インテル® Fortran コンパイラ Linux* 版 インストールと入門

エクセルソフト株式会社

www.xlsoft.com

ご注意

このガイドはインテル® Fortran コンパイラ 8.1 Linux 版の日本語オンライン・ヘルプから抜粋したものです。**各項目の詳細**や**最適化方法**の詳細は、**日本語オンライン・ヘルプ**をご参照ください。日本語オンライン・ヘルプは次のサイトより無償でダウンロードいただけます。

http://www.xlsoft.com/jp/products/intel/download.html

Distributed by

エクセルソフト株式会社

〒108-0014 東京都港区芝 5-1-9 ブゼンヤビル 4F

目次

はじめに

この入門ガイドはインテル® Fortran コンパイラ 8.1 Linux 版の日本語オンライン・ヘルプから抜粋したものです。各項目の詳細や最適化方法の詳細は、日本語オンライン・ヘルプをご参照ください。日本語オンライン・ヘルプは次のサイトより無償でダウンロードいただけます。

http://www.xlsoft.com/jp/products/intel/download.html

インテル® Fortran コンパイラについて

インテル® Fortran コンパイラは、IA-32 アーキテクチャ・システム向け、およびインテル® Itanium® アーキテクチャ・システム向けにコードをコンパイルします。

インテル Fortran コンパイラ製品には、開発環境用に次のコンポーネントが含まれています。

- 32 ビット・アプリケーション用のインテル Fortran コンパイラ
- Itanium ベース・アプリケーション用のインテル Fortran コンパイラ
- インテル® デバッガ (IDB)

Itanium ベース・アプリケーション用インテル Fortran コンパイラには、インテル Itanium アセンブラとインテル Itanium プロセッサ・リンカが含まれています。

インテル® エクステンデッド・メモリ 64 テクノロジ (インテル® EM64T) 対応 IA-32 システムについての詳細は、「Linux* 版インテル® Fortran コンパイラ 8.1 エクステンデッド・メモリ 64 テクノロジ・リリースノート」を参照してください。

本リリースの新機能

バージョン 8.1 の新機能と変更点

本リリースには、新しい定義済みプリプロセッサ・シンボル __INTEL_COMPILER_BUILD_DATE や新しいコンパイル・オプションを含む、いくつかの新機能と変更点が含まれています。本リリースの新規オプションをすべて記載したリストは、『インテル® Fortran コンパイラ・オプション・クイック・リファレンス・ガイド』の「新規のコンパイラ・オプション」を参照してください。

本リリースでは、インテル Fortran コンパイラ 8.0 でコンパイルされたオブジェクトおよび .mod ファイルを再コンパイルする必要はありません。

> **注:** 本リリースに実装された機能の最新情報は、リリースノートを参照してください。

バージョン 8.0 の新機能と変更点

バージョン 8.0 には、以下の新機能と変更点が含まれています。

- Compaq Visual Fortran と互換性のある、より多くのコマンドライン・コンパイラ・オプション

- 派生型拡張子の Fortran 2000 割り付けコンポーネントのサポート
- ランタイム・エラーとソースを関連付けるトレースバック・オプション
- 以前のアプリケーション実行プロファイルを基に、最も効果的なアプリケーション・テストを選択し、その優先度を示すテスト・プライオリタイゼーション・ツール
- 特定のワークロードにおけるアプリケーションのコード適用範囲情報をビジュアルに表示するコード・カバレッジ・ツール
- Itanium® ベース・システム用インテル® デバッガのサポート

注: バージョン 8.0 で実装された機能に関する最新情報は、リリースノートを参照してください。

1. インストール

システム要件

IA-32 プロセッサの動作環境

- インテル Pentium プロセッサまたは後継の IA-32 プロセッサを搭載したシステム (インテル Pentium 4 プロセッサ推奨)
- RAM 128MB (256MB 推奨)
- 100MB のディスク空き容量 (インストール時、ファイルのダウンロードおよびテンポラリ・ファイル用に別途 200MB の空き容量が必要)
- glibc 2.2.4、2.2.5、2.2.93、2.3.2 のいずれかとカーネル 2.4.X または 2.6.X の Linux システム。代表的なディストリビューションは下記のとおり。**注:** 下記のすべてのディストリビューションについて動作検証を行ったわけではありません。また、下記以外のディストリビューションでもご利用いただける場合があります。
    - Red Hat* Linux 7.3、8, 9
    - Red Hat Enterprise Linux* 2.1、3
    - SUSE* Linux 8.2、9.1
    - SUSE Linux Enterprise Server* 8 または 9

- gcc、g++ および関連ツールを含む、Linux 開発ツール・コンポーネント

Itanium プロセッサの動作環境

- Itanium プロセッサまたは Itanium 2 プロセッサを搭載したコンピュータ
- RAM 512MB (1GB 推奨)
- 150MB のディスク空き容量 (インストール時、ファイルのダウンロードおよびテンポラリ・ファイル用に別途 200MB の空き容量が必要)
- glibc 2.2.4、2.2.5、2.3.2 のいずれかとカーネル 2.4.X または 2.6.X の Linux システム。代表的なディストリビューションは下記のとおり。
  注: 下記のすべてのディストリビューションについて動作検証を行ったわけではありません。また、下記以外のディストリビューションでもご利用いただける場合があります。
    - Red Hat Linux 7.2
    - Red Hat Enterprise Linux AS 2.1、AS 3、WS 3
    - SUSE Linux Professional* 9.1

  - SUSE Linux Enterprise Server 8、9
  - United Linux* 1.0

- gcc、g++ および関連ツールを含む、Linux 開発ツール・コンポーネント

共有ライブラリを使用する際は binutils 2.14 以降を使用してください (binutils 2.11 には問題があることが報告されています)。

注: 数千行にも及ぶ非常にサイズの大きいソース・ファイルを、高度な最適化オプション (`-O3`、`-ipo`、`-openmp` など) をつけてコンパイルする際は、さらに大容量の RAM が必要となります。

インストール

以前のバージョンのコンパイラを使用している場合は、インテル Fortran コンパイラ 8.1 をインストールする前に、以前のコンパイラをアンインストールする必要があります。アンインストールに関しては、「コンパイラとデバッガのアンインストール」を参照してください。

インテル Fortran コンパイラのインストール・スクリプトは、システム・ユーティリティ RPM を使用してファイルをインストールします。RPM 4.0.2 および RPM 4.1 の両方に制限があるので注意してください。詳細はユーザーズ・ガイドの「既知の制限事項」を参照してください。

ライセンスのインストール

インテル Fortran コンパイラは、Macrovision 社の FLEXlm* 電子ライセンス・テクノロジを使用しています。ライセンスの管理は透過的に行われます。インテル Fortran コンパイラ 8.1 のインストール・プログラムは、製品コンポーネントをインストールする前に有効なライセンスをチェックします。また、プログラムのコンパイル時とビルド時にもライセンス・ファイルのチェックを行います。

インテル・ソフトウェア用の FLEXlm ライセンス・デーモンは、フローティング・ライセンスおよびノードロック・ライセンスで使用され、多くの一般的なプラットフォームで利用できます。ライセンス・デーモンは、ローカル・ネットワーク上のアクセス可能な任意のサポートされているプラットフォーム上にインストールされます。コンパイラ CD には、さまざまな Linux ディストリビューション用のライセンス・デーモンが含まれています。CD をお持ちでない場合、または別のプラットフォーム用のライセンス・デーモンが必要な場合は、インテル・プレミア・サポート (英語) の *[Downloards]* セクションからライセンス・デーモンをダウンロードしてください。

注: サポート・サービスの有効期間内であれば、インテル Fortran コンパイラ Linux 版の既存のライセンスをバージョン 8.1 でもそのまま使用できます。

インストール前に、以下の手順に従ってライセンス・ファイルをセットアップしてください。

- インテル Fortran コンパイラ 8.1 のダウンロード版をご利用の場合は、ライセンスは電子メールで送信されます。ライセンス・ファイルのインストール方法については、電子メールに記載の指示に従ってください。
- インテル Fortran コンパイラ 8.1 の CD-ROM 版パッケージをご利用の場合は、有効なライセンスは CD-ROM に含まれており、インストール・プログラムがこれを自動的に検出します。

しかし、テクニカル・サポートを受けたり、製品アップデートのダウンロードおよびインストールを行うには、**CD-ROM 版のユーザ**のみ以下の手順が必要となります:

1. **製品登録を行う:** 製品外箱の折り蓋に記載されているシリアル番号を確認してください。次に、http://www.intel.com/software/products/registrationcenter/ (英語) にアクセスし、指示に従ってください。登録が完了すると、24 時間以内に電子メールで新しいライセンスが送られます。
2. **新しいライセンスをインストールする:** 電子メールで送られた新しいライセンスは、1 年間のサポートサービスを保証するもので、この期間内に製品アップデートのダウンロードとインストール、およびフルのテクニカル・サポートを受けることができます。また、この電子メールにはライセンスのインストール方法についても記載されています。指示に従って新しいライセンスのインストールを完了してください。

注: ライセンス・ファイルの拡張子は、"`.lic`" です。
デフォルトのライセンス・ディレクトリは `/opt/intel_fc_80/licenses/` です。

サポート・サービス・ライセンスの詳細は、
http://www.intel.com/software/products/compilers/flin/pricelist.htm (英語) を参照してください。

インテル Fortran コンパイラのインストール

次の手順に従ってコンパイラをインストールします。

1. コンパイラ・パッケージをダウンロードします。
2. 書き込み権限があるディレクトリでパッケージを解凍 (untar) します。
   ```
   > tar -xvf l_fc_p[c]_8.1.xxx.tar
   ```
   または
   ```
   > tar -zxvf l_fc_p[c]_8.1.xxx.tar.gz
   ```
3. インストール・スクリプトを実行します。
   rpm コマンドを実行するために、**root ユーザ** でログインします。そして、解凍されたファイルが抽出されたディレクトリでインストール・スクリプトを実行します。
   ```
   > source ./install.sh
   ```
   `root` アカウントでログインできない場合は、`rpm2cpio` を使用して RPM ファイルを手動で解凍し、`ifortvars.sh (.csh)` ファイルを編集してコンパイラのインストール先ディレクトリを含めると、root 権限なしでもコンパイラをインストールできます。インストール・スクリプトでは、この手順を自動的に行います。
4. ライセンス・ファイルのディレクトリを入力します。
   これは上記でライセンス・ファイル (`*.lic`) を保存したディレクトリです。インストール・プログラムは、Linux コンポーネントのインテル Fortran コンパイラをインストールする前に、有効なライセンスをチェックします。
5. ライセンスのチェック終了後、インストール・プログラムは既にインストールされているインテルのソフトウェア製品と次のインストール・メニュー項目を表示します。

   - 32 ビット・アプリケーション用インテル・コンパイラ (IA-32 システムにインストールする場合) または
     Itanium アーキテクチャ用インテル・コンパイラ (Itanium ベース・システムにインストールする場合)

- 32 ビット・アプリケーション用 `Linux Application Debugger` または Itanium ベース・アプリケーション用 `Linux Application Debugger`

6. インストールするパッケージを選択します。
   推奨するインストールの順序は次のとおりです。

   - 最初にインテル Fortran コンパイラをインストールします。
   - 次に、Linux Application Debugger (インテル・デバッガ、idb) をインストールします。

   インストールを開始する前に、エンドユーザ・ライセンス契約 (EULA) が表示されます。`"accept"` と入力してライセンス契約に同意します。
   すべての製品を使用するために必要なパッケージがインストールされます。RPM パッケージが既にインストールされている場合、インストール・スクリプトはこれを報告し、既存のインストールを上書きするかどうか確認します。その後、次の RPM パッケージのインストールを続行します。既存のファイルを更新するには、デフォルトの RPM オプション `-U --replacefiles --force` を使用することを推奨します。
   デフォルトのインストール・ディレクトリは、インテル Fortran コンパイラでは `/opt/intel_fc_80/`、インテル・デバッガでは `/opt/intel_idb_80/` です。

7. インストールが完了すると、インストールされたインテル・パッケージ、それからインストール・メニューが再度表示されます。`'x'` を入力してインストール・スクリプトを終了します。

環境変数

インテル Fortran コンパイラは、定義されている場合、次の環境変数をサポートします:

- `FPATH` - `INCLUDE` および `USE` (コンパイル済み `.mod`) ファイルを検索する追加ディレクトリを指定します。ドライバは、これらを `-I` オプションに変換します。
- `LIBRARY_PATH` - リンカが検索する追加ディレクトリを指定します。ドライバは、これらを `-L` オプションに変換します。

コンパイラ環境の設定

インストール・スクリプト (`install.sh`) は、`PATH`、`LD_LIBRARY_PATH`、および `MANPATH` のような環境変数を設定するコンパイラ環境スクリプト・ファイル (`ifortvars.sh/idbvars.sh`) を作成します。これらのスクリプト・ファイルをログイン・スクリプト (`.login` ファイル) に追加することを強く推奨します。`".login"` ファイルで一旦、変数が設定されれば、セッションごとにスクリプト・ファイルを実行する必要がなくなります。

コンパイラ環境を設定するスクリプトの実行

- `source <インストール先ディレクトリ>/bin/ifortvars.sh(.csh)`
  (IA32 システムまたは Itanium ベース・システムで ifort を使用する場合)
- `source <インストール先ディレクトリ>/bin/idbvars.sh(.csh)`
  (IA32 システムまたは Itanium ベース・システムで idb を使用する場合)

また、インストール・プログラムにより、すべてのコンパイル処理に共通の設定を含むコンパイラ設定ファイル (**<インストール先ディレクトリ>**/bin/ifort.cfg) が作成されます。これらのファイルを編集して、デフォルトのオプションを追加することができます。**注:** コンパイラのアップデート・パッケージをインストールする場合は、変更した設定ファイルを別のファイル名で保存し、ファイルが上書きされないようにする必要があります。

インテル Fortran とインテル C++ の両方のコンパイラを使用する場合は、インテル C++ の環境変数スクリプト iccvars.sh も実行する必要があります。

製品のインストールが完了したら、サポートへ登録してください。登録方法については、ユーザーズ・ガイドの「テクニカル・サポート」を参照してください。

コンパイラとデバッガのアンインストール

下記の手順に従ってインテル・コンパイラとインテル・デバッガをアンインストールします。

1. root ユーザ権限でログインします。
2. コンパイラをアンインストールするには次のように指定します。
   `> source <コンパイラ・インストール先ディレクトリ>/bin/uninstall.sh`
   コンパイラをデフォルトのディレクトリにインストールした場合は、次のようになります。
   `> source /opt/intel_fc_80/bin/uninstall.sh`
3. デバッガをアンインストールするには次のように指定します。
   `> source <デバッガ・インストール先ディレクトリ>/bin/uninstall.sh`
   デバッガをデフォルトのディレクトリにインストールした場合は、次のようになります。
   `> source /opt/intel_idb_80/bin/uninstall.sh`

2. コンパイラの起動

インテル® Fortran コンパイラの使い方

インテル® Fortran コンパイラには、次のような種類があります。

- 32 ビット・アプリケーション用インテル Fortran コンパイラは IA-32 システム用です。IA-32 のコンパイルは、任意の IA-32 インテル® プロセッサ上で実行でき、IA-32 システム上で動作するアプリケーションを生成します。このコンパイラは、インテル® Pentium® M プロセッサ、Pentium 4 プロセッサ、インテル® Xeon™プロセッサなど、1 種類以上の IA-32 プロセッサ向けに最適です。
- Itanium® アーキテクチャ用インテル Fortran コンパイラ (またはネイティブ・コンパイラ) は、Itanium アーキテクチャ・システム用です。このコンパイラは Itanium ベースのシステムで動作し、Itanium ベース・アプリケーションが生成されます。Itanium アーキテクチャのコンパイラは Itanium ベースのシステムでしか実行できません。

これらのコンパイラを起動させるコマンドは `ifort` です。

> **注:** インテル® エクステンデッド・メモリ 64 テクノロジ (インテル® EM64T) 対応 IA-32 システムについての詳細は、「Linux* 版インテル® Fortran コンパイラ 8.1 エクステンデッド・メモリ 64 テクノロジ・リリースノート」を参照してください。

インテル Fortran コンパイラにはさまざまなオプションがあり、それによって、アプリケーションのパフォーマンスを向上させるコンパイラ機能を使用できます。

インテル Fortran コンパイラを使用すると、インテル® アーキテクチャ・ベースのコンピュータ上でソフトウェアの最高のパフォーマンスを引き出せます。このコンパイラにより、さまざまなハイ・パフォーマンスの最適化を行うことができます。以下の表は、それらの特徴と利点の一覧です。

特徴	説明
ストリーミング SIMD 拡張命令 (SSE)、ストリーミング SIMD 拡張命令 2 (SSE2)、およびストリーミング SIMD 拡張命令 3 (SSE3) のサポート	インテル® マイクロアーキテクチャの利点
自動ベクトライザ	コード内の自動並列処理
並列化	ループのマルチスレッド・コードの自動生成。OpenMP*での共用メモリ並列プログラミング
浮動小数点の最適化	浮動小数点のパフォーマンスが向上
データ・プリフェッチ機能	データ送信の高速化によりパフォーマンスが向上
プロシージャ間の最適化	大きなアプリケーションに対するパフォーマンスが向上
プログラム全体の最適化	大きなアプリケーション内のモジュール間のパフォーマンスが向上

プロファイルに基づく最適化	頻繁に使用されるプロシージャのプロファイリングに基づくパフォーマンスの向上
プロセッサ・ディスパッチ	最新のインテル・アーキテクチャの機能を利用すると同時に、前世代の インテル Pentium プロセッサとのオブジェクト・コードの互換性を確保

コンパイル・フェーズ

コンパイラは、Fortran 言語のソースを処理し、オブジェクト・ファイルを生成します。コンパイラの実行時にオプションを設定することによって、入力と出力を決定します。

コンパイラは、起動時に、ソースファイル名の拡張子、およびコマンドラインに指定されたコンパイル・オプションに基づいて、どのコンパイル・フェーズを実行するかを決定します。

次の表は、コンパイル・フェーズ、および各フェーズを制御するソフトウェアを示しています。

コンパイル・フェーズ	制御するソフトウェア	IA-32 または Itanium® ベース・アプリケーション
前処理 (オプション)	`fpp`	両方
コンパイル	`fortcom`	両方
アセンブル (オプション)	`as` または `ias`	`as` は IA-32 アプリケーション用で、`ias` は Itanium ベース・アプリケーション用です。
リンク	`ld(1)`	両方

デフォルトでは、コンパイラは、アセンブラを呼び出さずに直接オブジェクト・ファイルを生成します。しかし、特定のアセンブリ入力ファイルを使用して、プロジェクトのその他の部分とリンクする必要がある場合は、それらのファイルに対してアセンブラを使用できます。

コンパイラは、オブジェクト・ファイルと、認識できないファイル名とをリンカに渡します。次に、リンカは、そのファイルがオブジェクト・ファイル (`.o`) なのか、ライブラリ (`.a`) なのか、または 共有ライブラリ (`.so`) なのかを判断します。コンパイラは、すべてのタイプの入力ファイルを正しく処理するので、どのコンパイル・フェーズの実行にも使用できます。

前処理フェーズ

前処理は、プリプロセッサ・シンボル (マクロ) 置換、条件付きコンパイル、ファイルのインクルードといった処理を行います。コンパイルの最初のオプション・フェーズのときに、ファイルが前処理されます。`.fpp`、`.F`、`.F90`、`.FOR`、`.FTN`、または `.FPP` のファイル名の拡張子を持つソースファイルは、コンパイラにより 自動的に前処理されます。 例えば、次のコマンドは、標準的な Fortran プリプロセッサ・ディレクティブが格納されたソースファイルを前処理した後、このファイルをコンパイラとリンカに渡します。

```
ifort source.fpp
```

その他の拡張子を持つファイルを前処理する場合は、プリプロセッサを明示的に指定する必要があります。

通常は、Fortran ソース・プログラムのための前処理を指定する必要はありません。プログラムが `#if`、`#define` などの C 言語形式のプリプロセシング・コマンド使用している場合のみ前処理が必要になります。

ソース・プログラムを前処理したい場合は、プリプロセッサ fpp (インテル® Fortran コンパイラとともに提供されているプリプロセッサ) を使用するか、または C コンパイラの前処理機能を使用しなければなりません。fpp を使用することをお勧めします。

別のプリプロセッサを使用する場合は、コンパイラを起動する前にそのプリプロセッサを起動する必要があります。

fpp は、cpp に準拠しており、cpp 方式のディレクティブを受け付けます。cpp (および fpp) では、`#if` 式で文字列定数値を使用できません。

前処理オプションは、コマンドラインからプリプロセッサの操作を指示するのに使用できます。

> **注:** Fortran をサポートしていないプリプロセッサを使用すると、特に FORMAT 文で、Fortran コードが壊れる可能性があります。例えば、`FORMAT (\\I4)` は、バックスラッシュ「\」がレコードの終わりを示すため、プログラムの意味を変更します。

アセンブラとリンカ

次の表は、使用できるアセンブラとリンカをまとめたものです。

ツール	デフォルト	インテル® Fortran コンパイラに付属
IA-32 アセンブラ	Linux* アセンブラ、`as`	なし
Itanium® アセンブラ	インテル Itanium アセンブラ、`ias`	あり
リンカ	システムリンカ、`ld(1)`	なし

前処理、コンパイル、アセンブリ、およびリンクについては、代替ツールの場所およびオプションの指定ができます。

「インテル® Fortran が提供するライブラリ」も参照してください。

アセンブラ

32 ビット・アプリケーションについては、Linux が独自のアセンブラ `as` を用意しています。

Itanium ベースのアプリケーションについては、Itanium アセンブラ `ias` を使用してください。例えば、ある特定の入力ファイルを Fortran プロジェクトのオブジェクト・ファイルにリンクする場合は、次の手順を実行します。

1. `-S` オプションを指定してコマンドを発行し、アセンブリ・コード・ファイル `file.s`: **`ifort -S -c file.f`** を生成します。
2. *`file`*`.s` ファイルをアセンブルするには、次のコマンドで Itanium アセンブラを呼び出します: **`ias -Nso -p32 -o file.o file.s`**

次のアセンブラ・オプションが使用されます。

`-Nso` は、サインオン・メッセージを抑止します。

`-p32` は、32 ビット要素を再配置が可能なデータ要素として定義できるようにします。(このオプションには、下位互換があります。)

`-o file.o` は、出力オブジェクト・ファイル名を示します。

リンカ

コンパイラは、システムリンカ `ld(1)` を呼び出して、オブジェクト・ファイルから実行ファイルを作成します。

インテル® Fortran コンパイラのデフォルト動作

コンパイラは 1 つまたは複数の入力ファイルに対し、1 つまたは複数の出力ファイルを生成します。デフォルトでは、次の動作を実行します。

- 現在のディレクトリで、ライブラリ・ファイルを含むすべてのファイルを検索します。
- リンク用に指定されたオプションをリンカに渡します。
- ユーザ定義のライブラリをリンカに渡します。
- エラー・メッセージと警告メッセージを表示します。
- デフォルトのオプションが変更されない限り、デフォルトの設定と最適化を実行します。
- IA-32 アプリケーションの場合、`-tpp7` オプションを使用して、コードをインテル® Pentium® 4 プロセッサ用とインテル® Xeon™ プロセッサ用に最適化します。
- Itanium® ベース・アプリケーションの場合、`-tpp 2` オプションを使用して、コードをインテル® Itanium® 2 プロセッサ用に最適化します。

  注: Unicode* (マルチバイト) 形式の文字をサポートするオペレーティング・システムでは、コンパイラは Unicode* 文字を含むファイル名を処理します。

入力ファイルとファイル名拡張子

インテル® Fortran コンパイラでは、ファイル名拡張子から各入力ファイルの種類を解釈します。ファイル名拡張子には、`.a`、`.f`、`.for`、`.o` などがあります。

ファイル名	ファイルの種類	処理
`filename``.a`	オブジェクト・ライブラリ	`ld` に渡されます。

filename.f *filename*.ftn *filename*.for filename.i	Fortran の固定形式ソース	インテル Fortran コンパイラによってコンパイルされます。
filename.fpp *filename*.F *filename*.FOR *filename*.FTN *filename*.FPP	Fortran の固定形式ソース	インテル Fortran プリプロセッサ fpp によって前処理されてから、インテル Fortran コンパイラによってコンパイルされなければなりません。
filename.F90	Fortran の自由形式ソース	インテル Fortran プリプロセッサ fpp によって前処理されてから、インテル Fortran コンパイラによってコンパイルされなければなりません。
filename.f90 *filename*.i90	Fortran の自由形式ソース	インテル Fortran コンパイラによってコンパイルされます。
filename.s	アセンブリ・ファイル	アセンブラ (IA-32 コンパイラ) またはインテル Itanium® アセンブラ (Itanium コンパイラ) に渡されます。
filename.o	コンパイル済みのオブジェクト・ファイル	ld に渡されます。

コンパイラの設定ファイルを使用して、入力ライブラリのデフォルトのディレクトリを指定できます。入力ファイル、一時ファイル、ライブラリ、アセンブラ、およびリンカ用の追加ディレクトリを指定するには、出力ファイル名とディレクトリ名を指定するコンパイラ・オプションを使用します。

ファイル指定

ファイル指定では、任意で、ディレクトリを指定するパス名とそれに続くファイル名が使用されます。パス名は、次の 2 つの形式のいずれかで示します:

- ディレクトリがルート・ディレクトリを基準として決められた絶対パス名。最初の文字はスラッシュ (/) になります。例えば、次に示すディレクトリとファイル名は、/usr/users/gdata ディレクトリにある testdata ファイルを参照します: /usr/users/gdata/testdata
- ディレクトリが現在のディレクトリを基準として決められた相対パス名。相対パス名では、スラッシュ (/) を最初の文字として使用しません。次の例では、現在のディレクトリ /usr/users を基準とした相対パス名を使用して、gdata/ サブディレクトリ内の同一ファイル testdata を参照します: gdata/testdata

ディレクトリ名およびファイル名には、オペレーティング・システムのワイルドカード文字 (*、?、および [ ] 構文など) を含めることができません。C シェルで見られるように、パス名の最初の文字にチルド文字 (~) を使用することで、最上位ディレクトリを参照できます。

先頭の空白および末尾の空白は、文字式からは削除されますが、Hollerith (数値配列) 名からは削除されないため、ファイル指定を行う際は注意してください。

ファイル名では、大文字・小文字が区別されるので、両方を含めることができます。例えば、次の 3 つのファイルはすべて異なるファイルとして扱われます:

```
myfile.for
MYfile.for
MYFILE.for
```

出力ファイル

`ifort` コマンドによって生成される出力は以下のとおりです:

- コマンドラインで `-c` オプションを指定した場合、オブジェクト・ファイル (`test.o` など) が生成されます。オブジェクト・ファイルは、各ソースファイルごとに作成されます。
- `-c` オプションを省略した場合、実行ファイル (`a.out` など) が生成されます。
- ソースファイルに MODULE 文が 1 つまたはそれ以上ある場合、1 つまたはそれ以上のモジュール・ファイル (`datadef.mod` など) が生成されます。
- `-shared` オプションを使用した場合、共有ライブラリ (`mylib.so` など) が生成されます。

これらのファイルを生成するかどうかは、コマンドラインで適切なオプションを指定することで制御できます。

`-c` オプションが指定されていない場合、コンパイラによって各ソースファイルごとに一時オブジェクト・ファイルが生成されます。次に、リンカが起動され、オブジェクト・ファイルを 1 つの実行プログラム・ファイルにリンクし、最後に一時オブジェクト・ファイルを削除します。

`-c` オプションを指定した場合は、オブジェクト・ファイルが作成され、作業ディレクトリに保持されます。この場合、後でオブジェクト・ファイルを別の `ifort` コマンドからリンクする必要があります。この方法は、`make` コマンドで makefile を処理してコンパイルする方法のように、大きなアプリケーションを増分コンパイルするときに役立ちます。

コンパイルの途中で致命的なエラーが発生する場合、または `-c` などの特定のオプションを指定した場合、リンクは行われません。

> **注:** プログラム全体のすべてのオブジェクトをコンパイルするには、`-ipo` オプションを使用します。

実行プログラム・ファイル名を指定するには (`a.out` 以外の名前を指定する場合)、`-o` *`output`* オプションを使用します (*`output`* にはファイル名を指定します)。次のコマンドは、ソースファイル名 `test1.f` に対してファイル名 `prog1.out` を指定します:

```
ifort -o prog1.out test1.f
```

`-o` *`output`* オプションとともに `-c` オプションを指定する場合、オブジェクト名を変更できます (実行プログラム名は変更できません)。また、`-c` を指定し、`-o` *`output`* オプションを省略する場合、サフィックスが `.o` に変更されたソースファイル名がオブジェクト・ファイル名に使用されます。

注: 複数のソースファイルに対しては、-c および -o を一緒に使用できません。

デフォルトの最適化レベルは -O2 です (-g を指定しない場合のみ)。

コンパイラまたはリンカにより作成される一時ファイル

コンパイラまたはリンカによって作成される一時ファイルは、オペレーティング・システムが一時ファイルを格納するために使用するディレクトリに置かれます。

一時ファイルを格納する際に、ドライバはまず TMP 環境変数を確認します。定義されていれば、TMP が示すディレクトリが一時ファイルの格納に使用されます。

TMP 環境変数が定義されていなければ、ドライバは TMPDIR 環境変数を確認します。定義されていれば、TMPDIR が示すディレクトリが一時ファイルの格納に使用されます。

TMPDIR 環境変数が定義されていなければ、ドライバは TEMP 環境変数を確認します。定義されていれば、TEMP が示すディレクトリが一時ファイルの格納に使用されます。

TEMP 環境変数が定義されていなければ、/tmp ディレクトリが一時ファイルの格納に使用されます。

3. アプリケーションのビルド

コンパイル処理の制御

コンパイル時に使用される環境をカスタマイズするには、以下の変数、オプション、およびファイルを指定します。

- 環境変数はコンパイラがライブラリや「インクルード」ファイルなどの特殊なファイルを検索するパスを指定します。
- 設定ファイルは各コンパイルに使用するオプションを指定します。また、応答ファイルは個々のプロジェクトに使用するオプションとファイルを指定します。

環境変数の設定と表示

SET コマンドを使用することにより、環境変数を 1 つずつ表示するかまたは設定できます。また、`ifortvars.csh` および `ifortvars.sh` ファイルを使用することにより、一度に複数の環境変数を設定できます。これらのファイルは、`/opt/intel_fc_80/bin` ディレクトリにあります。詳細は、「シェル・スクリプトの実行による環境変数の設定」を参照してください。

C シェルで環境変数を設定するには、`setenv` コマンドを使用します:

```
setenv FORT8 /usr/users/smith/test.dat
```

C シェルで環境変数を無効にするには、`unsetenv` コマンドを使用します。

```
unsetenv FORT8
```

Bourne シェル (`sh`)、Korn シェル (`ksh`)、および bash シェルで環境変数を設定するには、`export` コマンドと割り当てコマンドを使用します:

```
export FORT8
FORT8=/usr/users/smith/test.dat
```

Bourne シェル、Korn シェル、または bash シェルで環境変数を無効にするには、`unset` コマンドを使用します:

```
unset FORT8
```

設定ファイル環境変数

デフォルトでは、コンパイラ実行ファイルと同じディレクトリにあるデフォルトの設定ファイル (`ifort.cfg`) が使用されます。ただし、異なるディレクトリにある他の設定ファイルを使用する場合は、`IFORTCFG` 環境変数を使って、その設定ファイルのディレクトリとファイル名を指定します。

ユーザーズ・ガイドの次のトピックも参照してください。

- コンパイル時の環境変数
- ランタイム時の環境変数

シェル・スクリプトの実行による環境変数の設定

コンパイラを起動する前に、環境変数を設定して、各種コンポーネントの場所を設定する必要があります。

インテル® Fortran コンパイラのインストールには、環境変数の設定に使用できるシェル・スクリプトが含まれています。

ソースコマンドを使用して、コマンドラインからシェル・スクリプトを実行します。例えば、bash シェルのスクリプト・ファイルを実行するには次のとおりです:

```
source /opt/intel_fc_80/bin/ifortvars.sh
```

C シェルを使用するには、`.csh` バージョンのスクリプト・ファイルを使用します:

```
source /opt/intel_fc_80/bin/ifortvars.csh
```

Linux* の起動時に `ifortvars.sh` を自動的に実行するには、`.bash_profile` ファイルを編集し、ファイルの最後に上記の行を追加します。次に例を示します。

```
# set up environment for Intel compiler
source /opt/intel_fc_80/bin/ifortvars.sh
```

インテル® Fortran コンパイラの起動

次の 2 つの方法のいずれかで、インテル® Fortran コンパイラを起動できます。

- `ifort` コマンドを使用する
- `make` コマンドを使用して makefile を指定する

ifort コマンドを使用する

構文は、次のとおりです:

```
ifort [options] input_file(s)
```

`option` はハイフンが先頭にある 1 文字以上の文字で指定されます。

オプションは、引数をファイル名、文字列、文字、または数字の形をとることができます。特に指定がない限り、オプションとその引数の間にスペースを挿入することができます。ユーザーズ・ガイドの「コンパイラ・オプションの概要」を参照してください。

スペースを区切り文字として使って、複数の *`input_file`* を指定できます。「入力ファイルとファイル名拡張子」を参照してください。

注: コマンドラインで指定されたオプションは、すべてのファイルに適用されます。例えば、次の `-c` および `-nowarn` コマンドライン・オプションは `x.f` および `y.f` ファイルに適用されます:

```
ifort -c x.f -nowarn y.f
```

make コマンドを使用する

さまざまなパスを持つ複数のファイルをコンパイルし、この情報を繰返しコンパイルできるようにするには、makefile を使用して インテル Fortran コンパイラを起動します。

makefile を使用して入力ファイルをコンパイルするには、`/usr/bin` と `/usr/local/bin` がパスになければなりません。

C シェルを使う場合は、`.cshrc` ファイルを編集して、次の行を追加します。

```
setenv PATH  /usr/bin:/usr/local/bin:yourpath
```

その後、次のようにコンパイルができます。

```
make -f yourmakefile
```

`-f` は特定の makefile を指定する `make` コマンドのオプションです。

ifort コマンドの例

複数のファイルのコンパイルおよびリンク

次の `ifort` コマンドは、Fortran の自由形式ソースファイル `aaa.f90`、`bbb.f90`、および `ccc.f90` をコンパイルします。このコマンドは、`ld` リンカを起動して、一時オブジェクト・ファイルをリンカに渡し、実行ファイル `a.out` を生成します:

```
ifort aaa.f90 bbb.f90 ccc.f90
```

次の `ifort` コマンドは、ファイル名が `.f` で終わるすべてのファイルを Fortran 固定形式ソースとしてコンパイルします。`a.out` ファイルがリンカによって生成されます:

```
ifort *.f
```

リンクの抑止

次の `ifort` コマンドは、`MODULE TYPEDEFS_1` を含む自由形式ソースファイル `typedefs_1.f90` をコンパイルしますが、リンクは行いません。このコマンドによって、`typedefs_1.mod` と `typedefs_1.o` が作成されます。オブジェクト・ファイルは自動的に削除されずに残されます。 `-c` オプションを指定することでリンクを抑止できます:

```
ifort -c typedefs_1.f90
```

出力ファイル名の変更

次の `ifort` コマンドは、自由形式の Fortran ソースファイル `circle-calc.f90` および `sub.f90` を続けてコンパイルします:

```
ifort -c circle-calc.f90 sub.f90
```

コンパイル時に、デフォルトの最適化レベルである `-O2` が両方のソースファイルに適用されます。`-c` オプションが指定されているので、オブジェクト・ファイルはリンカに渡されません。この場合、名前付けされた出力ファイルがオブジェクト・ファイルになります。

上記のコマンドと同じように、次の `ifort` コマンドは複数のソースファイルをコンパイルします:

```
ifort -o circle.out circle-calc.f90 sub.f90
```

この場合、`-c` オプションが指定されていないので、`circle.out` という名前を持つ実行プログラムが生成されます。

リンカ・ライブラリの追加

次の `ifort` コマンドは、デフォルトの最適化設定を使用して、自由形式ソースファイル `myprog.f90` をコンパイルし、検索に追加するライブラリをリンカに渡します:

```
ifort myprog.f90 typedefs_1.o -lmylib
```

ファイルは最適化レベル `-O2` で処理され、オブジェクト・ファイル `typedefs_1.o` とリンクされます。`-lmylib` オプションは、(`ifort` コマンドがリンカに渡す標準ライブラリ・リストの他に) `libmylib` ライブラリで未解決の参照を検索するように指定します。

モジュール (.mod) ファイルの使用

モジュール (`.mod` ファイル) とは、データ・オブジェクト、パラメータ、構造体、プロシージャ、および演算子のようなエンティティの仕様を持つプログラム・ユニットの種類です。これらのプリコンパイル済みの仕様と定義は、1 つまたはそれ以上のプログラム・ユニットで使用できます。モジュール・エンティティへの部分的あるいは完全なアクセスは、USE 文で提供されます。モジュールの一般的な用途は、グローバル・データの仕様、または派生型および関連した演算子の仕様を記述することです。

一部のプログラムでは、複数のディレクトリに格納されたモジュールが必要になります。プログラムのコンパイル時に `-I`*`dir`* オプションを使用することにより、プログラムに含める必要がある `.mod` ファイルの位置を指定できます。

`-module` *`path`* オプションを使用することにより、モジュール・ファイルを作成するディレクトリを指定できます。また、この指定したパスは、モジュール・ファイルを検索するパスとしても使用されます。このオプションを使用しない場合は、モジュール・ファイルが現在のディレクトリに作成されます。

他のプログラムやサブプログラムから参照される前に、モジュール・ファイルが作成されていることを確認する必要があります。

モジュールを持つプログラムのコンパイル

コンパイルするファイルに 1 つ以上のモジュールが定義されている場合、コンパイラは 1 つ以上の .mod ファイルを生成します。例えば、a.f90 ファイルには、以下のように定義されたモジュールが含まれています:

```
module test
integer:: a
contains
  subroutine f()
  end subroutine
end module test

module payroll
.
.
.
end module payroll
```

コンパイラ・コマンド: **ifort -c a.f90**

このコマンドにより、次のファイルが生成されます:

- test.mod
- test.o
- payroll.mod
- payroll.o

.mod ファイルには、プログラム a.f90 で定義されたモジュールに関する必要な情報が含まれています。

プログラムにモジュールが含まれていない場合、.mod ファイルは生成されません。モジュールが含まれていないサンプル・プログラム test2.f90 の例を以下に示します。

コンパイラ・コマンド: **ifort -c test2.f90**

上記のコマンドにより、オブジェクト・ファイル test2.o のみが生成されます。

他の例として、file1.f90 には 1 つ以上のモジュールが含まれており、file2.f90 には、USE 文を使ってそのモジュールを参照するプログラム・ユニットが 1 つ以上含まれていると仮定します。この場合、ソースを次のコマンドでコンパイルし、リンクを行います:

ifort file1.f90 file2.f90

マルチディレクトリ・モジュール・ファイルの処理

.mod ファイルを複数の異なるディレクトリに生成するときのモジュール管理の例を以下に示します。プログラム mod_def.f90 がディレクトリ /usr/yourdir/test/t にあり、このプログラムには、以下のように定義されたモジュールが含まれているとします:

```
file: mod_def.f90
module definedmod
.
.
.
end module
```

コンパイラ・コマンド:

```
ifort -c mod_def.f90
```

上記のコマンドによって、ディレクトリ /usr/yourdir/test/t 内に mod_def.o と definedmod.mod の 2 つのファイルが生成されます。

上記の .mod ファイルを別のディレクトリで使用する必要がある場合、例えば、プログラム usemod が /usr/yourdir/test/t2 ディレクトリにある definemod.mod を使用する場合、以下を行います:

```
file: use_mod_def.f90
program usemod
use definedmod
.
.
.
end program
```

上記のサンプル・プログラムをコンパイルするには、以下のコマンドを使用します:

```
ifort -c use_mod_def.f90 -I/usr/yourdir/test/t
```

-Idir オプションには、definedmod.mod ファイルが格納されているパスを指定します。

makefile を使用した並列呼び出し

モジュールが定義されているプログラムでは、makefile を使用した並列呼び出しがサポートされています。この makefile は、プロシージャ間の最適化およびプログラム全体の最適化に対応します。以下にサンプル・コードを示します:

```
test1.f90
module m1
.
.
.
end module
```

```
test2.f90
subroutine s2()
use m1
.
.
.
end subroutine
test3.f90
subroutine s3()
use m1
.
.
.
end subroutine
```

上記のコードをコンパイルするための makefile は、以下のようになります:

```
m1.mod: test1.o
test1.o: test1.f90
ifort -c test1.f90
test2.o: m1.mod test2.f90
ifort -c test2.f90
test3.o: m1.mod test3.f90
ifort -c test3.f90
```

インクルード・ファイルと .mod ファイルの検索

インクルード・ファイルは、`#include` プリプロセッサ・ディレクティブまたは Fortran の INCLUDE 文によってプログラムに取り込まれます。

ディレクトリからインクルード・ファイルが検索される順序は、次のとおりです。

1. include が含まれるソースファイルのディレクトリ
2. `-I`*`dir`* オプションで指定されたディレクトリ
3. 現在の作業ディレクトリ
4. `FPATH` 環境変数で指定されたディレクトリ

検索されるディレクトリの場所は、インクルード・ファイル・パスと呼ばれます。複数のディレクトリをインクルード・ファイル・パスに指定できます。

モジュール (`.mod`) ファイルは、USE 文を使用してプログラムで指定されます。モジュールファイルは、複数のディレクトリにわたって配置できます。

ディレクトリから `.mod` ファイルが検索される順序は、次のとおりです。

1. USE 文が含まれるソースファイルのディレクトリ
2. `-module path` オプションで指定されたディレクトリ
3. `-I`*`dir`* オプションで指定されたディレクトリ
4. 現在の作業ディレクトリ

5. FPATH 環境変数で指定されたディレクトリ

インクルード・ファイル・パスの指定と除外

インクルード・ファイルとモジュールファイルの場所を示すには、`-I`*`dir`* オプションを使用できます。

FPATH 環境変数で指定されたデフォルトのパスをコンパイラが検索しないようにするには、`-X` オプションを使用します。

これらのオプションは、`ifort.cfg` の設定ファイルまたはコマンドラインで指定できます。

例えば、デフォルトのパスの代わりに`/alt/include` のパスを検索するようにコンパイラに命令するときは次のコマンドラインを使用します。

```
ifort -X -I /alt/include newmain.f
```

設定ファイルと応答ファイル

*設定ファイル*と*応答ファイル* は、同じコマンドを何度も入力する手間を省く点では同じです。(応答ファイルは、間接コマンドファイルとしても知られています。) 次に、各ファイルについて説明します。

設定ファイル

設定ファイル (`.cfg`) を使用することにより、次のことが可能になります:

- コマンドライン・オプションを入力する時間を短縮できます。
- よく使用されるコマンドの整合性を保証します。

設定ファイルには、あらゆるコマンドライン・オプションを挿入できます。コンパイラは、設定ファイルに記述されたオプションを上から順番に処理し、次にコンパイラの起動時に指定されたコマンドライン・オプションを処理します。

> **注:** 設定ファイルに記述されたオプションは、コンパイラを実行するたびに使用されます。他のプロジェクトで、異なるオプションを使用する必要がある場合は、応答ファイルを使用してください。

デフォルトでは、`ifort.cfg` という名前の設定ファイルが使用されます。

このファイルは、コンパイラの実行ファイルが格納されているディレクトリにあります。

別の場所にある他の設定ファイルを使用する場合は、`IFORTCFG` 環境変数を使用し、その設定ファイルのディレクトリとファイル名を指定します。

設定ファイルのサンプル

以下に設定ファイルのサンプルを示します。ポンド記号 (#) は、その行がコメント行であることを示します。

```
## Example ifort.cfg file
##
## Define preprocessor macro MY_PROJECT.
-DMY_PROJECT
##
## Set extended-length source lines.
-extend_source
##
## Set maximum floating-point significand precision.
-pc80
##
```

応答ファイル

応答ファイル (間接コマンドファイルとしても知られています) を使用することにより、次のことが可能です:

- プロジェクトごとに、特定のコンパイルで使用するオプションを指定できます。
- この情報を個々のファイルに保存できます。

応答ファイルは、コマンドラインでオプションとして呼び出します。応答ファイルで指定したオプションは、コマンドライン上の応答ファイルが呼び出された場所に挿入されます。

応答ファイルは、設定ファイルと同じように、次のことが可能です:

- コマンドライン・オプションを入力する時間を短縮できます。
- よく使用されるコマンドの整合性を保証します。

設定ファイルに記述されたオプションは、コンパイラを実行するたびに使用されます。これに対して、応答ファイルは、個々のプロジェクトごとにオプションを維持できます。

応答ファイル (または間接コマンドファイル) では、1 行にオプションまたはファイル名をいくつも記述できます。同じコマンドラインで、複数のファイルを参照することもできます。

応答ファイルを使用するには、次の構文を使用します:

ifort @*responsefile* [@*responsefile2*...]

> **注:** コマンドラインでは、応答ファイル名の前にアットマーク (@) を入力する必要があります。

代替ツールの場所およびオプションの指定

インテル® Fortran コンパイラでは、前処理、コンパイル、アセンブリ、およびリンクのデフォルト・ツールに代わる代替ツールの場所およびオプションを指定できます。これを行うには、コマンドライン・オプションを使用します。

-Qlocation を使ってツールの代替場所を指定する

`-Qlocation` を使用すると、ツールのパス名を指定できます。このオプションの構文は次のとおりです:

`-Qlocation,`*`tool`*`,`*`path`*

`tool` には次のいずれかを入力します:

- `fpp` インテル Fortran プリプロセッサ (fpp)
- `f` インテル Fortran コンパイラ (fortcom)
- `as` アセンブラ
- `link` または `ld` リンカ
- `crt` スタートアップ・ルーチン (`crt%.o`)。実行ファイルにリンクされる。

`path` はツールの場所です。

例:

```
ifort -Qlocation,fpp,/usr/preproc myprog.f
```

-Qoption を使用してオプションをツールに渡す

`-Qoption` を使用して、オプションをプリプロセッサ、コンパイラ、アセンブラ、またはリンカに渡します。このオプションの構文は次のとおりです:

`-Qoption,`*`tool`*`,`*`options`*

`tool` には次のいずれかを入力します:

- `fpp` インテル Fortran プリプロセッサ (fpp)
- `f` インテル Fortran コンパイラ (fortcom)
- `as` アセンブラ
- `link` リンカ

`options` は、指定したツールに有効な引数文字列です。1 つでも複数個でも指定できます。

スペースかタブ文字を含む引数を指定するときは、その引数全体を二重引用符 (" ") で囲んでください。引数を複数指定するときは、それぞれをカンマで区切ってください。

次の例では、代替ライブラリとリンクします:

```
ifort -Qoption,link,-lmylib prog1.f
```

定義済みプリプロセッサ・シンボル

プリプロセッサ・シンボル (マクロ) を使用すると、コンパイルする前にプログラムの値を置換することができます。この処理は、前処理フェーズで行います。

一部のプリプロセッサ・シンボルはコンパイラ・システムによって事前定義されており、コンパイラ・ディレクティブおよび fpp で使用できます。他のシンボルを使用する場合は、コマンド行で指定します。

ユーザーズ・ガイドの「プリプロセッサ・オプション」も参照してください。

次の表は、インテル® Fortran コンパイラで使用できる定義済みプリプロセッサ・シンボルです。「デフォルト」欄では、デフォルトでそのプリプロセッサ・シンボルが有効 (オン) になるのか、あるいは無効 (オフ) になるのかを示します。

シンボル名	デフォルト	アーキテクチャ (IA-32 または Itanium® ベース)	説明
__INTEL_COMPILER=n	On, *n*=810	両方	インテル Fortran コンパイラを識別します。
__INTEL_COMPILER_BUILD_DATE =*YYYYMMDD*		両方	インテル Fortran コンパイラのビルドの日付を示します。
__linux__ __linux __gnu_linux__ linux __unix_ __unix unix __ELF__		両方	コンパイル開始時に定義されます。
__i386__ __i386 i386		IA-32 アーキテクチャ	
__ia64__ __ia64 ia64		Itanium アーキテクチャ	
_OPENMP=*n*	*n*=200011	両方	このプリプロセッサ・シンボルは、*YYYYMM* 形式で、*YYYY* と *MM* はそれぞれ OpenMP Fortran 使用がサポートされた年と月を表します。fpp と Fortran コンパイラの条件付きコンパイルの両方で、このプリプロセッサ・シンボルを使用できま

			す。また、-openmp が指定された場合のみ使用できます。
_PRO_INSTRUMENT	オフ	両方	-prof_gen が指定されたときに、定義されます。

プリプロセッサ・シンボルの定義

前処理中に使用されるシンボル名を定義するには、-D オプションを使用します。このオプションの機能は、#define というプリプロセッサ・ディレクティブと同じです。次の形式でこのオプションを指定します。

-D*name*[=*value*]

各アイテムの意味は次のとおりです。

- *name* は定義するシンボルの名前です。
- *value* は *name* を置き換える任意の *value* です。

value を入力しなかった場合、*name* は 1 に設定されます。スペースまたは特殊文字がある場合は、*value* が引用符で囲まれていなければなりません。

前処理を行うと、*name* が検出されるたびに、指定されている *value* に置換されます。例えば、*SIZE* という名前のシンボルを定義し、値を 100 にするには、次のコマンドを実行します。

ifort -fpp -D*SIZE* =100 prog1.f

前処理を行うと、前処理されたソースコードをコンパイラに渡す前に、*SIZE* が指定されている値の 100 に置換されます。プログラムに次の宣言が含まれていることを仮定します。

REAL VECTOR(*SIZE)*

コンパイラに送信されるコードでは、この宣言およびその他 *SIZE* という名前が検出されるすべての箇所において、値 100 が *SIZE* に取って代わります。

プリプロセッサ・シンボルの抑止

プリプロセッサ・シンボルの自動定義を抑止するには、-U オプションを使用します。このオプションは、指定された名前について現在有効になっているシンボル定義を抑止するために使用します。-U オプションは、#undef プリプロセッサ・ディレクティブと同じ機能を果たします。

コマンドライン出力からファイルへのリダイレクト

多くのテキストを表示するプログラムの場合、stdout に表示されているテキストをファイルにリダイレクトすることを考慮してください。テキストを多く表示すると、プログラムの実行が遅くなります。ワークステーションのウィンドウでテキストをスクロールすることで、I/O でボトルネック (経過時間の増加) が生じ、CPU 時間がより長くなる可能性があります。

次のコマンドでは、出力をリダイレクトしてから表示することによって、プログラムをより効率的に動作する方法を示します。

```
myprog > results.lis
more results.lis
```

プログラムの出力をリダイレクトすると、画面 I/O が減少するため、レポートされる時間が変わります。

実行プログラムの作成、実行、デバッグ

自由形式を使用し、モジュールおよび外部サブプログラムを使用するメイン・プログラムの例を以下に示します。

CALC_AVERAGE 関数は、別個に作成されたファイルに収められており、インターフェイス・ブロックの ARRAY_CALCULATOR モジュールに依存します。

USE 文が ARRAY_CALCULATOR モジュールにアクセスします。このモジュールには、CALC_AVERAGE の関数定義が含まれています。

5 要素の配列が CALC_AVERAGE 関数に渡され、PRINT される値が関数から AVERAGE 変数へ返されます。

サンプル・プログラム:

```
!File: main.f90
!This program calculates the average of five numbers
  PROGRAM MAIN
    USE ARRAY_CALCULATOR
    REAL, DIMENSION(5) :: A = 0
    REAL :: AVERAGE
    PRINT *, 'Type five numbers: '
    READ  (*,'(F10.3)') A

   AVERAGE = CALC_AVERAGE(A)
    PRINT *, 'Average of the five numbers is: ', AVERAGE
  END PROGRAM MAIN
```

メイン・プログラムが参照するモジュールを以下に示します。この例では、インターフェイス・ブロックや形状引継ぎ配列など、さらに多くの Fortran 95/90 機能が示されています:

```
!File: array_calc.f90.
!Module containing various calculations on arrays.
  MODULE ARRAY_CALCULATOR
    INTERFACE
      FUNCTION CALC_AVERAGE(D)
        REAL :: CALC_AVERAGE
        REAL, INTENT(IN) :: D(:)
      END FUNCTION CALC_AVERAGE
    END INTERFACE
```

```
  !Other subprogram interfaces...
  END MODULE ARRAY_CALCULATOR
```

メイン・プログラムが参照する CALC_AVERAGE の関数宣言を以下に示します:

```
!File: calc_aver.f90.
!External function returning average of array.
  FUNCTION CALC_AVERAGE(D)
    REAL :: CALC_AVERAGE
    REAL, INTENT(IN) :: D(:)
    CALC_AVERAGE = SUM(D) / UBOUND(D, DIM = 1)
  END FUNCTION CALC_AVERAGE
```

サンプル・プログラムを作成するためのコマンド

プログラム開発の初期段階では、上記の 3 つのサンプル・プログラムのようなファイルは、次のコマンドを使用して個別にコンパイルしリンクできます:

```
ifort -c array_calc.f90
ifort -c calc_aver.f90
ifort -c main.f90
ifort -o calc main.o array_calc.o calc_aver.o
```

コマンドの詳細:

- -c オプションはリンクを抑制し、.o ファイルを保持します。
- 最初のコマンドでは、ファイル array_calculator.mod および array_calc.o (MODULE 文にある名前によって、array_calculator.mod モジュール・ファイルの名前が決定します) が作成されます。モジュール・ファイルは作業ディレクトリに保存されます。
- 2 番目のコマンドでは、ファイル calc_aver.o が作成されます。
- 3 番目のコマンドでは、ファイル main.o が作成され、モジュール・ファイル array_calculator.mod が使用されます。
- 最後のコマンドでは、すべてのオブジェクト・ファイルが calc という名前の実行プログラムにリンクされます。ファイルをリンクするには、ld コマンドの代わりに ifort コマンドを使用します。

ファイル名を指定する順番は重要です。この ifort コマンドでは次の順でファイルを指定し、コンパイル、リンクが行われます:

- モジュールを定義するファイル array_calc.f90 がコンパイルされ、オブジェクト・ファイルおよび array_calculator.mod ファイルが作成されます。
- 外部関数 CALC_AVERAGE を含むファイル calc_aver.f90 をコンパイルします。
- ファイル main.f90 (メイン・プログラム) をコンパイルします。USE 文は、モジュール・ファイルの array_calculator.mod を参照します。
- ld を使用して、メイン・プログラムとすべてのオブジェクト・ファイルを calc という名前の実行プログラム・ファイルにリンクします。

サンプル・プログラムの実行

パス定義に calc が格納されたディレクトリが含まれている場合、プログラムの名前を入力するだけで実行できます。

```
calc
```

サンプル・プログラムを実行すると、メイン・プログラムの PRINT 文と READ 文によって、ユーザとプログラムの間で次のようなやり取りが行われます:

```
Type five numbers:
55.5
4.5
3.9
9.0
5.6
Average of the five numbers is:   15.70000
```

サンプル・プログラムのデバッグ

デバッガを使用してプログラムをデバッグするには、-g オプションを付けてソースファイルをコンパイルし、オブジェクト・ファイルおよび実行プログラム・ファイルにソース行単位のデバッグを行うためのシンボルテーブル情報を追加します。また、ifort コマンドで -o オプションを使用することにより、実行プログラム・ファイルに calc_debug という名前を付けられます:

```
ifort -g -o calc_debug array_calc.f90 calc_aver.f90 main.f90
```

ユーザーズ・ガイドの「idb を使用したデバッグの概要」および関連するセクションを参照してください。

共有ライブラリの作成

Fortran のソースファイルから共有ライブラリを作成するには、ifort コマンドを使用してファイルを処理します。

- .so ファイルを作成するには、-shared オプションを指定しなければなりません。
- -o *output* オプションを指定して、出力ファイルに名前を付けることができます。
- -c オプションを省略すると、コマンド行から直接、共有ライブラリ (.so ファイル) を単一ステップで作成できます。
  -o *output* オプションを省略すると、コマンド行の最初の Fortran ファイルの名前が、.so ファイルのファイル名を作成するのに使用されます。また、共有ライブラリ作成に関連付けられている追加オプションを指定できます。
- -c オプションを指定すると、オブジェクト・ファイル (.o ファイル) を作成できます。これは、-o オプションで名前を付けることができます。共有ライブラリを作成するには、.o ファイルを ld と一緒に処理します。この際、共有ライブラリ作成に関連付けられているオプションを指定します。
- Itanium® ベース・システムで共有ライブラリをビルドする場合、共有ライブラリに含まれる各オブジェクト・ファイルのコンパイルを行う -fpic オプションを指定します。このオプション

が使用されていない場合、リンカにより、「@gprel relocation against dynamic symbol.」というようなエラー・メッセージが発行されます。

ifort コマンドのみで共有ライブラリを作成する

ifort コマンドのみで共有ライブラリ (.so) ファイルを作成できます。

```
ifort -shared  octagon.f90
```

-shared オプションは共有ライブラリを作成するのに必要です。octagon.f90 はソースファイルの名前です。複数のソースファイルとオブジェクト・ファイルを指定できます。

-o オプションが省略されているため、共有ライブラリ・ファイルの名前は octagon.so になります。

-c オプションが省略されているため、Fortran ライブラリの標準リストを指定する必要はありません。

ifort コマンドおよび ld コマンドで共有ライブラリを作成する

最初に .o ファイルを作成する必要があります。次の例では、octagon.o ファイルを作成します。

```
ifort -c octagon.f90
```

octagon.o ファイルを ld コマンドの入力として使用して、octagon.so という共有ライブラリを作成します。

```
ld -shared -no_archive octagon.o \
     -lifport -lifcoremt -limf -lm -lirc -lcxa \
     -lpthread -lirc -lunwind -lc -lirc_s
```

次の点に注意してください。

- -shared オプションは共有ライブラリを作成するのに必要です。
- オブジェクト・ファイルの名前は octagon.o です。複数のオブジェクト (.o) ファイルを指定できます。
- -lifport とそれに続くオプションは、ライブラリの標準リストです。これは、ifort コマンドで、ld に渡すことになるリストです。共有ライブラリを作成する場合、すべてのシンボルが解決される必要があります。

正しくライブラリを指定するために、-dryrun コマンドの出力結果から、使用されるすべてのライブラリを確認するとよいでしょう。

-Qoption コマンドを使用してオプションを ld に渡します。

共有ライブラリの詳細は、ユーザーズ・ガイドの「ライブラリの作成」を参照してください。

ユーザーズ・ガイドの「ld(1)」リファレンス・ページも参照してください。

共有ライブラリの制約

ld で共有ライブラリを作成する場合、次の制約に注意してください。

- 共有ライブラリをアーカイブ・ライブラリにリンクすることはできません。
  共有ライブラリを作成する場合、外部参照を解決するために依存できるのは、他の共有ライブラリだけです。アーカイブ・ライブラリに存在するルーチンを参照する必要がある場合は、そのルーチンを個別の共有ライブラリに入れるか、または作成する共有ライブラリに含めます。共有ライブラリの作成時には、複数のオブジェクト (.o) ファイルを指定できます。
  ルーチンを個別の共有ライブラリに入れるには、そのルーチンのソースファイルかオブジェクト・ファイルを入手して、必要に応じて再コンパイルし、個別の共有ライブラリを作成します。ifort コマンドで再コンパイルする際、または ld コマンドで共有ライブラリを作成する際に、オブジェクト・ファイルを指定することができます。
  作成する共有ライブラリにルーチンを含めるには、そのルーチン (ソースファイルまたはオブジェクト・ファイル) を共有ライブラリを構成する他のソースファイルと一緒に配置し、必要に応じて再コンパイルします。
  次に、共有ライブラリを作成しますが、再コンパイル時または共有ライブラリ作成時に、そのルーチンが含まれるファイルを指定します。ifort コマンドで再コンパイルする際、または ld コマンドで共有ライブラリを作成する際に、オブジェクト・ファイルを指定することができます。
- 共有ライブラリを作成する場合、すべてのシンボルが定義 (解決) されている必要があります。
  共有ライブラリの作成時には、すべてのシンボルが ld に対して定義されていなければならないため (-Qoption コマンドを使用しない限り)、ld コマンド行にすべてのインテル® Fortran 標準ライブラリを含む共有ライブラリを指定する必要があります。インテル Fortran の標準ライブラリのリストは、-l*string* オプションを使って指定することができます。

共有ライブラリのインストール

共有ライブラリを作成したら、これを参照するプログラムを実行する前に、プライベートまたはシステム全体で使用するためにインストールする必要があります。

- プライベート用の共有ライブラリをインストールするには (テストを行う場合など)、ユーザーズ・ガイドの「ld(1)」で説明されているように環境変数を LD_LIBRARY_PATH に設定します。
- システム共通の共有ライブラリをインストールするには、ld が使用する標準ディレクトリ・パスの 1 つに共有ライブラリ・ファイルを配置します。ユーザーズ・ガイドの「ld(1)」を参照してください。

共通ブロックの割り当て

-dyncom (ダイナミック共用) オプションは、実行時に共通ブロックの割り当てを制御するために使用します。

このオプションは、共通ブロックを動的にするために指定します。そのデータの領域がコンパイル時ではなく実行時に割り当てられます。ダイナミック共通ブロックの宣言を含む各ルーチンへの入口

で、共通ブロックの領域が割り当てられているかどうかがチェックされます。ダイナミック共通ブロックがまだ割り当てられていない場合は、そのチェック時に領域が割り当てられます。

次の例のコマンドラインでは、実行時に動的に割り当てられる共通ブロックの名前とともに、ダイナミック共用オプションを指定しています。

```
ifort -dyncom "blk1,blk2,blk3" test.f
```

`blk1`、`blk2`、および `blk3` は、動的にされる共通ブロックの名前です。

-dyncom オプションの使用ガイドライン

次に、`-dyncom` (ダイナミック共用) オプションを使用する場合に注意する必要のあるいくつかの制限を示します。

- ダイナミック COMMON 内のエンティティを、DATA 文で初期化してはいけません。
- 指定された COMMON ブロックだけがダイナミック COMMON として指定されます。
- ダイナミック COMMON 内のエンティティを、スタティック COMMON 内のエンティティまたは DATA で初期化された変数とともに EQUIVALENCE 式で使用してはいけません。

ダイナミック共通ブロックを使用する理由

ダイナミック共通ブロックを使用する主な理由は、独自の割り当てルーチンの提供によって共通ブロックの割り当てが制御できるからです。独自の割り当てルーチンを使用するには、そのルーチンを Fortran ランタイム・ライブラリの前にリンクする必要があります。このルーチンは、正しいルーチン名を生成するために、C 言語で記述されなければなりません。

このルーチンのプロトタイプは、次のとおりです。

```
void _FTN_ALLOC(void **mem, int *size, char *name);
```

ここで

- *mem* は、共通ブロックの基底ポインタの場所です。このポインタは、このルーチンによって、割り当てられるブロックメモリを指すように設定しなければなりません。
- *size* は、プログラム内で宣言され、コンパイラが共通ブロックに割り当てる必要があると判断したメモリのバイト数 (整数)。この値を無視して、目的に合った任意の値を使用できます。
  注: 割り当てる領域のサイズ (バイト数) を返す必要があります。`_FTN_ALLOC()` を呼び出すライブラリ・ルーチンにより、この共通ブロックの他のすべてのオカレンスが、割り当てた領域に適合することが保証されます。size パラメータの変更によって、割り当てる領域のサイズ (バイト数) を返します。
- *name* は、動的に割り当てられる共通ブロックの名前です。

ダイナミック共通ブロックへのメモリの割り当て

ランタイム・ライブラリ・ルーチン `f90_dyncom` は、メモリ割り当てを実行します。コンパイラは、ダイナミック共通ブロックを含むプログラム内の各ルーチンの開始時に、このルーチンを呼び出します。

次に、このライブラリ・ルーチンが _FTN_ALLOC() を呼び出して、メモリを割り当てます。デフォルトでは、コンパイラが、各ルーチンで宣言されたとおりの共通ブロックのサイズ (バイト数) を f90_dyncom に渡し、次にそのサイズが _FTN_ALLOC() に渡ります。別のルーチンにおいて異なるサイズで宣言された同じ名前の共通ブロックを持つ非標準の拡張を使用する場合、その共通ブロックの宣言を含むルーチンが呼び出される順序によっては、ランタイム・エラーが発生する可能性があります。

Fortarn ランタイム・ライブラリには、単に要求されたバイト数を割り当てて返す _FTN_ALLOC() のデフォルトのバージョンが含まれます。

4. 参照情報

コンパイラの制限

データの記憶容量、配列の大きさ、および実行ファイルの合計サイズは、システム・パラメータによって定められている、使用可能なプロセス仮想アドレス空間の量によってのみ制限されます。

プログラムのサイズ、コードの複雑さなどの、単一のインテル® Fortran プログラム・ユニットがもつ制限と、プログラム内の個々の文が持つ制限を以下の表に示します:

言語要素	制限
CALL または関数参照ごとの引数の実数	メモリの制限に依存
宣言式における関数参照の引数の数	255
配列の次元	7
配列構造のネスト数	20
次元ごとの配列要素数	9,223,372,036,854,775,807 = 2**31-1 (IA-32 システム) 2**63-1 (Itanium® ベース・システム) + メモリ構成による制限
定数: 文字定数および Hollerith 定数	7198 文字
定数: リスト指定 I/O での入力文字数	2048 文字
継続行数	511
データおよび I/O を含む DO のネスト数	7
DO と 整構造 IF 文のネスト (結合された) 数	128
DO ループのインデックス変数	9,223,372,036,854,775,807 = 2**63-1
編集記述子のネスト数	8
書式文の長さ	2048 文字
Fortran ソース行の幅	固定形式: 72 文字 (または `-extend_source` が有効な場合は 132 文字) 自由形式: 7200 文字
INCLUDE ファイルのネスト数	20 レベル
計算型または割り当て型 GOTO リストのラベル数	メモリの制限に依存
文ごとの字句トークン数	20000
名前付き共通ブロック数	メモリの制限に依存
配列構造を含む DO のネスト数	7
入力/出力を含む DO のネスト数	7
インターフェイス・ブロックのネスト数	メモリの制限に依存

DO、IF または CASE 構文のネスト数	メモリの制限に依存
括弧形式のネスト数	メモリの制限に依存
数値定数の桁数	メモリの制限に依存
式中の括弧のネスト数	メモリの制限に依存
構造体のネスト数	30
シンボル名の長さ	63 文字

大きなデータ・オブジェクトを処理する際のメモリ制限に関する詳細は、製品リリース・ノートを参照してください。

索引

O

P

Q

S

T

U

あ

い

お

か

き

け

こ

ま

も

ら

り

わ