッグのためのオプション

データ位置合わせ	並列化
-qalign -qenum	-qsmp, -qnosmp -qthreaded, -qnothreaded
浮動小数点および数値機能	
サイズ -qldbl128, -qno1dbl128 -qlongdouble, -qno1ongdouble -qlonglit, -qno1onglit -qlonglong, -qno1onglong	浮動小数点値の丸め -qrndflt, -qnorndflt -y
単精度値 -qhsflt, -qnohsflt -qhssngl, -qnohssngl	その他の浮動小数点オプション -qfloat -qflttrap, -qnoflttrap -qmaf, -qnomaf
デバッグ	
-qcheck, -qnocheck -qdpcl, -qnodpcl -qdbxextra, -qnodbxextra -qextchk, -qnoextchk -qfullpath, -qnofullpath -g -qhalt -qheapdebug, -qnoheapdebug -qinitauto, -qnoinitauto	-qkeeparm, -qnokeeparm -qlinedebug, -qno1inedebug -qlist, -qno1ist -qlistopt, -qno1istopt -qsaveopt, -qnosaveopt -qsyntab -qxref, -qnoxref

リンクおよびライブラリー関連オプション

このグループのオプションは、コンパイル処理のリンク・フェーズに関連しています。またこのグループには、ライブラリーおよびヘッダー・ファイルを見つけるための、検索パスを指定する特殊な方法を提供するオプションを含んでいます。これらのコンパイラー・オプションは、一般に以下に関係しています。

- スtring・リテラルおよび定数の配置。
- 静的および動的リンクおよびライブラリー。
- 検索ディレクトリーの指定。

ld コマンドを制御するためのオプション

ストリング・リテラルおよび定数の配置	静的および動的リンクおよびライブラリー
-qkeyword, -qnokeyword -qro, -qnoro -qroconst, -qnoroconst	-b -brtl -e -G -qmkshrobj -qstdinc, -qnostdinc
検索ディレクトリー	その他のリンカー・オプション
-I -L -l (小文字の L) -qidirfirst, -qnoidirfirst -r -Z	-f -qinlglue, -qnoinglue

オプションの要約: C++ コンパイラー

C コンパイラー・オプションの多くは、C++ プログラムのコンパイルでも使用可能です。以下のテーブルは、C++ プログラムのコンパイル特有の追加のコンパイラー・オプション、および AIX プラットフォームで C++ プログラムをコンパイルする際に使用できない C オプションを示しています。

C++ プログラム用コンパイラー・オプション

C++ 固有のオプション	C のみのオプション
-+ -qcinc -qeh, -qnoeh -qhaltonmsg -qkeepinlines, -qnokeepinlines -qnamemangling -qobjmodel -goldpassbyvalue, -qnooldpassbyvalue -qpriority -qrtti, -qnortti -qstaticinline, -qnostaticinline -qtempinc, -qnotempinc -qtemplaterecompile, -qnottemplaterecompile -qtemplateregistry, -qnottemplateregistry -qtempmax -qtmplparse -qtwolink, -qnotwolink -qunique, -qnounique -qvftable, -qnovftable	-qalloca -qassert, -qnoassert -qc_stdinc -qcpluscmt, -qnocpluscmt -qdbxextra, -qnodbxextra -qgenproto, -qnoegenproto -ma -macpstr, -nomacpstr -qproto, -qnoproto -qsrcmsg, -qnosrcmsg -qsyntaxonly -qupconv, -qnoupconv

最適化について

単純なコンパイルとは、ソース・コードを実行可能ファイルまたは共有オブジェクトへ変換することです。最適化変換とは、アプリケーションの実行時のパフォーマンス全体を向上させる変換です。XL C/C++ は、PowerPC アーキテクチャーに合わせて調整された変換を最適化するポートフォリオを提供します。このような変換では、以下を行うことができます。

- ・ クリティカルな操作に対して実行する命令の数を減らす。
- ・ 生成されたオブジェクト・コードを再構成して、PowerPC アーキテクチャーの使用を最適化する。
- ・ メモリー・サブシステムの使用を改善する。
- ・ アーキテクチャーの機能を活用して、大量の共用メモリー並列化を処理する。

これらの目的は、アプリケーションの実行速度を速くすることです。

コードの変換に影響する使用可能な制御を理解し、優れたコードを記述すれば、比較的少ない開発努力で、パフォーマンスを大きく改善することができます。

OpenMP などのプログラミング・モデルを使用することにより、パフォーマンスの高いコードを記述できます。本節では、ランタイム・パフォーマンス、手動でコーディングしたマクロ最適化、一般的な読み易さ、およびソース・コードの全体的な移植性という点に関して、相反するこれらのバランスを取るために、コンパイラーが実行できる最適化の一部について説明します。

最適化は、アプリケーション開発サイクルの後半のフェーズ (例えば製品リリース・ビルドなど) でよく試行されます。可能であれば、コードを最適化する前に、まず最適化しない状態でコードをテストおよびデバッグしてください。最適化とは、プログラムに最も効率的なアルゴリズムを選択し、それらを正しくインプリメントすることです。言語標準へ準拠しているかどうかは、コードを正常に最適化できる度合いに直接大きく関係します。最適化プログラムは、最終的な規格合致試験です。

最適化は、コンパイラー・オプション、ディレクティブ、およびプラグマによって制御されます。ただし、コンパイラー・フレンドリー・プログラミングの技法が、オプションやディレクティブと同様に、パフォーマンスに対して有益です。コードを手動で最適化する (例えば手動でループをアンロールする) ことは不要であり、過度に行うことはお勧めできません。異常な構成によって、コンパイラー (およびその他のプログラマー) に混乱を招き、アプリケーションを新しいマシン用に最適化する作業が困難になる場合があります。

最適化は必ずしもすべてのアプリケーションに有益であるとは限りません。デバッグ機能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は、相反する関係にあると言えます。

関連資料

- ・ 「XL C/C++ プログラミング・ガイド」の『最適化レベルの使用』
- ・ 「XL C/C++ プログラミング・ガイド」の『アプリケーションの最適化』

最適化のための選択可能コンパイラー・オプション

次のテーブルは、プログラム・パフォーマンスを最適化するための基本コンパイラー・オプションの選択を示しています。完全なリストについては、「*XL C/C++ プログラミング・ガイド*」を参照してください。使用可能なサブオプションの資料については、「*XL C/C++ コンパイラー・リファレンス*」を参照するか、またはオプションのマニュアル・ページを参照してください。

表 1. 最適化のための基本コンパイラー・オプション

オプション	説明
-qnoot	コンパイラーは極めて限られた最適化を実行します。これがデフォルトです。アプリケーションを最適化し始める前に、 -qnoot と正しくコンパイルしているか確認してください。
-O2	コンパイラーは包括的な低レベルの最適化を実行しますが、これにはグラフ・カラーリング、共通副次式の除去、不要コードの除去、代数の単純化、定数伝搬、ターゲット・マシンの命令スケジューリング、ループのアンロール、およびソフトウェアのパイプラインを含みます。
-qarch -qtune -qcache	コンパイラーは、特定のハードウェアおよびアプリケーションが稼働する命令セットの特性を利用します。 -qarch を使用して、アプリケーション・コードを生成する対象のプロセッサ・アーキテクチャーのファミリーを指定します。 -qtune を使用して、最適化の対象を特定のマイクロプロセッサ上での実行に偏らせます。 -qcache を使用して、特定のキャッシュまたはメモリー形状を定義します。
-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、異なるコード・セクションが通常実行され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。 PDF プロセスは、非構造の分岐を含むアプリケーションに最も役立ちます。
-O3	コンパイラーは、 -O2 で、より積極的な最適化を実行します。ループのアンロールが深いほど、ループのスケジューリングが良く、暗黙のメモリー使用に関する限度を除去します。
-qhot	コンパイラーが高位の変換を実行し、さらなるループ最適化を提供し、選択的に配列埋め込みを実行します。このオプションは、大容量の数値処理を実行する科学計算アプリケーションに最も役立ちます。
-qipa	コンパイラーはプロシージャ間分析を実行し、アプリケーション全体を単位として最適化します (プログラム全体の分析)。このオプションは、頻繁に使用される大量のルーチンを含むビジネス・アプリケーションに最も役立ちます。また、高レベルの抽出をする C++ プログラムにも役立ちます。多くの場合、このオプションはコンパイル時間を大幅に増加させます。
-O4	これは、 -O3 -qipa -qhot -qarch=auto -qtune=auto -qcache=auto と同等です。コンパイルに時間がかかりすぎる場合、 -O4 -qnoipa でコンパイルを試行してください。
-O5	これは、 -O4 -qipa=level=2 と同等です。AIX プラットフォームでは、プロセッサが PowerPC 970 で、Altivec データ型がオペレーティング・システムによってサポートされている場合、このオプションは -qhot=vector -qhot=simd もオンにします。

プラグマの最適化入門

プラグマ・ディレクティブは、インプリメンテーション固有のプリプロセッサ・ディレクティブであり、プログラムのソース・コード内の行のグループをセクションに分けて、そのセクションに関するなんらかの情報についてコンパイラーに通知する機能を提供します。プラグマ・ディレクティブを使用すると、類似した名前の付いた、対応するコンパイラー・オプションよりも、より精密な制御が提供されることがよくあります。最適化に関連したプラグマ・ディレクティブの場合、自身での検出が不可能である最適化の潜在的な機会か、またはソース・コードの最適化に関する決定を促進する可能性のある、自身が作り出す想定について、コンパイラーに明示的に通知することを意図しています。

XL C/C++ は、XL Fortran ディレクティブ間で対応するものを持つ、パフォーマンス最適化をプラグマ・ディレクティブに提供します。最適化プラグマは、Fortran ディレクティブと同様、記述的なカテゴリーでは、命令、表明、および規定と考えることができます。これらのカテゴリーの区別は、コンパイラーがそのプラグマによって規定された指示または情報にどの程度準拠する必要があるかという、度合いを表します。

すべての OpenMP ディレクティブは、命令プラグマです。コンパイラーは、その OpenMP 標準のインプリメンテーションに厳密に適合して振る舞う必要があります。ただし、その結果得られるプログラムの振る舞いが、正しい結果が作成されることを保証するものではありません。XL C/C++ はすべての OpenMP V2.0 プラグマ・ディレクティブをインプリメントします。

表明プラグマは、プラグマが有効なソース・コードの行を変換する際に安全に行える前提事項についてコンパイラーに通知します。表明プラグマの意図は通知であり、通知は、コンパイラーが特定の仕方で振る舞うことに関する特定の義務を課すものではありません。XL C/C++ には、最適化に関連した以下の表明プラグマが提供されています。

選択された表明 #pragma ディレクティブ

名前	説明
isolated_call(function_list)	指名された関数に対する呼び出しに副次作用がないことを表明します。
disjoint(variable_list)	名前付き変数のいずれにも重複したストレージ域がないことを表明します。
ibm independent_loop	以下のループに、ループ発生の依存関係がないことを表明します。この融通性によって、局所性および並列変換が可能になります。
ibm independent_calls	以下のループ内の呼び出しによって、ループ発生の依存関係が生ずる原因にならないことを表明します。
ibm permutation	指定された配列の要素が、以下のループのそれぞれの反復において明確な値を取ることを表明します。この表明は、まばらなコードで有用な可能性があります。

選択された表明 #pragma ディレクティブ

名前	説明
ibm iterations(iteration_count)	以下のループの反復数を指定します。これは、コンパイラーがループを並列化することが有利かどうかの判別に役立ちます。プラグマは、コンパイル単位全体ではなく、単一ループに適用できます。
execution_frequency(very_low)	プラグマが含まれる制御パスがあまり頻繁に実行されないことを表明します。
leaves(function_list)	指定された関数に対する呼び出しが戻らないことを表明します。

規定プラグマは、パフォーマンスの向上を目的とした特定の方法での効果のもとでソース・コードを変換するようにコンパイラーに指示する提案です。規定プラグマは、**inline** キーワードのように機能します。コンパイラーが提案をインプリメントする必要はありませんが、提案を実行すると有利であると考えられる場合には、インプリメントする可能性があります。XL C/C++ には、最適化に関連した以下の規定プラグマが提供されています。

選択された規定 #pragma ディレクティブ

名前	説明
ibm sequential_loop	-qsmp=auto が指定されている場合でも、単一スレッドで次のループを実行するように、コンパイラーに指示します。
ibm snapshot(variable_name)	指定された変数を調べるため、プラグマのポイントでデバッグ・ブレイクポイントを設定します。
stream_unroll	for ループに含まれるストリームを複数のストリームに分割します。大規模な反復数および小規模なストリーム数を持つループを対象としています。
unroll	ディレクティブの直後にある for ループがアンロール可能であることを、コンパイラーに示します。このプラグマは、最内部と外部の for ループの両方に適用することができます。
unrollandfuse	コンパイラーに命令して、外部ループの for 本体部分、つまりループ・ネスト自体を複製し、複製したものを単一のループ・ネストに展開します。

関連資料

- すべての XL C/C++ プラグマは、「XL C/C++ コンパイラー・リファレンス」に説明されています。

移植の考慮事項

本節では、UNIX ベースのアプリケーションを AIX プラットフォームへ簡単に移植するために調査する一般的な項目を記載します。

アプリケーションを他のプラットフォームで実行するように移植する作業では、ソース・プラットフォームとターゲット・プラットフォームが必要です。コーディングを始める前に、最も基本的な段階として、次の質問を考えてください。ターゲット・プラットフォームへ移植することで、何が変更されるのか？ 純粋な「移植」とは、ハードウェアとオペレーティング・システムのみを変更することを指します。

理論的には、プログラムが良い形式で記述されており、プラットフォームに固有な点に依存せず、業界標準 (POSIX など) に従い、標準の言語定義に準拠して標準外の言語拡張を使用しなければ、新たなオペレーティング・システムに容易に移植することができ、再コンパイルとデバッグ以外には最小限の追加作業しか必要としません。ソース・プラットフォームが比較的最近の UNIX ベースのオペレーティング・システムである場合は、変更する点は、業界標準により完全に準拠させることと、同じ業界標準の新しいバージョンに準拠させることのみになることもあります。アプリケーションがすでに Linux システムで実行している場合は、再コンパイルして AIX でネイティブに実行するという選択肢があります。多くのアプリケーションは、変更せずとも再コンパイルして実行することができます。

しかも、異なる標準準拠コンパイラーでアプリケーションをコンパイルすることで、コンパイラー間で言語標準の実装が違うために、ソース・コードのわずかなぜい弱性を取り除くことができる場合があります。その結果、アプリケーションがより堅固になります。

移植を行う際に起こる問題は、内部移植性問題と外部移植性問題に分類することができます。内部移植性問題では、プログラム言語に組み込まれる暗黙の前提事項を処理します。例えば、C プログラムは、整数内での特定のバイト・オーダー、整数の相対サイズのセット、および構造体内の特定のフィールド・レイアウトを前提とします。内部移植性は、ハードウェアに対するプログラム・コードの関係と関連します。この移植性問題のクラスはプログラマーの制御下にあります。

一方、外部移植性問題は、プログラムが使用する外部インターフェース、プログラムが前提とするこれらのインターフェースのセマンティクス、およびプログラムに渡される引き数とプログラムから渡される戻り値の選択と関連しています。これらは、プログラムが依存するライブラリーおよびシステム呼び出し、プログラムが呼び出す (ただし、外部の) コードを取り扱っています。プログラムが標準化された外部インターフェースを使用する場合、外部移植性はプログラマーの制御下に置くことができます。

言語に組み込まれた移植性の問題

このセクションでは、AIX プラットフォームへの移植に関連する内部移植性の考慮事項の一部を示します。

- GNU C およびその他の言語拡張機能への依存の程度を調べる。ISO 言語仕様に厳密に適合するアプリケーションは、最大限の移植が可能です。IBM XL C/C++ は、C および C++ への GNU C および C++ 拡張機能のサブセットをサポートします。サポートされない拡張機能に依存するコードを再調査してください。
- ヌル・ポインターがどのように間接参照しているかを確認する。コード内の一部のエラーは、ハードウェア依存の特性によりプラットフォームで検出されません。これらの種類のエラーは、プログラムが他のプラットフォームに移植されると表示されることがあります。ソース・プラットフォームが AIX の場合は、オプション **-qcheck=nullptr** を使用すると、移植前にこのような状態を検出するのに役立ちます。

AIX では、最低の 4K メモリー（つまり、アドレス 0 ～ 4K-1）が読み取り可能で、ゼロが含まれますが、Linux および Mac OS X プラットフォームでは読み取り不可なので、このようなプラットフォームでアクセスするとセグメンテーション違反になります。例えば、次のような場合です。

```
if (strcmp(a, NULL) == 0) ...
```

この結果、Linux および Mac OS X ではセグメンテーション違反になりますが、AIX では違反になりません。

- 位置合わせを調べる。AIX でサポートされる位置合わせのタイプは、同じ名前になる場合がありますが、Linux または Mac OS X プラットフォームでサポートされるものと同じではありません。**-qalign** または **#pragma align** の特定の値に依存するプログラムを移植する場合、プログラムを変更しなければならないことがあります。
- データ構造の移植性を確実にする。あるプラットフォームでアプリケーションを使用してデータを生成し、他のプラットフォームでアプリケーションを使用してデータを読み取ると、データは読み取るアプリケーションが予想するのとは異なる位置合わせを持つことがあります。この問題を回避するために、構造内のデータのレイアウトには必ずプラットフォームに依存しないメカニズムを使用してください。例えば、**#pragma pack(1)** および **#pragma pack(pop)** ペアで構造を囲む場合、位置合わせはすべてのプラットフォームで同じになります。
- Make ファイル内のコマンドを変換するために **gxlc** または **gxlc++** ユーティリティを使用する。すべての gcc または g++ オプションに対応する XL C/C++ オプションが存在するわけではありません。
- 32 ビットまたは 64 ビット・アプリケーション、あるいは 128 ビット・サポートに異なるコンパイラ呼び出しモードを使用する。
- Linux または Mac OS X では、デフォルトのグローバル演算子 **new** が呼び出され、割り振り要求を実行することができない場合、タイプ **std::bad_alloc** の例外がスローされます。AIX では、グローバル演算子 **new** のデフォルトの振る舞いで、割り振りが失敗した場合にヌル・ポインターが戻されます。
- テンプレートを使用するアプリケーションの移植性を確実にする。C++ コンパイラは、ソース・コードのテンプレートを手動で保守する代替手段として、テンプレート・ファイルで機能する 2 つの異なるメソッドを提供します。各メソッドには関連するコンパイラ・オプションがあります。**-qtemplateregistry** コンパイラ・オプションはすべてのテンプレートのレコードを保守します。このメソッドをお勧めします。また、古いコンパイラ・オプション **-qtempinc** は、他

のプラットフォームから移植するアプリケーションのために提供されます。ただし、Mac OS X プラットフォームでは、コンパイラー・オプション `-qtempinc` は使用すべきでないと考えられます。

関連資料

以下の IBM Redbooks には、移植に関連する情報が含まれています。他の Redbook は、www.redbooks.ibm.com にてオンラインで参照することができます。

- 「AIX 5L ポーティング・ガイド」(2001 年)
- 「*Developing and Porting C and C++ Applications on AIX*」(2003 年)

コンパイル時エラーの診断

移植プロジェクトでは、オプション `-qinfo=por` でアプリケーションをコンパイルすることを基本的にお勧めしています。この `-qinfo` サブオプションは、特に移植性に関する診断メッセージを追加します。`-qinfo=warn64` オプションは、64 ビット・モードへのアプリケーションの移植に固有の診断メッセージを発行するようコンパイラーに命令します。これらのメッセージは、調査の範囲を狭め、特定のコーディング構成を正確に示すのに役立ちます。

以下のテーブルは、コンパイル時エラーを検出し、修正する役に立つその他のオプションを示しています。

コンパイル時エラーに関するその他の診断オプション

オプション	説明
<code>-qsrcmsg</code>	コンパイラーがエラーを含むと判断したソース行、その下にソース行の特定の箇所を指し示す行、そして診断メッセージを標準エラーに出力する。
<code>-qsource</code>	コンパイラー・リストを戻すことを要求する。リストには、行番号付きのソース・コード、指定されたオプション、コンパイルに使用したすべてのファイルのリスト、診断メッセージの重大度レベル別の要約、読み込まれた行数、およびコンパイルが成功したかどうかなどのセクションが含まれます。 <code>-qattr</code> および <code>-qxref</code> オプションを指定することでそれぞれリストの属性セクションと相互参照セクションを作成することができます。オブジェクト・セクションを作成するには、オプション <code>-qlist</code> を指定する必要があります。このセクションには、コンパイラーの生成する疑似アセンブリー・コードが表示され、診断実行時に問題が発生しており、コード生成エラーによってプログラムが期待通りのパフォーマンスを示していないことが疑われる場合に使用します。
<code>-qsuppress</code>	コンパイラーが特定のメッセージを出力しないようにする。メッセージ番号をコロン区切りでリストすることで、複数のメッセージを抑止できます。
<code>-qflag</code>	指定した診断メッセージが端末およびリスト・ファイルに出力されないようにする。このオプションで、どのレベル以下のメッセージを無視するかを指定するには、デフォルトのコンパイラー・メッセージ形式で 사용되는単一文字による重大度コードを使用します。

32 ビットおよび 64 ビット・アプリケーションの開発

XL C/C++ を使用して 32 ビットと 64 ビット・アプリケーションの両方を開発できます。このセクションでは、参照情報と、C および C++ プログラムを 32 ビットから 64 ビット・モードに移行する上での、移植性に関するその他の考慮事項を説明します。

移植を行う際、変更点がハードウェアとオペレーティング・システムのみであるなら、それは純粋な「移植」と言えます。ただし、AIX に移行する過程で、32 ビット・アプリケーションを 64 ビット・プログラミング・モデルに変更する場合は、この行いはもはや純粋な「移植」ではなく、ある種の開発活動であることを意味します。以下の特徴のいずれかを有する 32 ビット・アプリケーションは、64 ビット環境に移植する際に変更を必要とする可能性が非常に高いです。

- カーネルのメモリーを直接読み取り、解釈している。
- /proc ファイル・システムを使用して 64 ビット・プロセスにアクセスしている。
- 64 ビット・バージョンにしかないライブラリーを使用している。
- デバイス・ドライバである。
- AIX との間にインターオペラビリティ上の問題があるソース・プラットフォームからの移植である。

64 ビット・モードで開発すると、アプリケーションはより新しくて速い 64 ビット・ハードウェアとオペレーティング・システムを利用することが可能で、データベース・アプリケーションや科学計算アプリケーションなどの、大規模で複雑なメモリーを大量に使用するプログラムのパフォーマンスを向上することができます。データベース・アプリケーション、Web 検索エンジン、科学計算アプリケーションなどのアプリケーションで、32 ビットのアドレス・スペースによりパフォーマンスに制限が課せられているものは、多くの場合、64 ビット・モードに移行することで利便性が向上します。入出力によりパフォーマンスに制限が課せられているアプリケーションも、64 ビット・モードにすることで、データをディスクに書き込まずにメモリーに保持することができるため、より高いパフォーマンスを実現できます。これは、通常メモリー・アクセスの方が、ディスク I/O よりも速いためです。

64 ビット・マシンを使用することで、パフォーマンスに最も大きく貢献する点は、おそらく大規模な問題を物理メモリー内で直接処理できるようになる点でしょう。しかし、いくつかのアプリケーションは、64 ビット・モードで再コンパイルしたときよりも、32 ビット・モードでコンパイルした方が良いパフォーマンスを示します。この理由としては、以下のものが存在します。

- 64 ビット・プログラムの方が大きい。プログラム・サイズの増加により、物理メモリーへの負荷がより大きくなります。
- 64 ビットの long 型除法の方が、32 ビットの整数除法よりも時間がかかる。
- 64 ビット・プログラムで、配列指標に 32 ビットの符号付き整数を使用する場合は、配列を参照するたびに、符号拡張を行うための追加の命令が必要となる場合がある。

64 ビット・アプリケーションの他の欠点としては、より大きいレジスターを保持するためにより多くのスタック・スペースを必要とする点があります。ポインター・サイズが大きくなるため、アプリケーションのキャッシュ・フットプリントが大きくなります。64 ビット・アプリケーションは、32 ビット・プラットフォーム上では実行できません。

64 ビット・プログラムのパフォーマンス上のマイナス影響を補正するには、次の方法があります。

- 32 ビットと 64 ビット混合の演算処理を避ける。例えば、32 ビットの符号付きデータ型と 64 ビットのデータ型を加算すると、32 ビット型の方を符号拡張し、符号によって、レジスターの上位 32 ビットを設定する必要があります。レジスターの上位ビットを設定すると、計算が遅くなります。32 ビット型が符号なしの場合、レジスターの上位ビットはクリアされます。
- 64 ビットの long 型の除法を可能な限り避ける。乗算は、多くの場合、除法よりも高速です。同じ除数を使って多くの除法を実行する必要がある場合は、除数の逆数を一時変数に割り当て、すべての除法を一時変数に対する乗算に変更します。例えば、次の関数では、

```
double preTax(double total) { return total * (1.0 / 1.0825); }
```

とした方が、以下の直接除法を行った場合よりも、実行が速くなります。

```
double preTax(double total) { return total / 1.0825; }
```

その理由は、除法 (1.0 / 1.0825) は、定数フォールディングにより、コンパイル時に一度のみ評価されるからです。この最適化は通常、**-qnostrict** オプションに効力がある場合 (**-O2** 以上の最適化レベルでコンパイルすると起こります)、コンパイラーによって行われます。

- 配列指標には signed、unsigned、およびプレーンの **int** 型ではなく、**long** 変数を使用する。これにより、コンパイラーは配列を参照するときに符号拡張を行わずに済みます。

良い形式で書かれたコードは、64 ビット・プログラミング・モデルに移行すれば最小限の修正作業とデバッグで正常にコンパイルおよび実行できる可能性が高いです。「良い形式」という表現は、言語標準に準拠し、良いプログラミング慣習に従っていることを意味します。64 ビット・プログラミング・モデルへ移行することに当てはめると、「良い形式」とは、コードが特定のバイト・オーダーや外部データ・フォーマットに依存していないこと、および関数プロトタイプ、適切なシステム・ヘッダー・ファイル、およびシステム派生のデータ型を使用していることも含まれます。

AIX の 32 ビットおよび 64 ビット開発環境

AIX オペレーティング・システムの C および C++ コンパイラーでは、ILP32 と LP64 という 2 つの異なるプログラミング・モデルが提供されています。ILP32 は「integer/long/pointer 32」の頭字語で、AIX のネイティブな 32 ビット・プログラミング環境です。ILP32 データ・モデルは、32 ビット・アドレス・スペースを提供し、理論上のメモリーの限界は 4 GB です。LP64 は「long/pointer 64」の頭字語で、AIX の 64 ビット・プログラミング環境であり、主要なシステム・ベンダーの 64 ビットの UNIX ベース・システムではデファクト・スタンダードとなっているデータ・モデルです。概して、LP64 は、データ型のサイズと位置合わせを除き、ILP32 モデルと同じプログラミング・フィーチャーをサポートしており、最も広く使用されている **int** データ型と後方互換性があります。

コンパイラーのサポート

デフォルトでは、コンパイラーを呼び出すと、コンパイラーとリンカーを 32 ビット・モードで呼び出します。64 ビットで開発するために、次のフィーチャーが提供されています。

- **__64BIT__** プリプロセッサー・マクロ。これは、64 ビット・モードでコンパイルする際に事前定義します。このマクロを使用すると、プログラマーは同じソース・ファイルの中で、32 ビット実行を行うコード行と 64 ビット実行を行うコード行を別々に選択することができます。
- **OBJECT_MODE** 環境変数。この環境変数を指定すると、デフォルトのコンパイラ・モードを変更し、両方のビット・モードのオブジェクトを受け入れるようにします。C および C++ アプリケーション開発で一般的に使用されているいくつかの AIX ユーティリティでは、この環境変数を使用しています。ただし、AIX で提供されている C および C++ ライブラリーの多くが混成モードのアーカイブ (32 ビットと 64 ビットの両方のオブジェクトが含まれている) であっても、コンパイラーとリンカーはいずれも **OBJECT_MODE=32_64** をサポートしません。
- **-q64** コンパイラー・オプション。このオプションは、64 ビット・モードのサポートを設定するオプションで、ターゲット・アーキテクチャーの命令セットを指定する **-qarch** オプションと組み合わせて機能します。**-q32** と **-q64** は、**-qarch** オプションの設定に優先します。**-q32** と **-q64** が矛盾する場合は、最後に指定されたオプションが使用されます。
- 64 ビット・コンパイルでの **-qarch** のサポート。このオプションは、コンパイラーに、特定の 64 ビット・アーキテクチャーでの実行に最適なコードを生成するように指示します。これに応じて、**-qarch** および **-q64** の特定の組み合わせのみを受け入れます。

詳細については、「*XL C/C++ プログラミング・ガイド*」を参照してください。

AIX ユーティリティ・コマンドのサポート

AIX オペレーティング・システムは、オブジェクト・ファイルを扱ういくつかのユーティリティ・コマンドを提供します。これらのコマンドは、AIX のオブジェクトおよびライブラリーを操作するのに必要です。これらのユーティリティは、どのビット・モードのオブジェクト・ファイルを処理するかを指定する **-X** オプションを使用して呼び出すと、64 ビット XCOFF オブジェクト形式をサポートします。XCOFF (eXtended Common Object File Format) は、AIX のオブジェクト形式です。**-X** に指定できる値は、以下のとおりです。

32 32 ビットのオブジェクト・ファイルのみを処理

64 64 ビットのオブジェクト・ファイルのみを処理

32_64 32 ビットと 64 ビットの両方のオブジェクト・ファイルを処理

共用オブジェクトと共用ライブラリーを操作する AIX ユーティリティ・コマンド

コマンド	説明
ar	リンケージ・エディターが使用する索引付きライブラリーを保守する。32 ビット・モードのときは、64 ビット・オブジェクトを無視し、メッセージは出しません。

コマンド	説明
dump	オブジェクト・ファイルの選択した部分を表示する。リンクとロードにおいては、 dump コマンドは実行可能ファイルと共用オブジェクトのヘッダー情報を調べるために使用されます。 dump -H コマンドを使用すると、ヘッダー情報が表示されるので、実行時のシンボル解決における実行可能ファイルまたは共用オブジェクトの依存関係を判別することができます。 dump -Tv コマンドは、共用オブジェクトまたは実行可能ファイルに対するシンボル定義を表示します。この定義は、オブジェクトがリンク時にエクスポートしているシンボル、オブジェクトまたは実行可能ファイルがロード時にインポートしようとするシンボル、およびこれらのシンボルを含む共用オブジェクトの名前 (既知の場合) から成っています。
file	ファイルのビット・モードと、ファイルがストリップされているかどうかを表示する。
genkld	システム・メモリーにロードされた共用オブジェクトのうち、特にシステムの共用ライブラリー・セグメント内にあるものをリストする。プライベートな共用オブジェクトは、メモリーにロードされていても、 genkld コマンドの出力では表示されません。
ldd	実行可能ファイルを開始するためにロードされる、共用オブジェクトとアーカイブ・メンバーをリストする。
lorder	オブジェクト・ライブラリー内のメンバー・ファイルの最適な順番を探します。
nm	オブジェクト・ファイル、実行可能ファイル、およびオブジェクト・ファイル・ライブラリーのシンボル情報を表示する。 nm -g コマンドは、指定したライブラリー・アーカイブに含まれるすべてのアーカイブ・メンバーを処理します。 dump とは異なり、 nm は、シンボルを提供する予定の共用オブジェクトまたはアーカイブ・メンバーの名前を表示しません。
ranlib	アーカイブ・ライブラリーをランダム・ライブラリーに変換する。
rtlib_enable	デフォルト・リンク、静的リンク、または遅延ロードでコンパイルされた共用ライブラリーを、実行時リンクが可能なものに変換する。
size	XCOFF オブジェクト・ファイルのセクション・サイズを表示する。
slibclean	root ユーザーが、システムの共用ライブラリー・セグメントから、使用回数 0 の共用オブジェクトしかアンロードできないようにします。このユーティリティーは、ソフトウェアの保守段階に入った実動システム用での使用が意図されています。特に、不要なアプリケーションを除去する前や、インストール済みのアプリケーションを更新する前に使用します。
strip	バインダーの使用する情報や、シンボリック・デバッグ情報を除去することで、XCOFF オブジェクト・ファイルのサイズを削減します。

関連資料

- 「AIX 5L Version 5.2 Reference Documentation: Commands Reference」

AIX のオブジェクトとライブラリー

他の UNIX オペレーティング・システムと同様に、AIX オペレーティング・システムは、共用ライブラリーとそれを使用するアプリケーションを作成、開発、テスト、およびデバッグするための機能を提供しています。AIX 共用ライブラリーのフィーチャーは、他の UNIX オペレーティング・システムのものとは広く互換性があります。しかし、AIX では、動的バインディングされた共用ライブラリーを作成および使用するための機能も提供しています。動的バインディングでは、ユーザーのコードで参照され、共用ライブラリーで定義される外部シンボルは、実行時にローダーによって解決されます。動的リンクは、少ないメモリーでプログラムを実行でき、作成する実行可能ファイルも小さくなります。加えて、AIX は動的ロードをサポートします。動的ロードは、リンカー・オプションや特別なオブジェクト・ファイル・タイプによってではなく、サブルーチンのセットによって指定されるプログラミング方式です。

AIX でこの多種類のリンク方式を実装およびサポートするためには、ファイルが eXtensible Common Object File Format (XCOFF) になっている必要があります。ここでは、様々な用語が AIX 上のアプリケーション開発のコンテキストで使用されています。オブジェクトと共用ライブラリーに関する解説では、コンテキスト内で用語が持つ正確な意味を確認しておくことが重要です。

「オブジェクト・ファイル」とは、実行可能コード、データ、再配置情報、シンボル・テーブル、およびその他の情報を含むファイルを総称する用語です。複数のオブジェクト・ファイルを単一のライブラリー・アーカイブ・ファイルにアーカイブすることができます。このファイルは、単にライブラリーとも呼ばれます。ライブラリーに含まれるオブジェクト・ファイルは、アーカイブ・メンバーと呼ばれます。複数のオブジェクト・ファイルを使用することと比べ、単一のライブラリーを使用する利点は、実行可能ファイルを作成するために処理するファイル数が少なく済むことです。

AIX で実行可能ファイルをビルドするには、使用するすべてのオブジェクト・ファイルとアーカイブ・メンバーが同じビット・モードである必要があります。ただし、ライブラリー・アーカイブには、32 ビットと 64 ビットの両方のオブジェクト・モジュールを含むことが可能です。AIX の提供するシステム・ライブラリーの多くは、モード混成です。AIX オペレーティング・システムでは、オブジェクトおよびライブラリーを照会、保守、および操作するためにコマンド・ユーティリティーを提供しています。これらのユーティリティーには、ライブラリー・アーカイブ・ファイルを作成および保守するための **ar**、オブジェクト・ファイルのビット・モードを表示するための **file**、アーカイブでシンボルを照会するための **dump**、および指定のビット・モードのライブラリーで使用可能なシンボルを照会するための **nm** などが含まれます。

AIX における共用オブジェクトと共用ライブラリーの相違点

共用ライブラリー という用語と、共用オブジェクト という用語は、他の UNIX オペレーティング・システムにおいては同義語として使用されています。他の UNIX オペレーティング・システムで共用ライブラリー (別名ダイナミック・リンク・ライブラリー または *DLL*) と呼ばれているものは、AIX では共用オブジェクトに相当します。

AIX では、共用ライブラリーと共用オブジェクトとの間にはっきりとした相違があります。プログラムが複数のシステム・ライブラリーとリンクするとき、共用オブジェクトは、実行可能ファイルの従属モジュールです。AIX では、共用オブジェクトの名前は固有のものである必要はありません (ただし、同じライブラリー内では固有の名前である必要があります)。別々のライブラリーに含まれた同名の共用オブジェクトは、別々のモジュールとして認識されます。AIX の提供するシステム・ライブラリーのほとんどには、1 つ以上の共用オブジェクトが含まれます。

共用オブジェクトは、XCOFF ヘッダーに SHROBJ フラグを含む、単一のオブジェクト・ファイルです。AIX の共用オブジェクトの命名規則は `name.o` で、コンパイラの生成するデフォルトのファイル名拡張子を使用します。

AIX の共用ライブラリーは、**ar** で作成されたアーカイブ・ライブラリー・ファイルを参照します。このファイルに含まれるアーカイブ・メンバーのうち 1 つ以上は、共用オブジェクトです。ライブラリーには、非共用オブジェクト・ファイル (別名静的 オブジェクト・ファイル) を含むこともできます。AIX 共用ライブラリーの命名規則は、`libname.a` という形式で、リンカーの期待するファイル名拡張子を使用します。

XCOFF (eXtensible Common Object File Format) は、AIX オペレーティング・システムで使用するオブジェクト・ファイル・フォーマットです。これは、マシン・イメージ・オブジェクトと実行可能ファイルの正式な定義です。これらのオブジェクト・ファイルは、主にバインダーとシステム・ローダーによって使用されます。XCOFF は、標準の Common Object File Format (COFF) を拡張し、ダイナミック・リンクと、オブジェクト・ファイル内のユニットの置き換えを可能にします。また、XCOFF は、32 ビットと 64 ビットの両方のオブジェクトおよび実行可能ファイルに使用可能です。

32 ビット XCOFF ファイル、64 ビット XCOFF ファイル、またはその両方を認識するプログラムを作成することが可能です。プログラム自体に関しても、どちらのビット・モードでもコンパイルすることができ、32 ビットまたは 64 ビットのプログラムを作成できます。プリプロセッサ・マクロを定義することによって、アプリケーションで XCOFF ヘッダー・ファイルから適切な構造の定義を取り出すことができます。

アセンブラーとコンパイラーは、出力として XCOFF ファイルを生成します。バインダーは、個々のオブジェクト・ファイルを結合して 1 つの XCOFF 実行可能ファイルを作成します。システム・ローダーは、XCOFF 実行可能ファイルを読み込み、プログラムの実行可能なメモリー・イメージを作成します。シンボリック・デバッガーは、XCOFF 実行可能ファイルを読み込み、実行可能なメモリー・イメージの関数と変数へのアクセスを提供します。

AIX における共用オブジェクトと静的オブジェクトの相違点

多くの UNIX オペレーティング・システムでは、ファイル名拡張子 `.so` は共用オブジェクト・ファイルであることを示し、ファイル名拡張子 `.o` は静的オブジェクト・ファイルであることを示します。AIX では、静的オブジェクト・ファイルと共用オブジェクト・ファイルの両方に対して、ファイル名拡張子 `.o` が使用されます。共用オブジェクト・ファイルでは、拡張子 `.o` は、要件ではなく規則です。リンカーは、ファイル・ヘッダーを読み取り、ファイルの内容を判別します。共用オ

ブジェクト・ファイルは、完全にリンクおよび解決されるという点、ロードおよび実行可能であるという点、さらに XCOFF ファイル・ヘッダーの **FLAGS** フィールドに **SHROBJ** ビット・セットがあるという点で、静的オブジェクト・ファイルと区別されます。オブジェクトが動的に参照される場合は、**SHROBJ** ビットが設定されます。その他の場合は、リンカーはオブジェクトを静的として扱います。

AIX では、ファイル名拡張子が **.so** である共用オブジェクトもサポートします。**.o** ファイル名拡張子の規則以外は、任意の拡張子を使用して、AIX の共用オブジェクト・ファイルに名前を付けることができます。AIX では、ファイル名拡張子 **.so** は実行時リンクを示します。

AIX オペレーティング・システムは、ファイルの XCOFF ヘッダーを表示して、オブジェクトが共用オブジェクトであるか静的オブジェクトであるかを判別するためのさまざまな AIX コマンドを提供しています。**dump -ov** コマンドは、指定したファイルの XCOFF ヘッダーを表示します。**dump -gov** コマンドでは、指定したライブラリーのアーカイブ・メンバーを調べ、それぞれのアーカイブ・メンバーが共用オブジェクトであるか静的オブジェクトであるかを判別することができます。

リンク時とロード時

共用オブジェクトおよび共用ライブラリーは、AIX でプログラムを作成および実行する際に、2 つの段階で使用されます。

リンク時では、リンケージ・エディターは、指定された共用オブジェクトと共用ライブラリーを検索して、実行可能ファイルの作成に必要なすべての未定義シンボルを解決します。共用オブジェクトに参照されているシンボルが含まれている場合は、結果として生成される実行可能ファイルの XCOFF ヘッダーにあるローダー・セクションに、その共用オブジェクトまたは共用ライブラリーへの参照が含まれます。共用オブジェクトであるライブラリー・メンバーでの、参照されているシンボルでも同様です。シンボルは、これらの共用オブジェクトまたは共用ライブラリーからエクスポートされ、実行可能ファイルにインポートされます。

プログラムのロード時に、システム・ローダーは実行可能ファイルの XCOFF ヘッダー情報を読み取り、参照されている共用ライブラリーを検出しようとします。参照されている共用オブジェクトと共用ライブラリーがすべて検出されれば、実行可能ファイルが開始されます。システム・ローダーは、プログラムの実行可能コンポーネント・セクションを、プロセスのアドレス・スペースの適切なセグメントにロードしようとします。共用オブジェクトおよび共用ライブラリーに含まれるプログラム・テキストは、そのプログラム・テキストを必要とする最初のプログラムによってグローバル・システム・メモリーにロードされ、以降はそのプログラム・テキストを必要とするすべてのプログラムによって共用されます。

リンク時エラーの診断

一般的なリンク時エラーとして、未解決のシンボル、シンボルの重複、およびリンカーのメモリー不足があります。

未解決のシンボル

リンカー・エラー警告は、アプリケーションが多くのライブラリー (特にサード・パーティー製品の供給するライブラリー) とリンクされるときによく起こります。外部シンボルが解決できないと、リンクは失敗し、実行可能ファイルは生成されません。未解決シンボルによるエラーが起こる原因で最も多いものは、入力ファイルが見つからないことです。例えば、デフォルトの C ランタイム・ライブラリー `libc.a` にないライブラリー関数を呼び出すときは、そのシンボルがあるアーカイブ・ライブラリー・ファイルを指定する必要があります。

システムのリンケージ・エディターは、オブジェクト・ファイル、共用オブジェクト・ファイル、アーカイブ・オブジェクト・ファイル、ライブラリー、およびインポート・ファイルやエクスポート・ファイルなどを入力として受け入れます。未解決のシンボルが定義されているライブラリー・ファイルはどのようにして見つけばよいのでしょうか? 製品資料を検索するか、提供されているライブラリーをリンク・コマンドですべてインクルードしてリンケージ・エディターにシンボルの位置を探させるか、**nm** コマンドを使用して自分で探すことができます。

いくつかの特定のリンカー・オプションは、ログ・ファイルを生成するので、このログ・ファイルを分析すれば、未解決のシンボルを参照するライブラリーまたはオブジェクト・ファイルを判別できます。ログ・ファイルは、誤って使用されている、相互に依存するライブラリーや冗長なライブラリーを追跡するために使用できます。

- **-bnoquiet** オプションは、各バインダー・サブコマンドとその結果を `stdout` に書き出します。未解決のシンボル参照があると、出力に表示されます。出力には、指定したライブラリー・モジュールからインポートされたシンボルのリストも表示します。
- **-bmap:filename** オプションは、アドレス・マップを生成します。未解決のシンボルがファイルの先頭にリストされ、続いてインポートされたシンボルが表示されます。
- **-bloodmap:filename** オプションは、リンカーに受け渡されたすべての引き数、読み取られたすべての共用オブジェクト、およびインポートされているシンボルの数の情報を含むログ・ファイルを生成します。未解決のシンボルが検出された場合は、そのシンボルを参照するオブジェクト・ファイルまたは共用オブジェクトがログ・ファイルにリストされます。サード・パーティー製品で提供されているライブラリーの未解決シンボルを検出するには、ログ・ファイルを使用して、その製品の提供する他のライブラリー内を探します。

シンボルの重複

シンボルの重複によるエラーは、多くの場合プログラミング上の誤りによるものです。複数の外部関数定義を同じ名前にしてはなりません。ソース・コードが複数の外部関数定義を含む場合は、リンケージ・エディターは最初に処理した定義を使用するので、意図した結果にならない場合があります。推奨される解決策は、関数名を変更するか、静的関数を使用することです。

C++ でテンプレート関数を使用した場合にも、テンプレートが複数のソース・ファイルで暗黙的にインスタンス化されていると、リンク時にシンボルの重複によるエラーが発生することがあります。推奨される解決策は、**-qtemplateregistry** オプションおよび VisualAge C++ Professional バージョン 6 から追加されたテンプレート・ハンドリング機構を使用することです。

シンボルの重複の別の原因は、インライン化された関数です。すべてのインライン化された関数は、別個のインスタンス生成として表示されるシンボル・テーブルの項目を作成します。AIX 5.2 でオプション **-qweaksymbol** を使用してコンパイルすると、これらの警告の数を減らすことができます。

リンカーのメモリー不足

容量が非常に大きいファイルをリンクすると、リンカー・プロセスに割り当てられたメモリーを使い切ってしまうことがあります。解決策として、コマンドを呼び出しているユーザーのページング・スペースまたはリソース制限を増やすことが考えられます。**ulimit** コマンドを使用すると、リンカーを呼び出すユーザーのリソース制限を制御することができます。他の方法として、リンカー・プロセスを 32 ビット・ラージ・メモリー・モデルで実行することも考えられます。

実行時エラーの診断

プログラムが正常にコンパイルおよびリンクしても、実行時に予期しない結果を起こす場合があります。コンパイラーは、言語の構文規則に違反しないプログラミング・エラーは診断しません。このセクションでは、一般的なエラーとその発見方法、および訂正方法を説明しています。

未初期化変数

自動ストレージ期間を持つオブジェクトは、暗黙的に初期化されないため、初期値は未定です。**auto** 変数が値を設定される前に使用されると、プログラムの実行のたび、同じ結果を生成するかどうかは一定していません。コンパイラー・オプション **-qinfo=gen** を指定すると、値が設定される前に使用されている **auto** 変数の位置を表示します。**-qinitauto** オプションは、すべての自動変数を指定した値で初期化するようにコンパイラーに指示します。このオプションを使用すると、アプリケーションの実行時のパフォーマンスが落ちるため、デバッグ時にのみ使用することをお勧めします。

実行時検査

-qcheck オプションを指定すると、実行可能ファイルに実行時検査のコードを挿入します。サブオプションで、NULL ポインター、配列の範囲を超える指標付け、およびゼロによる除法を検査するように指定します。**-qinitauto** と同様に、**-qcheck** もアプリケーションのパフォーマンスが低下するため、デバッグ目的でのみ使用することが推奨されます。

ANSI 別名割り当て

型ベースの別名割り当て（「ANSI 別名割り当て」とも呼ばれます）では、データ・オブジェクトに安全にアクセスするために使用できる左辺値を制限します。言語標準への準拠を強制する言語レベルでコンパイルすると、C および C++ コンパイラーは、最適化を行うときに型ベースの別名割り当てを必ず実行します。ANSI の別名割り当て規則では、ポインターは、同じ型のオブジェクトか、互換性のある型のオブジェクトに対してのみ間接参照することができると規定されています。この規則の例外は、符号および型の修飾子には型ベースの別名割り当ては適用されないこと、および文字型ポインターはどのタイプも指し示すことができるということで

す。よく使用されるコーディング方法として、ポインタを非互換の型にキャストしてから間接参照することがありますが、これはこの規則に違反しています。

-qalias=noansi を設定して、ANSI 別名割り当てをオフにすると、プログラムの動作は訂正できるかもしれませんが、コンパイラーがアプリケーションを最適化できる機会を減らし、実行時のパフォーマンスが劣化します。推奨される解決策は、プログラムを修正して型ベースの別名割り当て規則に合わせることです。

#pragma option_override

最適化を使用している場合にのみエラーが表示される場合があります。複雑なアプリケーションでは、プログラミング・エラーを含むことが分かっている関数に対して最適化をオフにし、プログラムの残りの箇所では最適化をオンにすると、特に有益な場合があります。 `#pragma option_override` ディレクティブを使用して、特定の関数に別の最適化オプションを指定することができます。

`#pragma option_override` ディレクティブは、エラーの原因となっている関数を判別するために使用することもできます。問題の関数を見つけるには、エラーが出なくなるまで、ディレクティブ内でそれぞれの関数の最適化を順番にオフにしていきます。

共用メモリーの並列処理

プログラムの並列実行を実現し、シングル・プロセッサで実行するより高速でジョブを完了するための、証明された手法がいくつかあります。これらの手法には、以下のものがあります。

- ディレクティブ・ベースの共用メモリー並列処理 (SMP)
- コンパイラーへの共用メモリー並列処理の自動生成命令
- メッセージ引き渡しベースの共用または分散メモリー並列処理 (MPI)
- POSIX スレッド (pthread) 並列処理
- `fork()` および `exec()` を使用した下位 UNIX 並列処理

アプリケーションの移植性要件は、明らかに、使用に最適な技法を選択する際の要因です。この選択はまた、アプリケーション、プログラマーのスキルおよび設定、およびターゲット・マシンの特性に非常に依存します。AIX での並列処理を達成する 2 つの主要な方法は、手動でコーディングした POSIX スレッド (pthread) の使用と OpenMP ディレクティブの使用です。

AIX オペレーティング・システムの並列プログラミング機能はスレッドの概念に基づいています。並列プログラミングはマルチプロセッサ・システムの利点を活用しながら、既存の単一プロセッサ・システムで完全なバイナリー互換性を維持しています。これは、単一プロセッサ・システムで作業するマルチスレッドのプログラムは、再コンパイルなしでマルチプロセッサ・システムを利用することができることを意味します。

pthread は、並列処理プロセスの最大限の制御を提供するため、複数のプロセッサを効率的に使用する優れた柔軟性が実現します。そのトレードオフとして、コードが大幅に複雑化します。多くの場合、OpenMP ディレクティブ、SMP 対応ライブラリー、またはコンパイラーの自動 SMP 機能の使用といった簡単な方法が推奨さ

れます。スレッドを明示的に使用しても、必ずしもパフォーマンスが向上するわけではありません。マルチスレッドのアプリケーションをデバッグすると、どうしても問題が起こりがちです。ただし、一部のプログラムでは、これのみが実行可能な方法です。

以下に各技法の利点と欠点の一部をリストします。

コンパイラーによる自動並列処理

- 簡単なインプリメント (-qsmp=auto とコンパイル)。
- チームでの作業が容易。
- データ・スコープが無視されることによる限られたスケーラビリティ。
- コンパイラー依存 (特定のコンパイラーのリリースの場合でも)。
- 必ずしも移植可能ではない。

SMP 対応ライブラリー

- 作業がほとんど必要ない。
- 必ずしも移植可能ではない (通常専有)。
- 限られた柔軟性。

OpenMP ディレクティブ

- 移植可能。
- 自動並列処理において潜在的により優れたスケーラビリティ。
- ユニフォーム・メモリー・アクセスを前提とする。

ハイブリッド・アプローチ (OpenMP および pthread のサブセットまたは混合、または UNIX fork() および exec() 並列処理、プラットフォーム固有の構文)

- チームワークを可能にすることがある。
- パフォーマンスと移植性を確実にするため、十分に検証された概念が必要。
- 必ずしも移植可能ではない。

pthread

- 移植可能。
- 並列処理プロセスに対する最大限の制御を提供する。
- 自動並列処理の潜在的に最適なスケーラビリティ
- 複雑なコード化を処理するのに経験のあるプログラマーが必要。

OpenMP ディレクティブ

OpenMP ディレクティブは、特定のループをどのように並列処理するかをコンパイラーに命令するコマンド・セットです。ソースにディレクティブがあると、コンパイラーが並列コードで並列分析を実行する必要がなくなります。OpenMP ディレクティブを使用するには、並列処理に必要なインフラストラクチャーを提供する pthread ライブラリーが必要です。

OpenMP ディレクティブは、アプリケーションの並列処理に重要な 3 つの問題に取り組みます。まず、文節およびディレクティブはスコープ変数に使用可能です。ほとんどの場合、変数を共有するべきではありません。つまり、プロセッサはそれ

ぞれ、それ自身の変数のコピーを持っている必要があります。第 2 に、作業共用ディレクティブは、コードの並列領域に含まれる作業を複数の SMP プロセッサ間で分散させる方法を指定します。最後に、プロセッサ間で同期用のディレクティブがあります。

コンパイラーは OpenMP バージョン 2.0 仕様をサポートします。

関連資料

- 73 ページの『付録 B. OpenMP 準拠とサポート』

AIX のスレッド

AIX でのスレッドのインプリメンテーションは、外部インターフェースとプログラムが使用する前提となるセマンティクスを処理するため、外部移植問題として分類されます。AIX スレッド・ライブラリーは単一 UNIX 仕様バージョン 2 に準拠しており、この仕様には、POSIX スレッド規格として知られる IEEE POSIX 1003.1c 規格が含まれます。ライブラリーはまた、Open Group 98 仕様に準拠し、拡張スレッド機能を POSIX スレッド規格に追加します。これにより、pthread の標準化インターフェースにおける、スレッド化プログラムの移植性が確保されます。

AIX での POSIX スレッド・ライブラリー

AIX では、POSIX スレッド (pthread) は、C 言語プログラミング型およびサブルーチン呼び出しのセットとして定義され、ヘッダー・ファイル (/usr/include/pthread.h) および POSIX スレッド・ライブラリー (/usr/lib/libpthreads.a) でインプリメントされています。

以下のデータ型は pthread.h ヘッダー・ファイルでスレッド・ライブラリー用に定義されています。これらのデータ型の定義はシステムによって異なっても構いません。

pthread_t	スレッドを識別する。
pthread_attr_t	スレッド属性オブジェクトを識別する。
pthread_cond_t	条件変数を識別する。
pthread_condattr_t	条件属性オブジェクトを識別する。
pthread_key_t	スレッド固有のデータ・キーを識別する。
pthread_mutex_t	mutex を識別する。
pthread_mutexattr_t	mutex 属性オブジェクトを識別する。
pthread_once_t	一回限りの初期設定オブジェクトを識別する。

AIX は、POSIX スレッド規格のバージョン 7 のドラフトに従って作成された、既存のマルチスレッド・アプリケーションにバイナリー互換性を提供します。互換性 POSIX スレッド・ライブラリー、/usr/lib/libpthreads_compat.a は、これらのアプリケーションとの後方互換性用にのみ提供されています。互換性 POSIX スレッド・ライブラリーは、32 ビット・アプリケーションのみをサポートしています。

以下のオペレーティング・システム・ファイルは pthread の AIX インプリメンテーションを提供します。

/usr/include/pthread.h	ほとんどの pthread 定義をもつ C/C++ ヘッダー。
/usr/include/sched.h	一部のスケジューリング定義をもつ C/C++ ヘッダー
/usr/include/unistd.h	pthread_atfork() 定義をもつ C/C++ ヘッダー。
/usr/include/sys/limits.h	一部の pthread 定義をもつ C/C++ ヘッダー。
/usr/include/sys/pthdebug.h	ほとんどの pthread デバッグ定義をもつ C/C++ ヘッダー。
/usr/include/sys/sched.h	一部のスケジューリング定義をもつ C/C++ ヘッダー
/usr/include/sys/signal.h	pthread_kill() および pthread_sigmask() 定義をもつ C/C++ ヘッダー。
/usr/include/sys/types.h	一部の pthread 定義をもつ C/C++ ヘッダー。
/usr/lib/libpthreads.a	UNIX98 pthread および POSIX 1003.1c pthread を提供する 32 ビット/64 ビット・ライブラリー。
/usr/lib/libpthreads_compat.a	POSIX 1003.1c Draft 7 pthread を提供する 32 ビットのみのライブラリー。
/usr/lib/profiled/libpthreads.a	UNIX98 pthread および POSIX 1003.1c pthread を提供するプロファイルを作成された 32 ビット/64 ビット・ライブラリー。
/usr/lib/profiled/libpthreads_compat.a	POSIX 1003.1c Draft 7 pthread を提供するプロファイルを作成された 32 ビットのみのライブラリー。

スレッド・アンセーフ・ライブラリーはプログラムのたった 1 つのスレッドによって使用されることがあります。メイン・スレッドのみが、スレッド・アンセーフ・コードを安全に使用することができます。この制約事項は、スレッド固有の `errno` ではなく、グローバル `errno` のアクセスに起因するものであり、メイン・スレッドが行うライブラリー呼び出しにも適用されます。

AIX での Pthread

従来は、複数の単一スレッド化プロセスを使用して並列処理を行ってきましたが、一部のプログラムでは、並列処理のより細分化されたレベルから恩恵を受けることができます。マルチスレッド化プロセスはプロセス内で並列処理を提供し、複数の単一スレッド化プロセスのプログラミングに関連する多くの概念を共用します。

このセクションでは、他の UNIX システムと比べて、AIX における並列プログラミング機能のインプリメンテーションに関するポイントについて説明します。

用語: 従来の単一スレッド化プロセス・システムでは、スレッドおよびプロセスの特性は、プロセス と呼ばれる単一エンティティーに分類されます。他のシステムでは、スレッド は時に 軽量プロセス と同義語で、またスレッド の意味は時に プロセス とわずかに違うだけです。

AIX では、カーネル・スレッド と ユーザー・スレッド は区別されています。AIX では、軽量プロセス は通常カーネル・スレッドを意味します。ユーザー・スレッド は、プロセス内の他の独立した制御のフローと同じアドレス・スペース内で作動する、独立した制御のフローです。この説明の残りでは、スレッド という用語は、ユーザー・スレッドを意味します。

カーネル・スレッド はシステム・スケジューラーが処理するスケジュール可能なエンティティです。AIX では、カーネル・スレッドはプロセス内で稼働しますが、システムの他の任意のスレッドによって参照されます。ただし、プログラマーがカーネル・スレッドを直接制御することはできません。カーネル・スレッドは強くインプリメンテーションに依存するため、POSIX スレッド・ライブラリーのようなライブラリーは移植可能プログラムの書き込みを容易にするため、ユーザー・スレッドに提供されます。

ユーザー・スレッド は、プログラム内で複数の制御のフローを処理するため、プログラマーが使用するエンティティです。ユーザー・スレッドを処理するための標準化された API はスレッド・ライブラリーによって提供されます。ユーザー・スレッドは、プロセス内にのみ存在し、他のプロセスでユーザー・スレッドを参照することはできません。ライブラリーは専用のインターフェースを使用し、ユーザー・スレッドの実行のためにカーネル・スレッドを処理します。

プロパティ: 従来の単一スレッド化プロセス・システムでは、プロセスにプロパティ・セットがありました。AIX のようなマルチスレッド・プロセス・システムでは、これらのプロパティはプロセスとスレッドで分割されています。

マルチスレッド・システムの プロセス は、変更可能なエンティティです。プロセスは、実行フレームとして考慮される必要があります。これには、プロセス ID、プロセス・グループ ID、ユーザー ID、グループ ID、環境、作業ディレクトリーなどの従来のプロセス属性があります。プロセスはまた、ファイル記述子、シグナル・アクション、共用ライブラリー、およびプロセス間通信ツールなどの共通のアドレス・スペースと共通のシステム・リソースを提供します。

スレッド はスケジュール可能なエンティティで、独立したフローの制御を確保する必要があるプロパティのみを持っています。これらのプロパティには、スタック、スケジューリング・ポリシーまたは優先順位、保留およびブロック・シグナルのセット、スレッド固有データがあります。スレッド固有データの例は、**errno** エラー標識です。AIX では、各スレッドはそれ自身の **errno** エラー標識を持っていますマルチスレッド・システムでは、**errno** は通常、グローバル変数ではなく、スレッド固有の **errno** 値を戻すサブルーチンです。他のシステムでは、**errno** の他のインプリメンテーションを提供する可能性があります。

プロセス内のスレッドは、プロセスのグループとして考えるべきではありません。すべてのスレッドが同じアドレス・スペースを共有します。これは、2 つのスレッドで同じ値を持つ 2 つのポインターが同じデータを参照することを意味します。また、スレッドが共用システム・リソースの 1 つを変更すると、プロセス内のすべてのスレッドが影響を受けます。例えば、スレッドがファイルを閉じると、ファイルはすべてのスレッドのために閉じられます。

スレッドは作成したスレッドのリストを維持しませんし、それを作成したスレッドを知りません。スレッド作成は、親-子関係がスレッド間で存在するという点で、プ

ロセス作成とは異なります。最初のスレッドを除いて、すべてのスレッドは、同じ階層レベルにあります。最初のスレッドは、プロセスが作成される際に、自動的にオペレーティング・システムによって作成されます。

スレッド属性オブジェクトはスレッドの属性をカプセル化し、スレッドが作成される前に定義される必要があります。エンタリー・ポイント・ルーチンおよび引き数もまた、作成時に指定される必要があります。すべてのスレッドには 1 つの引き数と 1 つのエンタリー・ポイント・ルーチンがありますが、同じエンタリー・ポイント・ルーチンを幾つかのスレッドで使用することができます。

スレッド・モデルおよび仮想プロセッサ: ユーザー・スレッドはスレッド・ライブラリーの仮想プロセッサ (VP) によってカーネル・スレッドにマップされます。マッピングが行われる方法は、スレッド・モデル と呼ばれます。AIX のデフォルトのスレッド・モデルは M:N モデルで、すべてのユーザー・モデル・スレッドがカーネル・スレッドのプールにマップされ、すべてのユーザー・スレッドが仮想プロセッサのプールで稼働します。これは最も効果的で、複雑なスレッド・モデルで、ユーザー・スレッド・プログラミングの機能はスレッド・ライブラリーとカーネル・スレッド間で共有されます。M:N モデルはまた、単一ユーザー・スレッドが固有の仮想プロセッサ (1:1 スレッド・モデル) と結合し、すべての未結合ユーザー・スレッドが残りの VP を共用する場合に対応します。

1:1 スレッド・モデルはそれぞれのユーザー・スレッドを 1 つのカーネル・スレッドにマップし、すべてのユーザー・スレッドが 1 つの VP で稼働します。ユーザー・スレッド・プログラミングのほとんどの機能は直接カーネル・スレッドによって処理されます。1:1 モデルは、AIXTHREAD_SCOPE 変数の値を S に設定することによって使用可能にできます。

M:1 スレッド・モデルはどのシステムでも使用可能で、特に従来の単一スレッド・システムで使用されます。すべてのユーザー・スレッドは、ライブラリー・スケジューラーによって 1 つのカーネル・スレッドにマップされ、すべてのユーザー・スレッドは 1 つの VP で稼働します。ユーザー・スレッド・プログラミングのすべての機能は完全にライブラリーによって処理されます。

pthread ライフ・サイクルの概要: スレッド化されたプログラムの各スレッドには、それ自身の private プログラム・カウンター、スタック、およびレジスターがあります。メモリー状態およびファイル記述子は共用されます。

pthread.h ヘッダー・ファイル

pthread.h ヘッダー・ファイルは、スレッド・ライブラリーを使用する各ソース・ファイルの最初に含まれるファイルで、スレッド・セーフ・サブルーチンを確実に使用する必要があります。ヘッダー・ファイルには、すべてのサブルーチン・プロトタイプ、マクロ、およびスレッド・ライブラリーを使用するためのその他の定義が含まれています。また、**errno** グローバル変数をスレッド固有の **errno** 値を戻す関数として再定義します。このため、**errno** ID はもはや、マルチスレッド・プログラムの左辺値ではありません。

次のグローバル・シンボルは pthread.h ファイルで定義されています。

POSIX_REENTRANT_FUNCTIONS	すべての関数が再入可能であることを指定する。
---------------------------	------------------------

コンパイラ呼び出し

スレッド化アプリケーションは、コンパイラの `_r` をサフィックスとする呼び出しの 1 つによりコンパイルされ、リンクされる必要があります。再入可能呼び出しコマンドがあり、これによって確実に、適正かつ適切なオプションが使用され、プログラムが再入可能およびスレッド・セーフ・ライブラリーとリンクされます。例えば、`__xlc_r` はシンボル `_THREAD_SAFE` を定義し、`pthread` ライブラリーとリンクします。

スレッド作成

`pthread` プログラムの実行は、オペレーティング・システムが作成した単一スレッドとして開始されます。その後、使用可能なプロセッサで同時に作業をスケジューリングしながら、必要に応じて追加のスレッドが作成および終了されます。

最初のスレッドを除くスレッドは、`pthread_create` 関数を使用して作成されます。この関数には 4 つの引き数があります。正常に終了した場合に戻されるスレッド ID、スレッド属性オブジェクトに対するポインター、スレッドが実行する関数、およびスレッド関数に渡される引き数です。スレッド関数は、(型 `void *` の) 単一のポインター引き数を取り、(型 `void *` の) ポインターを戻します。実際、スレッド関数に対する引き数は構造体に対するポインターであることが多く、その構造体はスレッド関数にアクセスできる多くのデータ項目を含むことがあります。

プログラムは固定された数のスレッドを作成することができます。ただし、多くの場合、ランタイム時に作成するスレッド数をプログラムで決定する一方、環境変数の設定によってデフォルトの振る舞いをオーバーライドする機能を提供するのが有効です。例えば、`OpenMP` プログラムでのデフォルトは、使用可能なプロセッサと同じ数のスレッドを作成することです。AIX では、`libc` から `sysconf` ルーチンを読み出すことによって、オンライン・プロセッサの数を取得することができます。

スレッド終了

スレッドは、スレッド関数の実行が終了すると、暗黙的に終了します。スレッドは `pthread_exit` を呼び出すことによって明示的に終了することができます。また、あるスレッドで、`pthread_cancel` 関数を呼び出すことによって他のスレッドを終了させることも可能です。

最初のスレッドには、特別なプロパティがあります。最初のスレッドが実行ストリームの最後に達して戻ると、出口ルーチンが呼び出され、その時点で、そのプロセスに属するすべてのスレッドが終了します。ただし、最初のスレッドは切り離されたスレッドを作成でき、安全に `pthread_exit` を呼び出すことができます。この場合、残りのスレッドは各自のスレッド関数の実行を続け、プロセスは最後のスレッドが終了するまでアクティブなままになります。多くのアプリケーションでは、最初のスレッドがスレッド・グループを作成し、それらが終了するのを待って継続または終了するのが効率的です。これは、結合可能なスレッドを使用して実行することができます。AIX では、デフォルトの設定値は `detached` です。関数 `pthread_join` は、参照されたスレッドが終了するまで、呼び出しスレッドを中断し

ます。 `AIXTHREAD_SCOPE` 環境変数のシステム・スコープ属性は、N スレッドが N プロセッサで同時に実行される予定の場合に適切です。

同期: マルチスレッド・プログラムでは、同じ関数と同じリソースが幾つかの制御のフローによって同時にアクセスされることがあります。リソースの整合性を保護するには、マルチスレッド・プログラムのために書き込まれたコードは再入可能かつスレッド・セーフでなければなりません。再入可能とスレッド・セーフティは関数に関連する別々の概念です。関数は、再入可能またはスレッド・セーフのいずれか、その両方であるか、またはいずれでもない場合があります。

再入可能関数 は、連続する呼び出しで静的データを保持せず、静的データにポインタを戻すこともありません。すべてのデータは関数の呼び出し元によって指定されます。再入可能関数はまた、再入不可関数を呼び出すことはできません。ほとんどの場合、再入不可関数が再入可能関数となるためには、変更インターフェースを持つ関数と置き換えられる必要があります。再入不可関数は通常、スレッド・アンセーフでもあります。

スレッド・セーフ関数 は、ロックによって並行アクセスから共用リソースを保護します。スレッド・セーフティは関数のインプリメンテーションにのみ関係し、その外部インターフェースには影響しません。マルチスレッド・プログラムでは、複数のスレッドによって呼び出されるすべての関数がスレッド・セーフでなければなりません。

グローバル・データの使用は、明示的な同期または直列化を行わない場合は、スレッド・アンセーフです。グローバル・データは、そのアクセスを直列化できるように、スレッド毎に維持されるか、カプセル化される必要があります。スレッドは、他のスレッドによって起こったエラーに対応したエラー・コードを読み取ることがあります。AIX では、各スレッドがそれぞれの `errno` 値を持っています。

`ctime` および `strtok` サブルーチンなどの一部の標準 C サブルーチンは、ライブラリーに属し、スレッド・セーフとみなされても、再入不可です。サブルーチンの再入可能バージョンには、サフィックス `_r` を持つオリジナルのサブルーチンの名前があります。

`threadprivate` と共用変数間の区別はプログラムの正確さとパフォーマンスのために重要です。また、プログラムは `OpenMP` ディレクティブを使用して共用メモリーの並列処理を行うこともできます。共用変数へのアクセスは同期化され、競合を回避し、確実に正確な結果を出す必要があります。ただし、同期を使用する場合は、パフォーマンスとスケーラビリティの低下とバランスを取る必要があります。

共用メモリー用の並列プログラミングで主に困難なのは、ローカル変数およびグローバル変数の正しいバランスを見つけることです。これは、このスコープによって、どの変数を `private` または `shared` とするかを定義するからです。グローバル変数の競合は、`reduction sum` の場合と同様、パフォーマンス上の問題の主な原因です。一時ローカル変数の導入が、このような問題を解決する手助けになることがよくあります。

マルチスレッド・アプリケーションでは、共用メモリー・ロケーションの更新は通常、`mutex` ロックによって保護されます。オペレーティング・システムは確実に、

共用データへのアクセスを直列化します。一定の時間で、1 つのスレッドのみがロックとアンロックの間の領域に入ることができ、データを変更します。

AIX プロセス間のメモリーの共用: AIX とほとんどの UNIX システムによって、幾つかのプロセスが共用メモリーとして知られる共通データ・スペースを共有することができます。条件変数と mutex のプロセス共用属性は、これらのオブジェクトを共用メモリーに割り振り、異なるプロセスに属するスレッド間で同期をサポートすることができることを意味します。ただし、共用メモリー管理の業界標準インターフェースがないため、プロセス共用 POSIX オプションは AIX スレッド・ライブラリーにインプリメントされません。

マルチスレッド・プログラムのデバッグ



AIX オペレーティング・システムは **dbx** コマンドを提供し、C/C++ および Fortran プログラムのシンボリック・デバッグ・プログラムを呼び出します。**dbx** コマンドには、属性、条件、mutex、およびスレッドを含む、スレッド関連オブジェクトを表示するサブコマンドがあります。

カーネル・プログラミングでは、オペレーティング・システムはカーネル・デバッグ・プログラムを提供します。詳しくは、AIX 資料の一部である *AIX 5L Version 5.2 Kernel Extensions and Device Support Programming Concepts* を参照してください。




マルチスレッド・プログラムの調整

マルチスレッド・プログラムの調整の 1 つの特徴として、スレッド・ライフ・サイクルにおいて異なる状態で経過する時間量を調整することがあげられます。さまざまなスレッド化機能によって、ライフ・サイクル・イベント・シーケンスのペースおよびリソース消費量を制御します。スレッドの並列作業が終了すると、スレッドはしばらくの間 (spin 待機時間) 待機 (スピン) しますが、待機中にプロセッサ時間を消費します。spin 待機時間後、yield 待機時間が指定されている場合、スレッドはそのカーネル・スレッドを他の実行可能なスレッドに譲歩することができます。yield 待機時間の期限が切れると、スレッドはスリープ状態に入ります。スレッドをスリープ状態から復活させると、スレッドが yield 状態にある場合よりさらにリソース主体になります。

AIX オペレーティング・システムは、さまざまなスレッド化機能の振る舞いを変更する環境変数を提供します。次のテーブルでは、マルチスレッド・プログラムを調整する際に便利な選択された環境変数を示しています。

説明	構文
AIXTHREAD_SCOPE スレッド競合スコープは、システム・スケジューリング・キューの項目に対してアプリケーション・レベルの <code>pthread</code> のマッピングを制御します。 AIXTHREAD_SCOPE によって、アプリケーションを変更する必要なく、スレッド・スコープを変更可能とすることによって、アプリケーションの振る舞いを検討することができます。単一のプロセスには両スコープの <code>pthread</code> が含まれていることがあります。 <code>pthread_create()</code> に対する属性引き数が <code>NULL</code> でない場合、これを行うことができます。そして、属性構造体のコンテンツがスレッドのスコープを決定します。	 AIXTHREAD_SCOPE=  ここで、 P プロセス競合スコープ (デフォルト) を示し、M:N スレッド・モデルを暗黙指定します。プロセッサより多くのスレッドがある場合に使用するのが最適です。 S システム競合スコープを示し、1:1 スレッド・モデルを暗黙指定します。各ユーザー・スレッドは直接 1 つのカーネル・スレッドにマップされます。
AIXTHREAD_MNRRATIO <code>pthread</code> を作成および終了する際に、 <code>pthread</code> ライブラリー内で使用される位取り係数を制御します。プロセス・スコープ・スレッドにのみ適用されます。デフォルトの M:N 率は 8:1 で、つまり、すべてのカーネル・スレッドごとに 8 つの <code>pthread</code> ということです。環境変数の設定およびエクスポートによって M:N 率を変更します。	AIXTHREAD_MNRRATIO= <i>M:N</i> ここで、 <i>M</i> ユーザー・スレッドの数を示します。デフォルト値は 8 です。 <i>N</i> カーネル・スレッドの数を示します。デフォルト値は 1 です。
SPINLOOPTIME <code>mutex</code> の可用性をブロッキングする前に、 <code>pthread</code> が <code>mutex</code> を取得しようとする回数を制御します。環境変数の設定およびエクスポートによってスピン・ループ時間を変更します。	SPINLOOPTIME=<i>N</i> ここで、 <i>N</i> 他の <code>pthread</code> に譲歩する前に、ビジー・ロックを再試行する回数を示します。デフォルト値は 40 です。
YIELDLOOPTIME スリープになる前にビジー・スピン・ロックを取得しようとする際に、システムがプロセッサを譲歩する回数を制御します。 <code>pthread</code> がロックでスピンし、それを取得できない場合は、スレッドはスリープになります。 <code>yield</code> ループ時間の値を増やすと、 <code>pthread</code> の遅延と <code>pthread</code> がスリープ状態に入ることを防ぎます。この変数は、 SPINLOOPTIME も設定されている場合にのみ使用されます。 <code>yield</code> ループ時間の値は、環境変数の設定およびエクスポートによって変更できます。	YIELDLOOPTIME=<i>N</i> ここで、 <i>N</i> ビジー <code>mutex</code> またはスピン・ロックを取得するために譲歩する回数を示します。デフォルト値は 0 です。

スレッド化に影響を与える選択された環境変数

説明	構文
AIXTHREAD_MINKTHREADS プロセスに使用すべきカーネル・スレッドの最小回数を設定します。平均のスレッド化プロセスは、少なくともスケジューリング pthread に使用可能なこのカーネル・スレッド数を持っています。	▶▶—AIXTHREAD_MINKTHREADS=N—▶▶ ここで、 N カーネル・スレッドの数を示します。デフォルト値は 8 です。
AIXTHREAD_MUTEX_DEBUG スレッド・プロセスを実行する際の mutex に関するデバッグ情報のコレクションを制御するスイッチ。デフォルトは OFF です。デバッグ情報を維持すると、パフォーマンスに悪影響を及ぼすことがあります。	▶▶—AIXTHREAD_MUTEX_DEBUG=  —▶▶
AIXTHREAD_COND_DEBUG 条件変数に関するデバッグ情報のコレクションを制御するスイッチ。デフォルトは OFF です。デバッグ情報を維持すると、パフォーマンスに悪影響を及ぼすことがあります。	▶▶—AIXTHREAD_COND_DEBUG=  —▶▶
AIXTHREAD_RWLOCK_DEBUG pthread ライブラリーが、デバッグ・プログラムによって表示できるすべての読み取り/書き込みロックのリストを維持する原因となります。デフォルトは OFF です。	▶▶—AIXTHREAD_RWLOCK_DEBUG=  —▶▶
AIXTHREAD_GUARDPAGES 作成する 4 KB の保護ページ数を指定します。保護ページはスレッド・スタックがその最大数を超える場合を検出し、errant メモリー書き込みに対して保護するために使用されます。スレッド・スタックは、プロセス・ヒープに作成され、スタックのトップに読み取り専用保護ページを設定することによって保護できます。これらのページに書き込もうとすると、即時にセグメンテーション違反が起こります。スタック・オーバーフロー時に条件を調査すると、プログラムをデバッグし、適切な修正アクションを決定することができます。この変数を設定し、エクスポートします。	▶▶—AIXTHREAD_GUARDPAGES=N—▶▶ ここで、 N サイズ 4 KB のページ数を示します。デフォルト値は 0 です。 アプリケーションがそれ自身のスタックを指定し、そのプロセス・ヒープに大容量のページを使用すると、保護ページは作成されません。

スレッド化に影響を与える選択された環境変数

説明	構文
AIXTHREAD_SLPRATIO システムにスリープ率を設定します。その値は、カーネル・スレッドがスリープしている pthread に予約保持される必要がある回数をシステムに伝えます。この調整パラメーターによって、カーネル・リソースをさらに管理することができます。デフォルトのスリープ率は 1:12 です。	▶—AIXTHREAD_SLPRATIO=<i>k</i>:<i>p</i>—▶ ここで、 <i>k</i> カーネル・スレッドの数を示します。デフォルト値は 1 です。 <i>p</i> スリープしている pthread の数を示します。デフォルト値は 12 です。 正整数値は <i>k</i> および <i>p</i> に指定されることがあります。 <i>k</i> > <i>p</i> の場合は、その比率は 1:1 として処理されます (つまり、pthread よりカーネル・スレッドを指定することはできません)。
AIXTHREAD_STK スレッド・スタック・サイズを変更します。この環境変数を使用して、pthread 属性構造体でパラメーターを調整し、アプリケーションを再コンパイルおよび再ビルドする代わりに、最大サイズ 256MB までのスタックを指定します。スレッド・スタックのサイズを増やすと、各スレッド・スタックがプロセス・ヒープに作成されるため、動的に割り当てられたメモリーに使用可能なスペース量が減ることに注意してください。この環境変数をエクスポートして、スタック・サイズをプログラマチックに指定することなく作成した pthread のスタック・サイズを変更します。	▶—AIXTHREAD_STK=<i>N</i>—▶ ここで、 <i>N</i> バイト数を示します。デフォルトのスレッド・スタック・サイズは、32 ビット・アプリケーションで 96 KB で、64 ビット・アプリケーションで 192 KB です。

GNU C および C++ の移植性に関連したフィーチャー

GNU C で開発されるアプリケーションまたはコードの移植を容易にするために、XL C/C++ は C99 と標準 C++ に対する GNU C および C++ 言語拡張機能のサブセットをサポートします。このセクションの表では、サポートされるフィーチャー、サポートされないフィーチャー、および構文は受け入れられるがセマンティクスは無視されるフィーチャーがリストされます。

C コードでサポートされる 拡張機能を使用するために、**xlc** または **cc** 呼び出しコマンドを使用するか、または **-qlanglvl=extc89**、**-qlanglvl=extc99**、または **-qlanglvl=extended** のうち 1 つを指定してください。これらのフィーチャーをユーザーの C++ コードで使用するには、**-qlanglvl=extended** オプションを指定します。デフォルトでは、C++ のサポートされているすべての GNU C および C++ フィーチャーが受け入れられます。

以下の表では *accept/ignore* とマークされた拡張機能は、受け入れ可能なプログラミング・キーワードとしてコンパイラーに認識はされますが、GNU C/C++ セマンティクスはサポートされていません。これは、コンパイラーが *accept/ignore* キーワードまたは拡張機能を検出する場合、コンパイルは停止せず、GNU セマンティクスはアプリケーションでインプリメントされないということです。厳密な言語レベル (stdc89、stdc99) の下でこれらの拡張機能を使用するソース・コードをコンパイルすると、エラー・メッセージが出されます。

関連資料

GNU C および C++ 言語拡張機能は、<http://gcc.gnu.org/onlinedocs> の GNU マニュアルで完全に文書化されています。

関数属性

関数の宣言または定義を行う際に、特殊な属性を指定するには、キーワード `__attribute__` を使用します。このキーワードには、二重括弧内に属性仕様が続きます。XL C/C++ は、GNU C および C++ の関数属性のサブセットをサポートします。`accept/ignore` と記述された振る舞いの意味は、構文は受け入れられるものの、セマンティクスは無視され、コンパイルが続行するということです。

GNU C/C++ の XL C/C++ との関数属性の互換性

関数属性	動作
alias	supported
always_inline	supported
cdecl	accept/ignore
const	supported
constructor	accept/ignore
destructor	accept/ignore
dllexport	accept/ignore
dllimport	accept/ignore
eightbit_data	accept/ignore
exception	accept/ignore
format	supported
format_arg	supported
function_vector	accept/ignore
interrupt	accept/ignore
interrupt_handler	accept/ignore
longcall	accept/ignore
model	accept/ignore
no_check_memory_usage	accept/ignore
no_instrument_function	accept/ignore
noinline	supported
noreturn	supported
pure	supported
regparm	accept/ignore
section	accept/ignore
stdcall	accept/ignore
tiny_data	accept/ignore
weak	supported


関連資料

- 「XL C/C++ ランゲージ・リファレンス」の『関数属性』

変数属性

キーワード `__attribute__` を使用して、変数または構造化フィールドの特殊な属性を指定します。このキーワードには、二重括弧内に属性仕様が続きます。XL C/C++ は、GNU C および C++ の変数属性のサブセットをサポートします。*accept/ignore* と記述された振る舞いの意味は、構文は受け入れられるものの、セマンティクスは無視され、コンパイルが続行するということです。

GNU C/C++ の XL C/C++ との変数属性の互換性

変数属性	動作
aligned	supported
 init_priority	accept/ignore
mode	supported
model	accept/ignore
nocommon	accept/ignore
packed	supported
section	accept/ignore
transparent_union	accept/ignore
unused	accept/ignore
weak	accept/ignore


関連資料

- 「XL C/C++ ランゲージ・リファレンス」の『変数属性』

型属性

キーワード `__attribute__` を使用して、`struct` および `union` 型を定義する際に、それらの特殊な属性を指定します。このキーワードには、二重括弧内に属性仕様が続きます。XL C/C++ は、GNU C および C++ の型属性のサブセットをサポートします。*accept/ignore* と記述された振る舞いの意味は、構文は受け入れられるものの、セマンティクスは無視され、コンパイルが続行するということです。

GNU C/C++ の XL C/C++ との型属性の互換性

型属性	動作
aligned	supported
packed	supported
transparent_union	 supported
unused	accept/ignore

関連資料

- 「XL C/C++ ランゲージ・リファレンス」の『型属性』

GNU C および C++ アサーション

アサーション を使用して、コンパイル済みプログラムが実行するコンピューターまたはシステムの種類をテストします。アサーション `#cpu`、`#machine`、および `#system` が事前定義されています。また、プリプロセス・ディレクティブ `#assert`



と `#unassert` を使用してもアサーションを定義できます。

XL C/C++ での GNU C および C++ アサーション

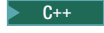
GNU C アサーション	動作
<code>#assert</code>	supported
<code>#unassert</code>	supported
<code>#cpu</code>	supported
<code>#machine</code>	supported
<code>#system</code>	supported 可能な値は <code>aix</code> と <code>unix</code>

GNU C および C++ 関連のその他の拡張子

GNU C および C++ に関連する以下のフィーチャーは、拡張言語レベル (`extc89`、`extc99`、`extended`) の下でサポートされます。

- ディレクティブ `#warning` を使用して、プリプロセッサに警告を出させ、処理を継続させます。
- ディレクティブ `#include_next` を使用して、ディレクトリーに現行のヘッダー・ファイルの後に次のヘッダー・ファイルを組み込むことを指定します。
- ローカル・ラベルは、それぞれの字句ブロックの開始点で宣言できます。
- 小括弧内で中括弧で囲まれた複合ステートメントを式として使用します。
- `__typeof__` キーワードで式のタイプを参照します。
- 複合式、条件式、および `lvalues` としてのキャストを使用します。
- 計算された `goto` 文を使用して、ラベルにジャンプします。そのアドレスは取り出され、アドレスは値として使用されます。
- キーワード `__alignof__` を使用して、変数位置合わせまたは、型ごとに通常必要な位置合わせについて照会します。
- これらのキーワードの代替スペルを使用します: `__asm__`、`__const__`、`__volatile__`、`__inline__`、`__signed__`、および `__typeof__`。
- 厳密な言語レベル・モードで拡張言語フィーチャーを使用する場合、`__extension__` キーワードを使用して、エラーを回避します。
- ゼロの長さの配列が、エラーなしで生成されることがあります。
-  他関数 (ネストされた関数) の定義内に現れる関数定義が許可されます。
-  共用体のメンバーは、属する共用体型にキャストされることがあります。

拡張言語レベル (`extc89`、`extc99`、`extended`) で、XL C/C++ は以下のフィーチャーの構文を認識しますが、それらのセマンティクスはサポートされていません。

-  レジスター変数の宣言は、グローバルであってもローカルであっても、優先されるレジスターを示すことができます。

付録 A. 言語サポート

この付録では、C および C++ プログラム言語のインプリメンテーションと、XL C/C++ によって提供される言語拡張機能について説明します。

ISO/IEC 国際規格との互換性

XL C/C++ を使用すると、移植性を重視したプログラミングを行うことができます。プログラム言語の完全な仕様は、構文とセマンティクスから構成されますが、インプリメンテーション時に言語拡張機能が使用されるため、特定言語仕様の準拠するインプリメンテーションはそれぞれ異なる可能性があります。

言語仕様に厳密に準拠するプログラムは、異なる環境間で最大の移植性を発揮します。理論上、標準に準拠した、あるコンパイラーで正しくコンパイルされ、拡張機能またはインプリメンテーション定義動作を使用しないプログラムは、ハードウェアの差異が許す範囲内で、他のすべての標準準拠のコンパイラーの下で適切にコンパイルされ、作動します。言語インプリメンテーションによって提供される言語への拡張機能を正しく活用したプログラムは、そのオブジェクト・コードの効率性を向上させることができます。

ISO/IEC 14882:2003(E) 国際規格の互換性


XL C/C++ は、ISO/IEC 国際規格 14882:2003(E) と整合しています。この規格は、書式を指定し、C++ プログラム言語で作成されたプログラムの解釈を確立するものです。この国際規格は、さまざまなインプリメンテーション間で、C++ プログラムの移植性を促進するために設計されています。ISO/IEC 14882:1998 は最初の C++ 言語でした。

ISO/IEC 9899:1990 国際規格の互換性

ISO/IEC 9899:1990 国際規格 (C89 として知られる) では、書式を指定し、C プログラミング言語で作成されたプログラムの解釈を確立しています。この規格は、さまざまなインプリメンテーション間で、C プログラムの移植性をプロモートするために設計されています。この規格は、ISO/IEC 9899/COR1:1994、ISO/IEC 9899/AMD1:1995、および ISO/IEC 9899/COR2:1996 で修正されました。修正および訂正された C89 規格をソース・コードが厳密に順守しているかを確認するには、コンパイラー・オプション `-qlanglvl=stdc89` を指定してください。

ISO/IEC 9899:1999 国際規格サポート

ISO/IEC 9899:1999 国際規格 (C99 としても知られる) は C プログラミング言語で作成されたプログラム用の、更新された規格です。これは、C 言語の機能を拡張し、C89 へ説明を提供し、技術修正を具体化するために設計されています。XL C/C++ は、この言語規格の多くのフィーチャーをサポートします。

 C コンパイラーは、C99 規格で指定されるすべての言語フィーチャーをサポートしています。この言語フィーチャーのセットをソース・コードが順守してい

るかを確認するには、**c99** 呼び出しコマンドを使用してください。また、この規格では、ランタイム・ライブラリーのフィーチャーも指定していることに注意してください。これらのフィーチャーは、現行のランタイム・ライブラリーおよびオペレーティング環境ではサポートされないことがあります。システム・ヘッダー・ファイルが使用可能かどうかにより、これらのフィーチャーがサポートされているかどうか分かります。

C99 での主なフィーチャー

XL C/C++ はすべての C99 言語フィーチャーをインプリメントしています。以下は、選択した主なフィーチャーの表です。関連資料は、「*XL C/C++ ランゲージ・リファレンス*」の項目を参照しています。

IBM C に対する ISO/IEC 9899:1999 国際規格の拡張

C99 フィーチャー	関連資料
ポインター用の restrict 型修飾子	restrict 型修飾子
汎用文字名	ユニコード規格
定義済み ID __func__	事前定義済み ID
変数と空引き数を持つ関数類似マクロ	関数類似マクロ
_Pragma 単項演算子	_Pragma 演算子
可変長配列	可変長配列
配列指標宣言内での static キーワード	配列
complex データ型	複素数型
long long int と unsigned long long int 型	整変数
16 進浮動小数点定数	16 進浮動小数点定数
集合体の複合リテラル	複合リテラル
指定された初期化指定子	初期化指定子
C++ 形式のコメント	コメント
許可されない暗黙の関数宣言	関数宣言
混合宣言とコード	for 文
_Bool 型	単純型指定子
inline 関数宣言	インライン関数
集合体の初期化指定子	指定された初期化指定子を使用した配列の初期化

C99 でサポートされる C89 の変更点と説明

C99 規格の一部の仕様は、言語の新しいフィーチャーに基づくものではなく、C89 規格を変更し、より明確にしたものです。XL C/C++ は、以下を含むすべての C99 言語フィーチャーをサポートしています。

- 配列メンバーを柔軟に使用できます。複数のメンバーを持つ構成の最後のメンバーは、サイズを指定せずに宣言できます。
- 暗黙の int 宣言はサポートされません。すべての宣言に型指定子が必要です。
- 列挙型指定子で、末尾のコンマが使用できます。

- 重複する型修飾子は、他に明示的な指定がなければ、受け取られて無視されます。
- `return` ステートメントから、必須の式が欠落している場合は、診断メッセージが出されます。
- プリプロセス中に評価されていた定数式は、`long long` と `unsigned long long` データ型を使用するようになりました。
- 空のマクロ引き数が関数類似マクロ内で使用できます。
- `#line` の最大値が 2 147 483 647 に増加しました。

XL C/C++ での C99 フィーチャー

ISO/IEC 9899:1999 国際規格 (C99) のいくつかのフィーチャーは、C++ でもインプリメントされます。これらの拡張は `-qlanglvl=extended` コンパイラー・オプションを指定すると有効になります。

IBM C++ に対する ISO/IEC 9899:1999 国際規格の拡張

C99 フィーチャー	参照
ポインター用の <code>restrict</code> 型修飾子	<code>restrict</code> 型修飾子
汎用文字名	ユニコード規格
定義済み ID <code>__func__</code>	事前定義済み ID
可変長配列	可変長配列
<code>complex</code> データ型	複素数型
16 進浮動小数点定数	16 進浮動小数点定数
集合体の複合リテラル	複合リテラル
変数と空引き数を持つ関数類似マクロ	関数類似マクロ
<code>_Pragma</code> 単項演算子	<code>_Pragma</code> 演算子

拡張言語レベル・サポート

`-qlanglvl` コンパイラー・オプションは、サポートしている言語レベルを指定するために使用され、そのためユーザーのコードがコンパイルされる方法に影響を与えます。別のコンパイラー呼び出しコマンドを使用して、暗黙的に言語レベルを指定することもできます。一般に、標準言語レベルの下で正しくコンパイルされ、実行される有効なプログラムは使用可能な直交拡張機能を使用して同じ結果を生成するために、正しくコンパイルし、実行されるはずで

例えば、ISO/IEC 9899:1990 国際規格 (C89) に厳密に準拠する方法で C プログラムをコンパイルするには、`-qlanglvl=stdc89` を指定する必要があります。`stdc89` サブオプションは、コンパイラーが規格に厳密に準拠し、どの言語拡張機能の使用も許可しないことを指示します。(**c89** コンパイラー呼び出しコマンドは、この言語レベルを暗黙的に指定します。)

標準言語レベルに拡張機能を使用することもできます。標準フィーチャーに干渉しない拡張機能は直交 拡張機能と呼ばれます。例えば、C プログラムをコンパイルする場合、`-qlanglvl=extc89` を指定して、C89 に直交する拡張機能を使用可能にできます。








ISO/IEC 9899:1999 国際規格 (C99) で説明されている言語フィーチャーのほとんどは、C89 への直交拡張機能と見なされます。

C++ プログラムをコンパイルする場合は、`-qlanglvl=extended` を指定して直交拡張機能を使用可能にできます。

一方では、直交以外の 拡張機能は、国際規格の 1 つに記載されているように、言語の観点から干渉され、競合することがあります。これらの拡張機能は、明示的に特定のコンパイラ・オプションで使用可能にしなければなりません。非直交拡張機能に依存すると、異なる環境へのアプリケーションの移植が容易にできなくなります。

`-qlanglvl` オプションの主なサブオプションは、以下にリストされています。

選択した `-qlanglvl` サブオプション

<code>-qlanglvl</code> サブオプション	サブオプションの説明
<code>-qlanglvl=stdc99</code>	 C99 標準への厳密な合致を指定します。
<code>-qlanglvl=stdc89</code>	 C89 標準への厳密な合致を指定します。
<code>-qlanglvl=ansi</code>	 C89 標準への厳密な合致を指定し、 <code>-qlonglong</code> コンパイラ・オプションを使用可能にします。
<code>-qlanglvl=extc99</code>	 C99 に直交するすべての拡張機能を使用可能にします。
<code>-qlanglvl=extc89</code>	 C89 に直交するすべての拡張機能を使用可能にします。
<code>-qlanglvl=extended</code>	 C89 に直交するすべての拡張機能を使用可能にし、 <code>-qpuconv</code> コンパイラ・オプションを指定します。  標準 C++ に加えてすべての直交拡張機能を使用可能にします。

付録 B. OpenMP 準拠とサポート

OpenMP アプリケーション・プログラム・インターフェース (API) は、移植可能で拡張が容易なプログラミング・モデルであり、マルチプラットフォームでメモリーを共用する並列アプリケーションを C、C++、および Fortran で開発するための標準インターフェースを提供します。OpenMP API の仕様は、OpenMP 組織という、IBM を含む主なコンピューター・ハードウェアおよびソフトウェア・ベンダーの集まりによって定義されます。

XL C/C++ は、OpenMP 仕様 2.0 に準拠しています。コンパイラーは、以下の OpenMP V2.0 エレメントのセマンティクスを認識し、保持します。

- `#pragma omp` ディレクティブ内の複数文節のコンマ区切り文字。
- `num_threads` 文節。
- `copyprivate` 文節。
- `threadprivate` 静的ブロック・スコープ変数。
- C99 可変長配列のサポート
- `private` 変数の冗長な宣言。
- タイミング・ルーチン `omp_get_wtick` および `omp_get_wtime`。

以下に説明するディレクティブ、ライブラリー関数、および環境変数を使用すると、移植性を維持しながら並列プログラムを作成し管理できます。

OpenMP 並列処理を使用できるようにするには、**-qsmp** コンパイラー・オプションを指定してください。

- 自動並列処理を選択するには、**-qsmp** または **-qsmp=auto** を指定します。このサブオプションを使用すると、コンパイラーはプログラム内のすべての OpenMP ディレクティブ、ライブラリー関数、および環境変数を認識しインプリメントすることに加え、*implicit parallelism* を実行できます。
- OpenMP 仕様 2.0 への厳格な準拠を選択する場合は、**-qsmp=omp** を指定します。このサブオプションを使用すると、コンパイラーはコードに指定された OpenMP ディレクティブ、ライブラリー関数、および環境変数のみをインプリメントします。他の自動化された並列処理は行いません。

関連資料

- <http://www.openmp.org>
- 「XL C/C++ コンパイラー・リファレンス」の『並列処理を制御するプラグマ』
- 「XL C/C++ コンパイラー・リファレンス」の『プログラムの並列化』

OpenMP ディレクティブ

それぞれのディレクティブは `#pragma omp` で始まり、他のプラグマ・ディレクティブとの潜在的な競合を削減します。

XL C/C++ での OpenMP ディレクティブ

ディレクティブ名	ディレクティブ説明
<code>parallel</code>	<code>parallel</code> ディレクティブは、並列領域を定義します。これはマルチスレッドによって並列して実行されるプログラムの領域です。
<code>for</code>	<code>for</code> ディレクティブは、反復作業共用構成を識別します。これは、関連するループの反復が並列して実行される必要のある領域を指定します。 <code>for</code> ループの反復は既存のスレッド間で分散されます。
<code>sections</code>	<code>sections</code> ディレクティブは反復しない作業共用構成を識別します。これはチーム内のスレッド間で分割される構成のセットを指定します。それぞれのセクションはチーム内のスレッドで 1 度実行されます。
<code>single</code>	<code>single</code> ディレクティブは構成を識別します。これは関連する構造化ブロックがチーム内のただ 1 つのスレッドでのみ実行されることを指定します (マスター・スレッドである必要はありません)。
<code>parallel for</code>	<code>parallel for</code> ディレクティブは、単一 <code>for</code> ディレクティブを含む並列領域用のショートカット形式です。セマンティクスは、 <code>for</code> ディレクティブのすぐ前に <code>parallel</code> ディレクティブを明示的に指定することと同じです。
<code>parallel sections</code>	<code>parallel sections</code> ディレクティブは、単一 <code>sections</code> ディレクティブを含んでいる並列セクションを指定するためのショートカット形式を提供します。セマンティクスは、 <code>sections</code> ディレクティブのすぐ前に <code>parallel</code> ディレクティブを明示的に指定することと同じです。
<code>master</code>	<code>master</code> ディレクティブは、チームのマスター・スレッドによって実行される構造化ブロックを指定する構成を識別します。
<code>critical</code>	<code>critical</code> ディレクティブは、関連した構造化ブロックの実行を行えるのは一時点では 1 つのスレッドのみに限定する構成を識別します。クリティカル領域を識別するために、オプションの <code>名前</code> を使用することもできます。スレッドは、同じ <code>名前</code> で他のスレッドがクリティカル領域を実行しなくなるまで、クリティカル領域の開始で待ちます。すべての <code>名前</code> なし <code>critical</code> ディレクティブは、同じ未指定の <code>名前</code> にマップします。
<code>barrier</code>	<code>barrier</code> ディレクティブは、チーム内のすべてのスレッドを同期化します。スレッドがこのディレクティブに遭遇すると、それぞれのスレッドは他のスレッドがこのポイントに達するまで待ちます。すべてのスレッドがバリアに遭遇したら、各スレッドは <code>barrier</code> ディレクティブ後でステートメントの実行を並列に開始します。

XL C/C++ での OpenMP ディレクティブ

ディレクティブ名	ディレクティブ説明
atomic	atomic ディレクティブは、自動的に更新されなければならない、複数の同時書き込みスレッドに公開されない特定のメモリ・ロケーションを識別します。
flush	flush ディレクティブは、コンパイラーによって並列領域のすべてのスレッドがメモリ内の指定されたオブジェクトの同じビューを持つようにするポイントを識別します。
ordered	ordered ディレクティブは、順序付けられた順で実行しなければならない構造化されたブロックのコードを識別します。
threadprivate	threadprivate ディレクティブは、スレッドに private になるファイル・スコープ、名前スペース・スコープ、または静的ブロック・スコープ変数を宣言します。

C C コンパイラーは、プログラム並列処理で使用される以下の追加ディレクティブを認識します。これらは、OpenMP 仕様 1.0 および 2.0 の一部ではなく、C++ コンパイラーでは認識できません。

- #pragma ibm critical
- #pragma ibm independent_calls
- #pragma ibm independent_loop
- #pragma ibm iterations
- #pragma ibm parallel_loop
- #pragma ibm permutation
- #pragma ibm schedule
- #pragma ibm sequential_loop

OpenMP データ・スコープ属性文節

文節は、並列または作業共用構成の間に変数のスコープ属性を制御するために、ディレクティブで指定できます。

XL C/C++ での OpenMP データ・スコープ属性文節

データ・スコープ属性文節名	データ・スコープ属性文節記述
private	private 文節は、リスト内の変数がチーム内のそれぞれのスレッドに対して private であることを宣言します。
firstprivate	firstprivate 文節は、private 文節によって提供される機能のスーパーセットを提供します。
lastprivate	lastprivate 文節は、private 文節によって提供される機能のスーパーセットを提供します。
copyprivate	copyprivate 文節は、チームに値をブロードキャストする共用変数を使用する代替手段を提供します。このメカニズムでは、private 変数を使用して、あるチーム・メンバーから他のメンバーに値をブロードキャストします。
num_threads	num_threads 文節は、並列構成でスレッドの特定の数を要求する機能を提供します。

XL C/C++ での OpenMP データ・スコープ属性文節

データ・スコープ属性文節名	データ・スコープ属性文節記述
shared	shared 文節は、チーム内のすべてのスレッド間でリストに表示される変数を共有します。チーム内のすべてのスレッドは、shared 変数用に同じストレージ域を使用できます。
reduction	reduction 文節は、指定された演算子で、リスト内に表示されたスカラー変数の縮小を実行します。
default	default 文節は、変数のデータ・スコープ属性をユーザーが操作できるようにします。

OpenMP ライブラリー関数

OpenMP ランタイム・ライブラリー関数は、ヘッダー<omp.h> に組み込まれます。これらは、並列実行環境を制御および照会するために使用できる実行環境関数、およびデータへのアクセスを同期化するために使用できるロック機能を含んでいます。

XL C/C++ での OpenMP ランタイム・ライブラリー関数

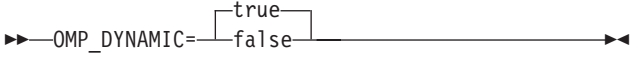
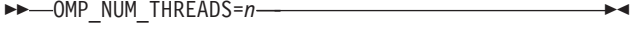
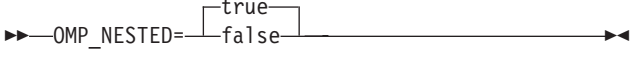
ランタイム・ライブラリー関数名	ランタイム・ライブラリー関数の説明
omp_set_num_threads	これに続く並列領域で使用するために、スレッド数を設定します。
omp_get_num_threads	呼び出される並列領域を実行しているチーム内で現在実行しているスレッド数を返します。
omp_get_max_threads	omp_get_num_threads への呼び出しによって返すことができる最大値を返します。
omp_get_thread_num	チーム内の関数を実行しているスレッドのスレッド番号を返します。チームのマスター・スレッドはスレッド 0 です。
omp_get_num_procs	プログラムに割り当てることができるプロセッサの最大数を返します。
omp_in_parallel	並列で実行している並列領域の動的エクステンション内で呼び出された場合に、ゼロ以外を返します。それ以外の場合は 0 を返します。
omp_set_dynamic	並列領域の実行で使用可能なスレッド数の動的調整を使用可能または使用不可にします。
omp_get_dynamic	動的スレッド調整が使用可能な場合にゼロ以外を返し、それ以外の場合は 0 を返します。
omp_set_nested	ネストされた並列性を使用可能または使用不可にします。
omp_get_nested	ネストされた並列性が使用可能な場合にゼロ以外を返し、使用不可の場合に 0 を返します。
omp_init_lock	単純ロックを初期設定します。
omp_destroy_lock	単純ロックを除去します。

ランタイム・ライブラリー関数名	ランタイム・ライブラリー関数の説明
omp_set_lock	単純ロックが使用可能になるのを待ちます。
omp_unset_lock	単純ロックをリリースします。
omp_test_lock	単純ロックをテストします。
omp_init_nest_lock	ネストできるロックを初期設定します。
omp_destroy_nest_lock	ネストできるロックを除去します。
omp_set_nest_lock	ネストできるロックが使用可能になるのを待ちます。
omp_unset_nest_lock	ネストできるロックをリリースします。
omp_test_nest_lock	ネストできるロックをテストします。
omp_get_wtick	連続する時計の 1 刻み間の秒数を戻します。
omp_get_wtime	経過した時刻を秒単位で戻します。

OpenMP 環境変数

OpenMP 環境変数は並列コードの実行を制御します。環境変数の名前は、常に大文字でなければなりませんが、それらの値は大/小文字の区別はありません。

説明	構文
<p>OMP_SCHEDULE</p> <p>ランタイム・スケジューリング・タイプとチャンク・サイズを設定します。 <code>runtime</code> にスケジューリング・タイプ・セットを持つ OpenMP ディレクティブにのみ適用します。</p>	<div data-bbox="800 247 1421 420"> <pre> OMP_SCHEDULE= [static, -n affinity dynamic, -n guided runtime] </pre> </div> <p>ここで、</p> <p>親和性</p> <p>C にのみ有効な IBM 拡張。最初にループの反復がスレッド数によって分割された反復数の上限に等しいサイズのローカル区画に分割されることを指定します</p> <p>$\text{CEILING}(\text{number_of_iterations} \div \text{number_of_threads})$。各ローカル区画はさらにローカル区画に残る反復数の半分の上限に等しいサイズのチャンクに細分されます</p> <p>$\text{CEILING}(\text{iterations_left_in_local_partition} \div 2)$。スレッドがフリーになると、そのローカル区画から次のチャンクを取ります。ローカル区画にチャンクがなくなると、スレッドは他のスレッドの区画から使用可能なチャンクを取ります。</p> <p>n が指定されると、各ローカル区画はサイズ n のチャンクに細分されます。 n が指定されない場合は、デフォルト値は 1 です。</p> <p>動的</p> <p>for ループの反復を一連の n サイズのチャンクに分ける必要があり、チャンクは次のプロセスに沿って処理されることを指定します。割り当てを待つスレッドに反復のチャンクが割り当てられると、スレッドはそのチャンクを実行し、次の割り当てを待ちます。このプロセスはすべてのチャンクが割り当てられるまで繰り返されます。 n が指定されない場合は、デフォルトのチャンク・サイズは 1 です。</p> <p>ガイド</p> <p>for ループの反復がサイズが減少しているチャンクのスレッドに割り当てられる必要があり、チャンクは次のプロセスに沿って処理されることを指定します。割り当てられた反復のチャンクを終了するスレッドには、動的に他のチャンクが割り当てられます。これは、すべてのチャンクが割り当てられるまで続きます。 n が指定されない場合は、最初のチャンク・サイズのデフォルト値は 1 です。</p> <p>静的</p> <p>for ループの反復を一連の n サイズのチャンクに分割する必要があり、チャンクは次のプロセスに沿って処理されることを指定します。使用可能なスレッドは、スレッド数によって決定された順序でチャンクを割り当てます。 n が指定されない場合、反復スペースはサイズがほぼ等しいチャンクに分割され、1 つのチャンクが各スレッドに割り当てられます。</p> <p>n 正数で、チャンク・サイズを示します。</p>

説明	構文
OMP_DYNAMIC 並列領域の実行で使用可能なスレッド数の動的調整を使用可能または使用不可にします。	 <p>ここで、</p> <p>true 使用可能なスレッド数の動的調整を使用可能にします。</p> <p>false 使用可能なスレッド数の動的調整を使用不可にします。</p>
OMP_NUM_THREADS 実行可能なスレッド数のセット。	 <p>ここで、</p> <p><i>n</i> スレッド数を示します。</p>
OMP_NESTED ネストされた並列性を使用可能または使用不可にします。	 <p>ここで、</p> <p>true ネストされた並列性を使用可能にします。</p> <p>false ネストされた並列性を使用不可にします。</p>

OpenMP インプリメンテーション定義動作

以下の情報は標準では指定されていません。標準のそれぞれのインプリメンテーションは、独自のインプリメンテーション定義値があります。

条件付きコンパイル

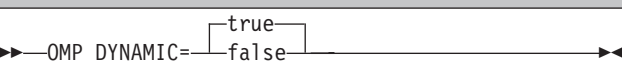

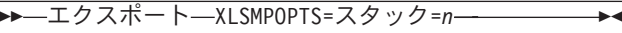
`_OPENMP` マクロは 199810 に定義されました。

スケジューリング

`schedule` 文節は、**for** ループの反復がどのようにチーム内のスレッド間で分割されるかを指定します。可能な OpenMP 標準値は、`static`、`dynamic`、`guided`、および `runtime` です。さらに、IBM C は値 `affinity` を拡張子として追加します。明示的に定義された `schedule` 文節がない場合は、XL C/C++ のデフォルトのスケジューリングは `static` です。

OpenMP プログラムの調整

`pthread` ライフ・サイクルでさまざまなタイミングを調整している環境変数に加え、次の環境変数は OpenMP アプリケーションの調整に役立ちます。

説明	構文
OMP_DYNAMIC 並列領域の実行で使用可能なスレッド数の動的調整を使用可能または使用不可にします。変数が使用可能な場合 (デフォルト設定)、ランタイム環境は並列領域に使用するスレッド数を調整できるため、システム・リソースを最も効果的に使用することができます。ベンチマーク、スケーリング・テストの場合、またはアプリケーションが特定のスレッド数に依存している場合は、動的検査によって少量のオーバーヘッドを追加できるため、変数を使用不可にする必要があります。	 <p>ここで、</p> <p>true 使用可能なスレッド数の動的調整を使用可能にします。</p> <p>false 使用可能なスレッド数の動的調整を使用不可にします。</p>
MALLOCMULTIHEAP 複数のヒープは便利で、スレッド化アプリケーションはメモリー割り振りサブルーチン呼び出しを発行するスレッドを複数持つことができます。単一のヒープでは、 <code>malloc()</code> 、 <code>free()</code> 、または <code>realloc()</code> 呼び出しを行おうとするすべてのスレッドが直列化されます (つまり、1 つのスレッドのみがこれら 3 つの機能を同時に行うことができます)。このような状態はマルチプロセッサ・マシンに重大な影響を与えることがあります。複数のヒープでは、各スレッドがそれ自身のヒープ (それぞれ最大 32 ヒープ) を取得します。この環境変数をエクスポートすることによって、デフォルトで設定されていない、複数のヒープを使用可能にします。	
XLSPMPOPTS 32 ビットの OpenMP アプリケーションでは、スレッド毎のスタック・サイズのデフォルト限度はかなり小さく、それを超えると、ランタイム・エラーになります。これが起こる場合は、スタック・サブオプションで XLSPMPOPTS 環境変数を設定することによってスタック・サイズを増やすことができます。	 <p>ここで、</p> <p>n スタック・サイズをバイトで示します。すべてのスレッドのスタック・サイズの合計は 256 MB を超えることができません (1 つのメモリー・セグメント)。1 つのセグメントの制限は 64 ビット・アプリケーションには適用されません。デフォルト値はスレッド毎に 4 MB です。</p>

関連資料

- 61 ページの『マルチスレッド・プログラムの調整』

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あり、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を提供し、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。

Lab Director
IBM Canada Ltd. Laboratory
B3/KB7/8200/MKM
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せんが、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 以外の製品に関する情報は、その製品の供給者、出版物、もしくはその他の公に利用可能なソースから入手したものです。IBM は、それらの製品のテストは行っておりません。したがって、他社製品に関する実行性、互換性、またはその他の要求については確証できません。IBM 以外の製品の性能に関する質問は、それらの製品の供給者をお願いします。

本書には、日常の業務処理で用いられるデータや報告書の例が含まれています。より具体性を与えるために、それらの例には、個人、企業、ブランド、あるいは製品などの名前が含まれている場合があります。これらの名称はすべて架空のものであり、名称や住所が類似する企業が実在しているとしても、それは偶然にすぎません。

著作権使用許諾:

本書には、様々なオペレーティング・プラットフォームでのプログラミング手法を例示するサンプル・アプリケーション・プログラムがソース言語で掲載されています。お客様は、サンプル・プログラムが書かれているオペレーティング・プラットフォームのアプリケーション・プログラミング・インターフェースに準拠したアプリケーション・プログラムの開発、使用、販売、配布を目的として、いかなる形式においても、IBM に対価を支払うことなくこれを複製し、改変し、配布することができます。このサンプル・プログラムは、あらゆる条件下における完全なテストを経ていません。従って IBM は、これらのサンプル・プログラムについて信頼性、利便性もしくは機能性があることをほのめかしたり、保証することはできません。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。

それぞれの複製物、サンプル・プログラムのいかなる部分、またはすべての派生的創作物にも、次のように、著作権表示を入れていただく必要があります。

© (お客様の会社名) (西暦年). このコードの一部は、IBM Corp. のサンプル・プログラムから取られています。 © Copyright IBM Corp. 1998, 2004 年. All rights reserved.

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。

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。

警告: 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。

AIX	POWER3	pSeries
@server	POWER4	Redbooks
IBM	POWER5	RS/6000
	PowerPC	VisualAge
	PowerPC Architecture	

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。

他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。

業界標準

以下の規格がサポートされます。

- C 言語は、International Standard C (ANSI/ISO-IEC 9899-1990 [1992]) に準拠しています。この規格は、American National Standard for Information Systems-Programming Language C (X3.159-1989) を正式に置き換えたもので、ANSI C 規格と技術的に同等です。コンパイラーは、ISO/IEC 9899:1990/Amendment 1:1994 によって C 標準に採用された変更をサポートしています。
- C 言語は、International Standard for Information Systems-Programming Language C (ISO/IEC 9899-1999 (E)) に準拠しています。
- C++ 言語は、言語の最初の公式定義である、International Standard for Information Systems-Programming Language C++ (ISO/IEC 14882:1998) に準拠しています。
- C++ 言語は、現在 *Standard C++* とも呼ばれている、International Standard for Information Systems-Programming Language C++ (ISO/IEC 14882:2002 (E)) にも準拠しています。
- C および C++ コンパイラーは、OpenMP C and C++ Application Programming Interface Version 2.0 をサポートしています。