FORTRAN 77 言語リファレンス

Sun WorkShop 6

Sun Microsystems, Inc.
901 San Antonio Road
Palo Alto, CA 94303
U.S.A. 650-960-1300

Part No. 806-4841-01
2000 年 6 月 Revision A

原典 : *FORTRAN 77 Language Reference (Sun WorkShop 6)*
Part No: 806-3594-10
Revision A

製品名の変更について

Sun は新しい開発製品戦略の一環として、Sun の開発ツール群の製品名を Sun WorkShop™ から Forte™ Developer に変更いたしました。製品自体の内容に変更はなく、従来通りの高品質をお届けいたします。

これまでの Sun の主力製品である基本プログラミングツールに、Forte Fusion™ や Forte™ for Java™ といった Forte 開発ツールの得意とする、マルチプラットフォームおよびビジネスアプリケーション実装の機能を盛り込むことで、より広範囲できめ細かな製品ラインが完成されました。

WorkShop 5.0 で使用されていた名称と、Forte Developer 6 で使用される新しい名称の対応については、以下の表をご覧ください。

旧名称	新名称
Sun Visual WorkShop™ C++	Forte™ C++ Enterprise Edition 6
Sun Visual WorkShop™ C++ Personal Edition	Forte™ C++ Personal Edition 6
Sun Performance WorkShop™ Fortran	Forte™ for High Performance Computing 6
Sun Performance WorkShop™ Fortran Personal Edition	Forte™ Fortran Desktop Edition 6
Sun WorkShop Professional™ C	Forte™ C 6
Sun WorkShop™ University Edition	Forte™ Developer University Edition 6

製品名の変更に加えて、次の 2 つの製品について大きな変更があります。

- Forte for High Performance Computing には Sun Performance WorkShop Fortran に含まれていたすべてのツール、および C++ コンパイラが含まれます。したがって、High Performance Computing のユーザーは開発用に 1 つの製品だけを購入すれば済むことになります。
- Forte Fortran Desktop Edition は以前の Sun Performance WorkShop Personal Edition と同じです。ただし、この製品に含まれる Fortran コンパイラでは、自動並列化されたコード、および明示的な指令に基づいた並列コードは生成できません。この機能は Forte for High Performance Computing に含まれる Fortran コンパイラでは使用できます。

Sun の開発製品を引き続きご利用いただきましてありがとうございます。今後もみなさまのご要望にお応えする製品をお届けできるよう努力してまいります。

目次

表目次

はじめに

このマニュアルでは、FORTRAN 77 プログラミング言語と、Sun WorkShop™ 6 `f77` コンパイラで対応する拡張機能について説明します。

このマニュアルは、Fortran と Solaris™ システムに関する実用的な知識を持つプログラマを対象にしています。

マルチプラットフォーム対応

この Sun WorkShop リリースは、Solaris 2.6、7、および 8 のオペレーティング環境 (SPARC™ プラットフォームおよび Intel プラットフォーム) をサポートしています。

特定のプラットフォームに対して、このリリースの `f77` コンパイラが提供されているかどうかは、Sun WorkShop `READMEs` ディレクトリにある Fortran 77 README ファイル `fortran_77` を参照してください (または、`f77 -xhelp=readme` コマンドを使用してください)。

Sun WorkShop 開発ツールへのアクセス方法

Sun WorkShop 製品コンポーネントとマニュアルページは標準ディレクトリ `/usr/bin` および `/usr/share/man` にはインストールされません。そのため `PATH` および `MANPATH` 環境変数を変更して Sun WorkShop コンパイラとツールにアクセスできるようにする必要があります。

`PATH` 環境変数を設定する必要があるかどうか判断するには以下を実行します。

1. **次のように入力して、PATH 変数の現在値を表示します。**

```
% echo $PATH
```

2. **出力内容から /opt/SUNWspro/bin を含むパスの文字列を検索します。**

パスがある場合は、PATH 変数は Sun WorkShop 開発ツールにアクセスできるように設定されています。パスがない場合は、この節の指示に従って、PATH 環境変数を設定してください。

MANPATH 環境変数を設定する必要があるかどうか判断するには以下を実行します。

1. **次のように入力して、workshop マニュアルページを表示します。**

```
% man workshop
```

2. **出力された場合、内容を確認します。**

workshop(1) マニュアルページが見つからないか、表示されたマニュアルページがインストールされたソフトウェアの現バージョンのものと異なる場合は、この節の指示に従って MANPATH 環境変数を設定してください。

注 – この節に記載されている情報は Sun WorkShop 6 製品が /opt ディレクトリにインストールされていることを想定しています。 Sun WorkShop ソフトウェアが /opt ディレクトリにインストールされていない場合は、システム管理者に連絡してください。

PATH 変数および MANPATH 変数は、C シェルを使用している場合はホームディレクトリの下の .cshrc ファイルに設定する必要があります。Bourne シェルか Korn シェルを使用している場合は、ホームディレクトリの下の .profile ファイルに設定する必要があります。

- Sun WorkShop コマンドを使用するには、 PATH 変数に以下を追加してください。

  /opt/SUNWspro/bin

- man コマンドで、Sun WorkShop マニュアルページにアクセスするには、MANPATH 変数に以下を追加してください。

  /opt/SUNWspro/man

PATH 変数についての詳細は、csh(1)、sh(1) および ksh(1) のマニュアルページを参照してください。MANPATH 変数についての詳細は、 man(1) のマニュアルページを参照してください。このリリースにアクセスするために PATH および MANPATH 変数を設定する方法の詳細は、『Sun WorkShop インストールガイド』を参照するか、システム管理者にお問い合わせください。

内容の紹介

このマニュアルは次の章と付録から構成されています。

第 1 章「Fortran の構成要素」では、Sun WorkShop FORTRAN 77 の基本的な構成要素を紹介しています。

第 2 章「データ型とデータ項目」では、Sun FORTRAN 77 のデータ型とデータ構造について説明しています。

第 3 章「式」では、Fortran の式について説明しています。

第 4 章「文」では、Sun WorkShop FORTRAN 77 コンパイラ f77 の文について説明しています。

第 5 章「入出力」では、Fortran 入出力の一般的な概念について説明しています。

第 6 章「組み込み関数」では、Sun WorkShop FORTRAN 77 の一部である組み込み関数を表に示します。

付録 A「ASCII 文字セット」では、ASCII 文字セットおよび制御文字に関する表を示します。

付録 B「文の例」では、f77 構文の型の選ばれた例を表に示します。

付録 C「データの表現」では、データ表現を簡単に説明しています。

付録 D「VMS 言語拡張」では、Fortran 77 がサポートする VMS 言語拡張について説明します。

書体と記号について

このマニュアルで使用している書体と記号について説明します。

表 P-1　このマニュアルで使用している書体と記号

書体または記号	意味	例
AaBbCc123	コマンド名、ファイル名、ディレクトリ名、画面上のコンピュータ出力、コーディング例。	.login ファイルを編集します。 ls -a を使用してすべてのファイルを表示します。 machine_name% You have mail.
AaBbCc123	ユーザーが入力する文字を、画面上のコンピュータ出力と区別して表わします。	machine_name% **su** Password:
AaBbCc123 または ゴシック	コマンド行の可変部分。実際の名前または実際の値と置き換えてください。	rm *filename* と入力します。 rm **ファイル名** と入力します。
『 』	参照する書名を示します。	『SPARCstorage Array ユーザーマニュアル』
「 」	参照する章、節、または、強調する語を示します。	第 6 章「データの管理」を参照してください。 この操作ができるのは、「スーパーユーザー」だけです。
\	枠で囲まれたコード例で、テキストがページ行幅を超える場合、バックスラッシュは、継続を示します。	machinename% grep '^#define \ XV_VERSION_STRING'
➤	階層メニューのサブメニューを選択することを示します。	作成: 「返信」➤「送信者へ」

- 小さな三角 (Δ) は、意味のある空白を示します。

```
ΔΔ36.001
```

- 標準外の機能には、シンボル記号“♦”を付けます。標準規格については、第 1 章に記述されています。
- FORTRAN 77 の例は原則として、固定カラムではなくタブ書式で示します。ソース行フォーマットの詳細については、『Fortran ユーザーズガイド』を参照してください。
- FORTRAN 77 規格では、「FORTRAN」とすべて大文字で表記する旧表記規則を使用しています。サンのマニュアルでは FORTRAN と Fortran の両方を使用しています。現在の表記規則では、「Fortran 95」と小文字を使用しています。

シェルプロンプトについて

シェルプロンプトの例を以下に示します。

表 P-2　シェルプロンプト

シェル	プロンプト
UNIX の C シェル	`machine_name%`
UNIX の Bourne シェルと Korn シェル	`machine_name$`
スーパーユーザー (シェルの種類を問わない)	`#`

関連マニュアル

以下の方法で、関連マニュアルにアクセスすることができます。

- インターネットの `docs.sun.com` の Web サイトからアクセスできます。特定の本のタイトルで検索するか、主題、マニュアルコレクションまたは製品別にブラウズすることができます。

  `http://docs.sun.com`

- ローカルシステムまたはローカルネットワークにインストールされた Sun WorkShop 製品からアクセスできます。Sun WorkShop 6 HTML 文書 (マニュアル、オンラインヘルプ、マニュアルページ、各コンポーネントの README ファイル、リリースノート) が、インストールした Sun WorkShop 6 製品から参照可能です。HTML 文書にアクセスするには、次のいずれかを実行します。
  - Sun WorkShop または Sun WorkShop™ TeamWare ウィンドウで、「ヘルプ」➤「オンラインマニュアルについて」を選択します。
  - Netscape™ Communicator 4.0 またはその互換バージョンのブラウザで、以下のファイルを開きます。

    `/opt/SUNWspro/docs/ja/index.html`

  参照できる Sun WorkShop 6 HTML 文書の一覧がブラウザに表示されます。一覧にあるマニュアルを開くには、マニュアルのタイトルをクリックしてください。

表 P-3 は、Sun WorkShop 6 関連マニュアルをマニュアルコレクション別に一覧にしたものです。

表 P-3 マニュアルコレクション別 Sun WorkShop 6 関連マニュアル

マニュアルコレクション	マニュアルタイトル	内容の説明
Forte Developer 6 / Sun WorkShop 6 リリースマニュアル	Sun WorkShop 6 マニュアルの概要	Sun WorkShop 6 で使用可能なマニュアルとそのアクセス方法について説明しています。
	Sun WorkShop の新機能	Sun WorkShop 6 の現在のリリースと以前のリリースでの新機能についての情報を記載しています。
	Sun WorkShop 6 リリースノート	インストールの詳細と Sun WorkShop 6 最終リリースの直前に判明した情報を記載しています。このマニュアルはコンポーネントごとの README ファイルにある情報を補足するものです。

表 P-3　マニュアルコレクション別 Sun WorkShop 6 関連マニュアル

マニュアルコレクション	マニュアルタイトル	内容の説明
Forte Developer 6 / Sun WorkShop 6	プログラムのパフォーマンス解析	新しい標本コレクタと標本アナラザの使い方について説明しています (上級者向けのプロファイリング事例と説明付き)。コマンド行解析ツール `er_print`、ループツール、ループレポートユーティリティおよび UNIX プロファイルツール `prof`、`gprof`、`tcov` についての情報も含んでいます。
	dbx コマンドによるデバッグ	dbx コマンドを使ってプログラムをデバッグする方法について説明しています。参考情報として、同じデバッグ処理を Sun WorkShop デバッグウィンドウを使って実行する方法も記載しています。
	Sun WorkShop の概要	Sun WorkShop 統合プログラミング環境の基本的なプログラム開発機能について説明しています。
Forte C 6 / Sun WorkShop 6 Compilers C	C ユーザーズガイド	C コンパイラオプション、サン固有の機能 (プラグマ、`lint` ツール、並列化、64 ビットオペレーティングシステムへの移行および ANSI/ISO 準拠 C) について説明しています。
Forte C++ 6 / Sun WorkShop 6 Compilers C++	C++ ライブラリ・リファレンス	C++ ライブラリについて説明しています。C++ 標準ライブラリ、Tools.h++ クラスライブラリ、Sun WorkShop Memory Monitor、`Iostream` および複素数の情報も含みます。
	C++ 移行ガイド	コードを本バージョンの Sun WorkShop C++ コンパイラに移行する方法について説明しています。

表 P-3　マニュアルコレクション別 Sun WorkShop 6 関連マニュアル

マニュアルコレクション	マニュアルタイトル	内容の説明
	C++ プログラミングガイド	新しい機能を使ってより効率的なプログラムを記述する方法について説明しています。テンプレート、例外処理、実行時の型識別、キャスト演算、パフォーマンス、およびマルチスレッド対応のプログラムに関する情報も記載されています。
	C++ ユーザーズガイド	コマンド行オプションとコンパイラの使い方についての情報を記載しています。
	Sun WorkShop Memory Monitor ユーザーズガイド	C および C++ のメモリー管理で生じた問題を Sun WorkShop Memory Monitor で解決する方法について説明しています。このマニュアルはインストールした製品 (`/opt/SUNWspro/docs/ja/index.html`) からのみ参照可能で、`docs.sun.com` Web サイトで参照することはできません。
Forte for High Performance Computing 6 / Sun WorkShop 6 Compilers Fortran 77/95	Fortran ライブラリ・リファレンス	Fortran コンパイラによって提供されるライブラリルーチンの詳細について説明しています。
	Fortran プログラミングガイド	入出力、ライブラリ、プログラム分析、デバッグおよびパフォーマンスに関連する内容を記述しています。
	Fortran ユーザーズガイド	コマンド行オプションとコンパイラの使い方についての情報を記載しています。
	FORTRAN 77 言語リファレンス	Fortran 77 言語の包括的な参照情報を記載しています。
	Fortran 95 区間演算プログラミングリファレンス	Fortran 95 コンパイラによってサポートされる組み込み `INTERVAL` データについて説明しています。
Forte TeamWare 6 / Sun WorkShop TeamWare 6	Sun WorkShop TeamWare ユーザーズガイド	Sun WorkShop TeamWare コード管理ツールの使用方法について説明しています。

表 P-3　マニュアルコレクション別 Sun WorkShop 6 関連マニュアル

マニュアルコレクション	マニュアルタイトル	内容の説明
Forte Developer 6/ Sun WorkShop Visual 6	Sun WorkShop Visual ユーザーズガイド	C++ と Java™ の GUI (グラフィカルユーザーインタフェース) を Sun WorkShop Visual を使用して作成する方法について説明しています。このマニュアルには、旧リリース (Sun WorkShop Visual 5.0) から変更のない機能が記載されています。
	Sun WorkShop Visual の新機能	Sun WorkShop Visual 6.0 で追加または変更された機能について説明しています。
Forte / Sun Performance Library 6	Sun Performance Library Reference (英語のみ)	コンピュータによる線形代数および高速フーリエ変換を実行するサブルーチンと関数の最適化ライブラリについて説明しています。
	Sun Performance Library User's Guide (英語のみ)	線形代数で発生した問題の解決に使用されるサブルーチンと関数のコレクションである Sun Performance Library のサン固有の機能の使用方法について説明しています。
数値計算ガイド	数値計算ガイド	浮動小数点演算における数値の精度に関する問題について説明しています。
標準ライブラリ 2	Standard C++ Library Class Reference (英語のみ)	標準 C++ の詳細について説明しています。
	標準 C++ ライブラリ・ユーザーズガイド	標準 C++ ライブラリの使用方法について説明しています。
Tools.h++ 7	Tools.h++ 7.0 ユーザーズガイド	Tools.h++ クラスライブラリの詳細について説明しています。
	Tools.h++ 7.0 クラスライブラリ・リファレンスマニュアル	C++ クラスを使用して、プログラム効率を向上させる方法について説明しています。

表 P-4 は、docs.sun.com の Web サイトからアクセスできる Solaris 関連マニュアルの一覧です。

表 P-4 Solaris 関連マニュアル

マニュアルコレクション	マニュアルタイトル	内容の説明
Solaris ソフトウェア開発	リンカーとライブラリ	Solaris リンクエディタと実行時リンカーの操作およびそれらが操作するオブジェクトについて説明しています。
	プログラミングユーティリティ	Solaris オペレーティング環境で使用可能な特殊組み込みプログラミングツールに関する開発者向けの情報を記載しています。

第1章

Fortran の構成要素

この章では、Sun WorkShop FORTRAN 77 の基本的な構成要素を紹介しています。

規格への準拠

- `f77` は、ANSI X3.9-1978 FORTRAN 規格およびこれに対応する国際標準化機構 (ISO) 1539-1980 規格に準拠しています。そのほか、FIPS 69-1、BS 6832、MIL-STD-1753 にも準拠しています。(これは現在の Fortran 規格ではありません。)
- 両コンパイラの浮動小数点演算は、IEEE 754-1985 規格および国際規格の IEC 60559:1989 に準拠しています。
- SPARC プラットフォームでは、どちらのコンパイラも SPARC V8 と SPARC V9 の最適化活用機能をサポートすると同時に、UltraSPARC™ も実装しています。これらの機能は、『SPARC アーキテクチャマニュアル バージョン 8』(トッパン刊) およびバージョン 9 (ISBN 0-13-099227-5) に定義されています。
- このマニュアルでは、上に挙げた各種規格のバージョンに適合していることを「規格」と呼び、これらの規格の範囲外の機能を「規格外」または「拡張機能」と呼んでいます。

上記の規格は、標準化団体の責任によって改訂される場合があります。また、コンパイラが準拠すべき規格のバージョンに改訂または変更が生じた場合は、今後リリースされる Fortran コンパイラの機能と旧リリースとの互換性がなくなる可能性もあります。

拡張

標準の FORTRAN 77 言語では、再帰、ポインタ、倍精度複素数、4 倍精度実数、4 倍精度複素数などが拡張され、さらに NAMELIST、DO WHILE、構造体、記録、共用体、マップ、変数書式などの多くの VAX とVMS Fortran 5.0 も拡張されています。マルチプロセッサ Fortran では、自動および明示的なループの並列化処理が行えます。

Sun FORTRAN 77 では VMS 拡張機能を使用することができるので、VAX システム用に書かれたプログラムを、Solaris で実行できるように簡単に移植することができます。

このマニュアルでは、Sun f77 に実装されている機能のうち、1 ページの「規格への準拠」に当てはまらない機能については、特別な記号 (♦) を付けて説明しています。

基本概念

Fortran の基本的な用語と概念は以下のとおりです。

- プログラムは、1 つ以上のプログラム単位で構成されます。
- プログラム単位は、文の連続で、END で終わります。
- 文は、0 個以上のキーワード、英字名、定数、文番号、演算子、特殊文字で構成されます。
- 各キーワード、英字名、定数、および演算子は、Fortran 文字セットの 1 つ以上の文字から構成されます。
- 文字定数には、任意の ASCII 文字を使用できます。
- 文番号は、1 から 5 桁の数字で構成されますが、ゼロ以外の数字が最低 1 つは必要です。

文字セット

文字セットは、以下のもので構成されます。

- 大文字の A から Z、小文字の a から z

- 数字の 0 から 9
- 以下の表の特殊文字

表 1-1　特殊文字の用途

文字	名前	用途
	スペース	文字定数の一部になる場合を除き、文中では無視される
	タブ	タブフォーマットのソース行を設定する ♦
=	等号	代入
+	プラス	加算、単項演算子
–	マイナス	減算、単項演算子
*	アスタリスク	乗算、選択戻り、注釈、べき乗、stdin、stdout、並びによる入出力
/	スラッシュ	除算、データの区切り、名前付き共通ブロック、構造体、記録の終わり
( )	括弧	式、複素定数、記憶列共有結合、パラメータ、または implicit 文でのグループ、書式、引数の並び、および 添字を囲む
,	コンマ	データ、式、複素定数、結合グループ、書式、引数の並びおよび添字を分離する分離文字
.	ピリオド	小数点、論理定数と演算子の区切り文字、記録欄
'	アポストロフィ	引用符付き文字定数
"	引用符	引用符付き文字定数、8 進定数 ♦
$	ドル記号	変数群 (ネームリスト) 入力用区切り文字、編集記述子、指令 ♦
!	感嘆符	注釈 ♦
:	コロン	配列宣言子、部分列、編集記述子
%	パーセント	特殊関数: %REF、%VAL、%LOC ♦
&	アンパサンド	継続、選択戻り、変数群入力用区切り文字。 第 1 桁で使用する場合は、行をタブフォーマットのソース行に設定する ♦
?	疑問符	変数群中の名前を要求する ♦
\	バックスラッシュ	エスケープ文字 ♦
< >	角括弧	書式中で変数式を囲む ♦

以下の使用法と制約があります。

- 大文字小文字は、Fortran の文のキーワードや英字名では、区別されません。`f77` の `-U` オプションを使用すると、英字名の大小文字は区別されます。♦
- 制御文字は文字セットには含まれませんが、データとしては大部分が許可されます。ただし Control-A、Control-B、Control-C の 3 文字はデータとして使用できません。これらの文字は、`char()` 関数を使用するなどの方法でプログラム中に取り入れることはできます。♦
- 文字列では、任意の ASCII 文字が文字データとして有効です。♦

バックスラッシュ文字 (`\`) について、エスケープシーケンスかコンパイルオプション `-xl` が必要なことがあります。改行文字 (`\n`) については、エスケープシーケンスを使用します。27 ページの表 2-3 を参照してください。

英字名

次の項目には英字名を付けることができます。

表 1-2　英字名が付く項目

定数	名前付き共通ブロック
変数	変数群 (ネームリスト) ♦
配列	主プログラム (Main programs)
構造体 ♦	初期値設定プログラム単位 (Block data)
記録 ♦	
関数 ♦	サブルーチン (Subroutines)
	関数
	入口

英字名には、以下の制約があります。

- 文字数に制限はありません。♦ 標準は 6 です。
- アルファベット、数字、ドル記号 `($)`、下線文字 (_) で構成されます。`$` と `_` は規格外です。♦

- 通常、先頭はアルファベットです。数字やドル記号 ($) で始めることはできません。下線 (_) で始めることはできますが、Fortran およびシステムのライブラリで使用されている名前と重複する可能性があるので、使用しない方が安全です。

注 – 2 本の下線で始まる手続き名は、コンパイラ内部で特別にサポートされている関数とみなされます。先頭に 2 本の下線を使用して関数やサブルーチンを命名すること
(_ _xfunc など)は避けてください。コンパイラの使用方法と重複してしまいます。♦

- 大小文字を区別しません。コンパイラが全部小文字に変換します。f77 のコマンド行の -U オプションを使用すると、デフォルトが変更されてソースファイルに使われている大文字はそのままになります。♦

例： デフォルトでは、次のどちらを用いても結果は同じです。

```
ATAD = 1.0E-6
Atad = 1.0e-6
```

- スペースには意味がありません。

例： 次のどの形式をとっても結果は同じです。

```
IF ( X .LT. ATAD ) GO TO 9
IF ( X .LT. A TAD ) GO TO 9
IF(X.LT.ATAD)GOTO9
```

次に英字名の例を示します。

表 1-3 英字名の例

有効	無効	無効の理由
X2	2X	先頭に数字がある
DELTA_TEMP	_ _DELTA_TEMP	先頭に _ _がある (コンパイラ用)
Y$Dot	Y | Dot	正しくない文字 | がある

- 一般的には、1 つのプログラム単位の中で、別の項目に同じ英字名を付けることはできません。

  次は例外です。
  - 変数または配列には、共通ブロックと同じ名前を付けることができます。
  - 記録欄には、構造体と同じ名前を付けることができます。♦
  - 記録欄には、その構造体の別のレベルにある欄と同じ名前を付けることができます。♦

- プログラム単位が 2 つ以上あるプログラム全体で、以下のうちの 2 つに同じ名前を使用することはできません。
  - 初期値設定副プログラム
  - 共通ブロック
  - 入口
  - 関数副プログラム
  - 主プログラム
  - サブルーチン

プログラム単位

プログラム単位は一連の文であり、END 文で終わります。各プログラム単位は、主プログラムか副プログラムのどちらかです。プログラムを実行可能にするには、主プログラムが必要です。

副プログラムには 3 種類あります。サブルーチン、関数、初期値設定副プログラムです。サブルーチンと関数は手続きと呼ばれ、他の手続きまたは主プログラムから呼び出されます。初期値設定副プログラムはローダーによって処理されます。

文

文は 1 つ以上のキーワード、英字名、定数、演算子で構成され、適切な句読文字が入ります。Fortran 77 では、どのキーワードも予約語ではありません。文関数と代入文を除いて、すべての文はキーワードで始まります。

実行文と非実行文

文には実行文と非実行文があります。

一般には、実行時の動作を指定する文を実行文といいます。そうでないものを非実行文といいます。

非実行文は型やサイズのような属性の指定、並び方や順序の規定、データの初期値の定義、編集命令の指定、文関数の定義、プログラム単位の分類、および入口の定義をします。一般に非実行文は、最初の実行文の実行前に解析されます。

Fortran 77 文

表 1-4 Fortran 77 文

ACCEPT*	DOUBLE COMPLEX	GOTO (割り当て型)*	PRINT*
ASSIGN*	DOUBLE PRECISION	GOTO (単純)*	PRAGMA
代入*	ELSE*	IF (算術)*	PROGRAM
AUTOMATIC	ELSE IF*	IF (ブロック)*	REAL
BACKSPACE*	ENCODE*	IF (論理)*	RECORD
BLOCK DATA	END*	IMPLICIT	RETURN*
BYTE	END DO*	INCLUDE	REWIND*
CALL*	END FILE*	INQUIRE*	SAVE
CHARACTER	END IF*	INTEGER	文関数
CLOSE*	END MAP	INTRINSIC	STATIC*
COMMON	END STRUCTURE	LOGICAL	STOP*
COMPLEX	END UNION	MAP	STRUCTURE
CONTINUE*	ENTRY	NAMELIST	SUBROUTINE
DATA	EQUIVALENCE	OPEN*	TYPE
DECODE*	EXTERNAL	OPTIONS	UNION
DIMENSION	FORMAT	PARAMETER	VIRTUAL
DO*	FUNCTION	PAUSE*	VOLATILE
DO WHILE*	GOTO*	POINTER	WRITE*

アスタリスク (*) は実行文を示します。

ソース行形式

文は 1 行以上で構成されます。第 1 行を開始行、それに続く行を継続行といいます。

ソース行には次の 2 つの形式があります。

- 標準固定フォーマット
- タブフォーマット ♦

標準固定フォーマット

標準固定フォーマットのソース行の定義は、次のとおりです。

- 各行の最初の 72 桁が有効です。9 ページの「拡張行」を参照してください。
- 最初の 5 桁は空白か、または数値文番号です。
- 6 桁目に空白もゼロもない行が継続行です。
- 短い行はパディングされて、72 文字になります。
- 長い行は短く切られます。9 ページの「拡張行」を参照してください。

タブフォーマット

タブフォーマットのソース行の定義は次のとおりです。♦

- 第 1 桁から第 6 桁のどこかにタブがあるか、または第 1 桁にアンパサンドがあると、タブフォーマットのソース行になります。
- タブが最初の非空白文字の場合、タブに続くテキストは、7 桁目から始まるとみなされます。
- タブの前に注釈指示文字または文番号を入れることができます。
- 第 1 桁にアンパサンド (`&`) があるか、最初のタブの後にゼロ以外の数字がある行が継続行です。

フォーマットの混用

同じプログラム単位内では両方のフォーマットを混用できますが、同じ行内ではできません。

継続行

継続行のデフォルトの最大数は 99 (1 開始行と 99 の継続行) です。♦

この行数を変えるには、`-Nl`*n* オプションを使用します。♦

拡張行

ソース行の長さを 132 文字に延ばすには -e オプションを使います。♦
そうでない場合はデフォルトによって、f77 は 72 桁目より後の文字を無視します。
行の長さを延ばす例を示します。

```
demo% f77 -e prog.f
```

パディング

DATA 文中の以下に示すような行ではパディングは重要です。

```
C         1         2         3         4         5         6         7
C234567890123456789012345678901234567890123456789012345678901234567890123456789012
       DATA SIXTYH/60H
     1                              /
```

注釈と空白行

1 桁目に c、C、*、d、D、! のいずれかがある行は注釈行です。-xld オプションが指定されていれば、D か d で始まる行はデバッグ行としてコンパイルされます。d、D、! は規格外です。♦

文欄のいずれかの桁に感嘆符 (!) を入れると、文字定数表現の中を除いて、その行の ! の後は全部注釈になります。♦

すべて空白の行は注釈行です。

例： c、C、d、D、*、!、および空白の注釈

```
c   式解析の始まり
    CHARACTER S, STACK*80
    COMMON /PRMS/ N, S, STACK
    …
*   式を解読:
    IF ( S .GE. '0' .AND. S .LE. '9' ) THEN ! 行末コメント
    CALL PUSH ! スタックに保存。行末コメント
d   PRINT *, S! デバッグコメントと行末コメント
    ELSE
    CALL TOLOWER ! 小文字の行末コメント
    END IF
D   PRINT *, N!  デバッグコメントと行末コメント
    …
C   終わり
!   式解析
```

指令

指令は注釈の特殊な形式で、コンパイラに情報を伝えます。♦ 指令はコンパイラ指令とも呼ばれます。指令には次の 2 種類があります。

- 一般的な指令
- 並列化の指令

f77 で使用可能な指令の詳細については、『Fortran ユーザーズガイド』および『Fortran プログラミングガイド』を参照してください。

一般的な指令

指令の一般形式は次のいずれかです。♦

- C$PRAGMA *id*
- C$PRAGMA *id* (*a* [, *a*] ...) [, *id* (*a* [, *a*] ...)] ,...
- C$PRAGMA SUN *id* [=*options*]

id は指令キーワードの識別子で、*a* は引数です。

構文

注釈中に指令を次のように入れることができます。

- 第 1 桁に注釈指示文字 `c`、`C`、`!`、`*` のいずれかを入れます。
- 任意の桁に注釈指示文字 `!` を入れます。
- 続けて `$PRAGMA` の 7 文字を入れます。空白は入れられませんが、大小文字の使用は自由です。

規則と制約

先頭の 8 文字の後、空白は無視され、Fortran のテキスト同様に、大文字小文字は同等になります。

注釈であるため、指令は継続できません。しかし、必要であれば続けて多くの `C$PRAGMA` 行を付けられます。

コメントが上記の構文の条件を満たしていると、コンパイラは認識された命令を 1 つ以上含むと考えます。そうでなければ、警告が出されます。

並列化指令

並列化指令は、その後に続く DO ループの並列化をコンパイラに実行させる明示的な要求です。並列化指令の構文は、一般指令の構文と異なります。

並列化指令は、コンパイル オプション `-parallel` または `-explicitpar` が使用された場合のみ認識されます。`f77` で使用される並列化オプションについては、『Fortran プログラミングガイド』を参照してください。

並列化指令は、以下のような構文になります。

- 先頭文字は 1 桁目に入ります。
- 先頭文字には、`c`、`C`、`*`、`!` のいずれかが入ります。
- 続けて `$PAR` の 4 文字が入ります。空白は使用できませんが、大小文字の使用は自由です。
- 次に、空白で区切った指令キーワードとオプションが入ります。

明示的並列化指令のキーワードには、以下のものがあります。

`TASKCOMMON`、`DOALL`、`DOSERIAL`、`DOSERIAL*`

各並列化指令のキーワードの後には、任意で独自の修飾子を入れることができます。

例 : 共用変数のループは次のように指定します。

```
C$PAR DOALL SHARED(yvalue)
```

並列化および並列化指令の詳細については、『Fortran プログラミングガイド』を参照してください。Fortran 並列化機能には、Sun WorkShop HPC ライセンスが必要です。

第2章

データ型とデータ項目

この章では、Sun FORTRAN 77 のデータ型とデータ構造について説明します。

規格外の機能には、普通小さな塗り潰した菱形 (♦) が付きます。

データの型

通常、定数、定数式、変数、配列、配列要素、部分列、および関数は、 型を持つデータを表しています (型のない定数は例外です)。

一方、プログラムまたはサブルーチン、初期値設定副プログラム、共通ブロック、変数群、構造化記録は、データ型をもっていません。

データの型に関する規則

名前により型が決定されます。つまりデータや関数の名前により、以下のデータ型規則に従って、明示的または暗黙的にデータの型が決まります。

- 定数、変数、配列、および関数の英字名が持つデータの型は、総称関数以外は、1 つのプログラム単位について 1 つだけです。
- 型宣言文で明示的に名前の型を宣言するとデータの型が決まります。
- 型宣言文で明示的に名前の型を宣言しないと、名前の最初のアルファベットがデータの型を暗黙的に決めます。
- デフォルトの暗黙の型宣言規則では、名前の最初のアルファベットが I、J、K、L、M、N のいずれかであればそのデータ型は整数であり、それ以外の文字であれば実数です。

- IMPLICIT 文を使用して、デフォルトの暗黙の型宣言を変更できます。IMPLICIT NONE 文を使用して、暗黙の型宣言をすべて無効にすることもできます。コマンド行にコンパイラフラグ -u を指定して、暗黙の型宣言をすべて無効にすることもできます。この方法は、各プログラム単位を IMPLICIT NONE 文で始める方法と結果が同じです。

配列要素

配列要素の型は配列名の型と同じです。

関数

組み込み関数は、あらかじめ指定された型を持っています。組み込み関数には、明示的な型宣言文は必要ありませんが、型宣言をすることもできます。総称関数には、前もって決められた型はなく、311 ページの「組み込み関数」で説明しているように、引数の型で決まります。

外部関数の型は以下のいずれかの方法で指定できます。

- 型宣言文で明示的に名前を指定します。
- FUNCTION 文で明示的に FUNCTION という語の前にデータの型名を付けます。
- 変数の場合のように名前で暗黙的に示します。

例：型宣言文で明示的に名前を指定します。

```
      FUNCTION F ( X )
      INTEGER F, X
      F = X + 1
      RETURN
      END
```

例：FUNCTION 文で明示的に名前を指定します。

```
      INTEGER FUNCTION F ( X )
      INTEGER X
      F = X + 1
      RETURN
      END
```

例：変数の場合のように名前で暗黙的に示します。

```
FUNCTION NXT ( X )
INTEGER X
NXT = X + 1
RETURN
END
```

暗黙の型宣言は関数の型にも有効です。デフォルトでの暗黙の型宣言または IMPLICIT 文によって関数の型を決めます。関数の副プログラムとそれを呼び出すプログラム単位とで関数の型を同じにするのは、ユーザーの責任です。つまり、f77 コンパイラ はプログラム単位間の型の一致を調べません。

データの型の性質

本節では、Sun FORTRAN 77 のデータ型について説明しています。

データサイズを明示的に宣言しないデフォルトのデータ宣言は、特定のコンパイラオプションによって、その意味が変わってくる場合があります。22 ページの「データ型のサイズと整列条件」の節では、データのサイズと整列条件とこれらのオプションの作用について説明します。

BYTE ♦

データの型 BYTE は、記憶領域を 1 バイトだけ使用する型です。これは論理型で、LOGICAL *1 と同義です。

BYTE 型の変数は以下の値のいずれかをとることができます。

- 8 ビットの 1 つのデータ項目
- -128 から 127 までの整数
- 論理値 .TRUE. または .FALSE.

論理値と解釈された場合、0 は .FALSE. を表し、その他の値はすべて .TRUE. です。

f77 では、配列の索引として (REAL 型と同様に) BYTE 型を使用できますが、DO ループの索引としては使用できません (INTEGER、REAL、DOUBLE PRECISION だけが使用できます)。コンパイラが INTEGER の明示的チェックを必要とする場合、BYTE は使用できません。

例を示します。

```
    BYTE Bit3 / 8 /, C1 / 'W' /,
&    Counter / 0 /, Switch / .FALSE. /
```

BYTE 項目は 1 バイト (8 ビット) の記憶領域を占め、 1 バイトの境界を割り当てられます。

CHARACTER

文字データ型 CHARACTER は、CHARACTER*1 と同義で 1 文字のデータを表現します。

文字はアポストロフィ (') または引用符 (") で囲みます。♦ 引用符 (") は規格外であり、-xl オプションを指定してコンパイルする場合は、引用符は別の意味になるので、文字列を囲むにはアポストロフィを使用しなければなりません。

CHARACTER 型のデータには符号が付きません。CHARACTER 項目は 1 バイト (8 ビット) の記憶領域を占め、1 バイト境界に割り当てられます。

CHARACTER *n

文字列データ型 CHARACTER*n (n > 0) は n 個の文字の列を表現します。

CHARACTER*n のデータ型は n バイトの記憶領域を占め、1 バイトの境界に割り当てられます。

文字列定数は 2 バイトの境界に割り当てられ、DATA 文中でない場合は、C ルーチンとのデータの受け渡しを容易にするため、その後に空文字が 1 つ付きます。

COMPLEX

複素数データは複素数の近似値です。複素数データ型 COMPLEX は普通 COMPLEX*8 と同義で、複素数を表す一対の REAL*4 の値です。前の要素は実部を、後の要素は虚部を表します。

サイズを指定しない場合の COMPLEX 項目のサイズはデフォルトで 8 バイトです。デフォルトでは 4 バイト境界に割り当てられます。しかし、特殊なオプションを指定してコンパイルすれば、これらのデフォルトを変更できます。(22 ページの「データ型のサイズと整列条件」を参照)

COMPLEX*8 ♦

複素数データ型 COMPLEX*8 は 普通 COMPLEX と同義ですが、ただしコンパイラオプションに関係なく、サイズは 8 バイトです。

COMPLEX*16 (倍精度複素数) ♦

複素数データ型 COMPLEX*16 は、DOUBLE COMPLEX と同義ですが、コンパイラオプションに関係なく、サイズは常に 16 バイトです。

COMPLEX*32 (4 倍精度複素数) ♦

(SPARC のみ) 複素数データ型 COMPLEX*32 は 4 倍精度複素数です。これは一対の REAL*16 要素で構成されます。各要素には符号ビット、15 ビットの指数部、112 ビットの小数部があります。f77 では、REAL*16 要素は IEEE 規格に準拠します。

COMPLEX*32 項目のサイズは 32 バイトです。

DOUBLE COMPLEX ♦

複素数データ型 DOUBLE COMPLEX は、普通 COMPLEX*16 と同義で、複素数を表す一対の DOUBLE PRECISION (REAL*8) の値です。前の要素は実部を、後の要素は虚部を表します。

サイズを指定しない場合の DOUBLE COMPLEX 項目のサイズはデフォルトで 16 です。

DOUBLE PRECISION

倍精度データは実数の近似値です。倍精度データ型 DOUBLE PRECISION は、REAL*8 と同義で、倍精度のデータを表現します。

サイズを指定しない場合の DOUBLE PRECISION 項目のサイズはデフォルトで 8 バイトです。

DOUBLE PRECISION の要素には符号ビット、11 ビットの指数部、52 ビットの小数部があります。f77 では、DOUBLE PRECISION 要素は、倍精度浮動小数点データに関する IEEE 規格に準拠します。データのレイアウトについては、353 ページの「データの表現」を参照してください。

INTEGER

整数データ型 INTEGER は符号付き整数値を表現します。

サイズを指定しない場合の INTEGER 項目のサイズはデフォルトで 4 です。INTEGER は 4 バイト境界に割り当てられます。しかし、特殊なオプションを指定してコンパイルすれば、これらのデフォルトを変更できます。(22 ページの「データ型のサイズと整列条件」を参照)

INTEGER*2 ♦

短い整数データ型 INTEGER*2 は符号付き整数値を表現します。INTEGER*2 型のオブジェクトだけを持つ式の型は INTEGER*2 です。この機能を使用すると、性能に影響することがありますので注意してください。

総称関数は、デフォルトの整数型に従って、短いか、または長い整数を返します。-i2 フラグを使用して手続きをコンパイルすると、その大きさに収まる整定数の全部と (サイズは明示されていない) INTEGER 型の変数の全部が INTEGER*2 型になります。整数値の組み込み関数の精度が総称関数の規則で決まらない場合、-i2 オプションが有効な時に優先される長さ (INTEGER*2) を戻すものが選ばれます。-i2 オプションが有効なとき、LOGICAL 量のデフォルトの長さは 2 バイトです。

通常の整数は、Fortran 77 の規則に従い、REAL 変数と同じスペースを占めます。これらの整数は C 言語の long int 型相当であり、2 バイトの整数は C 言語の short int 型に相当します。これらの短い整数と論理量は、記憶領域割り当ての標準規則には従いません。

INTEGER*2 は 2 バイトを占めます。

INTEGER*2 は 2 バイト境界に割り当てられます。

INTEGER*4 ♦

整数データ型 INTEGER*4 は符号付き整数値を表現します。

INTEGER*4 は 4 バイトを占めます。

INTEGER*4 は 4 バイト境界に割り当てられます。

INTEGER*8 ♦

整数データ型 INTEGER*8 は符号付き 64 ビット整数を表現します。これは、-dbl オプションが設定されている場合のみです。

INTEGER*8 は 8 バイトを占めます。

INTEGER*8 は 8 バイト境界に割り当てられます。

LOGICAL

論理データ型 LOGICAL は、論理値 .TRUE. または .FALSE. を保持します。値 0 は .FALSE. を表し、他の値はすべて .TRUE. を表します。

サイズを指定していない場合の LOGICAL 項目の通常のデフォルトのサイズは 4 です。LOGICAL は 4 バイト境界に割り当てられます。しかし、特殊なオプションを指定してコンパイルすれば、これらのデフォルトを変更できます。

LOGICAL*1 ♦

1 バイトの論理データ型 LOGICAL*1 は BYTE と同義で、以下の値のいずれかをとることができます。

- 1 文字
- -128 から 127 までの整数
- 論理値 .TRUE. または .FALSE.

値については LOGICAL の定義と同様ですが、LOGICAL*1 は 1 文字または小さい整数を表現できます。例を示します。

```
      LOGICAL*1 Bit3 / 8 /, C1 / 'W' /,
&     Counter / 0 /, Switch / .FALSE. /
```

LOGICAL*1 項目は 1 バイトの記憶領域を占めます。

LOGICAL*1 は 1 バイトの境界に割り当てられます。

LOGICAL*2 ♦

データ型 LOGICAL*2 は、論理値 .TRUE. または .FALSE. をとります。値は LOGICAL の場合と同様に定義されています。

LOGICAL*2 は 2 バイトを占めます。

LOGICAL*2 は 2 バイトの境界に割り当てられます。

LOGICAL*4 ♦

論理データ型 LOGICAL*4 は、論理値 .TRUE. または .FLASE. をとります。値は LOGICAL について定義されている値と同じです。

LOGICAL*4 は 4 バイトを占めます。

LOGICAL*4 は 4 バイト境界に割り当てられます。

LOGICAL*8 ♦

論理データ型 LOGICAL*8 は、論理値 .TRUE. または .FLASE. をとります。値は LOGICAL データ型について定義されているものと同じです。

LOGICAL*8 は 8 バイトを占めます。

LOGICAL*8 は 8 バイト境界に割り当てられます。

REAL

実数データは実数の近似値です。実数データ型 REAL は通常、REAL*4 と同義で、実数データを表現します。

サイズを指定していない REAL 項目のデフォルトのサイズは通常 4 バイトです。REAL は 4 バイト境界に割り当てられます。しかし、特殊なオプションを指定してコンパイルすれば、これらのデフォルトを変更できます。

REAL の要素は、符号ビット、8 ビットの指数部、23 ビットの小数部で構成されます。f77 での REAL の要素は IEEE 規格に準拠します。

REAL*4 ♦

データ型 REAL*4 は REAL と同義ですが、コンパイラオプションに関係なく、サイズは常に 4 バイトです。

REAL*8 (倍精度実数)♦

データ型 REAL*8 は DOUBLE PRECISION と同義ですが、コンパイラオプションに関係なく、サイズは常に 8 バイトです。

REAL*16 (4 倍精度実数)♦

(SPARC のみ) データ型 REAL*16 は 4 倍精度の実数です。

REAL*16 項目のサイズは 16 バイトです。

REAL*16 の要素は、符号ビット、15 ビットの指数部、112 ビットの小数部で構成されます。f77 での REAL*16 の要素は、拡張された精度では IEEE 規格に準拠します。

データ型のサイズと整列条件

記憶領域と整列条件は、常にバイト単位で規定されます。1 バイトに収まる値は、バイト単位で整列されます。

それぞれの型のサイズと整列条件は、コンパイラオプションの指定とプラットフォーム、さらに変数の宣言方法によって異なります。COMMON ブロックの最大整列条件は、4 バイト境界になります。

デフォルトのデータの整列条件や記憶領域の割り当ては、`-aligncommon`、`-f`、`-dalign`、`-dbl_align_all`、`-dbl`、`-xmemalign`、`-r8`、`-i2`、`-xtypemap` などのオプションを指定してコンパイルすることにより、変更することができます。このマニュアルでデフォルトについて説明するときは、これらのオプションは指定されていないものと想定します。

個々のコンパイラオプションの詳細については、『Fortran ユーザーズガイド』を参照してください。

次の表に、デフォルトのサイズと整列条件についてまとめています。ただし、それ以外の型やオプションに関する項目は無視しています。

表 2-1　デフォルトのデータサイズと整列条件 (バイト単位)

		デフォルトの整列条件		COMMON **の整列条件**	
Fortran 77 **のデータ型**	**サイズ**	SPARC	x86	SPARC	x86
`BYTE X`	1	1	1	1	1
`CHARACTER X`	1	1	1	1	1
`CHARACTER*n X`	n	1	1	1	1
`COMPLEX X`	8	4	4	4	4
`COMPLEX*8 X`	8	4	4	4	4
`DOUBLE COMPLEX X`	16	8	4	4	4
`COMPLEX*16 X`	16	8	4	4	4
`COMPLEX*32 X`	32	8/16	--	4	--
`DOUBLE PRECISION X`	8	8	4	4	4
`REAL X`	4	4	4	4	4
`REAL*4 X`	4	4	4	4	4
`REAL*8 X`	8	8	4	4	4

表 2-1　デフォルトのデータサイズと整列条件 (バイト単位) (続き)

Fortran 77 のデータ型	サイズ	デフォルトの整列条件		COMMON の整列条件	
		SPARC	x86	SPARC	x86
REAL*16 X	16	8/16	--	4	--
INTEGER X	4	4	4	4	4
INTEGER*2 X	2	2	2	2	2
INTEGER*4 X	4	4	4	4	4
INTEGER*8 X	8	8	4	4	4
LOGICAL X	4	4	4	4	4
LOGICAL*1 X	1	1	1	1	1
LOGICAL*2 X	2	2	2	2	2
LOGICAL*4 X	4	4	4	4	4
LOGICAL*8 X	8	8	4	4	4

次の点に注意してください。

- REAL*16 と COMPLEX*32 は、SPARC のみ使用できます。64 ビット環境 (-xarch=v9 または v9a を使ってコンパイルする) では、表に 8/16 と記載されているように、デフォルトの整列条件は 16 バイト境界になります (8 バイトではありません)。
- 配列と構造体はそれらの要素または欄によって整列条件が決まります。配列はその配列要素と同様に整列します。構造体は整列の幅が最大の欄と同様に整列します。

オプション -i2、-r8、または -dbl を指定してコンパイルすると、明示的なサイズ指定なしで出力される特定のデータ宣言のデフォルトが変更されます。

表 2-2　-i2、-r8、-dbl によって変更されるデータのデフォルト

デフォルトの型	-i2 を使用	-r8 または -dbl を使用
INTEGER	INTEGER*2	INTEGER*8
LOGICAL	LOGICAL*2	LOGICAL*8
REAL	REAL*4	REAL*8

表 2-2 -i2、-r8、-dbl によって変更されるデータのデフォルト

デフォルトの型	-i2 を使用	-r8 または -dbl を使用
DOUBLE	REAL*8	REAL*16
COMPLEX	COMPLEX*8	COMPLEX*16
DOUBLE COMPLEX	COMPLEX*16	COMPLEX*32

-i2 と -r8 を同時に使用しないでください。同時に使用した場合、予期しない結果が出ることがあります。REAL*16 と COMPLEX*32 は、SPARC のみ使用できます。

-dbl または -r8 を使用すると、INTEGER および LOGICAL は、上記に示すように大きいスペースが割り当てられます。これは、整数項目と実数項目は同じ大きさの記憶領域を使用するという Fortran の要件を満たすための処置です。-r8 を使用すると 8 バイトが割り当てられますが、行われる計算は 4 バイトのみです。-dbl を使用すると 8 バイトが割り当てられ、8 バイトの計算が実行されます。-dbl と -r8 のどちらを使用しても結果は同じになります。-r8 または -dbl を使用した場合の不利な点は、DOUBLE PRECISION 型データまでも QUAD PRECISION 型に拡張してしまうため、性能が劣化する可能性があることです。

-r8 や -dbl などの古いオプションよりも、順応性のある -xtypemap オプションを使用することをお勧めします。-dbl と -r8 に相当する -xtypemap オプションは以下のとおりです。

- SPARC マシン

-dbl は、-xtypemap=real:64,double:128,integer:64 と同じ
-r8 は、-xtypemap=real:64,double:128,integer:mixed と同じ

- x86

-dbl は、-xtypemap=real:64,double:64,integer:64 と同じ
-r8 は、-xtypemap=real:64,double:64,integer:mixed と同じ

integer:mixed は、8 バイトの整数の割り当てを示します。ただし行われる計算は 4 バイトのみです。

SPARC では、さらに以下の 2 つの -xtypemap オプションを使用できます。

-xtypemap=real:64,double:64,integer:mixed
-xtypemap=real:64,double:64,integer:64

これらはデフォルトの REAL と DOUBLE を両方とも 8 バイトに割り当てるため、-r8 や -dbl よりも実用的です。

INTEGER と LOGICAL は同等に処理され、COMPLEX は 2 つの REAL 値として割り当てられます。また、DOUBLE COMPLEX は DOUBLE と同じ方法で割り当てられます。

オプション -f または -dalign (SPARCのみ) を指定すると、8 バイト、16 バイト、32 バイトのデータは、すべて 8 バイト境界に割り当てられます。オプション -dbl_align_all を指定すると、すべてのデータが 8 バイト境界に割り当てられます。これらのオプションに依存するプログラムは、移植性に欠けることがあります。

コンパイラオプションの詳細については、『Fortran ユーザーズガイド』を参照してください。

定数

定数とはプログラム単位を通して値が変化しないデータです。定数を表現する文字列の形式によりその定数の値とデータの型が決まります。(PARAMETER 文で定義される名前付き定数の場合は、名前によってデータ型が決まります。)

一般的な定数には次の 3 つがあります。

- 算術定数
- 論理定数
- 文字定数

算術定数や論理定数に含まれる空白文字は定数の値に影響しません。文字定数に含まれる空白文字は定数の値に影響します。

算術定数には以下のものがあります。

型のある定数	型なしの定数
複素定数	2 進定数
倍精度複素定数	8 進定数
倍精度定数	16 進定数
整定数	ホレリス定数
実定数	

符号付き定数はプラスまたはマイナス符号が付いた算術定数です。符号なし定数は符号が付いていない算術定数です。

整数、実数、および倍精度のデータでは、ゼロは正数でも負数でもありません。符号付きのゼロと符号なしのゼロの値は同じです。

オプション `-i2`、`-dbl`、`-r8`、または `-xtypemap` のいずれかを指定してコンパイルすると、整数、実数、複素数、および倍精度の各定数のデフォルトサイズが変更されます。これらのオプションについては、第 2 章「データ型とデータ項目」と『Fortran ユーザーズガイド』を参照してください。

文字定数

文字列定数とはアポストロフィまたは引用符で囲まれた文字の列です。アポストロフィは規格ですが、引用符は規格外です。♦

`-xl` オプションを指定してコンパイルする場合、引用符は別の意味に解釈されるので、アポストロフィで文字列を囲む必要があります。

アポストロフィで囲んだ文字列の中にアポストロフィを文字として含めるには、アポストロフィを 2 個続けます。引用符の場合も同様です。次にその例を示します。

```
'abc'   "abc"
'ain''t'"vi では ""h9Y と入力"
```

文字列がどちらの区切り文字で始まる場合も、別の区切り文字は文字列の中に埋め込むことができます。この場合引用符を 2 つ続けたり、バックスラッシュ・エスケープを使用する必要はありません。表 2-3 を参照してください。
次にその例を示します。

```
"abc" "abc"
"ain't"'vi では "h9Y と入力'
```

空文字 ♦

`DATA` 文中を除き、文字列定数の後ろには 1 個の 空文字を続けて、C ルーチンとのデータの受け渡しを容易にします。空文字だけの文字列定数を作ることもできます。このような空文字列定数は Fortran 規格外です。

以下に空文字列の例を示します。

```
demo% cat NulChr.f
    write(*,*) 'a', '', 'b'
    stop
    end
demo% f77 NulChr.f
NulChr.f:
 MAIN:
demo% a.out
ab
demo%
```

ただし、空文字定数を文字変数に入れると、この変数には空白が含まれ長さは最小で 1 バイトになります。

以下の例は空文字列の長さを示しています。

```
demo% cat NulVar.f
    character*1 x / 'a' /, y / '' /, z / 'c' /
    write(*,*) x, y, z
    write(*,*) len( y )
    end
demo% f77 NulVar.f
NulVar.f:
 MAIN:
demo% a.out
a c
  1
demo%
```

エスケープシーケンス ♦

C 言語の用法との互換性のために、以下の表に示すバックスラッシュ・エスケープシーケンスがあります。文字列の中にエスケープシーケンスを書くと、それに対応した文字に置き換えられます。

表 2-3　バックスラッシュ・エスケープシーケンス

エスケープシーケンス	文字
\n	改行 (Newline)

表 2-3　バックスラッシュ・エスケープシーケンス (続き)

エスケープシーケンス	文字
\r	キャリッジリターン
\t	タブ
\b	バックスペース
\f	用紙送り （Form feed）
\v	垂直タブ
\0	空 （NULL）
\'	アポストロフィ （文字列を終了させない）
\"	引用符 （文字列を終了させない）
\\	\
x	*(*上記以外の任意の文字）

-xl オプションを指定してコンパイルした場合、バックスラッシュ文字 (\) は普通の文字として扱われます。つまり、-xl オプションを指定すると、上記のエスケープシーケンスを使用して特殊文字を入力することはできなくなります。

技術的にはエスケープシーケンスは規格外ではありませんが、処理系によって定義が異なります。

複素定数

複素定数は実定数または整定数 (またはパラメータ定数 ♦) を順序付けて対にしたものです。2 つの定数は括弧に入れ、コンマで区切ります。最初の定数は実部で、2 番目の定数は虚部です。複素定数COMPLEX*8 は 8 バイトの記憶領域を使用します。

以下に複素定数の例を示します。

```
( 9.01, .603 )
( +1.0, -2.0 )
( +1.0, -2 )
( 1, 2 )
( 4.51, )   無効 — 2 番目が必要
```

COMPLEX*16 定数

倍精度複素定数 COMPLEX*16 は、実定数または整定数を順序付けて対にしたもので、一方の定数は REAL*8 で、他方の定数は INTEGER、REAL*4 または REAL*8 です。♦

2 つの定数は括弧に入れ、コンマで区切ります。最初の定数は実部で、2 番目の定数は虚部です。倍精度複素定数 COMPLEX*16 は 16 バイトの記憶領域を使用します。

以下に倍精度複素定数の例を示します。

```
( 9.01D6, .603 )
( +1.0, -2.0D0 )
( 1D0, 2 )
( 4.51D6, ) 無効 — 2 番目が必要
( +1.0, -2.0 ) DOUBLE COMPLEX ではない — REAL*8 が必要
```

COMPLEX*32 (4 倍精度複素数) 定数

(SPARC のみ) 4 倍精度複素定数 ♦ は実定数または整定数を順序付けて対にしたもので、一方の定数は REAL*16 で、他方の定数は INTEGER、REAL*4、REAL*8 または REAL*16 です。♦

2 つの定数は括弧に入れ、コンマで区切ります。最初の定数は実部で、2 番目の定数は虚部です。4 倍精度複素定数 COMPLEX*32 ♦ は 32 バイトの記憶領域を使用します。

以下に 4 倍精度複素定数の例を示します (SPARC のみ)。

```
( 9.01Q6, .603 )
( +1.0, -2.0Q0 )
( 1Q0, 2 )
( 3.3Q-4932, 9 )
( 1, 1.1Q+4932 )
( 4.51Q6, ) 無効 — 2 番目が必要
( +1.0, -2.0 ) 4 倍精度ではない — REAL*16 が必要
```

整定数

整定数は、10 進数字列の前にプラス符号 (省略可能) またはマイナス符号が付いたものです。

制限事項

- 10 進数字以外の文字は使用できません (スペースは例外です)。
- 符号がない場合、その定数は負ではないとみなされます。
- 定数値の範囲は、整数を 64 ビットに拡張するオプションでコンパイルしない限り、INTEGER*4 (-2147483648、2147483647) にならなくてはなりません。64 ビットに拡張するオプションでコンパイルした場合は、INTEGER*8 (-9223372036854775808、9223372036854775807) になります。22 ページの「データ型のサイズと整列条件」を参照してください。

以下に整定数の例を示します。

```
-2147483648
-2147483649 無効 — 小さすぎる、エラーメッセージ
-10
0
+199
29002
2.71828     INTEGER ではない — 10 進小数点は使えない
1E6         INTEGER ではない — E は使えない
29,002      無効 — コンマは使えない、エラーメッセージ
2147483647
2147483648 無効 — 大きすぎる、エラーメッセージ
```

代替 8 進表記 ♦

整定数の指定には次のような代替 8 進表記を使用することもできます。整数の数字列の前に二重引用符 (") を付け、-xl オプションを指定してコンパイルしてください。このように表記された整定数は 8 進定数です。型は INTEGER です。
次の 2 つの文は等価です。

```
    JCOUNT = ICOUNT + "703
    JCOUNT = ICOUNT + 451
```

整定数として 2 進、8 進、16 進またはホレリスのような型なし定数を指定することもできます。34 ページの「型なし定数 (2 進、8 進、16 進)」を参照してください。

長い整数 ♦

定数の範囲を INTEGER*4 (-21474836、21474836) から INTEGER*8 (-9223372036854775808、9233720368547758O7) に拡張するオプションを指定してコンパイルします。この整定数はデータの型が INTEGER*8 の 8 バイト整数として格納されるかまたは引き渡されます。

短い整数 ♦

定数の引数が (-32768, 32767) の範囲内である場合、この引数は通常、データの型が INTEGER*4 の 4 バイト整数に拡張されます。-i2 オプションを指定してコンパイルした場合、この引数はデータの型が INTEGER*2 の 2 バイト整数として格納されるかまたは引き渡されます。

論理定数

論理定数とは真または偽の論理値です。論理定数の値は .TRUE. と .FALSE. で、それ以外はありません。区切り文字のピリオドは必要です。

論理定数は 4 バイトの記憶領域をとります。論理定数を実引数として使用する場合、この引数は 4 バイトで引き渡されます。-i2 オプションを指定してコンパイルした場合、この引数は 2 バイトで引き渡されます。

実定数

実定数は実際の数の近似値であり、正数、負数、ゼロのいずれかの値をとります。小数点か指数またはその両方を含んでいます。符号がない場合、その定数は負ではないとみなされます。

実定数 REAL*4 は 4 バイトの記憶領域を使用します。

基本実定数

基本実定数は、プラス符号 (省略可能) またはマイナス符号、整数部、小数点、小数部の順で構成されます。

整数部と小数部はそれぞれ数字の列であり、どちらか一方を省略することはできますが、両方を省略することはできません。

次に基本実定数の例を示します。

```
+82.
-32.
90.
98.5
```

実指数部

実指数部は、文字 E、プラス符号 (省略可能) またはマイナス符号、整数の順で構成されます。実指数部の例を示します。

実指数部の例を示します。

```
E+12
E-3
E6
```

実定数

実定数は次のいずれかの形式をとります。

- 基本実定数
- 基本実定数と実指数部
- 整定数と実指数部

実指数部は 10 のべき乗で示されます。実定数の値はこの 10 のべき乗と E の前にある定数との積です。

実定数の例を示します。

```
-32.
-32.18
1.6E-9
7E3
1.6E12
$1.0E2.0    無効 — $ は使えない、エラーメッセージ
82          REAL ではない — 10 進小数点か指数が必要
29,002.0    無効 — コンマは使えない、 エラーメッセージ
1.6E39      無効 — 大きすぎる、マシンの無限大が使用される
1.6E-39     無効 — 小さすぎる、いくらか精度が失われる
```

以下の制限事項があります。

- プラス符号 (省略可能) またはマイナス符号、小数点、0 から 9 までの数字、および文字 E 以外の文字は使用できません。
- 正規化された単精度浮動小数点値の近似値は、(1.175494E-38, 3.402823E+38) の範囲内でなければなりません。

REAL*8 (倍精度実) 定数

倍精度実定数は実際の数の近似値であり、正数、負数、ゼロのいずれかの値をとります。符号がない場合、その定数は負ではないとみなされます。倍精度実定数は倍精度指数部と省略できる小数点を含みます。倍精度定数 REAL*8 は 8 バイトの記憶領域を使用します。REAL*8 表記は規格外です。♦

倍精度指数部

倍精度指数部は、文字 D、プラス符号 (省略可能) またはマイナス符号、整数の順で構成されます。

倍精度指数部は 10 のべき乗で示されます。倍精度定数の値はこの 10 のべき乗と D の前にある定数との積です。形式と解釈は、E の代わりに D が使用されることを除いて、実指数部の場合と同じです。

以下に倍精度定数の例を示します。

```
1.6D-9
7D3
$1.0D2.0    無効 — $ は使えない、エラーメッセージ
82          DOUBLE PRECISION ではない  — 10 進小数点が指数が必要
29,002.0D0 無効 — コンマは使えない、 エラーメッセージ
1.8D308     無効 — 大きすぎる、マシンの無限大が使用される
1.0D-324    無効 — 小さすぎる、いくらか精度が失われる
```

制限事項は次のとおりです。

- プラス符号 (省略可能) またはマイナス符号、小数点、0 から 9 までの数字、空白、および文字 D 以外の文字は使用できません。
- IEEE 形式の正規化された倍精度浮動小数点値の近似値は、(2.225074D-308, 1.797693D+308) の範囲内でなければなりません。

REAL*16 (4 倍精度実) 定数

(SPARC のみ) 4 倍精度定数は基本実定数または整定数の後ろに 4 倍精度指数部が付いたものです。 31 ページの「実定数」を参照してください。 ♦

4 倍精度指数部は、文字 Q、プラス符号 (省略可能) またはマイナス符号、整数の順で構成されます。

4 倍精度定数は、正数、負数、ゼロのいずれかの値をとります。符号がない場合、その定数は負ではないとみなされます。

以下に 4 倍精度定数の例を示します。

```
1.6Q-9
7Q3
3.3Q-4932
1.1Q+4932
$1.0Q2.0 無効 — $ は使えない、エラーメッセージ
82       quad ではない — 指数が必要
29,002.0Q0 無効 — コンマは使えない、 エラーメッセージ
1.6Q5000 無効 — 大きすぎる、マシンの無限大が使用される
1.6Q-5000 無効 — 小さすぎる、いくつか精度が失われる
```

形式と解釈は、E の代わりに Q が使用されることを除いて、実定数の場合と同じです。

以下の制限事項があります。

- プラス符号 (省略可能) またはマイナス符号、小数点、0 から 9 までの数字、空白、および文字 Q 以外の文字は使用できません。
- IEEE 形式の正規化された 4 倍精度浮動小数点値の近似値は、(3.362Q-4932, 1.20Q+4932) の範囲内でなければなりません。
- 4 倍精度定数は 16 バイトの記憶領域をとります。
- 各 4 倍精度定数は 8 バイト境界に割り当てられます。

型なし定数 (2 進、8 進、16 進)

型なしの数値定数は、式の中でどのように使用されるかによってデータの型が決められるので、この名称が付けられています。♦

この定数は使用されるまでは変換されません。ただし、f77 では、この定数は文字列と区別される必要があります。

この定数の一般形はアポストロフィで囲んだ数字の列の前に文字 B、O、X または Z を付けます。B は 2 進、O は 8 進、X または Z は 16 進の定数であることを示します。

DATA 文および PARAMETER 文の中の 2、8、16 進定数の例を次に示します。

```
      PARAMETER ( P1 = Z'1F' )
      INTEGER*2 N1, N2, N3, N4
      DATA N1 /B'0011111'/, N2/O'37'/, N3/X'1f'/, N4/Z'1f'/
      WRITE ( *, 1 ) N1, N2, N3, N4, P1
1     FORMAT ( 1X, O4, O4, Z4, Z4, Z4 )
      END
```

FORMAT 文の中の編集記述子に注意してください。O は 8 進を、Z は 16 進を表します。上記の整定数はすべて 10 進で 31 になります。

DATA 文および PARAMETER 文以外の式での 2、8、16 進定数の例を次に示します。

```
      INTEGER*4   M, ICOUNT/1/, JCOUNT
      REAL*4   TEMP
      M = ICOUNT + B'0001000'
      JCOUNT = ICOUNT + O'777'
      TEMP = X'FFF99A'
      WRITE(*,*) M, JCOUNT, TEMP
      END
```

この例では B'0001000' と O'777' は INTEGER*4 と定義され、X'FFF99A' は REAL*4 と定義されています。実数に対して IEEE 形式の浮動小数点を使用すれば、アーキテクチャが異なっても与えられたビットパターンから生成される値は同じになります。

上記の文は以下のように扱われます。

```
      M = ICOUNT + 8
      JCOUNT = ICOUNT + 511
      TEMP = 2.35076E-38
```

制御文字

制御文字は次の方法で入力できます。CHAR 関数は規格内ですが、次の方法は規格外です。

例：型なし定数での制御文字

```
CHARACTER BELL, ETX / X'03' /
PARAMETER ( BELL = X'07' )
```

型なし定数の代替表記

Fortran 77 の他のバージョンとの互換性のため、8 進と 16 進表記には以下の代替表記を使用することができます。この代替表記は 2 進数には無効です。また、DATA 文または PARAMETER 文の中では使用できません。

8 進表記の場合、8 進数字の列をアポストロフィで囲み、その後ろに文字 O を付けます。

例：型なし定数の 8 進代替表記

```
'37'O
 37'O無効 — 最初のアポストロフィがない
'37'数値でない — O の文字がない
'397'O無効 — 無効な数字
```

16 進表記の場合、16 進数字の列をアポストロフィで囲み、その後ろに文字 X を付けます。

例：型なし定数の 16 進代替表記

```
'ab'X
3fff'X
'1f'X
'1fX無効 — 後ろに続くアポストロフィがない
'3f'数値でない — X がない
'3g7'X無効 — 無効な数 g
```

2 進、8 進および 16 進定数には、次の規則と制限事項があります。

- これらの定数は数値定数が許可される箇所ならどこでも使用できます。

- これらの定数には型がありません。これらの定数は、変数の型と一致させるための変換なしで変数に格納されますが、それを受け取る欄の適切な部分 (下位部または上位部) に格納されます。
- 定数を受け取るデータ型の桁数が定数の桁数よりも多い場合、左側にゼロが詰められます。
- 定数を受け取る欄のデータ型の桁数が定数の桁数よりも少ない場合、定数の桁の左側が切り捨てられます。その結果としてゼロ以外の桁が切り捨てられた場合、エラーメッセージが表示されます。
- 定数の先頭部分のゼロは無視されます。
- これらの定数として最大 8 バイトまでのデータを指定できます。
- 型なし定数を実引数として使用した場合、この実引数にはデータ型はありませんが、常に 4 バイトのデータとして引き渡されます。
- 2 進定数の場合、各桁は 0 か 1 でなければなりません。
- 8 進定数の場合、各桁は 0 から 7 までの数字でなければなりません。
- 16 進定数の場合、各桁は 0 から 9 までの数字か、`A` から `F` まで、または `a` から `f` までの英字でなければなりません。
- `DATA` 文以外ではこれらの定数は前後関係から決まる型として扱われます。型なし定数が 2 項演算子とともに使用された場合、その定数はもう一方のオペランドと同じデータ型になります (`8.0 + '37'O`)。
- `DATA` 文ではこれらの定数は型なしの 2 進、16 進または 8 進定数として扱われます。

ホレリス定数 ♦

ホレリス定数は、符号なしのゼロ以外の整定数、文字 `H`、印刷可能文字列の順で構成されます。整定数はスペースとタブも含めて印刷可能文字列中の文字数を指定します。

ホレリス定数は各文字 1 バイトずつの記憶領域を占めます。

ホレリス定数は 2 バイト境界に割り当てられます。

Fortran 77 規格にはこの旧式のホレリス表記はありませんが、古いプログラムとの互換性を維持するためにホレリス機能を装備することをお勧めします。

ホレリスデータを文字列定数の代わりに使用することができます。また、ホレリスデータを IF 文の条件に使用したり、DATA 文や代入文中の変数 (文字変数以外) を初期化するのにも使用できますが、このような使用法は勧められませんし、すべて規格外です。ホレリスデータは型なし定数です。

以下に型なし定数の例を示します。

```
CHARACTER C*1, CODE*2
INTEGER TAG*2
DATA TAG / 2Hok /
CODE = 2Hno
IF ( C .EQ. 1HZ ) CALL PUNT
```

ホレリス定数には、次の規則と制限事項があります。

- 文字数には実際上の制限はありません。
- 文字を継続行に続けることはできますが、複雑な方法になります。短い標準固定フォーマットの行は 72 桁目まで右に空白が詰められますが、短いタブフォーマットの行は改行で終わります。
- ホレリス定数が 2 項演算子とともに使用された場合、その定数はもう一方のオペランドと同じデータ型になります。
- ホレリス定数を変数に代入した場合、その定数の長さが変数のデータ型の長さより短かければ、スペース (ASCII 32) が右側に追加されます。

ホレリス定数または変数の長さが変数のデータ型より長い場合は、文字の右側が切り捨てられます。

- ホレリス定数を実引数として使用した場合、4 バイトの項目として引き渡されます。
- ホレリス定数が使用されていて、そのデータ型が前後関係から決まらない場合、INTEGER*4 が使用されます。

Fortran 95 のスタイルによる定数 ♦

Sun WorkShop Fortran 77 コンパイラは、データ項目のサイズのリテラルな指定を可能にする整数と実数の定数の Fortran 95 のスタイルによる構文を認識します。Fortran 95 の用語の用法では、定数のリテラルには、末尾のアンダースコア (オプション) に続けて「kind type parameter」を入れることができます。♦

Sun Fortran 77 における実装では、「kind type parameter」に使用可能な数値は、1、2、4、8、16 に限定され、それによってリテラルな定数のデータサイズをバイト単位で指定することができます。

たとえば、次のようになります。

```
12_88 バイトの整定数を指定、値 = 12
12.012_1616 バイトの実定数を指定、値 = 12.012
1.345E-10_8 8 バイトの実定数を指定、値 = 1.345E-10
(-1.5_8,.895E-3_8)8 バイトの実部と虚部を持つ複素定数を指定
```

複素数の定数では、実部と虚部は、別々の kind type parameter (1.0_8、2.0_4) で指定されますが、最終的なデータ項目は、同じサイズの実部と虚部から構成されることになります。これは大きい値を指定した方のサイズになります。

このような構成は、次の例からも分かるように、特定のデータ型が必要なサブプログラムを、定数を引数として呼び出す際に役立ちます。

```
call suby(A,1.5_8,0_8,Y)
...
subroutine suby(H0,M,N,W)
INTEGER *8 M,N,
...
```

変数

変数とは記憶場所と対になっている英字名です。変数は、名前、値、および型を持ちます。その記憶場所に格納されているデータが変数の値です。配列もしくは配列要素、記録、または記録欄は変数には含まれないことに注意してください。したがって、この定義は通常の「変数」という語の使用法よりも限定的です。

変数の型は型宣言文で指定できます。型が型宣言文で明示的に指定されない場合、その変数名の最初のアルファベットによって型が暗黙的に決められます。この場合、通常のデフォルトの暗黙の型宣言か、IMPLICIT 文で指定された暗黙の型宣言に従います。データの型宣言規則の詳細については、13 ページの「データの型」を参照してください。

プログラム実行中のある時点での変数は定義されているか未定義であるかのどちらかです。変数が予測可能な値を持つ場合は変数は定義されており、そうでない場合は未定義です。副プログラムが終了すると、定義されていた変数が未定義になることがあります。

代入文、入力文、または DATA 文で変数を定義することができます。DATA 文で変数に値を割り当てると、その変数は初期定義されたことになります。

2 つの変数が同じ記憶場所に関連付けられている場合、それら 2 つの変数は関連付けられています。EQUIVALENCE 文、COMMON 文、または MAP 文を使用すれば変数を関連付けることができます。実引数と仮引数を使用しても変数を関連付けることができます。

配列

配列とは同じ型の要素の集合に名前を付けたものです。配列は空ではない一連のデータであり、連続する記憶領域を占めます。配列は、名前、一連の要素、および型を持ちます。

配列名は、一連のデータ全体に対して付けられた英字名です。

配列要素は一連のデータのメンバーの 1 つです。各記憶場所は配列の 1 要素を保持します。

配列要素名は添字で修飾された配列名です。詳しくは 44 ページの「配列の添字」を参照してください。

配列は次の文のいずれかで宣言します。

- DIMENSION 文
- COMMON 文
- 型宣言文：BYTE、CHARACTER、INTEGER、REAL など

配列宣言子

配列宣言子は配列の名前と特性を指定します。

配列宣言子の構文は次のとおりです。

```
a ( d [, d ] ... )
```

ここで、

- *a* は配列の名前、*d* は寸法宣言子です。
- 寸法宣言子の形式は次のとおりです。

[*dl* :] *du*

ここで、

- *dl* は下限 (lower dimension bound) です。
- *du* は上限 (upper dimension bound) です。

1 つの配列は、1 プログラム単位 (主プログラム、サブルーチン、関数、または共通ブロック)内の 1 配列宣言に 1 度だけ指定できます。同じ単位内に複数または重複した配列宣言がある場合、コンパイラはそれらの宣言をエラーとして通知します。

配列の次元数は寸法宣言子の数です。次元数の最小は 1、最大は 7 です。大きさ引き継ぎ配列の場合、最後の寸法がアスタリスクで指定されます。

下限はその次元の最初の要素を示し、上限はその次元の最後の要素を示します。1 次元の配列では上下限は配列の最初と最後の要素です。

次に配列宣言子の上下限の例を示します。

```
    REAL V(-5:5)
```

上記の例で `V` は 1 次元で 11 の要素を持つ実数の配列です。最初の要素は `V(-5)` で、最後の要素は `V(5)` です。

下限の指定を省略した (その場合、下限は 1) 配列の例を次に示します。

```
    REAL V(1000)
```

上記の例で V は 1 次元で 1,000 個の要素を持つ実数の配列です。最初の要素は V(1) で、最後の要素は V(1000) です。

配列は最大で 7 次元にすることができます。次にその例を示します。

```
    REAL TAO(2,2,3,4,5,6,10)
```

下限が 1 以外の配列の例を次に示します。

```
    REAL A(3:5, 7, 3:5), B(0:2)
```

次に文字配列の例を示します。

```
    CHARACTER M(3,4)*7, V(9)*4
```

配列 M は 12 個の要素を持ち、各要素は 7 文字からなります。
配列 V は 9 個の要素を持ち、各要素は 4 文字からなります。

上下限には、次の制限事項があります。

- 上下限とも、負数、ゼロ、正数を指定できます。
- 上限は下限よりも大きいか等しくなければなりません。
- 上下限が 1 つしか指定されない場合、上限の指定とみなされ、下限は 1 になります。
- 大きさ引き継ぎ配列は最後の次元の上限がアスタリスクで指定されます。
- 上下限は整数式で、その式の各オペランドは、定数、仮引数、または共通ブロック内の変数です。配列の引用やユーザー定義の関数は指定できません。

整合配列

整合配列とは仮引数として指定された配列、あるいは、1 つ以上の次元または上下限が整変数の式として指定された局所配列 ♦ です。その整変数は、それ自身が仮引数または共通ブロック内の変数になります。

整合配列は、通常の DIMENSION 文、または型宣言文で宣言できます。f77 では整合配列は RECORD 文でも宣言できます。ただし、その RECORD 文が構造体宣言ブロックの中にはないとします。

次に整合配列の例を示します。

```
    SUBROUTINE POPUP ( A, B, N )
    COMMON / DEFS / M, L
    REAL A(3:5, L, M:N), B(N+1:2*N) !仮引数の配列
    REAL C(N+1,2*N) !局所配列
```

制限事項は以下のとおりです。

- 整合配列の大きさは対応する実引数の大きさを超えてはいけません。
- 一連の呼び出しの最初の呼び出し元では、対応する配列の次元が定数で決定されていなければなりません。
- COMMON 文で整合配列を宣言することはできません。

配列がルーチンに対して局所指定される場合は、そのルーチンの入り口でメモリーが割り当てられ、呼び出し元への戻りで割り当てが解除されます。♦

大きさ引き継ぎ配列

大きさ引き継ぎ配列とは仮引数として指定された配列であり、かつ、最後の次元の上限がアスタリスクで指定された配列です。

大きさ引き継ぎ配列は、通常の DIMENSION 文、COMMON 文、または型宣言文で宣言できます。

f77 の拡張機能では、以下のことが認められています。♦

- RECORD 文における大きさ引き継ぎ配列の宣言。ただしその RECORD 文が構造体宣言ブロックの中にあってはいけません。
- 入出力文における内部ファイルの装置識別子としての大きさ引き継ぎ配列の指定。
- 入出力文における実行時の書式指定子としての大きさ引き継ぎ配列の指定。

最後の次元の上限がアスタリスクで指定された大きさ引き継ぎ配列の例を次に示します。

```
    SUBROUTINE PULLDOWN ( A, B, C )
    INTEGER A(5, *), B(*), C(0:1, 2:*)
```

大きさ引き継ぎ配列は入出力並びの中では指定できません。

添字なしの配列名

添字なしの配列名はその配列全体を示し、以下の文の中で指定できます。

- COMMON 文
- DATA 文
- 入出力文
- NAMELIST 文
- RECORD 文
- SAVE 文
- 型宣言文

EQUIVALENCE 文では、添字なしの配列名はその配列の最初の要素を示します。

配列の添字

配列要素名は添字で修飾された配列名です。

添字の形式

添字は括弧で囲まれた添字式の並びです。配列の各次元に対して 1 つの添字式を指定しなければなりません。

添字の形式は次のとおりです。

(s [, s] ...)

ここで s は添字式です。括弧も添字の一部です。

たとえば、次のような宣言子で 2 × 3 の配列を宣言しているとします。

```
REAL M(2,3)
```

上記について、次のようにすれば、その配列の特定の要素に値を代入することができます。

```
M(1,2) = 0.0
```

上記の文は、配列 M の行 1、列 2 の要素に 0.0 を代入しています。

添字式

添字式には以下の特性と制限事項があります。

- 添字式は整数式、実数式、複素式、論理式またはバイト式です (FORTRAN 77 規格に従う場合は整数式でなければなりません)。
- 添字式に配列要素の引用や関数の引用を含めることができます。
- 関数の引用が評価された結果、同じ添字内の他の添字式の値を変更してはいけません。
- 各添字式は配列の対応する次元への索引です。
- 各添字式の値は配列の対応する次元の上下限の範囲内になければなりません。
- (*L1*,⋯,*Ln*) という形式の添字 (各 *Li* は対応する次元の下限) はその配列の最初の要素を引用します。
- (*U1*,⋯,*Un*) という形式の添字 (各 *Ui* は対応する次元の上限) はその配列の最後の要素を引用します。
- 配列要素 A(n) は必ずしも配列 A の n 番目の要素であるとは限りません。

```
REAL V(-1:8)
V(2) = 0.0
```

上記の例では V の 4 番目の要素がゼロに設定されます。

32 ビット環境では、添字式が INTEGER*4 の範囲を越えることはできません。添字式が範囲内 (-2147483648, 2147483647) にない場合、結果は予測できません。64 ビット環境用としてコンパイルする場合は、INTEGER*8 の添字式を使用することができます。

配列の順序付け

配列要素はイメージ的には通常、最初の添字を行番号、2 番目の添字を列番号として並べられます。これは伝統的な数学の *nxm* 行列の表記に対応します。

a1,1	a1,2	a1,3	...	a1,m
a2,1	a2,2	...		a2,m
...	...	ai,j	...	ai,m
an,1	an,2	...		an,m

要素 ai,j は、*i* 行 *j* 列にあります。

次に例を示します。

```
    INTEGER*4 A(3,2)
```

A の要素は概念的に 3 行と 2 列で並べられます。

A(1,1)	A(1,2)
A(2,1)	A(2,2)
A(3,1)	A(3,2)

配列要素は列番号順で格納されます。

たとえば配列 A の場合、その要素はメモリー上に次のように配置されます。

A(1,1)	A(2,1)	A(3,1)	A(1,2)	A(2,2)	A(3,2)

内側 (左端) の添字が先に変化します。

部分列

文字データは連続した 1 個以上の文字です。部分列とは、文字変数、文字配列要素、または構造体の記録の文字欄の中の連続した部分のことです。

部分列名は次の 2 つの形式のどちらかで表されます。

v ([*e1*] : [*e2*])

a (*s* [, *s*] ...) ([*e1*] : [*e2*])

上記の形式中の各記号の意味は次のとおりです。

v	文字変数名
a (*s* [, *s*] ...)	文字配列要素名
e1	部分列の左端の文字の位置
e2	部分列の右端の文字の位置

e1 と *e2* は整数式です。32 ビット環境では、整数式が INTEGER*4 の範囲を越えることはできません。整数式が範囲内 (-2147483648, 2147483647) にない場合、結果は予測できません。64 ビット環境用としてコンパイルする場合は、部分列の文字位置を示す式を INTEGER*8 の範囲内で指定することができます。

次の例は、文字変数名 S の I 番目の文字を最初の文字とし、L 番目の文字を最後の文字とする文字列です。

```
S(I:L)
```

上記の例の部分列の文字数は *(L-I+1)* です。

次の例は、配列要素 A(J,K) の M 番目の文字を最初の文字とし、N 番目の文字を最後の文字とする文字列です。

```
A(J,K)(M:N)
```

上記の例の部分列の文字数は *(N-M+1)* です。

部分列の規則と制限事項は次のとおりです。

- 部分列の中の文字位置は左から右へと番号が付けられます。
- 文字列の最初の文字位置には番号 1 (0 ではない) が付けられます。
- 部分列の最初と最後の文字位置は整数式で表します。
- 最初の式を省略すると 1 になります。
- 2 番目の式を省略すると宣言された文字列の長さになります。
- *0* ＜ *I* ≦ *L* ≦ 宣言された長さ (*I* は部分列の最初の位置、*L* は最後の位置) という条件に従っていない場合、結果は未定義となります。
- 部分列は代入文の左辺と右辺で使用できます。また、手続きの実引数としても使用できます。
- 部分列はオーバーラップすることはできません。`ASTR(2:4) = ASTR(3:5)` は不正です。

以下に部分列の例を示します。行 2、列 3 の要素の値は e23 です。

```
demo% cat sub.f
    character v*8 / 'abcdefgh' /,
&   m(2,3)*3 / 'e11', 'e21',
&   'e12', 'e22',
&   'e13', 'e23' /
    print *, v(3:5)
    print *, v(1:)
    print *, v(:8)
    print *, v(:)
    print *, m(1,1)
    print *, m(2,1)
    print *, m(1,2)
    print *, m(2,2)
    print *, m(1,3)
    print *, m(2,3)
    print *, m(1,3)(2:3)
    end
demo% f77 sub.f
sub.f:
 MAIN:
demo% a.out
 cde
 abcdefgh
 abcdefgh
 abcdefgh
 e11
 e21
 e12
 e22
 e13
 e23
 13
demo%
```

構造体

構造体は配列を一般化したものです。♦

配列は同じ型の要素の集まりですが、構造体は必ずしも同じ型ではない要素の集まりです。

配列の要素は数値の添字を使用することにより引用されますが、構造体の要素は要素 (または欄) 名を使用することにより引用されます。

構造体宣言では、記録を構成する欄の名前、型、サイズ、および順序を指定して記録の形式を定義します。構造体が定義されて名前が付けられると、以下で説明するようにその構造体を RECORD 文で使用することができます。構造体宣言の構文を以下に示します。

構造体宣言

構造体宣言の構文は以下のとおりです。

```
STRUCTURE [/構造体名/] [欄の並び]
    欄宣言
    [欄宣言]
    . . .
    [欄宣言]
END STRUCTURE
```

構造体名	構造体の名前
欄の並び	指定された構造体の欄の並び
欄宣言	記録内の欄を定義する欄宣言の定義は次項を参照

欄宣言

欄宣言は以下のいずれかです。

- 部分構造体 (別の構造体宣言または以前に定義された記録)
- 共用体宣言 (後の説明を参照)
- Fortran 77 の型宣言

次に STRUCTURE 宣言の例を示します。

```
STRUCTURE /PRODUCT/
INTEGER*4 ID
CHARACTER*16 NAME
CHARACTER*8 MODEL
REAL*4 COST
REAL*4 PRICE
END STRUCTURE
```

上記の例では、PRODUCT という名前の構造体を５つの欄 ID、NAME、MODEL、COST および PRICE で構成するように定義しています。欄の並びの例は 55 ページの「構造体の中の構造体」を参照してください。

構造体の規則と制限事項

以下の点に注意してください。

- 名前はスラッシュで囲みます。入れ子にした構造体でのみ名前を省略できます。
- スラッシュがある場合は名前もなければなりません。
- 欄の並びは入れ子にした構造体の中だけで指定できます。
- 欄宣言は必ず 1 つ以上なければなりません。
- 各構造体名はすべての構造体を通じて一意に決められなければなりません。しかし、構造体名を他の構造体の欄名に使用したり、変数名として使用することはできます。
- STRUCTURE 文から END STRUCTURE 文までの間では欄宣言文と PARAMETER 文だけが使用できます。構造体宣言ブロック内にある PARAMETER 文はその外側にある PAPAMETER 文と同じ効果を持ちます。

欄の規則と制限事項

型宣言のある欄は通常の Fortran 77 の型宣言文と同じ構文を使用し、f77 のすべての型が使用できます。以下の規則と制限事項に従います。

- 必要な配列の形状指定は型宣言文の中で行わなければなりません。欄名に対して DIMENSION 文は無効です。

- 欄名として擬似名 %FILL を指定できます。%FILL は Fortran 77 の他のバージョンとの互換性のために提供されています。f77 では整列の問題は考慮されているので %FILL を使用する必要はありません。擬似名 %FILL は、ある特定のサブルーチン内に引用できない欄を作りたい場合に役に立つ機能です。%FILL の機能は、指定されたサイズと型の欄を作り、その欄の引用を不可能にすることだけです。
- すべての欄名は明示的に型宣言されなければなりません。STRUCTURE 宣言の文には、IMPLICIT 文も、暗黙の I、J、K、L、M および N の規則も適用されません。
- 欄宣言の中では、整合配列または大きさ引き継ぎ配列を使用したり、CHARACTER 宣言の文字の長さを (*) で指定することはできません。

構造体宣言での欄のオフセット *n* は、1 つ前にある欄のオフセットにその前欄の長さを加算した値です。整列条件を満たすための調整が行われた場合は、さらにその分が補正されます。記憶領域割り当ての要約については、353 ページの「データの表現」を参照してください。

記録宣言

RECORD 文は、変数が指定された構造体の記録であることを宣言するか、あるいは配列が指定された構造体の記録の配列であることを宣言します。

RECORD 文の構文は以下のとおりです。

```
RECORD [/構造体名/ ]記録の並び
    [,/構造体名/ 記録の並び]
    …
    [,/構造体名/ 記録の並び]
```

構造体名	あらかじめ宣言されている構造体の名前
記録の並び	変数、配列、または、形状が指定された配列の、コンマで区切られた並び

前の STRUCTURE 文の例を用いた RECORD の例を次に示します。

```
    RECORD /PRODUCT/ CURRENT, PRIOR, NEXT, LINE(10)
```

3 つの変数 CURRENT、PRIOR および NEXT はそれぞれ PRODUCT 構造体を持つ記録で、LINE はその記録が 10 個集まった配列です。

記録について次の規則と制限事項に注意してください。

- 各記録はメモリー上では分離して割り当てられます。
- 明示的に初期化されない限り、記録の初期値は未定義です。
- 記録、記録欄、記録配列、および記録配列要素を実引数および仮引数として使用できます。記録を引数として渡す場合、その欄は型、順序、および寸法が一致していなければなりません。呼び出す側と呼び出される側の手続きで、記録宣言は一致していなければなりません。共用体宣言の内部ではマップ欄の順序に意味はありません。56 ページの「UNION と MAP」を参照してください。
- 記録欄は COMMON 文では指定できません。
- 記録と記録欄は DATA 文、EQUIVALENCE 文、または NAMELIST 文では指定できません。記録欄は SAVE 文では指定できません。

記録と欄の引用

記録全体または記録内の個々の欄を引用することができます。また、構造体は入れ子可能なので欄自身を構造体にすることができます。したがって、欄内の欄を引用したり、そのまた内側の欄を引用することができます。

記録と欄の引用の構文は以下のとおりです。

記録名[.欄名] ... [.欄名]	
記録名	あらかじめ定義されている記録変数の名前
欄名	直前 （左側） に指定されている記録の中にある欄の名前

前記の 2 つの例の構造体と記録を用いた引用例を以下に示します。

```
…
RECORD /PRODUCT/ CURRENT, PRIOR, NEXT, LINE(10)
…
CURRENT = NEXT
LINE(1) = CURRENT
WRITE ( 9 ) CURRENT
NEXT.ID = 82
```

上記の例では、

- 最初の代入文が記録全体 (5 つの欄全部) を別の記録にコピーします。
- 2 番目の代入文が記録全体を記録の配列の最初の要素にコピーします。
- WRITE 文が記録全体を出力します。
- 最後の文が記録の ID を 82 に設定しています。

構造体と記録の宣言、および記録と欄への代入の例を以下に示します。

```
demo% cat str1.f
* str1.f 単純構造体
    STRUCTURE / S /
    INTEGER*4 I
    REAL*4 R
    END STRUCTURE
    RECORD / S / R1, R2
    R1.I = 82
    R1.R = 2.7182818
    R2 = R1
    WRITE ( *, * ) R2.I, R2.R
    STOP
    END
demo% f77 -silent str1.f
demo% a.out
82 2.718280
demo%
```

部分構造体の宣言

構造体の中の欄をさらに構造体にすることができます。このような欄を部分構造体と呼びます。部分構造体は次のどちらかの方法で宣言します。

- 構造体宣言の中の RECORD 宣言
- 構造体宣言の中の構造体宣言 (入れ子にされている)

構造体の中の記録

入れ子にした構造体宣言とは、構造体宣言または共用体宣言の中に含まれている構造体宣言のことです。以前に定義された記録を構造体宣言の中で使用することができます。

あらかじめ定義された記録 PRODUCT を使用して構造体 SALE を定義する例を以下に示します。

```
STRUCTURE /SALE/
CHARACTER*32  BUYER
INTEGER*2  QUANTITY
RECORD     /PRODUCT/  ITEM
END STRUCTURE
```

上記の例では、構造体 SALE は BUYER、QUANTITY および ITEM の 3 つの欄を含んでいます。ITEM は構造体 /PRODUCT/ をもつ記録です。

構造体の中の構造体

宣言の中で宣言を入れ子にすることができます。

/PRODUCT/ があらかじめ宣言されていない場合、SALE の宣言の中で宣言することができます。以下に例を示します。

```
STRUCTURE /SALE/
CHARACTER*32  BUYER
INTEGER*2  QUANTITY
STRUCTURE /PRODUCT/ ITEM
INTEGER*4  ID
CHARACTER*16  NAME
CHARACTER*8  MODEL
REAL*4  COST
REAL*4  PRICE
END STRUCTURE
END STRUCTURE
```

この例でも、前記の例と同じく、構造体 SALE は 3 つの欄 BUYER、QUANTITY および ITEM を含みます。欄 ITEM は「構造体の宣言」で定義された欄の並び (この場合は要素数が 1 個の並び) の例です。

それぞれの構造体のサイズと複雑さにより、与えられた状況に最も適した部分構造体宣言の方式を決めます。

部分構造体内の欄の引用

部分構造体内の欄を引用することができます。

部分構造体 (前記の例の PRODUCT および SALE が現行プログラム単位内で定義されているとする) の欄を引用する例を次に示します。

```
…
RECORD /SALE/ JAPAN
…
N = JAPAN.QUANTITY
I = JAPAN.ITEM.ID
…
```

部分構造体の規則と制限事項

次のことに注意してください。

- 部分構造体に対して少なくとも 1 つの欄名を定義しなければなりません。
- 同じ入れ子レベルの 2 つの欄が同じ名前を持つことはできません。異なるレベルにある欄であれば同じ名前を持つことができます。ただし、そのようにするとプログラミング上で問題が生じやすくなります。
- 擬似名 %FILL を使用して記録内で欄を整列させることができます。これは名前なしの空の欄を作ります。
- どの入れ子レベルでも構造体がそれ自身を部分構造体として含むことは許可されません。

UNION と MAP

共用体宣言は実行時に記憶領域を共有する欄のグループを定義します。

構文

共用体宣言の構文は以下のとおりです。

```
UNION
マップ宣言
マップ宣言
[マップ宣言]
…
[マップ宣言]
END UNION
```

マップ宣言の構文は以下のとおりです。

```
MAP
欄宣言
[欄宣言]
…
[欄宣言]
END MAP
```

マップ内の欄

マップ宣言の中の各欄宣言は以下のうちのいずれかです。

- 構造体宣言
- 記録
- 共用体宣言
- データ欄の型宣言

マップ宣言は、共用体の中に代替の欄グループを定義します。実行中のある時点では、どちらか一方のマップが共有の記憶場所に関連付けられています。マップ内の欄を引用すると、それ以前に記憶場所に関連付けられていたマップの欄グループは未定義になり、新しく引用された欄が属するマップの欄グループが記憶場所を引き継ぎます。共用体が使用するメモリー量はその共用体の中の最大のマップが使用するメモリー量です。

以下の例では、構造体 /STUDENT/ が NAME、CLASS および MAJOR か、もしくは NAME、CLASS、CREDITS および GRAD_DATE を含むように宣言しています。

```
STRUCTURE /STUDENT/
CHARACTER*32  NAME
INTEGER*2  CLASS
UNION
    MAP
        CHARACTER*16 MAJOR
    END MAP
    MAP
        INTEGER*2  CREDITS
        CHARACTER*8  GRAD_DATE
    END MAP
END UNION
END STRUCTURE
```

変数 PERSON が上記の例の構造体 /STUDENT/ を持つように定義した場合、PERSON.MAJOR は最初のマップの欄を引用し、PERSON.CREDITS は 2 番目のマップの欄を引用します。2 番目のマップの欄の変数が初期化された後でプログラムが変数 PERSON.MAJOR を引用した場合、最初のマップが有効になり、2 番目のマップの各変数は未定義になります。

ポインタ

POINTER 文は変数とポインタの対を設定します。♦
各ポインタには対になる変数のアドレスが含まれています。

構文の規則

POINTER 文の構文は次のとおりです。

```
POINTER ( p1, v1 ) [, ( p2, v2 ) ... ]
```

ここで、

- *v1* と *v2* はポインタ基底付き変数で、指示先とも呼ばれます。
- *p1* と *p2* は対応するポインタです。

ポインタ基底付き変数とは、POINTER 文で指定されるポインタと対になっている変数です。ポインタ基底付き変数は、通常は単に基底付き変数または指示先と呼ばれます。ポインタはアドレスを含んでいる整変数です。(POINTER 文で指定する変数名は、コンパイラでは VOLATILE と見なされます)

次に単純な POINTER 文の例を示します。

```
POINTER ( P, V )
```

上記の例で、V は指示先、P はそれに関連付けられたポインタです。

その他の例については、197 ページの「POINTER」を参照してください。

ポインタの使用法

ポインタ基底付き変数を使用するには以下の手順が必要です (最初の 2 つの手順はどちらを先に実行しても構いません)。

1. **POINTER 文でポインタ基底付き変数とポインタの対を定義します。**
2. **ポインタ基底付き変数の型を定義します。**

   ポインタそのものは整数型になりますが、型宣言には使用しません。
3. **適切な大きさと型を持つ記憶領域のアドレスをポインタに設定します。**

   通常は、他にはポインタに対する明示的な処置は必要ありません。
4. **ポインタ基底付き変数を引用します。**

   一般的な Fortran 77 の文でポインタ基底付き変数を使用すれば、この変数のアドレスは必ずそれに関連付けられているポインタから取られます。

アドレスとメモリー

ポインタ基底付き変数が定義された時には、その変数には記憶領域が割り当てられていません。通常、ユーザーは自分の責任で適切な型とサイズを持つ変数のアドレスを用意し、普通の代入文または DATA 文によりそのアドレスをポインタに割り当てなければなりません。

loc()、malloc()、および free() の各ルーチンは、メモリーアドレスとポインタとの関連付けと関連解除を行います (これらのルーチンについては、第 6 章「組み込み関数」を参照)。

64 ビット環境用としてコンパイルする場合は、POINTER 文で宣言されるポインタは INTEGER*8 の値になります。

LOC() 関数によるアドレスの獲得

アドレスは組み込み関数 LOC() から得ることができます。

LOC() 関数を使用してアドレスを得る例を以下に示します。

```
* ptr1.f: LOC() を介してアドレスを割り当てる
    POINTER ( P, V )
    CHARACTER A*12, V*12
    DATA A / 'ABCDEFGHIJKL' /
    P = LOC( A )
    PRINT *, V(5:5)
    END
```

上記の例では、CHARACTER 文は A には 12 バイトの記憶領域を割り当てますが、V には記憶領域を割り当てません。V はポインタ基底付き変数なので V の型だけを指定します。次に A のアドレスが P に割り当てられるので、V を使用するとポインタ P を通して A が引用されます。このプログラムは E を出力します。

64 ビット環境用としてコンパイルする場合は、LOC() によって INTEGER*8 の値が戻されます。受け取る変数は、アドレスが切り捨てられる可能性を避けるため、ポインタまたは INTEGER*8 のどちらかの変数でなければなりません。

MALLOC() 関数による記憶領域とアドレスの獲得

関数 MALLOC() は記憶領域を割り当て、その領域の開始アドレスを戻します。この関数の引数は整数で、割り当てるメモリーの量をバイト単位で指定します。この関数が正常に実行されるとその領域の最初の項目へのポインタを戻し、失敗すると整数 0 を戻します。記憶領域は初期化されません。

64 ビット環境のコンパイルでは、LOC () によって INTEGER*8 の値が戻されます。受け取る変数は、アドレスが切り捨てられる可能性を避けるため、ポインタまたは INTEGER*8 のどちらかの変数でなければなりません。

MALLOC を使用してポインタに記憶領域を割り当てる例を以下に示します。

```
    COMPLEX Z
    REAL X, Y
    POINTER ( P1, X ), ( P2, Y ), ( P3, Z )
    …
    P1 = MALLOC ( 10000 )
    …
```

上記の例では、MALLOC() が 10,000 バイトのメモリーを割り当て、そのメモリーブロックのアドレスとポインタ P1 を関連付けています。

FREE() サブルーチンによる記憶領域の割り当て解除

サブルーチン FREE() は先に MALLOC() により割り当てられた記憶領域を解放します。
FREE() に与える引数は、MALLOC() により返されたポインタでなければなりません。これはプログラマが FREE() に与える必要があります。記憶領域はメモリーマネージャに戻され、メモリーマネージャはプログラマがその記憶領域を使用できないようにします。

FREE による割り当て解除の例を以下に示します。

```
    POINTER ( P1, X ), ( P2, Y ), ( P3, Z )
    …
    P1 = MALLOC ( 10000 )
    …
    CALL FREE ( P1 )
    …
```

上記の例では、MALLOC() が 10,000 バイトのメモリーを割り当て、そのメモリーがポインタ P1 に関連付けられます。その後、FREE() がその同じ 10,000 バイトをメモリーマネージャに戻しています。

特記事項

malloc()、loc()、free() を使用してポインタとメモリー割り当てを操作する場合は、以下の点に特に注意してください。

- ポインタはコンパイラによって自動的に整数型に宣言されます。プログラマが自分でポインタを型宣言してはいけません。
- ポインタ基底付き変数自身をポインタにすることはできません。
- ポインタ基底付き変数は構造体も含めてどの型にすることもできます。
- ポインタ基底付き変数は、型宣言文の中でサイズが指定されていても、宣言された時点では記憶領域は割り当てられません。
- ポインタ基底付き変数を仮引数として使用したり、COMMON 文、EQUIVALENCE 文、DATA 文、または NAMELIST 文の中で使用することはできません。

- 主プログラム内では、ポインタ基底付き変数の寸法式は定数式でなければなりません。サブルーチンと関数内では、仮引数に対するのと同じ規則がポインタ基底付き配列変数に対して適用されます。つまり、式の中に仮引数と共通ブロックの中の変数を含むことができます。式の中のすべての変数は、サブルーチンまたは関数の呼び出し時に整数値で定義されなければなりません。

- 32 ビット環境では、アドレス式が `INTEGER*4` の範囲を超えることはできません。式が範囲内 (-2147483648, 2147483647) にない場合、結果は予測できません。

- 64 ビット環境用としてコンパイルする場合は、`malloc64()` を使用して 64 ビットのアドレス領域にアクセスします。ルーチン `malloc64()` は、`INTEGER*8` の引数を取り、64 ビットのポインタ値を戻します。64 ビットのプログラムでは、`POINTER` 文で定義されたポインタは 64 ビットの `INTEGER*8` になります。『Fortran ライブラリ・リファレンス』および `malloc`(3F) のマニュアルページを参照してください。

最適化とポインタ

ポインタを使用すると、大域的オプティマイザの処理の前提範囲が限定されます。次の 2 つの場合を比較してください。

- ポインタが使用されていない場合、サブルーチンまたは関数の呼び出しによって変更されるのは、共通ブロックの変数、または引数として渡される変数だけであることをオプティマイザは認識しています。

- ポインタが使用されている場合、この前提はもはや無効となります。なぜなら、ルーチンは引数のアドレスを取り出して共通ブロックにあるポインタの中に保存することができ、以後このポインタを自分自身または他のルーチンの呼び出しで使用するからです。

したがって、オプティマイザは、サブルーチンまたは関数の呼び出しにおいて引数として渡される変数が、他の呼び出しにより変更される可能性を仮定しなければなりません。このようなポインタの無制約な使用は、ポインタを使用しない大部分のプログラムに対しても最適化のレベルを低下させることになります。

一般的なガイドライン

ポインタが最適化を行うには、 2 つの選択があります。

- 最適化レベルを `-O4` にして、ポインタを使用しない。

- ポインタを計算するデータの場所を指定し、副プログラムに渡すことのみに使用する。ポインタにその他の操作を行うと、不正な結果をもたらします。

2 番目の選択には更に選択肢があります。ポインタをルーチンに局所的に使用し、それは最適化せず、計算を行うルーチンのみを最適化します。呼出しルーチンを別々のファイルに入れることにより、あるルーチンは最適化し、あるものは最適化しないということができます。

例：`-O3` または `-O4` でも比較的安全なコーディング

```
REAL A, B, V(100,100)   このプログラム単位は、
POINTER ( P, V )        アドレスを取得して渡す以外に
P = MALLOC(10000)       P では何も行わない
…
CALL CALC ( P, A )
...
END

SUBROUTINE CALC ( ARRAY, X )
...
RETURN
END
```

レベル `-O4` で `CALC` のみを最適化したい場合は、`CALC` 内ではポインタを使用しないでください。

注意事項

次のようなコーディングの仕方では、レベル `-O3` または `-O4` での最適化で問題が生じる可能性があります。

- プログラム単位が、ポインタで算術計算を行う。
- 副プログラムが各呼び出し間で引数のアドレスを保存する。
- 関数がポインタ引数の値を戻さずに、引数のアドレスを戻す。
- ポインタを通して変数の引用があるが、その変数アドレスは `LOC()` または `MALLOC()` から明示的に得られたものではない。

例： -O3 または -O4 で問題を発生させるコード

```
      COMMON A, B, C
      POINTER ( P, V )
      P = LOC(A) + 4    最適化を行うと、ここで問題が起こる可能性がある
      …
```

コンパイラは、P を通した引用により A が変更されると仮定しますが、B が変更されることは仮定しません。この仮定により誤ったコードが生成される可能性があります。

第3章

式

この章では、Fortran の式について説明しています。

式と演算子とオペランド

式は 1 つ以上のオペランド、ゼロ以上の演算子、およびゼロ以上の括弧の組み合わせから成ります。

式には次の 3 種類があります。

- 算術式は結果として単一の算術値をもたらします。
- 文字式は結果として文字型の単一値をもたらします。
- 論理式または関係式は結果として単一の論理値をもたらします。

演算子はどのような処理または演算を行うかを指示します。

オペランドは処理を適用する項目を指示します。オペランドは以下のいずれかのデータ項目です。

- 定数
- 変数
- 配列要素
- 関数
- 部分列
- 構造体の記録欄 (結果としてスカラーデータ項目をもたらす場合)
- 式

算術式

算術式は結果として単一の算術値をもたらします。算術式のオペランドは次の型を持っています。♦ は規格外の機能を示します。

- BYTE ♦
- COMPLEX
- COMPLEX*32 (SPARC のみ) ♦
- DOUBLE COMPLEX ♦
- DOUBLE PRECISION
- INTEGER
- LOGICAL
- REAL
- REAL*16 (SPARC のみ) ♦

算術式の演算子は以下のいずれかです。

表 3-1 算術演算子

演算子	意味
**	べき乗
*	乗算
/	除算
-	減算または単項マイナス
+	加算または単項プラス

BYTE または LOGICAL オペランドを算術演算子と組み合わせると、整数データとして解釈されます。

これらの演算子は次の形式では二項演算子です。

$$a \oplus b$$

a と *b* はオペランドであり、⊕ の位置には **、*、/、-、または + のいずれかの演算子が入ります。

例： 二項演算子

```
A-Z
X*B
```

演算子の + と - は次の形式では単項演算子です。

$$\oplus\ b$$

ここで *b* はオペランドであり、⊕ の位置には - か + 演算子のどちらかが入ります。

例： 単項演算子

```
-Z
+B
```

基本的な算術式

各々の算術演算子の基本的な式を以下に示します。

表 3-2　算術式

式	意味
`a ** z`	a を z 乗する
`a / z`	a を z で割る
`a * z`	a に z を掛ける
`a - z`	a から z を引く
`-z`	z の符号を反転する
`a + z`	z を a に足す
`+z`	z と同じ

括弧がなく、式に 2 つ以上の演算子がある場合、演算順序は演算子の優先順位によって決まります。1 つの例外を除き、演算子の優先順位が等しければ左から右へ評価されます。

表 3-3　算術演算子の優先順位

演算子	優先順位
**	最初
* /	2 番目
+ -	最後

左から右への規則の例外を以下に示します。

```
F ** S ** Z
```

上記の式は次のように評価されます。

```
F ** (S ** Z)
```

f77 では 2 つの連続する演算子を使用できます。♦

例： 2 つの連続する演算子

```
    X ** -A * Z
```

上記の式は次のように評価されます。

```
    X ** (-(A * Z))
```

上の例において、コンパイラはまず ** を評価しますが、X を何乗するかを知る必要が生じます。式の残りの部分を見て - か * を選択しなければなりません。コンパイラはまず * を、次に - を、最後に ** を選択します。

混合モード

BYTE オペランドを除いて、両方のオペランドが同じ型を持っている場合、結果の値も同じ型を持ちます。(両方のオペランドの型が BYTE である算術演算の結果は、デフォルトの整数になります。) オペランドが異なる型を持っている場合、弱い型が強い型に拡張されます。弱い型とは精度が低い、または記憶単位が小さい方の型です。以下の強度の一覧表にまとめます。

データ型	強度
BYTE または LOGICAL*1	1 （弱い）
LOGICAL*2	2
LOGICAL*4	3
INTEGER*2	4
INTEGER*4	5
INTEGER*8	6
LOGICAL*8	6
REAL*4 （実数）	6
REAL*8 （倍精度）	7
REAL*16 （4 倍精度）（SPARC のみ）	8
COMPLEX*8 （複素数）	9
COMPLEX*16 （倍精度複素数）	10
COMPLEX*32 （4 倍精度複素数）（SPARC のみ）	11 （強い）

注 – REAL*4、INTEGER*8、LOGICAL*8 は同じ強度ですが、異なるオペランドの組み合わせ結果です。たとえば、INTEGER*8 と強度 1〜5 の型とを組み合わせると、結果は INTEGER*8 になります。1 つのオペランドが REAL*4 で、もう 1 つが強度 1〜5 の型の場合、結果は INTEGER*4 です。LOGICAL*8 は、結果のサイズが 8 バイトであるということを意味するにすぎません。

混合モードの例： R が実数で、I が整数であれば、次の式は実数型を持ちます。

```
R * I
```

まず I が実数に拡張され、乗算が行われるからです。

規則

式のデータ型に関する規則は次のとおりです。

- 式に 2 つ以上の演算子がある場合、最後に行われた演算の型が式の値の型になります。
- 整数の演算子は両オペランドが整数のときだけ適用されます。

例： 次の式は結果としてゼロになります。

```
    2/3 + 3/4
```

- INTEGER*8 オペランドが REAL*4 オペランドと混合されると、結果は REAL*8 になります。

これに対する拡張が 1 つあります。算術演算の中の論理型またはバイト型オペランドは整数として使用されるということです。

- 実数の演算子は両オペランドが実数か、またはバイト、論理、または整数と、実数のオペランドの組み合せに適用されます。実数オペランドと組み合わされた整数オペランドは実数に拡張されます。拡張後の実数の小数部はゼロです。たとえば、R が実数で、I が整数であれば、R+I は実数になります。ただし、(2/3)*4.0 は 0 になることに注意してください。
- 倍精度の演算子は倍精度オペランドに適用され、これより低い精度のオペランドは倍精度に拡張されます。拡張後の倍精度数値の新しい最下位ビットはゼロに設定されます。実数オペランドを拡張してもオペランドの精度は上がりません。
- 複素数の演算子は複素数オペランドに適用されます。整数オペランドは実数に拡張され、複素数オペランドの実部として使用されます。虚部はゼロに設定されます。
- 論理変数に対して数値演算子が許可されます。♦ Fortran 規格が数値を要求するあらゆる場所に論理値を使用することができます。ここでの数値とは integer、real、complex、double precision、double complex、real*16 (SPARC のみ) のいずれかです。コンパイラは論理値を暗黙的に適切な数値に変換します。これらの機能を使用すると、プログラムは移植できない場合があります。

次の例は integer 型と logical 型の組み合せを示しています。

```
      COMPLEX  C1 / ( 1.0, 2.0 ) /
      INTEGER*2 I1, I2, I3
      LOGICAL L1, L2, L3, L4, L5
      REAL R1 / 1.0 /
      DATA I1 / 8 /, I2 / 'W' /, I3 / 0 /
      DATA L1/.TRUE./, L2/.TRUE./, L3/.TRUE./, L4/.TRUE./,
     &   L5/.TRUE./
      L1 = L1 + 1
      I2 = .NOT. I2
      L2 = I1 .AND. I3
      L3 = I1 .OR. I2
      L4 = L4 + C1
      L5 = L5 + R1
```

結果の型

整数オペランドに論理演算を適用すると、演算はビットごとに行われます。結果は整数です。

オペランドが整数と論理オペランドの組み合せであれば、論理オペランドは整数に変換され、結果は整数になります。

算術代入

算術代入文は変数、配列要素、記録、または記録欄に値を代入します。構文は次のとおりです。

v = *e*	
e	算術式、文字定数、または論理式
v	数値変数、配列要素、または記録欄

論理値を数値として代入することは許可されますが、規格外なので移植できない場合があります。結果のデータ型は *v* のデータ型になります。♦

算術代入文を実行すると、式 *e* が評価され、(型が異なる場合) v の型に変換され、v に代入されます。結果の値の型は次表に従って決まります。

文字定数は整数型または実数型の変数に代入できます。文字定数とはホレリス定数、またはアポストロフィか引用符で囲まれた文字列です。文字はデータの変換を伴わずに変数に移されます。これは規格外なので移植できない場合があります。♦

v の型	*e* の型
INTEGER*2、INTEGER*4、INTEGER*8	INT (*e*)
REAL	REAL (*e*)
REAL*8	DBLE (*e*)
REAL*16 (SPARC のみ)	QREAL (*e*) (SPARC のみ)
DOUBLE PRECISION	DBLE (*e*)
COMPLEX*8	CMPLX (*e*)
COMPLEX*16	DCMPLX (*e*)
COMPLEX*32 (SPARC のみ)	QCMPLX (*e*) (SPARC のみ)

注 – オプション -i2、-dbl、-r8、または -xtypemap のいずれかを指定してコンパイルした場合、*e* の想定される型に影響があります。これについては第 2 章「データ型とデータ項目」を参照してください。これらのオプションについては、『Fortran ユーザーズガイド』も参照してください。

例： 算術代入

```
INTEGER I2*2, J2*2, I4*4
LOGICAL L1, L2
REAL R4*4, R16*16
DOUBLE PRECISION DP
COMPLEX C8, C16*16
J2 = 29002
I2 = J2
I4 = (I2 * 2) + 1
DP = 6.4D0
QP = 9.8Q1
R4 = DP
R16 = QP
C8 = R1
C8 = ( 3.0, 5.0 )
I2 = C8
C16 = C8
C8 = L1
R4 = L2
```

文字式

文字式とはオペランドが文字型を持つ式のことです。文字式は結果として、 1 つ以上の文字を持つ文字型の 1 つの値をもたらします。唯一の文字演算子は連結演算子 // です。

式	意味
a // *z*	*a* と *z* を連結する

2 つの文字列を連結した結果は、左オペランドの文字のすぐ後に右オペランドの文字が続く 3 番目の文字列になります。連結演算 *a*//*z* の値は文字列であり、この文字列の値は *z* の値を右側に連結した *a* の値です。この文字列の長さは *a* と *z* の長さの合計です。

オペランドは以下のいずれかのデータ項目です。

- 文字定数
- 文字変数

- 文字配列要素
- 文字関数
- 部分列
- 構造体の記録欄 (結果としてスカラー文字データ項目をもたらす場合)

例： 文字式 (C、S、および R.C は文字と仮定します)

```
'wxy'
'AB' // 'wxy'
C
C // S
C(4:7)
R.C
```

以下に、(規格外の) 例外を示します。♦

- **制御文字** — 制御文字を入力するには、Control キーを押しながら別のキーを押します。たいていの制御文字はこの方法で入力できますが、Control-A、 Control-B、Control-C、または Control-J は入りません。

例： Control-C を入力する有効な方法

```
CHARACTER etx
etx = CHAR(3)
```

- **複数バイト文字** — 漢字などの複数バイト文字は、コメントや文字列で使用することができます。

文字列の代入

文字列の代入形式は次のとおりです。

v = *e*	
e	代入値を与える式
v	変数、配列要素、部分列、文字記録欄

文字代入の意味は、文字を右側から左側へコピーすることです。文字代入文を実行すると、文字式が評価され、結果の値が *v* に代入されます。

- *e* が *v* より長い場合、文字の右側が切り捨てられます。
- *e* が *v* より短い場合、空白文字が右側に詰められます。

例： 次のプログラムは "joinedΔΔ" を表示します。

```
CHARACTER A*4, B*2, C*8
A = 'join'
B = 'ed'
C = A // B
PRINT *, C
END
```

また、次のプログラムは equal 文字列を表示します。

```
IF ( ('ab' // 'cd') .EQ. 'abcd' ) PRINT *, 'equal'
END
```

例： 文字代入

```
CHARACTER BELL*1, C2*2, C3*3, C5*5, C6*6
REAL Z
C2 = 'z'
C3 = 'uvwxyz'
C5 = 'vwxyz'
C5(1:2) = 'AB'
C6 = C5 // C2
I = 'abcd'
Z = 'wxyz'
BELL = CHAR(7)    制御文字 (^G)
```

上記の結果は次のとおりです。

C2	'zΔ'	後続の空白
C3	'uvw'	
C5	'ABxyz'	
C6	'ABxyzz'	すなわち、C2 からの 'z'

C2	'zΔ'	後続の空白
I	'abcd'	
Z	'wxyz'	
BELL	07 hex	Control-G、ベル

例 : ホレリス代入 ♦

```
      CHARACTER S*4
      INTEGER I2*2, I4*4
      REAL R
      S = 4Hwxyz
      I2 = 2Hyz
      I4 = 4Hwxyz
      R = 4Hwxyz
```

代入の規則

文字代入の規則は次のとおりです。

- 左側が右側より長い場合、左側に後続の空白が詰められます。
- 左側が右側より短い場合、後続文字は切り捨てられます。
- 文字列代入文の左辺と右辺はオーバーラップすることはできません。47 ページの「部分列」を参照してください。

論理式

論理式は 1 つ以上の論理オペランドと論理演算子の組み合わせから成ります。論理式は結果として単一の論理値をもたらします。演算子は以下のいずれかです。

表 3-4　論理演算子

演算子	標準名
.AND.	論理積
.OR.	論理和 （内包的論理和）
.NEQV.	論理非等価
.XOR.	排他的論理和
.EQV.	論理等価
.NOT.	論理否定

区切り文字のピリオドが必要です。

2 番めの論理演算子が .NOT. 演算子でないかぎり、2 つの論理演算子を連続して使用することはできません。

論理演算子は以下に示す優先順位に従って評価されます。

表 3-5　論理演算子の優先順位

演算子	優先順位
.NOT.	高位
.AND.	--
.OR.	--
.NEQV.、.XOR.、.EQV.	低位

論理演算子の優先順位が等しければ、左から右へ評価されます。

論理式の中で論理演算子を他の演算子と混在して使用する場合、優先順位は次のようになります。

表 3-6　演算子の優先順位

演算子	優先順位
算術演算子	高位
文字演算子	--
関係演算子	--
論理演算子	低位

基本となる論理式の意味は次のとおりです。

表 3-7　論理式とその意味

論理式	意味
X .AND. Y	X、Y ともに真
X .OR. Y	X、Y の少なくとも一方が真
X .NEQV. Y	X、Y ともに真ではなく、ともに偽でもない
X .XOR. Y	X、Y の一方だけが真
X .EQV. Y	X、Y ともに真またはともに偽
.NOT. X	論理否定

サイズが違うオペランドに対する論理演算では、結果の論理値は最大オペランドのサイズになります。たとえば L2.AND.L4 の場合、L2 が LOGICAL*2 で、L4 が LOGICAL*4 であると、結果は LOGICAL*4 になります。

BYTE ♦ の変数は、LOGICAL*1 として処理されます。

論理代入

論理式の値を論理変数に代入する構文は次のとおりです。

v = *e*

e	論理式、-128 と 127 の間の整数、または単一の文字定数 ♦
v	論理変数、配列要素、または記録欄

論理代入文を実行すると論理式 *e* が評価され、結果の値が *v* に代入されます。*e* が (-128 と 127 の間の整数または単一の文字定数ではなく) 論理式の場合、*e* は真か偽のどちらかの値を持っていなければなりません。

任意のサイズの論理式を任意のサイズの論理変数に代入できます。

数値を論理値として代入することはできます (ゼロ以外のすべての値が .TRUE. として処理され、ゼロは .FALSE. として処理される) が、規格外なので移植できない場合があります。♦

例： 論理代入

```
LOGICAL B1*1, B2*1
LOGICAL L3, L4
B2 = B1
B1 = L3
L4 = .TRUE.
```

関係演算子

関係演算子は 2 つの算術式または 2 つの文字式を比較し、結果として単一の論理値をもたらします。演算子は以下のいずれかです。

演算子	意味
.LT.	小さい
.LE.	小さいか等しい
.EQ.	等しい
.NE.	等しくない
.GT.	大きい
.GE.	大きいか等しい

区切り文字のピリオドが必要です。

すべての関係演算子の優先順位は同じです。文字演算子と算術演算子は関係演算子に優先します。

関係式の場合、2 つのオペランドの各々が評価され、次にその 2 つの値が比較されます。指定した関係が成り立つ場合の値は真であり、そうでない場合は偽です。

例：関係演算子

```
NODE .GE. 0
X .LT. Y
U*V .GT. U-V
M+N .GT. U-V 混合モード：整数  M+N  は実数になる
STR1 .LT. STR2 STR1 と STR2 は文字型
S .EQ. 'a' S は文字型
```

文字関係式の場合、

- 「小さい」は「ASCII コード系の大小順序において先行する」ことを意味します。
- 一方のオペランドが短い場合、短いほうのオペランドには長い方のオペランドの長さに合わせて右側に空白が詰められます。

定数式

定数式は明示的な定数とパラメータ (定数名)、そして Fortran 演算子から構成されています。各々のオペランドは、それ自体が別の定数式、定数、定数の英字名、または定数引数によって呼び出される組み込み関数になっています。

例： 定数式

```
PARAMETER (L=29002), (P=3.14159), (C='along the ')
PARAMETER ( I=L*2, V=4.0*P/3.0, S=C//'riverrun' )
PARAMETER ( M=MIN(I,L), IA=ICHAR('A') )
PARAMETER ( Q=6.4Q6, D=2.3D9 )
K = 66 * 80
VOLUME = V*10**3
DO I = 1, 20*3
```

定数式の制限事項は次のとおりです。

- 定数式は定数が許可される場所ではどこでも許可されますが、DATA 文または規格に合った FORMAT 文の規格範囲では許可されません。
- 定数式は変数書式の式で許可されます。♦
- 浮動小数点のべき乗は許可されず、警告が出されます。

例： 浮動小数点のべき乗は許可されません。

```
demo% cat ConstExpr.f
    parameter (T=2.0*(3.0**2.5))
    write(*,*) t
    end
demo% f77 ConstExpr.f
ConstExpr.f:
 MAIN:
"ConstExpr.f", line 1: 警告:
    パラメータ t が定数以外に設定されています
demo% a.out
    31.1769
demo%
```

記録代入

記録代入の一般的な形式は次のとおりです。♦

v = *e*	
e	記録または記録欄
v	記録または記録欄

e と *v* は同じ構造体を持つ必要があります。つまり、各々が同じ数の欄を持ち、対応する欄は同じ型とサイズでなければなりません。

例： 記録代入と記録欄代入

```
STRUCTURE /PRODUCT/
INTEGER*4 ID
CHARACTER*16 NAME
CHARACTER*8 MODEL
REAL*4 COST
REAL*4 PRICE
END STRUCTURE
RECORD /PRODUCT/ CURRENT, PRIOR, NEXT, LINE(10)
…
CURRENT = NEXT
LINE(1) = CURRENT
WRITE ( 9 ) CURRENT
NEXT.ID = 82
```

上記の例では、最初の代入文は 1 つの記録全体 (5 つの欄全部) を別の記録に、2 番目の代入文は記録全体を記録の配列の最初の要素にコピーします。WRITE 文は記録全体を出力し、最後の文は記録の ID 欄を 82 に設定します。

式の評価

次の制限事項はすべての算術式、文字式、関係式、および論理式に適用されます。

- 式、変数、配列要素、部分列、記録欄、ポインタ、または関数のいずれかの項目を引用する場合、その項目は引用するときに定義されている必要があります。
- 整数のオペランドは整数値で定義する必要があり、ASSIGN 文による文番号値で定義してはいけません。
- 引用する部分列のすべての文字は、引用するときに定義されている必要があります。
- 関数を引用することにより、同じ文の中における他の項目の値が変更されてはなりません。
- 関数を引用することによりブロック内の項目を変更するときは、同じ文中の他の関数引用の値に影響するような変更はできません。

第4章

文

この章では Sun WorkShop FORTRAN 77 コンパイラ `f77` の文について説明しています。規格外の文はシンボル記号 (♦) で示しています。(規格準拠の詳細については、第 1 章を参照してください。) 文の例は、付録 B「文の例」を参照してください。

ACCEPT

`ACCEPT` ♦ 文は標準入力から読み取り、以下の構文を使用します。

構文

`ACCEPT` *f [, iolist]*

`ACCEPT` *grname*

パラメータ	説明
f	書式識別子
iolist	変数、部分列、配列、および記録の並び
grname	変数群の名前

説明

`ACCEPT` *f [,iolist]* 文は `READ` *f [,iolist]* に相当し、Fortran 77 の旧バージョンとの互換用に提供されています。

例 : 並び入力

```
REAL VECTOR(10)
ACCEPT *, NODE, VECTOR
```

ASSIGN

ASSIGN 文は文番号を変数に代入します。

ASSIGN *s* TO *i*

パラメータ	説明
s	文番号
i	整変数

説明

文番号 *s* は実行文または FORMAT 文の文番号です。

文番号は、 ASSIGN 文と同じプログラム単位内に定義されている文の文番号でなければなりません。

整数の変数 *i* には、一度文番号を代入した後、同じ文の番号、異なる文の番号、整数のいずれかを再代入することができます。ただし、INTEGER*2 で宣言することはできません。

文番号を変数に代入すると、以下で引用することができます。

- 割り当て型 GO TO 文
- 入出力文の書式識別子として

制限事項

変数を割り当て型 GO TO 文の文番号、または書式識別子として引用するには、その前に変数に文番号を代入しなければなりません。

i に文番号値を代入しているときは、*i* で演算を行ってはいけません。

64 ビット プラットフォームでは、ASSIGN 文によって変数 i に格納された実数値はプログラムで使用できません。ただし、割り当て型 GO TO 文中で使用する場合、または入出力文中で書式識別子として使用する場合は除きます。さらに、割り当て型 GO TO 文の中、または書式識別子として使用できる変数は、ASSIGN 文によって指定された変数に限ります。

例

例 1 : 実行文の文番号を代入

```
    IF(LB.EQ.0) ASSIGN 9 TO K
    …
    GO TO K
    …
9   AKX = 0.0
```

例 2 : FORMAT 文の文番号を代入

```
    INTEGER PHORMAT
2   FORMAT ( A80 )
    ASSIGN 2 TO PHORMAT
    …
    WRITE ( *, PHORMAT ) 'FORMAT 文の文番号が割り当てられました。'
```

代入

代入文は値を変数、部分列、配列要素、記録、または記録欄に代入します。

v = *e*

パラメータ	説明
v	変数、部分列、配列要素、記録、または記録欄
e	代入値を与える式

説明

値は定数または式の結果です。代入文の種類には算術代入、論理代入、文字代入、および記録代入があります。

算術代入

v は数値型であり、変数、配列要素、または記録欄の名前です。

e は算術式、文字定数、または論理式です。論理値を数値として代入することは規格外のため移植できない場合があります。結果のデータ型は *v* のデータ型になります。♦

算術代入文を実行すると、式 *e* が評価され、(型が異なる場合) *v* の型へ変換され、*v* に代入されます。結果の値の型は下表に従って決まります。

v の型	*e* の型
INTEGER*2、INTEGER*4 **または** INTEGER*8	INT(*e*)
REAL	REAL(*e*)
REAL*8	REAL*8
REAL*16 (SPARC のみ)	QREAL(*e*) (SPARC のみ)
DOUBLE PRECISION	DBLE(*e*)
COMPLEX*8	CMPLX(*e*)
COMPLEX*16	DCMPLX(*e*)
COMPLEX*32 (SPARC のみ)	QCMPLX(*e*) (SPARC のみ)

注 – オプション -i2、-dbl、-r8、または -xtypemap のいずれかを指定してコンパイルした場合、変数と式のデフォルトのデータサイズが変更されることがあります。これについては第 2 章「データ型とデータ項目」を参照してください。これらのオプションについては、『Fortran ユーザーズガイド』も参照してください。

例 : 代入文

```
REAL A, B
DOUBLE PRECISION V
V = A * B
```

上記の例は、次の例とまったく同じようにコンパイルされます。

```
REAL A, B
DOUBLE PRECISION V
V = DBLE( A * B )
```

論理代入

v は論理型の変数、配列要素、または記録欄の名前です。

e は論理式、-128 と 127 の間の整数、または単一の文字定数です。

論理代入文を実行すると、論理式 *e* が評価され、結果の値が *v* に代入されます。*e* が (-128 と 127 の間の整数または単一の文字定数ではなく) 論理式の場合、*e* は真か偽のどちらかの値を持ちます。

任意のサイズの論理式を任意のサイズの論理変数に代入できます。`LOGICAL` 文に関する節で、論理変数のサイズについてさらに詳しく述べます。

文字代入

この定数はホレリス定数またはアポストロフィ (') か引用符 (") で区切られた文字列です。文字列には制御文字の Control-A、Control-B、Control-C を含むことはできません。すなわち、Control キーを押しながら、A、B、C のキーを押すことはできません。これらの制御文字を必要とする場合は、関数 `CHAR()` を使用してください。

引用符を使用して文字定数を区切ると、`-xl` オプションを指定してコンパイルすることができません。この場合の引用符は 8 進定数を意味するからです。文字は変換されずに変数に移されます。これは移植できない場合があります。

演算子 `//` を含む文字式は `CHARACTER` 型の項目にだけ代入できます。ここで *v* は `CHARACTER` 型で、変数、部分列、配列要素、または記録欄の名前です。*e* は文字式です。

文字代入文を実行すると、文字式が評価され、結果の値が *v* に代入されます。*e* の長さが *v* の長さより長い場合、右側の文字が切り捨てられます。*e* の長さが *v* の長さより短かければ、空白文字が右側に詰められます。

記録代入

v と *e* は各々が記録または記録欄です。♦

e と *v* は同じ構造体である必要があります。以下の場合は同じ構造体です。

- *e* と *v* が同じ要素データ型を持つ欄である。
- *e* と *v* が同じ数の欄を持つ記録で、対応する欄が同じ要素データ型である。
- *e* と *v* が同じ数の欄を持つ記録で、対応する欄が上記で定義されたもの (同じ要素データ型であること) と同じ構造体を持つ部分構造体である。

RECORD 文と STRUCTURE 文に関する節で、記録の構造についてさらに詳しく述べます。

例

例 1 : 算術代入

```
INTEGER I2*2, J2*2, I4*4
REAL R1, QP*16  ! (REAL*16 は SPARC のみ)
DOUBLE PRECISION DP
COMPLEX C8, C16*16, QC*32  ! (COMPLEX*32 は SPARC のみ)
J2 = 29002
I2 = J2
I4 = (I2 * 2) + 1
DP = 6.4D9
QP = 6.4Q9
R1 = DP
C8 = R1
C8 = ( 3.0, 5.0 )
I2 = C8
C16 = C8
C32 = C8
```

例 2 : 論理代入

```
LOGICAL B1*1, B2*1
LOGICAL L3, L4
L4 = .TRUE.
B1 = L4
B2 = B1
```

例 3 : ホレリス代入

```
CHARACTER S*4
INTEGER I2*2, I4*4
REAL R
S = 4Hwxyz
I2 = 2Hyz
I4 = 4Hwxyz
R = 4Hwxyz
```

例 4 : 文字代入

```
CHARACTER BELL*1, C2*2, C3*3, C5*5, C6*6
REAL Z
C2 = 'z'
C3 = 'uvwxyz'
C5 = 'vwxyz'
C5(1:2) = 'AB'
C6 = C5 // C2
BELL = CHAR(7)     制御文字 (^G)
```

上記の結果は次のとおりです。

```
C2    は、'zΔ'          後続に空白があります
C3    は、'uvw'
C5    は、'ABxyz'
C6    は、'ABxyzz'      C5 の左側に余分の z があります
BELL は、   16 進の 07  Control-G、ベルです
```

例 5 : 記録代入と記録欄代入

```
STRUCTURE /PRODUCT/
INTEGER*4 ID
CHARACTER*16 NAME
CHARACTER*8 MODEL
REAL*4 COST
REAL*4 PRICE
END STRUCTURE
RECORD /PRODUCT/ CURRENT, PRIOR, NEXT, LINE(10)
…
CURRENT = NEXT          記録から記録へ
LINE(1) = CURRENT       記録から配列要素へ
WRITE ( 9 ) CURRENT     記録全体を書き込む
NEXT.ID = 82            値を欄に割り当てる
```

AUTOMATIC

AUTOMATIC ♦ 文を使用すると、副プログラムが再帰的に呼び出されるごとに各指定項目のコピーが確実に行われます。

また、この文を使用すると、副プログラムが RETURN 文で終了する場合に、副プログラムの外で指定項目が確実に未定義になります。

AUTOMATIC *vlist*

パラメータ	説明
vlist	変数および配列の並び

説明

自動変数を使用すると、手続きの呼び出しごとに 1 つコピーが行われます。呼び出したときに局所変数が未定義になるのを避けるために、f77 はすべての局所変数をデフォルトで静的に定義すると共に、あらゆる変数を静的か自動のどちらかに分類します。STATIC ♦ 文、AUTOMATIC ♦ 文、または IMPLICIT 文で明示的に変数を静的または自動に定義することもできます。『Fortran ユーザーズガイド』の第 3 章「式」にある -stackvar オプションも参照してください。

AUTOMATIC の使用法として、関数のスタート時にすべての自動変数を宣言することができます。

例：暗黙の自動変数を持つ再帰関数

```
    INTEGER FUNCTION NFCTRL( I )
    IMPLICIT AUTOMATIC (A-Z)
    ...
    RETURN
    END
```

局所変数および局所配列はデフォルトでは静的になるため、一般に SAVE を使用する必要はありませんが、SAVE を使用することにより移植性を確実にすることができます。また、RETURN 以外の方法により副プログラムを終了する場合にも SAVE を使用すると安全です。

制限事項

自動変数と自動配列は、DATA または SAVE 文では使用できません。

また、引数と関数値は f77 によって常に自動にされるため、 DATA、RECORD、STATIC、SAVE 文では使用できません。

例

例 : AUTOMATIC のその他の使用例を示します。

```
    AUTOMATIC A, B, C
    REAL P, D, Q
    AUTOMATIC P, D, Q
    IMPLICIT AUTOMATIC (X-Z)
```

例 : AUTOMATIC の場合、構造体の動作は予測できません。

```
demo% cat autostru.f
    AUTOMATIC X
    STRUCTURE /ABC/
      INTEGER I
    END STRUCTURE
    RECORD /ABC/ X X は自動だが、構造体にすることはできない。
    X.I = 1
    PRINT '(I2)', X.I
    END
demo% f77 -silent autostru.f
demo% a.out
*** TERMINATING a.out (訳:停止 a.out)
*** Received signal 10 (SIGBUS)(訳: シグナル 10 SIGBUS を受け取りました)
Bus Error (core dumped)          (訳: バスエラー)
demo%
```

制限事項

AUTOMATIC 文と型宣言文を結合して AUTOMATIC 型宣言文を作ることはできません。

たとえば、AUTOMATIC REAL X では、変数 X が AUTOMATIC と REAL であると宣言しているのではなく、変数 REALX が AUTOMATIC であると宣言しています。

BACKSPACE

BACKSPACE 文は先行する記録の直前に指定ファイルを位置付けます。

```
BACKSPACE u

BACKSPACE( [UNIT= ] u [, IOSTAT=ios ] [, ERR=s ] )
```

パラメータ	説明
u	ファイルに結合された外部装置の装置識別子
ios	入出力状態指定子、整変数、または整数の配列要素
s	誤り指定子：*s* は、BACKSPACE 文が使用されているプログラム単位内にある実行文の文番号であること。BACKSPACE 文の実行中にエラーが生じるとプログラムの制御はこの文番号に移される。

説明

端末ファイルにおける BACKSPACE は無効です。

u は順番探査に結合する必要があります。直接探査ファイルにおける BACKSPACE 文の実行は FORTRAN 77 規格では定義されておらず、どのようになるか保証できません。直接探査ファイル、または追加探査ファイルにおいて BACKSPACE 文を使用することは好ましくありません。FORM='BINARY' として開いたファイルに対しては BACKSPACE は指定できず、実行時にエラーとなります。

BACKSPACE 文を実行すると、ファイル位置が次のように変更されます。

実行前	実行後
ファイルの始め	そのまま
ファイル終了記録の後	ファイル終了記録の前
直前の記録の始め	同じ記録の開始位置

例

例 1 : 簡単なバックスペース

```
    BACKSPACE 2
    LUNIT = 2
    BACKSPACE LUNIT
```

例 2 : エラートラップを伴うバックスペース

```
    INTEGER  CODE
    BACKSPACE ( 2, IOSTAT=CODE, ERR=9 )
    …
9   WRITE (*,*) 'BACKSPACE 中にエラー'
    STOP
```

BLOCK DATA

BLOCK DATA 文は、名前付き共通ブロック内の変数と配列を初期化する副プログラムを識別します。

BLOCK DATA [*name*]

パラメータ	説明
name	BLOCK DATA 文が使用される初期値設定副プログラムの英字名。 このパラメータは省略可能。これは大域名。

説明

初期値設定副プログラムには、複数の名前付き共通ブロックの複数のデータの初期値を含むことができます。

BLOCK DATA 文は初期値設定副プログラムの最初の文でなければなりません。

初期値設定副プログラムで使用できるのは以下の文です。

- COMMON

- DATA
- DIMENSION
- END
- EQUIVALENCE
- IMPLICIT
- PARAMETER
- RECORD
- SAVE
- STRUCTURE
- **型宣言文**

名前付き共通ブロックに定義された項目だけが初期値設定副プログラムで初期設定できます。

名前付き共通ブロックの 1 つの項目を初期設定すると、その共通ブロック内に記憶単位を持つすべての項目は、初期設定していなくても COMMON 文で宣言する必要があります。

制限事項

実行可能プログラムでは、名前なし初期値設定副プログラムを 1 つしか使用できません。

同じ名前付き共通ブロックは、同じ実行可能プログラムの 2 つ以上の初期値設定副プログラムに指定することはできません。

省略可能なパラメータの名前は同じ実行可能プログラムの外部手続き、主プログラム、共通ブロック、または他の初期値設定副プログラムの名前と同じであってはなりません。名前は副プログラムの局所名と同じであってはなりません。

例

```
    BLOCK DATA INIT
    COMMON /RANGE/ X0, X1
    DATA X0, X1 / 2.0, 6.0 /
    END
```

BYTE

BYTE ♦ 文は型が 1 バイトの整数であることを指定します。配列次元を指定したり、初期値を与えたりすることもできます。

BYTE *v [/c/]* ...

パラメータ	説明
v	定数名、変数、配列、配列宣言子、関数、または仮関数のどれかの名前
c	直前の名前に対する定数の並び

説明

これは LOGICAL*1 と同義です。BYTE 型の項目は論理値の .TRUE.、.FALSE.、1 つの 8 ビットデータ項目、または -128 と 127 の間の整数を持つことができます。

例

```
    BYTE BIT3 /8/, C1/'W'/, M/127/, SWITCH/.FALSE./
```

CALL

CALL 文は指定サブルーチンに分岐し、そのサブルーチンを実行し、終了した後、呼び出し元のプログラムへ戻ります。

CALL *sub [([ar [, ar] ...])]*

パラメータ	説明
sub	呼び出されるサブルーチンの名前
ar	サブルーチンに引き渡す実引数

説明

引数はコンマで区切られます。

FORTRAN 77 規格では CALL 文における実引数の順序、数値、型が、引用されるサブルーチンの対応する仮引数と一致することを必要とします。-XlistE オプションを設定した場合のみ、コンパイラはこれをチェックします。

再帰は許可されます。副プログラムがそれ自体を直接呼び出すか、またはこのサブルーチンを続いて呼び出す別の副プログラムを呼び出すことにより、間接的に呼び出すことができます。このような再帰は規格外です。♦

実引数 ar は次のいずれかでなければなりません。

- 式
- 引数として渡されることが許可された組み込み関数。
  実引数として引き渡すことができない組み込み関数については、177 ページの表 4-2 を参照してください。
- サブルーチン名
- 文番号があとに続く選択戻り指定子の * または &。& は規格外です。♦

最も基本的な式、および最も多く使用される式には次のようなものがあります。

- 定数
- 変数名
- 配列名
- 仮引数 (CALL 文がサブルーチンの中にある場合)

- 記録名

サブルーチンに引数がない場合、そのサブルーチンを引用する CALL 文に実引数があってはなりません。空の引数をサブルーチン名の後に続けることができます。

CALL 文の実行は次のように進みます。

1. **すべての式 (引数) が評価されます。**
2. **すべての実引数が対応する仮引数に関連付けられ、サブルーチンの本体が実行されます。**
3. **通常、RETURN または END 文をサブルーチンで実行するとき、制御は CALL 文の後の文に移されます。RETURN n という形式による選択戻りが実行されると、制御は CALL 文の n という選択戻り指定子により指定された文へ移されます。**

注 – SUBROUTINE としてでなく FUNCTION として定義された副プログラムに対して CALL 文を使用すると、予測できない結果になる可能性があるので、使用しないでください。このように不適切な CALL 文があっても、警告メッセージは表示されません。

例

例 1 : 文字列

```
CHARACTER *25 TEXT
TEXT = 'Some kind of text string'
CALL OOPS ( TEXT )
SUBROUTINE OOPS ( S )
CHARACTER S*(*)
WRITE (*,*) S
END
```

例 2 : 選択戻り

```
    CALL RANK ( N, *8, *9 )
    WRITE (*,*) 'OK - 正常戻り'
    STOP
8   WRITE (*,*) '重要でない - 最初の代替戻り'
    STOP
9   WRITE (*,*) '重要 - 2 番目の代替戻り'
    STOP
    END

    SUBROUTINE RANK ( N, *, * )
    IF ( N .EQ. 0 ) RETURN
    IF ( N .EQ. 1 ) RETURN 1
    RETURN 2
    END
```

例 3 : 選択戻りの別の形式。& は規格外。♦

```
    CALL RANK ( N, &8, &9 )
```

例 4 : 配列、配列要素、および変数。

```
    REAL M(100,100), Q(2,2), Y
    CALL SBRX ( M, Q(1,2), Y )
    …
    END
    SUBROUTINE SBRX ( A, D, E )
    REAL A(100,100), D, E
    …
    RETURN
    END
```

この例では、実数配列 M は実数配列 A に対応し、実数配列要素 Q(1,2) は実数変数 D に対応します。

例 5 : 構造体の記録と欄。記録は規格外。♦

```
      STRUCTURE /PRODUCT/
      INTEGER*4 ID
      CHARACTER*16 NAME
      CHARACTER*8 MODEL
      REAL*4 COST
      REAL*4 PRICE
      END STRUCTURE
      RECORD /PRODUCT/ CURRENT, PRIOR
      CALL SBRX ( CURRENT, PRIOR.ID )
      …
      END
      SUBROUTINE SBRX ( NEW, K )
      STRUCTURE /PRODUCT/
      INTEGER*4 ID
      CHARACTER*16 NAME
      CHARACTER*8 MODEL
      REAL*4 COST
      REAL*4 PRICE
      END STRUCTURE
      RECORD /PRODUCT/ NEW
      …
      RETURN
      END
```

上記の例では、記録 NEW は記録 CURRENT に、整数の変数 K は記録欄 PRIOR.OLD に対応します。

CHARACTER

CHARACTER 文は定数名、変数、配列、関数、および仮関数の型が文字であることを指定します。

この文で任意に項目を初期化したり、配列寸法を指定します。

CHARACTER *[* len [,]] v [* len /c/]] …*

パラメータ	説明
v	定数名、変数、配列、配列宣言子、関数、仮関数の名前
len	定数名、変数、配列要素、および関数の文字における長さ
c	直前の名前に対する定数の並び

説明

それぞれの文字は文字境界に整列した 8 ビットの記憶領域を占有します。文字変数を含む共通ブロックと文字配列は文字変数の 1 つの集合に組み込まれています。ある要素の最初の文字は先行要素の最後の文字に隙間なしに続きます。

長さ len は 0 より大きくなければなりません。len を省略すると、1 に等しいとみなされます。

局所文字変数、共通文字変数、定数名、仮引数、または関数名の場合は、len は整定数または括弧で囲まれた整数の定数式です。

仮引数または関数名の場合、len は別の形式を持つこともあります。この形式は括弧で囲まれたアスタリスク、すなわち CHARACTER*(*) です。この形式は、関数名の長さはプログラム単位の参照によって定義され、仮引数が実引数の長さを持つことを意味します。

定数名の場合、len は括弧で囲まれたアスタリスクでもよく、これは名前がその定数の長さを持つものとして定義されていることを示します。これは後述の例 5 で示します。

定数の並び *c* は変数、配列、または配列宣言子に対してだけ使用できます。直前の変数に対して 1 つだけと、直前の配列の各要素に対して 1 つずつ定数が許可されます。

例

例 1 : 文字列と文字列の配列

```
CHARACTER*17 A, B(3,4), V(9)
CHARACTER*(6+3) C
```

上記のコードは以下のコードと全く同じです。

```
CHARACTER A*17, B(3,4)*17, V(9)*17
CHARACTER C*(6+3)
```

上記の 2 つの例はどちらも規格外です。♦

```
CHARACTER A*17, B*17(3,4), V*17(9) 非標準
```

空文字列 (長さがゼロ) の変数は存在しません。空文字を代入された 1 バイトの文字列の長さは 1 です。

例 2 : 空文字列の変数は存在しません。

```
CHARACTER S*1
S = ''
```

代入文の実行中、変数 S はあらかじめ消去されて空白になり、0 個の文字が S に移されるので、S は空白を 1 つ含みます。宣言により、LEN(S) は 1 という長さを戻します。1 より小さいサイズを宣言することはできないので、これは最も短い文字列変数になります。

例 3 : 固定長の仮引数文字列

```
SUBROUTINE SWAN ( A )
CHARACTER A*32
```

例 4 : 対応する実引数と同じ長さを持つ仮引数文字列

```
SUBROUTINE SWAN ( A )
CHARACTER A*(*)
…
```

例 5 : 括弧に囲まれたアスタリスクを持つ定数名

```
CHARACTER *(*) INODE
PARAMETER ( INODE = '警告: INODE が破壊されています !' )
…
```

組み込み関数 LEN (INODE) は、文字列の実際に宣言された長さを返します。この関数は主に仮引数 CHAR*(*) と一緒に使用されます。

例 6 : 組み込み関数 LEN

```
CHARACTER A*17
A = "xyz"
PRINT *, LEN( A )
END
```

上記のプログラムは 3 ではなく、17 を表示します。

CLOSE

CLOSE 文は装置からファイルを切り離します。

```
CLOSE(  [UNIT=] u [, STATUS=sta] [, IOSTAT=ios]
[, ERR=s] )
```

パラメータ	説明
u	外部装置のための装置識別子。UNIT= を使用しない場合、*u* を最初に指定すること。
sta	ファイルの処理を決める。*sta* は文字式で、後続の空白が取り除かれると値は KEEP または DELETE になる。状態指定子のデフォルト値は KEEP。一時 (スクラッチ) ファイルの場合、*sta* は常に DELETE になる。一時ファイル以外の他のファイルの場合、デフォルトの *sta* は KEEP。
ios	入出力状態指定子 *ios* は整変数または整数の配列要素であること。
s	誤り指定子 *s* は CLOSE 文を含むプログラムの中の実行文の文番号であること。CLOSE 文の実行時にエラーが生じると、プログラムの制御はこの文に移される。

説明

磁気テープを使用する場合は、TOPEN() ルーチンを使用する方が確実です。

オプションの指定はどのような順序でもかまいません。

DISP= と DISPOSE= オプションは STATUS= に対する代替として許可されていますが、
-ansi フラグが設定されていると警告が出されます。

CLOSE 文の実行は次のように進みます。

1. **指定された装置が切り離されます。**
2. ***sta* が DELETE であれば、指定された装置に結合されたファイルが削除されます。**
3. **引数 IOSTAT を指定すると、エラーが発生しなければ、*ios* はゼロに、そうでない場合は正の値に設定されます。**

注釈

開いているすべてのファイルは、通常のプログラム終了時にデフォルトの *sta* で閉じられます。指定された *sta* に関係なく、一時ファイルは閉じられると常に削除されます。

存在しない装置、またはファイルが結合されていない装置を指定する CLOSE 文を実行しても無効です。

装置にゼロを指定した CLOSE 文 (規格ではエラーになる) の実行は許可されませんが、装置を他のファイルとして再び開くことができます。

CLOSE 文の実行により切り離された装置 / ファイルは、同じファイル / 装置または異なるファイル / 装置に再度結合できます。

注 – テープ入出力の場合、TOPEN() ルーチンを使用します。

例

例 1 : 閉じて保持する

```
CLOSE ( 2, STATUS='KEEP')
```

例 2 : 閉じて削除する

```
CLOSE ( 2, STATUS='DELETE', IOSTAT=I )
```

例 3 : STATUS が SCRATCH であっても、一時ファイルを閉じ残す

```
OPEN ( 2, STATUS='SCRATCH')
…
CLOSE ( 2, STATUS='KEEP', IOSTAT=I )
```

COMMON

COMMON 文は主記憶領域のブロックを定義して、異なるプログラム単位が引数を使用しないで同一データを共有できるようにします。

COMMON [/[*cb*]/] *nlist* [[,]/[*cb*] / *nlist*] ...

パラメータ	説明
cb	共通ブロック名
nlist	変数名、配列名、および配列宣言子の並び

説明

共通ブロック名が省略されると無名共通ブロックになります。

無名共通ブロックを含め、どの共通ブロック名も同じプログラム単位内の COMMON 文に複数指定することができます。同一共通ブロック名に続く並び *nlist* は、その共通ブロック名への並びの連続として扱われます。

共通ブロックの大きさは共通ブロックのすべての構成要素の大きさと整列に要するスペースの合計です。

1 つのプログラム内では、異なるプログラム単位内の同一の名前を持つ共通ブロックはすべて同一の大きさでなければなりません。ただし、無名共通ブロックは同じ大きさでなくても構いません。

制限事項

仮引数名と関数名は COMMON 文に指定することはできません。

EQUIVALENCE 文は、同一プログラム単位内の 2 つの異なる共通ブロックの一連の要素を関連付けないでください。例 2 を参照してください。

EQUIVALENCE 文は共通ブロックを左側に拡張しないでください。例 4 を参照してください。

例

例 1 : 無名共通ブロックと名前付き共通ブロック

```
DIMENSION V(100)
COMMON V, M
COMMON / LIMITS / I, J
…
```

上記において V と M は無名共通ブロックにあります。一方、I と J は名前付き共通ブロック LIMITS 内で定義されています。

例 2 : 同一プログラム単位内の 2 つの異なる共通ブロックの要素を関連付けることはできません。

```
COMMON /X/ A
COMMON /Y/ B
EQUIVALENCE ( A, B)   できない
```

例 3 : EQUIVALENCE 文は共通ブロックを右側に拡張することができます。

```
DIMENSION A(5)
COMMON /X/ B
EQUIVALENCE ( B, A)
```

例 4 : EQUIVALENCE 文は共通ブロックを左側に拡張しないでください。

```
COMMON /X/ A
REAL B(2)
EQUIVALENCE ( A, B(2))   できない
```

COMPLEX

COMPLEX 文は、定数名、変数、配列、関数、または仮関数の型を複素数として指定し、また配列の次元と大きさを指定し、初期値を与えることもできます。

```
COMPLEX [*len[,]] v [* len [/c/]] [, v [* len [/c/]] ...
```

パラメータ	説明
v	定数名、変数、配列、配列宣言子、関数、仮関数の名前
len	定数名、変数、配列要素、または関数のバイト単位の長さで、8、16、または 32 (32 は SPARC のみ)
c	直前にくる名前の定数の並び

説明

宣言は、COMPLEX、COMPLEX*8、COMPLEX*16、COMPLEX*32 のいずれかです。サイズの指定は規格外となります。♦

COMPLEX

COMPLEX W のように宣言すると、変数 W は通常メモリー内で隣接した 2 つの REAL*4 要素で、複素数と解釈されます。

サイズを指定しない場合、デフォルトサイズが使用されます。

COMPLEX W などの宣言のデフォルトサイズを変更するには、オプション -dbl、-r8、または -xtypemap のいずれかを指定してコンパイルします。詳細については、第 2 章「データ型とデータ項目」を参照してください。

COMPLEX*8 ♦

COMPLEX*8 W のように宣言すると、変数 W はメモリー内で隣接した 2 つの REAL*4 要素で、複素数と解釈されます。

COMPLEX*16 ♦

COMPLEX*16 W のように宣言すると、W はメモリー内で隣接した 2 つの REAL*8 要素で、倍長の複素数と解釈されます。

COMPLEX*32 ♦

(SPARC のみ) COMPLEX*32 W のように宣言すると、変数 W はメモリー内で隣接した 2 つの REAL*16 要素で、4 倍長の複素数と解釈されます。

注釈

各複素組み込み関数には倍精度複素数用のものがあります。一般に、その関数名は C の代わりに Z または CD で始まります。2 つの関数 DIMAG と DREAL は例外で、実数値を返します。

4 倍精度用の特定の複素関数があります (SPARC のみ)。一般に、特定の REAL と、これに対応する C 接頭辞の付いた COMPLEX、CD 接頭辞の付いた COMPLEX DOUBLE があります。さらに CQ 接頭辞の付いた 4 倍精度 COMPLEX 関数もあります。たとえば、SIN()、CSIN()、CDSIN()、CQSIN() など。

例

例 1 : 複素変数。以下の文は等価です。

```
COMPLEX U, V
COMPLEX*8 U, V
COMPLEX U*8, V*8
```

例 2 : 複素変数の初期化

```
COMPLEX U / (1, 9.0) /, V / (4.0, 5 ) /
```

複素定数が整数または実数のどちらかの数字の対であることに注意してください。

例 3 : 倍精度複素定数、初期化

```
COMPLEX U*16 / (1.0D0, 9 ) /, V*16 / (4.0, 5.0D0) /
COMPLEX*16 X / (1.0D0, 9.0) /, Y / (4.0D0, 5 ) /
```

倍精度複素定数が数字の対であり、少なくとも一方の数字が倍精度でなければならないので注意してください。

例 4 : 4 倍精度複素定数、初期化 (SPARC のみ)

```
COMPLEX U*32 / (1.0Q0, 9 ) /, V*32 / (4.0, 5.0Q0) /
COMPLEX*32 X / (1.0Q0, 9.0) /, Y / (4.0Q0, 5 ) /
```

4 倍精度複素定数が数字の対であり、少なくとも一方の数字が 4 倍精度でなければならないので注意してください。

例 5 : 複素数配列。次のすべての文は規格外です (SPARC のみ)。♦

```
COMPLEX R*16(5), S(5)*16
COMPLEX U*32(5), V(5)*32
COMPLEX X*8(5), Y(5)*8
```

CONTINUE

CONTINUE 文は何も行わない文です。(SPARC のみ)

[*label*] CONTINUE

パラメータ	説明
label	文番号

説明

CONTINUE 文は文番号を付ける場所として使用されることが多く、通常 DO ループの終わりにあります。

CONTINUE 文は文番号を付けるのに便利なので、特に DO ループの終了文として使用されます。CONTINUE 文は、何も実行しません。

CONTINUE 文が DO ループの終了文として使用される場合、実行される次の文は DO ループの出口の条件によって決まります。

例

```
    DIMENSION U(100)
    S = 0.0
    DO 1 J = 1, 100
    S = S + U(J)
    IF ( S .GE. 1000000 ) GO TO 2
1   CONTINUE
    STOP
2   CONTINUE
    . . .
```

DATA

DATA 文は、変数、部分列、配列、配列要素を初期化します。

DATA *nlist* / *clist* / [[,] *nlist* / *clist* /] ...

パラメータ	説明
nlist	コンマで区切った変数、配列、配列要素、部分列、および DO 型並び
clist	形式 *c [, c]...* をもつ並び
c	形式 *c* または *r*c* の *1* つ。*c* は定数または定数の定数名
r	ゼロでない符号なし整定数、またはこのような定数の定数名

説明

実行可能プログラムの実行開始時に初期値を定義されていた項目は、すべて指定された値で定義されます。

*r*c* は定数 *c* を *r* 個連続させたものと同等です。

DATA 文は非実行文であり、すべての宣言文の後に現れなければなりませんが、文関数と実行文を混在させることができます (ただし、これは規格外となります)。♦

注 – コンパイルに `-stackvar` オプションを使用した場合、実行文から局所変数が参照された後、`DATA` 文でその局所変数を初期化するとエラーとして扱われます。『Fortran ユーザーズガイド』を参照してください。

clist の *(*反復因子を考慮に入れた*)* 定数の数は *nlist* の項目数と同じでなければなりません。*nlist* における配列の出現はその配列のすべての要素の並びを指定するのと同等です。配列要素は定数の添字によってだけ索引できます。

`DATA` 文には自動変数や自動配列を記述できません。

通常の型変換が *clist* の文字型でない各メンバーに対して起こります。

DATA 文の文字定数

nlist の文字項目の長さが対応する *clist* の定数より長い場合、右側に空白文字が詰められます。

nlist の文字項目の長さが対応する *clist* の定数より短い場合、余分な右側の文字が無視されます。

clist の定数が整数型で *nlist* の項目が文字型である場合、下記の規則に従ってください。

- 文字項目は 1 文字の長さでなければなりません。
- 定数は整数型で、0 から 255 までの範囲の値を持たなければなりません。`^A`、`^B`、`^C` は、Control キーを押しながら、A、B、または C キーを押して入力せずに、組み込み関数 `CHAR` を使用してください。

clist の定数が文字定数またはホレリス定数で、*nlist* の項目が `INTEGER` 型である場合、割り当てられる文字の数は `INTEGER*2` と `INTEGER*4` に対しそれぞれ 2 と 4 です。文字定数またはホレリス定数が項目の容量より少ない文字数を持つ場合、定数は右側にスペースで拡張されます。文字またはホレリス定数が格納できるより多くの文字を持つ場合、定数の右側が切り捨てられます。

DO 型並び

nlist は配列要素を初期化するために `DO` 型並びを指定することができます。

`DO` 型並びの形式は下記のとおりです。

(dlist, iv = m1, m2 [,m3])

パラメータ	説明
dlist	配列要素名と DO 型並びの並び
iv	DO 変数と呼ばれる整変数
m1	*iv* の初期値を指定する整定数式
m2	*iv* の終値を指定する整定数式
m3	*iv* の増分値を指定する整定数式。*m3* が省略されるとデフォルトの値 *1* がとられる。

DO 型ループの範囲は *dlist* です。DO 型ループの繰り返し回数は *m1*、*m2*、および *m3* から計算され、必ず正です。

DO 型並びは、入出力文 PRINT、READ、WRITE の変数並びにも使用されることがあります。

変数

変数は型宣言文でも初期化できます。これは FORTRAN 77 規格の拡張です。これらの例を個々の型宣言文と一般的な型宣言文のところで説明します。♦

例

例 1： 文字、整数、および実数スカラー。実数配列。

```
    CHARACTER TTL*16
    REAL VEC(5), PAIR(2)
    DATA TTL / '任意のタイトル' /,
&    M / 9 /, N / 0 /,
&    PAIR(1) / 9.0 /,
&    VEC / 3*9.0, 0.1, 0.9 /
    …
```

例 2 : 配列－DO 型

```
      REAL R(3,2), S(4,4)
      DATA ( S(I,I), I=1,4) / 4*1.0 /,
     &    (( R(I,J), J=1,3), I=1,2) / 6*1.0 /
      …
```

例 3 : 整数と文字の混在

```
      CHARACTER CR*1
      INTEGER I*2, N*4
      DATA I / '00' /, N / 4Hs12t /, CR / 13 /
      …
```

DECODE/ENCODE

ENCODE は文字変数、配列、または配列要素に書き込みます。♦ DECODE は文字変数、配列、または配列要素から読み取ります。データは書式識別子に従って編集されます。

同様の機能は内部ファイルに書式付き順次 WRITE 文、READ 文を使用して達成することができます。ENCODE および DECODE は FORTRAN 77 規格にはありませんが、Fortran 77 の旧バージョンとの互換性のために提供されています。

ENCODE(*size, f, buf* [, IOSTAT= *ios*] [, ERR= *s*]) [*iolist*]

DECODE(*size, f, buf* [, IOSTAT= *ios*] [, ERR= *s*]) [*iolist*]

パラメータ	説明
size	変換される文字数、整数式
f	書式識別子で、FORMAT 文の文番号。書式列を指定する文字式またはアスタリスクのどれか。
buf	変数、配列、または配列要素
ios	入出力状態指定子 *ios* は整変数または整数配列要素であること。
s	誤り指定子 (文番号) *s* は ENCODE と DECODE 文が出現するプログラム単位の中の実行文の文番号であること。

パラメータ	説明
iolist	入出力項目の並び

説明

入出力並びの構成要素としては、変数、部分列、配列、配列要素、記録欄が可能です。単純な添字なし配列名は、配列のすべての要素を記憶領域順で指定します。左端の添字が最も速く増加します。

実行順序は以下のとおりです。

1. ENCODE 文は書式識別子に従って並びの項目を文字形式に変換し、文字を *buf* に格納します。内部ファイルの WRITE 文も同じ操作を行います。
2. DECODE 文は書式識別子に従って *buf* の文字データを内部 (2 進) 形式に変換し、並びの項目に格納します。READ 文も同じ操作を行います。
3. *buf* が配列の場合、その要素は左端の添字が最も速く増加する順序で処理されます。
4. ENCODE または DECODE 文が処理することのできる文字数は *buf* のデータ型によります。たとえば、INTEGER*2 配列は 1 要素当たり 2 文字を持つことができるので、文字の最大数は配列の要素数の 2 倍になります。文字変数または文字配列要素はその長さに等しい数の文字を持つことができます。文字配列は各要素の長さと要素数の積に等しい数の文字を持つことができます。
5. 書式識別子と入出力並びの間の関係は書式付き入出力文と同様です。

例

DECODE/ENCODE プログラム

```
      CHARACTER S*6 / '987654' /, T*6
      INTEGER V(3)*4
      DECODE( 6, '(3I2)', S ) V
      WRITE( *, '(3I3)') V
      ENCODE( 6, '(3I2)', T ) V(3), V(2), V(1)
      PRINT *, T
      END
```

上記のプログラムは下記を出力します。

```
 98 76 54
547698
```

DECODE は文字 S を 3 つの整数として読み取り、それらを V(1)、V(2)、および V(3) に格納します。

ENCODE 文は値 V(1)、V(2)、および V(3) を文字として T に書き込みます。T は '547698' になります。

DIMENSION

DIMENSION 文は配列の次元数とそれぞれの次元の要素数を指定します。

DIMENSION 文で任意に各項目の初期値を与えることができます。

DIMENSION *a* (*d*) [,*a* (*d*)] ...

パラメータ	説明
a	配列名
d	配列の形状を指定します。コンマにより区切られた 1 個から 7 個までの 宣言子の並び。

説明

ここでは、寸法宣言子と配列について説明します。

寸法宣言子

それぞれの寸法の上下限は寸法宣言子により決められます。寸法宣言子の形式は次のとおりです。

```
[ dd1 :] dd2
```

dd1 と *dd2* は上下限値を指定する寸法上下限式です。これらには整数型または実数型の算術式が使用できます。これらは定数、定数名、仮引数、または COMMON 文で定義された変数を使用して作成することができます。配列引用およびユーザー定義関数の引用は次元上下限式では使用できません。*dd2* はアスタリスクも可能です。*dd1* が指定されないと値 1 がとられます。*dd1* の値は *dd2* 以下でなければなりません。

非定数の寸法上下限式は、整合配列を定義するために副プログラムで使用できます。しかし主プログラムでは使用できません。

非整数の上下限式は使用する前に整数に変換されます。小数部は切り捨てられます。

整合配列

寸法宣言子が仮引数または COMMON 文で定義された変数を含む算術式である場合、この配列は整合配列と呼ばれます。この場合、寸法は副プログラムへの入口の引数の初期値に等しくなります。

大きさ引き継ぎ配列

配列は、寸法宣言子にアスタリスクがある場合、大きさ引き継ぎ配列と呼ばれます。この場合、寸法の上限は規定されません。アスタリスクは仮配列にだけ使用可能で、配列宣言子の最後の寸法の上限として現れます。

例

例 1： 主プログラムの配列

```
DIMENSION M(4,4), V(1000)
…
END
```

上記の例では、M は形状 4 × 4 の配列として指定され、V は形状 1000 の配列として指定されます。

例 2 : サブルーチンの整合配列

```
SUBROUTINE INV( M, N )
DIMENSION M( N, N )
…
END
```

上記の例では仮引数は配列 M と変数 N です。M は形状 N × N の平方配列用として指定されます。

例 3 : 上下限

```
DIMENSION HELIO (-3:3, 4, 3:9)
…
END
```

上記では HELIO は三次元配列です。最初の要素は HELIO(-3,1,3) で、最後の要素は HELIO(3,4,9) です。

例 4 : 上下限付き仮配列

```
SUBROUTINE ENHANCE( A, NLO, NHI )
DIMENSION A(NLO : NHI)
…
END
```

例 5 : 非整数の上下限

```
PARAMETER ( LO = 1, HI = 9.3 )
DIMENSION A(HI, HI*3 + LO )
…
END
```

上記の例では A は形状 9 × 28 の配列です。

例 6 : 非整数の上下限付き整合配列

```
SUBROUTINE ENHANCE( A, X, Y )
DIMENSION A(X : Y)
…
END
```

例 7 : 大きさ引き継ぎ配列

```
SUBROUTINE RUN(A,B,N)
DIMENSION A(*), B(N,*)
...
```

DO

DO 文は 1 組の文を繰り返し実行します。

DO *s [,] loop-control*

または

DO *loop-control* ♦

s　実行文の文番号

loop-control の形式は以下のとおりです。

variable = e1, e2 [, *e3*]

パラメータ	説明
variable	整数型、実数型、倍精度型、またはバイト型の変数
e1, e2, e3	整数型、実数型、または倍精度型の式で、それぞれ初期値、限界値、および増分値を指定する。

説明

DO 文は、以下の構文を含みます。

文番号付き DO ループ

文番号付き DO ループは以下の文から構成されます。

- DO 文
- ブロックと呼ばれる実行文の集合
- 端末文、通常は CONTINUE 文

端末文

s により識別される文は端末文と呼ばれます。端末文は DO 文と同一のプログラム単位内の一連の文の中で、 DO 文の後に続かなければなりません。

端末文は以下の文であってはなりません。

- **単純** GO TO
- **割り当て型** GO TO
- **算術** IF
- **ブロック** IF / ELSE IF
- ELSE
- END IF
- RETURN
- STOP
- END DO

端末文が論理 IF 文である場合、下記以外のすべての実行文を含むことができます。

- DO / DO WHILE
- **ブロック** IF / ELSE IF
- ELSE IF
- ELSE
- END IF
- END
- **論理** IF

DO ループの範囲

DO ループの範囲は、端末文までの DO 文に続いて現れる (端末文を含む) すべての実行文から構成されます。

DO 文が他の DO ループの範囲内に現れる場合、その範囲は外側の DO ループの範囲内に完全に含まれなければなりません。2 つ以上の文番号付き DO ループが同一の端末文を持つことができます。

DO 文が IF、ELSE IF、または ELSE ブロック内に現れる場合、関連 DO ループの範囲は完全にそのブロック内に含まれなければなりません。

ブロック IF 文が DO ループの範囲内に現れる場合、対応する END IF 文もその DO ループの範囲内に現れなければなりません。

ブロック DO ループ ♦

ブロック DO ループは以下の文から構成されます。

- DO 文
- ブロックと呼ばれる実行文の集合
- 端末文、END DO 文

このループは規格外です。

実行手順は以下のとおりです。

1. **式 *e1*、*e2*、および *e3* が評価されます。*e3* が存在しない場合、その値は 1 とされます。**
2. **DO 変数は *e1* の値で初期化されます。**
3. **繰り返し回数は式の値として確立されます。**

   MAX(INT((*e2* - *e1* + *e3*)/),*e3*, 0)

   下記のどれかが真の場合、繰り返し回数はゼロなので注意してください。

   - *e1* > *e2* で *e3* > ゼロ
   - *e1* < *e2* で *e3* < ゼロ

   コンパイル時オプション -onetrip が指定されると、繰り返し回数は 1 より小さくはなりません。

4. **繰り返し回数が検査され、それがゼロより大きい場合、DO ループの範囲が実行されます。**

端末文処理

DO ループの端末文が実行された後、以下の処理が実行されます。

1. **DO 変数の値があると、その値は DO 文の実行時に計算された *e3* の値だけ増分されます。**
2. **繰り返し回数は 1 ずつ減ります。**
3. **繰り返し回数が検査され、それがゼロより大きい場合、DO ループの範囲内の文が再び実行されます。**

制限事項

DO 変数は DO ループの範囲内ではどの様な方法でも変更しないでください。

DO ループの範囲外から DO ループの範囲内に飛び込まないでください。

注釈

場合によっては、終値に対して DO 変数を検査する前に実行される増分の演算の結果、DO 変数がオーバーフローする場合があります。これが起こるときは、プログラムに誤りがあっても、コンパイラも実行システムもそれを検出しません。この状態では、DO 変数がオーバーフローしてもループは予定通り終了します。

DO ループの範囲外から DO ループの範囲内に飛び込んだ場合、警告が発生されますが、実行は継続されます。

範囲外からの飛び先が端末文で、それが CONTINUE でもあり、入れ子になった DO ループの端末文でもある場合、常に内側の DO ループのほとんどが実行されます。

例

例 1 : 入れ子になった DO ループ

```
    N = 0
    DO 210 I = 1, 10
    J = I
    DO 200 K = 5, 1
    L = K
    N = N + 1
200 CONTINUE
210 CONTINUE
    WRITE(*,*)'I =',I, ', J =',J, ', K =',K,
&                                ', N =',N, ', L =',L
    END
demo % f77 -silent DoNest1.f
"DoNest1.f", line 4: 警告: DO の範囲が実行されていません。
demo % a.out
I =  11, J =  10, K =  5, N =  0, L =  0
demo %
```

内部ループは実行されず、WRITE のときは L が未定義です。ここで L は 0 として示されていますが、これは処理系によって異なります。この例とは同じではない場合があります。

例 2 : プログラム DoNest2.f (常に定義されている DO 変数)

```
    INTEGER COUNT, OUTER
    COUNT = 0
    DO OUTER = 1, 5
            NOUT = OUTER
        DO INNER = 1, 3
            NIN = INNER
            COUNT = COUNT+1
        END DO
    END DO
    WRITE(*,*) OUTER, NOUT, INNER, NIN, COUNT
    END
```

上記のプログラムは下記を出力します。

```
    6 5 4 3 15
```

DO WHILE

DO WHILE ♦ 文は、指定された条件が真である限り、1 組の文を繰り返し実行します。

```
DO   [ s [,]] WHILE (e)
```

パラメータ	説明
s	実行文の文番号
e	論理式

説明

実行手順は以下のとおりです。

1. **指定された式が評価されます。**
2. **式の値が真の場合、DO WHILE ループの範囲内の文が実行されます。**
3. **式の値が偽の場合、制御は DO WHILE ループに続く文に移ります。**

端末文

s が指定されると、それにより識別される文は端末文と呼ばれ、DO WHILE 文の後に続かなければなりません。

端末文は以下の文であってはなりません。

- **単純** GO TO
- **割り当て型** GO TO
- **算術** IF
- **ブロック** IF / ELSE IF
- ELSE
- END IF
- RETURN
- STOP
- END

- DO / DO WHILE

端末文が論理 IF 文である場合、下記以外のすべての実行文を含むことができます。

- DO / DO WHILE
- **ブロック** IF / ELSE IF
- ELSE IF
- ELSE
- END IF
- END
- **論理** IF

s が指定されない場合、DO WHILE ループは END DO 文で終わらなければなりません。

DO WHILE ループの範囲

DO WHILE ループの範囲は、DO WHILE 文に続いて現れる端末文までの (端末文を含む) すべての実行文から構成されます。

DO WHILE 文が他の DO WHILE ループの範囲内に現れる場合、その範囲は外側の DO WHILE ループの範囲内に完全に含まれなければなりません。 2 つ以上の DO WHILE ループが同一の端末文を持つことができます。

DO WHILE 文が IF、ELSE IF、または ELSE ブロック内に現れる場合、関連する DO WHILE ループの範囲は完全にそのブロック内になければなりません。

ブロック IF 文が DO WHILE ループの範囲内に現れる場合、対応する END IF 文もその DO WHILE ループの範囲内に現れなければなりません。

端末文処理

DO WHILE ループの端末文が実行された後、制御は対応する DO WHILE 文に返されます。

制限事項

DO WHILE ループの外側から DO WHILE ループの内側に飛び込むと、予測できない結果を生ずることがあります。

注釈

e で使用される変数は、DO WHILE ループの範囲内ではどのようにでも修飾することができます。

例

例 1 : 文番号のない DO WHILE

```
    INTEGER A(4,4), C, R
    …
    C = 4
    R = 1
    DO WHILE ( C .GT. R )
    A(C,R) = 1
    C = C - 1
    END DO
```

例 2 : 文番号付き DO WHILE

```
    INTEGER A(4,4), C, R
    …
    DO 10 WHILE ( C .NE. R )
    A(C,R) = A(C,R) + 1
    C = C+1
10  CONTINUE
```

DOUBLE COMPLEX

DOUBLE COMPLEX ♦ 文は、倍精度複素数型を指定します。この文で配列次元と大きさを指定し、初期値を与えることもできます。

DOUBLE COMPLEX *v* [/*c*/] [, *v* [/*c*/] …

パラメータ	説明
v	定数名、変数、配列、配列宣言子、関数、または仮関数の名前
c	この前にくる名前についての定数の並び

説明

宣言は DOUBLE COMPLEX または COMPLEX*16 です。

DOUBLE COMPLEX ♦

DOUBLE COMPLEX Z のように宣言すると、変数 Z はメモリー内で連続する 2 つの REAL*8 要素で、1 つの倍長の複素数と解釈されます。

サイズを指定しない場合、デフォルトサイズが使用されます。

DOUBLE COMPLEX Z などの宣言のデフォルトサイズを変更するには、オプション -dbl、-r8、または -xtypemap のいずれかを指定してコンパイルします。詳細については、第 2 章「データ型とデータ項目」を参照してください。

COMPLEX*16 ♦

COMPLEX*16 Z のような宣言では、変数 Z はメモリー内で連続する 2 つの REAL*8 要素で、1 つの倍長の複素数と解釈されます。

注釈

各複素数組み込み関数には倍精度複素数用のものがあります。一般に、その関数名は C の代わりに Z または CD で始まります。たとえば、SIN()、CSIN()、CDSIN() などです。2 つの関数 DIMAG と DREAL は例外で、実数値を返します。

例 : 倍精度複素数スカラーと配列

```
    DOUBLE COMPLEX U, V
    DOUBLE COMPLEX W(3,6)
    COMPLEX*16 X, Y(5,5)
    COMPLEX  U*16(5), V(5)*16
```

DOUBLE PRECISION

DOUBLE PRECISION 文は、型を倍精度に指定し、配列形状を指定し、初期値を与えることもできます。

DOUBLE PRECISION *v* [/*c*/] [, *v* [/*c*/] …

パラメータ	説明
v	定数名、変数、配列、配列宣言子、関数、または仮関数の名前
c	この前にくる名前についての定数の並び

説明

宣言は DOUBLE PRECISION または REAL*8 です。

DOUBLE PRECISION

DOUBLE PRECISION X のように宣言すると、変数 X はメモリー内の REAL*8 要素で、1 つの倍長の実数と解釈されます。

サイズを指定しない場合、デフォルトサイズが使用されます。DOUBLE PRECISION X などの宣言のデフォルトサイズを変更するには、オプション -dbl、-r8、または -xtypemap のいずれかを指定してコンパイルします。詳細については、第 2 章「データ型とデータ項目」を参照してください。

REAL*8 ♦

REAL*8 X のように宣言すると、変数 X は常にメモリー内の REAL*8 型の要素で、倍長の実数と解釈されます。

例

```
    DOUBLE PRECISION R, S(3,6)
    REAL*8 T(-1:0,5)
```

ELSE

ELSE 文は ELSE ブロックの始まりを示します。

```
IF ( e ) THEN
 …
ELSE
 …
END IF
```

*e*は論理式。

説明

ELSE 文の実行はプログラムに何の影響も与えません。

ELSE ブロックは、ELSE 文に続く、END IF 文までの (ただし、END IF 文は含まない) すべての実行文から構成されます。このときの END IF 文は ELSE 文と同じ IF の深さです。詳細については、159 ページの「IF ブロック」を参照してください。

ELSE ブロックは空でも構いません。

制限事項

ELSE ブロックの外側から ELSE ブロックに移ることはできません。

ELSE 文の文番号はどの文からも参照できません。

ELSE と同じ IF の深さを有する END IF 文は、同じ IF の深さを有する ELSE IF、または ELSE 文の前に現れなければなりません。

例

例 1： ELSE

```
CHARACTER S
…
IF ( S .GE. '0' .AND. S .LE. '9' ) THEN
CALL PUSH
ELSE
CALL TOLOWER
END IF
…
```

例 2： END IF が期待されるところに無効の ELSE IF がある場合

```
IF ( K .GT. 5 ) THEN
N = 1
ELSE
N = 0
ELSE IF ( K .EQ. 5 ) THEN  正しくない
…
```

ELSE IF

ELSE IF は複数の条件決定構造を提供します。

ELSE IF (*e2*) THEN

IF (*e1*) THEN

END IF …

e1 と *e2* は**論理式。**

説明

一連の独立した検査を行うことが可能で、各検査は複数の文を指定できます。

ELSE IF ブロックは、ELSE IF 文に続く、ELSE IF 文と同じ IF の深さを有する次の ELSE IF、ELSE、または END IF 文までの (ただし、これらの文は含まない) すべての実行文から構成されます。

ELSE IF ブロックは空でも構いません。

ELSE IF (*e*)の実行手順は以下のとおりです。

1. ***e* が評価されます。**
2. ***e* が真の場合、ELSE IF ブロックの最初の文から実行されます。*e* が真で ELSE IF ブロックが空の場合、その ELSE IF 文と同じ IF の深さを有する次の END IF 文に制御が移ります。**
3. ***e* が偽の場合、その ELSE IF 文と同じ IF の深さを有する次の ELSE IF、ELSE、または END IF 文に制御が移ります。**

制限事項

ELSE IF ブロックの外側から ELSE IF ブロックに移ることはできません。

ELSE IF 文の文番号があったとしても、いかなる文もそれを引用することはできません。

ELSE IF と同じ IF の深さを有する END IF 文は、同じ IF の深さを有する ELSE IF または ELSE 文の前に現れなければなりません。

例

例 : ELSE IF

```
READ (*,*) N
IF ( N .LT. 0 ) THEN
WRITE(*,*) 'N<0'
ELSE IF ( N .EQ. 0) THEN
WRITE(*,*) 'N=0'
ELSE
WRITE(*,*) 'N>0'
END IF
```

ENCODE/DECODE

ENCODE ♦ 文は記憶領域の並びからデータを書き込みます。

ENCODE(*size*, *f*, *buf* [, IOSTAT=*ios*] [, ERR=*s*]) [*iolist*]

パラメータ	説明
size	変換される文字数
f	書式識別子
buf	変数、配列、または配列要素
ios	入出力状態識別子
s	誤り識別子 *(*文番号*)*
iolist	入出力項目の並びで、それぞれ文字変数、配列、または配列要素

説明

ENCODE は Fortran 77 の旧バージョンとの互換性のために提供されています。同様の機能は、内部ファイルに書式付き順次 WRITE 文を使用して達成することができます。ENCODE は FORTRAN 77 規格にはありません。

データは書式識別子に従って編集されます。

例

```
    CHARACTER S*6, T*6
    INTEGER V(3)*4
    DATA S / '987654' /
    DECODE( 6, 1, S ) V
1   FORMAT( 3 I2 )
    ENCODE( 6, 1, T ) V(3), V(2), V(1)
```

DECODE は文字 S を 3 つの整数として読みとり、それらを V(1)、V(2)、および V(3)に格納します。ENCODE 文は値 V(3)、V(2)、および V(1) を文字として T に書き込みます。T は '547698' になります。

詳細および具体的な例については、114 ページの「DECODE/ENCODE」を参照してください。

END

END 文はプログラム単位の終わりを示し、以下の構文を使用します。

```
END
```

説明

END 文は以下のとおりです。

- プログラム単位の最後の文でなければなりません。
- 1 行内に 1 文だけでなければなりません。
- 文番号を持つことができます。

主プログラムでは END 文はプログラムの実行を終了します。

関数またはサブルーチンでは END 文は RETURN と同じ効果を持ちます。♦

FORTRAN 77 規格では END 文は継続行を持つことはできませんが、f77 ではこれを許可します。♦

END IF 文のような他のいかなる文も、END 文とみなされるような開始行を持つことはできません。

例

例： END

```
      PROGRAM MAIN
      WRITE( *, * ) '極めて小さい'
      END
```

END DO

END DO ♦ 文は DO ループを終了し、以下の構文を使用します。

```
END DO
```

説明

END DO 文はブロック DO 文の区切り文です。文番号が DO 文で指定されない場合、対応する終了文は END DO 文でなければなりません。DO ループの範囲内からだけ END DO 文に分岐することができ、それにより DO ループを終了します。

例

例 1 : 文番号付き DO

```
    DO 10 N = 1, 100
    …
10  END DO
```

例 2 : 文番号なし DO

```
    DO N = 1, 100
    …
    END DO
```

END FILE

END FILE 文は、指定された装置に結合したファイルに、次の記録としてファイル終了記録を書き込みます。

```
END FILE   u

END FILE (   [ UNIT= ] u [, IOSTAT=ios ] [, ERR=s ])
```

パラメータ	説明
u	ファイルと結合した外部装置の装置識別子。オプションは任意の順に指定できるが、UNIT= を省略する場合は *u* を最初にする。
ios	入出力状態指定子。整変数または整数の配列要素。
s	誤り指定子 *s* は、END FILE 文が現れたプログラム中の実行文の文番号。END FILE 文の実行時にエラーが発生すると、プログラムの制御はこの文番号に渡される。

説明

磁気テープに、END FILE 文とその他の Fortran 77 規格入出力文を使用している場合は、代わりに信頼性の高い TOPEN() ルーチンを使用することをお勧めします。

2 個のファイル終了記録は、テープ終了マークを意味します。テープファイルに書き込む場合、END FILE は 2 個のファイル終了記録を書き込み、それから テープは 2 番目のファイル終了記録の前まで後退 (バックスペース) します。この場所でファイルを閉じると、ファイルの終了とテープの終了が表示されます。この場所から (次に現れる WRITE 文、または磁気テープを巻き戻さないで他のプログラムで使用することによって) 更に記録が書き込まれる場合、最初のテープマーク (ファイル終了記録) の次に別のデータファイルが続き、そして更に 2 個のテープマークが書き込まれます。

制限事項

u は順番探査ファイルに結合されます。直接探査ファイルでの END FILE 文の実行は FORTRAN 77 規格で定義されていないので、結果は予測できません。したがって直接探査ファイルに END FILE 文を使用しないでください。

例

例 1 : 定数

```
    END FILE 2
    END FILE ( 2 )
    END FILE ( UNIT=2 )
```

例 2 : 変数

```
    LOGUNIT = 2
    END FILE LOGUNIT
    END FILE ( LOGUNIT )
    END FILE ( UNIT=LOGUNIT )
```

例 3 : エラートラップ

```
    NOUT = 2
    END FILE ( UNIT=NOUT, IOSTAT=KODE, ERR=9)
    …
9   WRITE(*,*) 'END FILE でエラー。装置番号:', NOUT
    STOP
```

END IF

END IF 文は、IF で始まるブロック IF の終わりを示し、以下の構文を使用します。

```
END IF
```

説明

各ブロック IF 文に対応する END IF 文は同じプログラム単位内になければなりません。対応する END IF 文はブロック IF 文と同じ IF の深さにあります。

例

例 1 : IF/END IF

```
IF ( N .GT. 0 )THEN
N = N+1
END IF
```

例 2 : IF/ELSE/END IF

```
IF ( N .EQ. 0 ) THEN
N = N+1
ELSE
N = N-1
END IF
```

END MAP

END MAP ♦ 文は、MAP 宣言の終わりを示し、以下の構文を使用します。

```
END MAP
```

説明

詳細については、234 ページの「UNION と MAP」を参照してください。

制限事項

MAP 文は UNION 文の中になければなりません。

例

```
    …
    MAP
    CHARACTER *16 MAJOR
    END MAP
    …
```

END STRUCTURE

END STRUCTURE ♦ 文は、構造体宣言の終わりを示し、以下の構文を使用します。

```
END STRUCTURE
```

説明

詳細については、224 ページの「STRUCTURE」を参照してください。

例

```
    STRUCTURE /PROD/
    INTEGER*4ID
    CHARACTER*16NAME
    CHARACTER*8MODEL
    REAL*4COST
    REAL*4PRICE
    END STRUCTURE
```

END UNION

END UNION ♦ 文は、共用体宣言の終わりを示し、以下の構文を使用します。

```
END UNION
```

説明

詳細については、234 ページの「UNION と MAP」を参照してください。

例

```
UNION
MAP
CHARACTER*16
END MAP
MAP
INTEGER*2    CREDITS
CHARACTER *8 GRAD_DATE
END MAP
END UNION
```

ENTRY

ENTRY 文は、副プログラム内の代替入口を定義します。

ENTRY *en* [([*fa* [, *fa*] …])]

パラメータ	説明
en	関数またはサブルーチン副プログラムの中の入口の英字名
fa	仮引数。仮引数は変数名、配列名、仮手続き名、または選択戻り文番号を指定するアスタリスク。

説明

入口名による引用手続き

サブルーチン副プログラム内で使用される入口名は、サブルーチンのように扱われ、CALL 文で引用できます。同様に、関数副プログラム内で使用される入口名は関数のように扱われ、関数として引用できます。

入口名を EXTERNAL 文の中で指定し、実引数として使用できます。仮引数としては使用できません。

入口副プログラム (サブルーチンまたは関数) の実行は、ENTRY 文の後の最初の実行文から始まります。

ENTRY 文は非実行文です。

ENTRY 文より物理的に前にある実行文で、その ENTRY 文に出現する入口名を使用することはできません。

引数の一致

ENTRY 文の仮引数は、同じ副プログラム内の FUNCTION、SUBROUTINE、および他の ENTRY 文と、順序、数、型および名前が同一である必要はありません。関数、サブルーチン、または入口の引用で使用する実引数の並びは、対応する FUNCTION、SUBROUTINE または ENTRY 文の仮引数の並びと、順序、数、型および名前が一致しなければなりません。

ENTRY 文内の選択戻り引数を指定する場合、仮引数の並びの中にアスタリスクを指定します。アンパサンドは代替として有効です。♦　選択戻り引数を指定する ENTRY 文は、サブルーチン、副プログラムの中でだけ使用でき、関数の中では使用できません。

制限事項

ENTRY 文はブロック IF 構造または DO ループの中では使用できません。

ENTRY 文が文字関数副プログラム中に現れる場合、その関数副プログラムと同じ長さを持つ CHARACTER 型として定義されます。

例

例 1 : サブルーチンの複数の入口

```
SUBROUTINE FIN( A, B, C )
INTEGER A, B
CHARACTER C*4
…
RETURN

ENTRY HLEP( A, B, C )
…
RETURN

ENTRY MOOZ
…
RETURN
END
```

上記の例では、サブルーチン FIN が 2 つの選択的入口を持ちます。入口 HLEP は引数の並びを持ち、入口 MOOZ は引数の並びを持ちません。

例 2 : 呼び出すルーチンの中で上記のサブルーチンと入口名を次のように呼び出します。

```
INTEGER A, B
CHARACTER C*4
…
CALL FIN( A, B, C )
…
CALL MOOZ
…
CALL HLEP( A, B, C )
…
```

上記のように CALL 文の順序は ENTRY 文の順序に合わせる必要はありません。

例 3 : 関数の複数の入口

```
REAL FUNCTION F2 ( X )
F2 = 2.0 * X
RETURN

ENTRY F3 ( X )
F3 = 3.0 * X
RETURN

ENTRY FHALF ( X )
FHALF = X / 2.0
RETURN
END
```

EQUIVALENCE

EQUIVALENCE 文は、プログラム単位内の複数の変数、または配列が同じ記憶領域を共有することを宣言します。

EQUIVALENCE (*nlist*) [, (*nlist*)] ...

パラメータ	説明
nlist	コンマで分離された変数名、配列要素名、配列名、および部分列名の並び

説明

EQUIVALENCE 文は、*nlist* の並びに現れる項目の記憶領域が同じ先頭アドレスを持つことを宣言します。

EQUIVALENCE 文は、*nlist* で指定されていない項目の関連付けを引き起こすことがあります。

配列名の指定は、配列の最初の要素を指定したのと同じ意味を持ちます。

EQUIVALENCE 文の中に配列要素名が現れる場合、添字の数は配列名の配列宣言子で指定される次元数より少ないか、または同じ数をとることができます。

制限事項

仮引数、および関数の名前は、*nlist* には指定できません。

配列要素の添字は、下限より大きくかつ上限より小さいか等しい整定数でなければなりません。

EQUIVALENCE 文は、自動変数を他の自動変数または記憶クラスと関連付けることができます。これらのクラスは、COMMON、STATIC、SAVE、DATA 文、または仮関数であってはなりません。

EQUIVALENCE 文は、文字型の要素と文字型でない要素とを関連付けることができます。♦

EQUIVALENCE 文は、同じ記憶単位が 1 つの記憶領域の中に 2 回以上現れるように宣言することはできません。たとえば次の文は実行できません。

```
    DIMENSION A (2)
    EQUIVALENCE (A(1),B), (A(2),B)
```

EQUIVALENCE 文は、連続する記憶単位を非連続になるように宣言することはできません。たとえば次の文は実行できません。

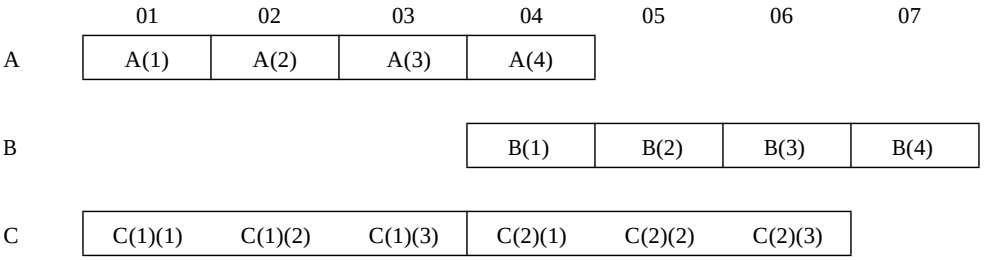
```
    REAL A (2)
    DOUBLE PRECISION D (2)
    EQUIVALENCE (A(1), D(1)), (A(2), D(2))
```

COMMON 文と EQIVALENCE 文が一緒に使用される場合、追加される規則があります。この規則については 106 ページの「COMMON」を参照してください。

例

```
    CHARACTER A*4, B*4, C(2)*3
    EQUIVALENCE (A,C(1)),(B,C(2))
```

A、B、および C の関連は、以下のようになります。先頭 7 文字の位置は、以下のようにメモリーに割り当てられます。



EXTERNAL

EXTERNAL 文は、外部手続きまたは外部仮手続きであることを宣言し、それらの英字名を実引数として使用することを可能にします。

EXTERNAL *proc* [, *proc*] ...

パラメータ	説明
proc	外部手続き、外部仮手続き、または初期値設定副プログラムの名前

説明

外部手続きまたは外部仮手続きが実引数ならば、この実引数は同じプログラム単位の中の EXTERNAL 文内になければなりません。

組み込み関数名が EXTERNAL 文内に現れる場合、この組み込み関数名は外部サブルーチンまたは外部関数とみなされます。その名前の組み込み関数は、このプログラム単位内では使用できません。

制限事項

1 つのサブルーチン名または関数名は、プログラム単位内で EXTERNAL 文に 1 回だけ指定することができます。

文関数名は EXTERNAL 文内には表示できません。

例

例 1： ユーザー作成の TAN の使用

```
EXTERNAL TAN
T = TAN( 45.0 )
…
END
FUNCTION TAN( X )
…
RETURN
END
```

例 2： ユーザー定義関数名を引数として引き渡す

```
REAL AREA, LOW, HIGH
EXTERNAL FCN
…
CALL RUNGE ( FCN, LOW, HIGH, AREA )
…
END

FUNCTION FCN( X )
…
RETURN
END

SUBROUTINE RUNGE ( F, X0, X1, A )
…
RETURN
END
```

FORMAT

FORMAT 文は、入力または出力記録の配置を指定します。

label FORMAT *(f)*

パラメータ	説明
label	文番号
f	書式仕様並び

f の中の項目形式:

[*r*] *desc*

r	反復因子 (オプション)
desc	編集記述子。ただし *r* が表示されている場合 *desc* は反復可能な編集記述子であること。

反復可能な編集記述子には、次のものがあります。

I	F	E	D	G
I*w*	F*w*	E*w*	D*w*	G*w*
I*w.d*	F*w.d*	E*w.d*	D*w.d*	G*w.d*
O	A	E*w.d.e*	D*w.d.e*	G*w.d.e*
O*w*	A*w*	E*w.dEe*	D*w.dEe*	G*w.dEe*
O*w.d*	L			
Z	L*w*			
Z*w*				
Z*w.d*				

まとめ

- I、O、Z は整数用 (10進、8進、16進)
- F、E、D、G は実数用 (固定小数点、指数、倍精度、一般)
- A は文字用
- L は論理値用

反復不可能な編集記述子は以下のとおりです。

- *'a1a2 … an'* 単引用符で区切られた文字列
- *"a1a2 … an"* 二重引用符で区切られた文字列
- *n*H*a1a2 … an* ホレリス文字列
- $
- /
- :
- [*k*]R (*k* のデフォルトは 10)
- [*k*]P (*k* のデフォルトは 0)
- B,BN および BZ
- S,SU,SP および SS
- T*n* および *n*T
- TL[*n*] および TR[*n*] (*n* のデフォルトは 1)
- [*n*]X(n のデフォルトは 1)

編集記述子の詳細は 256 ページの「書式付き入出力」を参照してください。

説明

FORMAT 文には、記録の配置を作成または使用するための明示的な編集指示が含まれます。この編集指示は書式付き入出力文と ENCODE/DECODE 文で使用されます。

反復因子

r はゼロ以外の符号なし整定数でなければなりません。

反復可能な編集記述子

記述子 I、O、Z、F、E、D、G、L、および A は、編集方法を示し反復可能です。

w と *e* は、ゼロ以外の符号なし整定数です。

d と *m* は、符号なし整定数です。

反復不可能な編集記述子

以下に反復不可能な記述子を示します。

(")、($)、(')、(/)、(:)、B、BN、BZ、H、P、R、Q、S、SU、SP、SS、T、TL、TR、X

これらの記述子は編集の方法を示し、反復不可能です。

- *ai* は ASCII 文字です。
- *n* はゼロ以外の符号なし整定数です。
- *k* は符号なしまたは符号付き整定数です。

項目分離文字

書式仕様並びの項目はコンマで分離されます。スラッシュまたはコロン編集記述子の前または後と、P 編集記述子とその直後の F、E、D、G 編集記述子の間ではコンマは省略可能です。コンマがなくても意味が明白な所では省略できますが、上記の場合以外は規格外です。♦

変数式 ♦

一般に書式中の整定数は、山括弧で囲んだ任意の式に置換できます。

```
1     FORMAT(  … < e > … )
```

*n*H... 編集記述子の中の *n* は、変数式にはなりません。

制限事項

FORMAT 文の文番号は、GO TO 文、算術 IF 文、DO 文、選択戻り文の中では使用できません。

注意事項

明示的書式の場合、無効な書式文字列には、コンパイル時に警告とエラーメッセージ、またはそのどちらかが出されます。

変数書式の場合、無効な書式文字列には、実行時に警告とエラーメッセージ、またはそのどちらかが出されます。

<e> の変数書式の場合、無効な書式文字列には、コンパイル時または実行時に警告とエラーメッセージ、またはそのどちらかが出されます。

詳細については、290 ページの「実行時書式」を参照してください。

例

例 1： A、I、および F 書式

```
    READ( 2, 1 ) PART, ID, HEIGHT, WEIGHT
1   FORMAT( A8, 2X, I4, F8.2, F8.2 )
    WRITE( 9, 2 ) PART, ID, HEIGHT, WEIGHT
2   FORMAT( 'Part:', A8, ' Id:', I4, ' Height:', F8.2,
&   ' Weight:', F8.2 )
```

例 2： 変数式

```
    DO 100 N = 1, 50
    …
1   FORMAT( 2X, F<N+1>.2 )
```

FUNCTION (外部)

FUNCTION 文で始まるプログラム単位は関数副プログラムとして識別されます。

[**型**] FUNCTION *fun* ([*ar* [, *ar*] …])

パラメータ	説明
型	BYTE ♦ CHARACTER CHARACTER*n（n は 1 以上であること） CHARACTER*(*) COMPLEX COMPLEX*8 ♦ COMPLEX*16 ♦ COMPLEX*32 ♦ DOUBLE COMPLEX ♦ DOUBLE PRECISION INTEGER INTEGER*2 ♦ INTEGER*4 ♦ INTEGER*8 ♦ LOGICAL LOGICAL*1 ♦ LOGICAL*2 ♦ LOGICAL*4 ♦ LOGICAL*8 ♦ REAL REAL*4 ♦ REAL*8 ♦ REAL*16 ♦
fun	*fun*関数に割り当てられた英字名
ar	*fun*仮引数名

(COMPLEX*32と REAL*16 は、SPARC のみ使用できます)。

長さを指定するための代替構文を次に示します。♦

```
[ 型 ] FUNCTION fun [*m]([ar[,ar] ...])
```

パラメータ	説明
m	データ型の長さを指定するゼロ以外の符号なし整定数
ar	仮引数名

説明

FUNCTION 文の型、値、仮引数について説明します。

関数の型

FUNCTION 文は、型、名前、および仮引数を含みます。

FUNCTION 文に**型**がない場合、関数の型はデフォルトと、その後の IMPLICIT 文または型宣言文によって決定されます。**型**がある場合、その関数名は他の型宣言文には指定できません。

注 – データ型のサイズが明示的に宣言されていない限り、オプション -dbl、-r8、-i2、または -xtypemap のいずれかを指定してコンパイルすると、関数への呼び出しまたは関数の定義で想定されるデフォルトのデータサイズが変更されます。これらのオプションの詳細については、第 2 章「データ型とデータ項目」と『Fortran ユーザーズガイド』を参照してください。

関数の値

関数の英字名は、変数名として副プログラム内になければなりません。関数副プログラム内の RETURN 文または END 文が実行される時の、この変数の値が関数の値です。

仮引数

引数の並びは、仮引数の数を定義します。これら仮引数の型はデフォルトの値、型宣言文、IMPLICIT 文、および DIMENSION 文の組み合せによって定義されます。

仮引数の数は、この関数副プログラムを呼び出す時の実引数の数と同じでなければなりません。

関数は仮引数に値を割り当てることができます。割り当てられた値は、関数副プログラム内の RETURN 文または END 文が実行されると、呼び出し側プログラムに返されます。

制限事項

選択戻り指定子は、FUNCTION 文の中では許可されません。

f77 では再帰的な呼び出しが可能です。関数またはサブルーチンがそれ自身を直接呼び出すと、再帰的な呼び出しになります。関数またはサブルーチンが、他の関数またはサブルーチンを呼び出し、それが復帰する前に元の関数またはサブルーチンを呼び出した場合も、再帰的な呼び出しになります。

例

例 1： 文字関数

```
CHARACTER*5 FUNCTION BOOL(ARG)
BOOL = 'TRUE'
IF (ARG .LE. 0) BOOL = 'FALSE'
RETURN
END
```

上記の例で、BOOL は 5 文字長の CHARACTER 型の関数として定義されています。この関数は呼び出されると、変数 ARG の値に応じて文字列 TRUE または FALSE を返します。

例 2： 実数関数

```
FUNCTION SQR (A)
SQR = A*A
RETURN
END
```

上記の例で、関数 SQR は (デフォルトで) REAL 型の関数として定義され、引き渡された数の 2 乗を返します。

例 3 : 関数のサイズ、 代替構文 ♦

```
    INTEGER FUNCTION FCN*2 ( A, B, C )
```

上記の規格外形式は、次のように処理されます。

```
    INTEGER*2 FUNCTION FCN ( A, B, C )
```

GO TO (割り当て型)

割り当て型 GO TO 文は、変数に割り当てられた文番号値によって識別される文に分岐します。

GO TO *i* [[,] (*s*[, *s*]...)]

パラメータ	説明
i	整変数名
s	実行文の文番号

説明

実行は次のように進みます。

1. **割り当て型 GO TO 文の実行時、変数 *i* には、割り当て型 GO TO 文と同じプログラム単位内にある実行文の文番号値が割り当てられます。**
2. **割り当て型 GO TO 文が実行されると、制御は *i* によって識別される文に移されます。**
3. **文番号の並びが指定されている場合、*i* に割り当てられる文番号は並び内の文番号の 1 つでなければなりません。**

制限事項

i は、GO TO 文と同じプログラム単位内にある ASSIGN 文によって割り当てられます。

i は INTEGER*4 または INTEGER*8 でなければなりません。 INTEGER*2 であってはなりません。

s は GO TO 文と同じプログラム単位内になければなりません。

GO TO 文の中に同じ文番号が 2 回以上現れてもかまいません。

分岐先の文は実行文でなければなりません。DATA、ENTRY、FORMAT、INCLUDE 文は指定できません。

DO、IF、ELSE IF、ELSE ブロックにブロックの外部から分岐することはできません。

例

例：割り当て型 GO TO 文

```
    ASSIGN 10 TO N
    …
    GO TO N ( 10, 20, 30, 40 )
    …
10  CONTINUE
    …
40  STOP
```

GO TO (計算型)

計算型 GO TO 文は、整数式または実数式の値に応じて並びから 1 つの文番号を選択し、制御をその文番号に渡します。

GO TO (*s*[,*s*]…)[,]*e*

パラメータ	説明
s	**実行文の文番号**
e	**整数型または実数型の式**

説明

実行は以下のように進みます。

1. **最初に *e* が評価され、必要であれば整数に変換されます。**
2. ***1* ≦ *e* ≦ *n* (*n* は指定された文番号の数) のとき、指定された並びから *e* 番目の文番号が選択され、制御がその文番号に渡されます。**
3. ***e* の値が範囲外、つまり *e* < *1* または *e* > *n* の場合、計算型 GO TO 文は CONTINUE 文として実行されます。**

制限事項

s は GO TO 文と同じプログラム単位の中になければなりません。

GO TO 文の中に同じ文番号が 2 回以上現れてもかまいません。

分岐先の文は実行文でなければなりません。DATA、ENTRY、FORMAT、INCLUDE 文は指定できません。

DO、IF、ELSE IF、ELSE ブロックにブロック外部から分岐することはできません。

例

例 : 計算型 GO TO 文

```
    …
    GO TO ( 10, 20, 30, 40 ), N
    …
10  CONTINUE
    …
20  CONTINUE
    …
40  CONTINUE
```

上記の例で

- N=1 のとき 10 へ分岐。
- N=2 のとき 20 へ分岐。
- N=3 のとき 30 へ分岐。
- N=4 のとき 40 へ分岐。
- N<1 または N>4 のとき無効。10 へ進む。

GO TO (単純)

単純 GO TO 文は、制御を指定された文に渡します。

GO TO *s*

パラメータ	説明
s	実行文の文番号

説明

GO TO 文の実行により、制御は *s* の文番号が付いた文に渡されます。

制限事項

s は GO TO 文と同じプログラム単位内になければなりません。

分岐先の文は実行文でなければなりません。DATA、ENTRY、FORMAT、INCLUDE 文は指定できません。

DO、IF、ELSE IF、ELSE ブロックにブロックの外部から分岐することはできません。

例

```
    A = 100.0
    B = 0.01
    GO TO 90
    …
90  CONTINUE
```

IF (算術)

算術 IF 文は、算術式の値に応じて 3 個の指定された文の 1 つへ分岐します。

IF (*e*) *s1, s2, s3*

パラメータ	説明
e	算術式 (整数、実数、倍精度、または 4 倍精度)
s1, s2, s3	実行文の文番号

説明

IF 文は算術式の値が負か、ゼロか、正かに応じて、それぞれ第 1、第 2、および第 3 文番号に制御を渡します。

制限事項は次のとおりです。

- *s1*、*s2*、*s3*は、IF 文と同じプログラム単位内になければなりません。
- 同じ文番号が 2 回以上 IF 文の中に現れてもかまいません。
- 分岐先の文は実行文でなければなりません。DATA、ENTRY、FORMAT、INCLUDE 文は指定できません。
- DO、IF、ELSE IF、ELSE ブロックにブロックの外部から分岐することはできません。

例

```
    N = 0
    IF ( N ) 10, 20, 30
```

N の値はゼロなので、制御は文番号 20 に渡されます。

IF (ブロック)

ブロック IF 文は、論理式の値に応じて複数の文の集合から 1 つを選択して実行します。

```
IF ( e ) THEN
…
END IF
```

パラメータ	説明
e	論理式

説明

ブロック IF 文は論理式を評価し、論理式が真のとき IF ブロックと呼ばれる 1 組の文を実行します。論理式が偽のとき、制御は同じ IF の深さの次の ELSE、ELSE IF、END IF 文に渡されます。

IF の深さ

文 *S* の IF の深さは、*n1-n2* の値です。ここで *n1* はプログラム単位の最初から *S* までの *S* を含むブロック IF 文の数です。*n2* はプログラム単位の中の *S* までの END IF 文の数です。*S* は含みません。

例 : 次のプログラムの中の文 9 の IF の深さは 2-1、つまり 1 です。

```
      IF ( X .LT. 0.0 ) THEN
          MIN = NODE
      END IF
      …
9     IF ( Y .LT. 0.0 ) THEN
      MIN = NODE - 1
      END IF
```

あらゆる文の IF の深さは、ゼロか正の値です。IF、ELSE IF、ELSE、END IF 文の各ブロックの IF の深さは、正の値でなければなりません。各プログラム単位の END 文の IF の深さは、ゼロでなければなりません。

IF ブロック

IF ブロックは、ブロック IF 文からそのブロック IF 文と同じ IF の深さを持った次の ELSE、ELSE IF、または END IF 文までのすべての実行文 (ただし ELSE、ELSE IF、END IF を含まない) で構成されています。IF ブロックは空のことがあります。次の例では、2 つの代入文が IF ブロックを構成します。

```
IF ( X .LT. Y ) THEN
M = 0
N = N+1
END IF
```

以下のように実行されます。

1. **論理式 *e* が最初に評価されます。*e* が真のとき、IF ブロックの最初の文から実行されます。**
2. ***e* が真で IF ブロックが空のとき、制御はブロック IF 文と同じ IF の深さを持つ次の END IF 文 に渡されます。**
3. ***e* が偽のときは、制御はブロック IF 文と同じ IF の深さを持つ次の ELSE IF、ELSE、または END IF 文に渡されます。**
4. **IF ブロックの最後の文が、文番号への分岐にならなかった場合、制御はその IF ブロックの前のブロック IF 文と同じ IF の深さを持つ、次の END IF 文に渡されます。**

制限事項

IF ブロックの外部からその IF ブロックに分岐することはできません。

例

例 1 : IF-THEN-ELSE

```
IF ( L ) THEN
N=N+1
CALL CALC
ELSE
K=K+1
CALL DISP
END IF
```

例 2 : ELSE-IF を含む IF-THEN-ELSE-IF

```
IF ( C .EQ. 'a' ) THEN
NA=NA+1
CALL APPEND
ELSE IF ( C .EQ. 'b' ) THEN
NB=NB+1
CALL BEFORE
ELSE IF ( C .EQ. 'c' ) THEN
NC=NC+1
CALL CENTER
END IF
```

例 3 : 入れ子にされた IF-THEN-ELSE

```
IF ( PRESSURE .GT 1000.0 ) THEN
IF ( N .LT. 0.0 ) THEN
X = 0.0
Y = 0.0
ELSE
Z = 0.0
END IF
ELSE IF ( TEMPERATURE .GT. 547.0 ) THEN
Z = 1.0
ELSE
X = 1.0
Y = 1.0
END IF
```

IF (論理)

論理 IF 文は、論理式の値に応じて 1 つの文を実行するかしないかを決定します。

```
IF ( e ) st
```

パラメータ	説明
e	論理式
st	実行文

説明

論理 IF 文は論理式を評価し、論理式の値が真のときは指定された文を実行します。論理式の値が偽のときは指定された文は実行されず、CONTINUE 文が実行されたように実行が継続されます。

st は任意の実行文です。DO ブロック、IF、ELSE IF、ELSE、END IF、END、または論理 IF 文を除きます。

例

```
IF ( VALUE .LE. ATAD ) CALL PUNT ! THEN がないことに注意
IF ( TALLY .GE. 1000 ) RETURN
```

IMPLICIT

IMPLICIT 文は、デフォルトの名前の型を確認または変更します。

```
IMPLICIT 型 ( a [, a ] … ) [, 型 ( a [, a ] … ) ]
IMPLICIT NONE
IMPLICIT UNDEFINED(A-Z) ♦
```

パラメータ	説明
型	BYTE ♦ CHARACTER CHARACTER*n （n は 1 以上であること） CHARACTER*(*) COMPLEX COMPLEX*8 ♦ COMPLEX*16 ♦ COMPLEX*32 ♦(SPARCのみ) DOUBLE COMPLEX ♦ DOUBLE PRECISION INTEGER INTEGER*2 ♦ INTEGER*4 ♦ INTEGER*8 ♦ LOGICAL LOGICAL*1 ♦ LOGICAL*2 ♦ LOGICAL*4 ♦ LOGICAL*8 ♦ REAL REAL*4 ♦ REAL*8 ♦ REAL*16 ♦(SPARCのみ) AUTOMATIC ♦ STATIC ♦
a	1 文字の英文字、またはアルファベット順に並んだ 1 文字ずつの英文字の範囲。英文字の範囲は、マイナス記号で区切られた最初と最後の文字で指定される。

説明

暗黙の型宣言と非暗黙の型宣言の違いは、次のとおりです。

暗黙の型宣言

IMPLICIT 文は、プログラム単位内に暗黙の型宣言規則を適用しないことも宣言できます。

IMPLICIT 文は、その指定の中に現れる英文字 (1 つの英文字またはある範囲の英文字) で始まるすべてのユーザー定義の名前について型と大きさを宣言します。

IMPLICIT 文は、組み込み関数の型を変更しません。

IMPLICIT 文は、IMPLICIT 文を含むプログラム単位の中でだけ適用されます。

プログラム単位は、2 つ以上の IMPLICIT 文を含むことができます。

型宣言文によって宣言されたユーザー定義の名前に対しては、IMPLICIT 文の宣言は無効になります。

注 – オプション -dbl、-i2、-r8、または -xtypemap のいずれかを指定してコンパイルすると、IMPLICIT REAL (A-Z) のようにサイズを指定しない IMPLICIT 文によって型宣言された名前の想定サイズが変更されることがあります。詳細については、第 2 章「データ型とデータ項目」と『Fortran ユーザーズガイド』を参照してください。

非暗黙の型宣言

IMPLICIT 文の第 2 の形式は、ユーザー定義の名前に暗黙の型宣言を適用しないこと、およびすべてのユーザー定義の名前は明示的に宣言された型を持つことを指定します。

IMPLICIT NONE または IMPLICIT UNDEFINED (A-Z) が指定された場合には、プログラム単位内に他の IMPLICIT 文は存在できません。

制限事項

IMPLICIT 文は、他のすべての宣言文の前になければなりません。

同じ英文字が 1 つの英文字、またはある範囲の英文字群として、1 つのプログラム単位のすべての IMPLICIT 文の中に 2 回以上現れてもかまいません。♦

FORTRAN 77 規格は、これを 1 回だけに制限しています。f77 の場合には、英文字が 2 回以上使用されると使用のたびに順番に宣言されます。例 4 を参照してください。

例

例 1 : IMPLICIT、すべてが整数

```
IMPLICIT INTEGER (A-Z)
X = 3
K = 1
STRING = 0
```

例 2 : U、V、または W で始まる場合は複素数、C または S で始まる場合は文字

```
IMPLICIT COMPLEX (U,V,W), CHARACTER*4 (C,S)
U1 = ( 1.0, 3.0)
STRING = 'abcd'
I = 0
X = 0.0
```

例 3 : すべての項目に宣言が必要な例

```
IMPLICIT NONE
CHARACTER STR*8
INTEGER N
REAL Y
N = 100
Y = 1.0E5
STR = 'Length'
```

上記の例のように最初に IMPLICIT NONE が指定されると、すべての変数が明示的に宣言されなければなりません。

例 4 : 英文字を 2 回使用 ♦

```
IMPLICIT INTEGER (A-Z)
IMPLICIT REAL (A-C)
C = 1.5E8
D = 9
```

上記の例で A-Z は INTEGER を、A-C は REAL を意味します。

INCLUDE

INCLUDE ♦ 文は、ファイルを原始プログラムに挿入します。

```
INCLUDE 'file'
```

パラメータ	説明
file	挿入されるファイルの名前

```
INCLUDE "file"
```

説明

名前を指定されたファイルの内容が INCLUDE 文に置き換えられます。

探索パス

INCLUDE 文で引用されている名前が文字 / で始まる場合、f77 は INCLUDE ファイルの絶対パス名を意味するものと解釈します。それ以外の場合、f77 は次に示す順序で以下のディレクトリの中からファイルを探します。

1. INCLUDE 文を持つ原始ファイルを含むディレクトリ。
2. -Iloc オプションで名前を指定されたディレクトリ。
3. f77 のコマンドが出された現在のディレクトリ。
4. デフォルトリストのディレクトリ。標準インストールの場合、デフォルトリストは次のようになります。

```
/opt/SUNWspro/<release>/include/f77 /usr/include
```

ディレクトリ /*mydir*/ への非標準インストールの場合、デフォルトリストは次のようになります。

```
/mydir/SUNWspro/<release>/include/f77 /usr/include
```

<release> は、コンパイラのリリースによって異なります。

これらの INCLUDE 文は、10 段階の深さまで入れ子にすることができます。

プリプロセッサの #include

INCLUDE 文で検索したパスと順序は、『Fortran ユーザーズガイド』の -I オプションの説明にあるように、プリプロセッサの #include 指令で検索するものと同じではありません。プリプロセッサの #include 指令によって取り込まれたファイルは、#define などを含むことができるのに対して、コンパイラの INCLUDE 文で取り込まれたファイルは Fortran 文以外を含むことはできません。

INCLUDE 文の中の VMS 論理ファイル名

f77 は次の場合に INCLUDE 文の VMS 論理ファイル名を解釈します。

- コンパイラオプション -xl[d] または -vax=*spec* が設定されている場合
- 論理名と UNIX パス名の間のマッピングを定義する環境変数 LOGICALNAMEMAPPING がある場合

f77 は解釈のため以下の規則を使用します。

- 環境変数は、次の構文で文字列として設定されます。

```
"lname1=path1; lname2=path2; ... "
```

ここで各 *lname* は論理名、*path1*、*path2* はディレクトリのパス名 (終わりの '/' は付かない) です。

- この文字列を解析するとき、すべての空白は無視されます。終わりの /[no]*list* は、INCLUDE 文のファイル名から取り去られます。
- ファイル名の中の論理名は VMS ファイルの中で最初の : によって区切られるため、f77 は *lname1*:*file* 形式のファイル名を *path1*/*file* 形式に変換します。
- 論理名は、大文字/小文字が区別されます。論理名が、LOGICALNAMEMAPPING で指定されていない INCLUDE 文で発見された場合には、ファイル名は変更されずそのまま使用されます。

例

例 1： 基本的な INCLUDE の例

```
    INCLUDE 'stuff'
```

これはファイル *stuff* の内容に置換されます。

例 2： INCLUDE の探索パス

条件は次のとおりです。

- ソースファイルが次の行を持つ

```
    INCLUDE 'ver1/const.h'
```

- 現在のディレクトリが、/usr/ftn
- ソースファイルが、/usr/ftn/projA/myprg.f

上記の場合、f77 は 以下に示す順序でこれらのディレクトリの中の const.h を探します。

標準インストールの場合、f77 は以下のディレクトリを探索します。

- /usr/ftn/projA/ver1
- /usr/ftn/ver1
- /opt/SUNQspro/<release>/include/f77/ver1
- /usr/include

ディレクトリ /mydir への非標準インストールの場合は、/opt を /mydir に置き換えます。*<release>* パスは、コンパイラのリリースごとに変わります。

INQUIRE

INQUIRE 文は、装置またはファイルについての情報を返します。

INQUIRE([UNIT=] *u, slist*)

INQUIRE(FILE= *fn, slist*)

パラメータ	説明
fn	問い合わせされるファイルの名前
u	問い合わせされるファイルの番号

slist　　指定子並び。下記の中の 1 つまたは複数を任意の順で含むことができる。

- ERR = *s*
- EXIST = *ex*
- OPENED = *od*
- NAMED = *nmd*
- ACCESS = *acc*
- SEQUENTIAL = *seq*
- DIRECT = *dir*
- FORM = *fm*
- FORMATTED = *fmt*
- UNFORMATTED = *unf*
- NAME = *fn*
- BLANK = *blnk*
- IOSTAT = *ios*
- NUMBER = *num*
- RECL = *rcl*
- NEXTREC = *nr*

説明

ファイルが存在するか、開かれているか、または順番入出力に結合されているかを確認できます。ファイルは名前、存在する (またはしない)、および特定の方法 (FORMATTED、UNFORMATTED、SEQUENTIAL、または DIRECT) で結合可能か否かなどの属性を持ちます。

装置またはファイル名で問い合わせを行うことができますが、同一の文の中で両方で問い合わせることはできません。

このシステム環境の中で、あるファイルに関してユーザーが持つアクセス権を知る唯一の方法は、ACCESS(3F) 関数を使用することです。INQUIRE 文では、アクセス権を検出できません。

次の表は、INQUIRE 文の指定子についてまとめたものです。

表 4-1　INQUIRE 文の指定子

形式：指定子＝変数

指定子	変数の値	変数のデータ型
ACCESS	'DIRECT' 'SEQUENTIAL'	CHARACTER
BLANK	'NULL','ZERO'	CHARACTER
DIRECT *	'YES' 'NO' 'UNKNOWN'	CHARACTER
ERR	文番号	INTEGER
EXIST	.TRUE., .FALSE.	LOGICAL
FORM	'FORMATTED' 'UNFORMATTED' 'BINARY'♦	CHARACTER
FORMATTED *	'YES' 'NO' 'UNKNOWN'	CHARACTER
IOSTAT	エラー番号	INTEGER
NAME †	ファイルの名前	CHARACTER
NAMED †	.TRUE., .FALSE.	LOGICAL
NEXTREC	次記録番号	INTEGER
NUMBER *	装置番号	INTEGER
OPENED	.TRUE., .FALSE.	LOGICAL
RECL	記録長	INTEGER
SEQUENTIAL *	'YES' 'NO' 'UNKNOWN'	CHARACTER
UNFORMATTED *	'YES' 'NO' 'UNKNOWN'	CHARACTER

*は、装置による問い合わせが規格外であるが、f77 では受け付けられることを示す。

†は、ファイルによる問い合わせが規格外であるが、f77 では受け付けられることを示す。

- 一時ファイルの場合には、NAMED と NUMBER は返されません。
- 指定された名前を持つファイルがない場合、次に示す指定子は返されません。DIRECT、FORMATTED、NAME、NAMED、SEQUENTIAL、および UNFORMATTED
- OPENED=.FALSE.の場合、次に示す指定子は返されません。ACCESS、BLANK、FORM、NEXTREC、および RECL
- 指定された装置にファイルが結合されていない場合、次に示す指定子は返されません。ACCESS、BLANK、DIRECT、FORM、FORMATTED、NAME、NAMED、NEXTREC、NUMBER、RECL、SEQUENTIAL、および UNFORMATTED
- ACCESS='SEQUENTIAL' のとき、RECL と NEXTREC は不定です。
- FORM='UNFORMATTED' のとき、BLANK は返されません。

INQUIRE 指定子のキーワード

INQUIRE 指定子のキーワードに関する詳細は以下のとおりです。

ACCESS = *acc*

- *acc* は、結合が順番入出力用のときは値 'SEQUENTIAL' を、直接入出力用のときは 'DIRECT' を代入される文字変数です。結合されていないときの値は未定義です。

BLANK = *blnk*

- *blnk* は、書式付き入出力に結合されているファイルに対して NULL 空白制御が有効なときは 'NULL' が代入され、空白がゼロに変換されているときは 'ZERO' が代入される文字変数です。

DIRECT = *dir*

- *dir* は、ファイルが直接入出力に結合できるときは値 'YES' を、結合できないときは値 'NO' を、システムが回答できないときは 'UNKNOWN' を代入される文字変数です。

ERR = *s*

- *s* は INQUIRE 文の実行時にエラーが起きたときに分岐する文の文番号です。

EXIST = *ex*

- *ex* は、ファイルまたは装置が存在するときは .TRUE. に設定され、存在しないときは
  .FALSE. に設定される論理変数です。リンクされたファイルが存在しない場合でも、ファイルがリンクである時は、INQUIRE は、.TRUE. を常に返します。

FILE = *fn*

- *n* はファイルの名前を持つ文字式または * です。ファイル名に続く空白は無視されます。ファイル名がすべて空白のときは現在のディレクトリです。ファイルは現行プログラムの中で装置に結合している必要はありません。

FORM = *fm*

- *fm* は、ファイルが書式付き入出力に結合されているときは値 'FORMATTED' を、書式なし入出力に結合されているときは値 'UNFORMATTED' を、非構造化入出力でファイルが開かれたときは値 'BINARY' を代入される文字変数です。♦

FORMATTED = *fmt*

- *fmt* は、ファイルが書式付き入出力に結合できるときは値 'YES' を、結合できないときは 'NO' を、システムが回答できないときは 'UNKNOWN' を代入される文字変数です。

IOSTAT = *ios*

- *ios* は OPEN 文の中の *ios* と同じです。

NAME = *fn*

- *fn* は、装置に結合されているファイルの名前を代入される文字変数です。ユーザーが装置番号で問い合わせたとき、OPENED と NAMED 変数が両方とも真でなければ、名前パラメータは未定義です。ファイル名で問い合わせたときは、この名前パラメータが返されます。ただし FORTRAN 77 規格では未定義のままです。

NAMED = *nmd*

- *nmd* は、ファイルが名前を持っているときは .TRUE. に設定され、ファイルが名前を持っていないときは .FALSE. に設定される論理変数です。

NEXTREC = *nr*

- *nr* は、直接探査に結合されているファイルから読み取った最終記録の番号に 1 を加えた整変数です。ファイルが結合されていない場合は、-1 が戻されます。

NUMBER = *num*

- *num* は、装置に結合されているファイルがあれば、装置の番号に設定される整変数です。結合されているファイルがない場合、- 1 に設定されます。

OPENED = *od*

- *od* は、ファイルが装置に、または装置がファイルに結合されているときは .TRUE. に設定され、結合していないときは .FALSE. に設定される論理変数です。

RECL = *rcl*

- *rcl* は、ファイルが直接探査に結合されているときにファイル記録の記録長が割り当てられる整変数です。-xl[d] オプションが設定されていると、f77 は INQUIRE 文が戻す *rcl* を調整しません。-xl[d] オプションが設定されていると、OPEN 文がこの調整を行います。-xl[d] についての説明は、366 ページの「-xl[d] オプションを必要とする機能の詳細」を参照してください。結合されているファイルがない場合、rcl は -1 に設定されます。

SEQUENCIAL = *seq*

- *seq* は、ファイルが順番入出力に結合できるときは値 'YES' を、順番入出力に結合できないときは値 'NO' を、システムが回答できないときは 'UNKNOWN' を代入される文字変数です。

UNFORMATTED = *unf*

- *fmt* は、ファイルが書式なし入出力に結合できるときは値 'YES' を、結合できないときは 'NO' を、システムが回答できないときは 'UNKNOWN' を代入される文字変数です。

UNIT = *u*

- *u* は装置番号を指定する整数式または * です。FILE か、または UNIT の 1 つだけを使用します。

例

例 1 : 装置による問い合わせ

```
LOGICAL OK
INQUIRE( UNIT=3, OPENED=OK )
IF ( OK ) CALL GETSTD ( 3, STDS )
```

例 2 : ファイルによる問い合わせ

```
LOGICAL THERE
INQUIRE( FILE='.profile', EXIST=THERE )
IF ( THERE ) CALL GETPROFILE( FC, PROFILE )
```

例 3 : 複数の応答、UNIT= を省略

```
CHARACTER FN*32
LOGICAL HASNAME, OK
INQUIRE ( 3, OPENED=OK, NAMED=HASNAME, NAME=FN )
IF ( OK .AND. HASNAME ) PRINT *, 'ファイル名="', FN, '"'
```

INTEGER

INTEGER 文は、定数名、変数、配列、関数、仮関数の型を整数型に宣言します。

また配列の次元と大きさを指定し、初期値を与えることもできます。

INTEGER [* *len*[,]] *v* [**len* [/*c*/]] [, *v* [* *len* [/*c*/]] ...

パラメータ	説明
v	定数名、変数、配列、配列宣言子、関数または仮関数の名前
len	定数名、変数、配列要素または関数のバイト単位の長さで、2、4 または 8。8 は -dbl がオンのときのみ指定できます。
c	直前にくる名前の定数の並び

説明

宣言としては、INTEGER、INTEGER*2、INTEGER*4、INTEGER*8 が可能です。

INTEGER

INTEGER H などの宣言の場合、通常、変数 H はメモリー内の 1 つの INTEGER*4 要素であり、1 つの整数値として解釈されます。サイズの指定は規格外です。♦

サイズを指定しない場合、デフォルトサイズが使用されます。INTEGER H などの宣言のデフォルトサイズを変更するには、オプション -dbl、-i2、-r8、または -xtypemap のいずれかを指定してコンパイルします。詳細については、第 2 章「データ型とデータ項目」を参照してください。

INTEGER*2 ♦

INTEGER*2 H などの宣言の場合、変数 H は必ずメモリー内の 1 つの INTEGER*2 要素であり、1 つの整数値として解釈されます。

INTEGER*4 ♦

INTEGER*4 H などの宣言の場合、変数 H は必ずメモリー内の 1 つの INTEGER*4 要素であり、1 つの整数値として解釈されます。

INTEGER*8 ♦

INTEGER*8 H などの宣言の場合、変数 H は必ずメモリー内の 1 つの INTEGER*8 要素であり、1 つの整数値として解釈されます。

制限事項

配列の添字には、INTEGER*8 変数および 8 バイトの定数や式を使用しないでください。使用した場合は、末尾の 4 バイトのみが使用されます。インデックス値が 4 バイトの整数の幅を超えた場合、この動作によるプログラムの結果は予測できません。

例

例 1 : 以下の整数宣言は、それぞれ等価です。

```
INTEGER U, V(9)
INTEGER*4 U, V(9)
INTEGER U*4, V(9)*4
```

例 2 : 初期化

```
INTEGER U / 1 /, V / 4 /, W*2 / 1 /, X*2 / 4 /
```

INTRINSIC

INTRINSIC 文は実引数として引き渡すことのできる組み込み関数を指定します。

INTRINSIC *fun* [, *fun*] ...

パラメータ	説明
fun	関数名

説明

組み込み関数の名前を実引数として使用する場合、それらは同一プログラム単位内の INTRINSIC 文に書かれていなければなりません。

例 : 次の例は実引数として引き渡される組み込み関数です。

```
INTRINSIC SIN, COS
X = CALC ( SIN, COS )
```

制限事項

1 つの英字名は同一プログラム単位内の EXTERNAL 文と INTRINSIC 文のどちらか一方でしか使用できません。

実引数は個別名でなければなりません。ほとんどの総称名は個別名と同じですが、IMAG、LOG、LOG10 のようにそうでないものもあります。

英字名は INTRINSIC 文で 2 回以上使用することができます。ただし、FORTRAN 77 規格では 1 回に限定されています。♦

以下の表の組み込み関数はインラインまたは総称なので実引数として引き渡すことはできません。

表 4-2　実引数にできない組み込み関数

LOC	INT	SNGL	AIMAX0	AMIN1
AND	IINT	SNGLQ	AJMAX0	DMIN1
IAND	JINT	REAL	IMAX0	IMIN1
IIAND	IQINT	DREAL	JMAX0	JMIN1
JIAND	IIQINT	DBLE	MAX1	QMIN1
OR	JIQINT	DBLEQ	AMAX1	IMAG
IOR	IFIX	QEXT	DMAX1	LOG
IIOR	IIFIX	QEXTD	IMAX1	LOG10
IEOR	JIFIX	QFLOAT	JMAX1	QREAL
IIEOR	IDINT	CMPLX	QMAX1	QCMPLX
JIOR	IIDINT	DCMPLX	MIN	SIZEOF
JIEOR	JIDINT	ICHAR	MIN0	EPBASE
NOT	FLOAT	IACHAR	AMIN0	EPEMAX
INOT	FLOATI	ACHAR	AIMIN0	EPEMIN
JNOT	FLOATJ	CHAR	AJMIN0	EPHUGE
XOR	DFLOAT	MAX	IMIN0	EPMRSP
LSHIFT	DFLOTI	MAX0	JMIN0	EPPREC
RSHIFT	DFLOTJ	AMAX0	MIN1	EPTINY
LRSHFT	IZEXT	JZEXT	ZEXT	

LOGICAL

LOGICAL 文は、定数名、変数、配列、関数、仮関数の型を論理型に宣言します。

また配列の形状を指定し、初期値を与えることもできます。

LOGICAL [*len[,]] v[* len [/c/]] [, v [* len [/c/]] ...

パラメータ	説明
v	定数名、変数、配列、配列宣言子、関数または仮関数の名前
len	定数名、変数、配列要素または関数のバイト単位の長さで、1、2、4、8 のいずれか。8 は -dbl オプションがオンの場合のみ指定できます。♦
c	この前にくる名前の定数の並び

説明

宣言としては、LOGICAL、LOGICAL*1、LOGICAL*2、LOGICAL*4、LOGICAL*8 が可能です。

LOGICAL

LOGICAL H などの宣言の場合、通常、変数 H はメモリー内の 1 つの INTEGER*4 要素であり、1 つの論理値として解釈されます。サイズの指定は規格外です。♦

サイズを指定しない場合、デフォルトサイズが使用されます。LOGICAL Z などの宣言のデフォルトサイズを変更するには、オプション -dbl、-i2、-r8、または -xtypemap のいずれかを指定してコンパイルします。詳細については、第 2 章「データ型とデータ項目」を参照してください。

LOGICAL*1 ♦

LOGICAL*1 H などの宣言の場合、変数 H は必ずメモリー内の BYTE 要素であり、1 つの論理値として解釈されます。

LOGICAL*2 ♦

LOGICAL*2 H などの宣言の場合、変数 H は必ずメモリー内の 1 つの INTEGER*2 要素であり、1 つの論理値として解釈されます。

LOGICAL*4 ♦

LOGICAL*4 H などの宣言の場合、変数 H は必ずメモリー内の 1 つの INTEGER*4 要素であり、1 つの論理値として解釈されます。

LOGICAL*8 ♦

LOGICAL*8 H などの宣言の場合、変数 H は必ずメモリー内の 1 つの INTEGER*8 要素であり、1 つの論理値として解釈されます。

例

例 1 : 以下の整数宣言は、それぞれ等価です。

```
LOGICAL U, V(9)
LOGICAL*4 U, V(9)
LOGICAL U*4, V(9)*4
```

例 2 : 初期化

```
LOGICAL U /.false./, V /0/, W*4 /.true./, X*4 /'z'/
```

MAP

MAP ♦ 宣言は共用体における欄の代替グループを定義します。

```
MAP
      欄宣言
      …
      [欄宣言]
END MAP
```

説明

欄宣言は次のどれかです。

- 型宣言 (初期値を含めることもできます)
- 部分構造体 (別の構造体宣言または先に定義された記録のどちらか)
- 共用体宣言 (詳細については、234 ページの「UNION と MAP」を参照してください)

例

例： MAP

```
    STRUCTURE /STUDENT/
    CHARACTER*32  NAME
    INTEGER*2  CLASS
    UNION
    MAP
        CHARACTER*16 MAJOR
    END MAP
    MAP
        INTEGER*2  CREDITS
        CHARACTER*8  GRAD_DATE
    END MAP
    END UNION
    END STRUCTURE
```

NAMELIST

NAMELIST ♦ 文は変数または配列名の並びを定義し、それを一意のグループ名と関連付けます。

NAMELIST / *grname* / *namelist* [[,] / *grname* / *namelist*] ...

パラメータ	説明
grname	グループの英字名
namelist	変数と配列の並び

説明

NAMELIST 文に含まれる、グループ名および変数群について説明します。

グループ名

グループ名は変数群入出力文で使用され、読み取りまたは書き込みの対象となる変数または配列の並びを示します。変数群入出力文では入出力並びの代わりにこの名前が使用されます。グループ名は一意とし、読み取りまたは書き込みの可能な項目の並びを表します。

同一グループの変数はいくつかの NAMELIST 文によって別々に同一のグループ名で定義できます。これらの定義はまとめて 1 つの NAMELIST によるグループの定義とみなされます。

変数群の項目

変数群の項目は任意のデータ型とすることができます。

変数群の項目には変数または配列を指定することができ、これらの項目は複数の変数群に属することができます。変数群で指定された項目に限り変数群入出力での読み取りまたは書き込みが可能ですが、すべての項目に対して入力記録の中のデータを指定する必要はありません。

変数群の項目の順序により変数群出力への値の書き込み順序が制御されます。入力記録中では、項目はどのような順序で並んでいてもかまいません。

制限事項

入力データは、変数群に現れる配列の要素または文字列の部分列に値を割り当てることができます。

NAMELIST 文 に以下のものを使用することはできません。

- 定数 (パラメータ)
- 配列要素
- 記録と記録欄
- 部分列
- 大きさ引き継ぎ仮配列

例

例： NAMELIST 文

```
CHARACTER*16 SAMPLE
LOGICAL*4 NEW
REAL*4 DELTA
NAMELIST /CASE/ SAMPLE, NEW, DELTA
```

この例ではグループ CASE が SAMPLE、NEW、DELTA の 3 つの変数で構成されます。

OPEN

OPEN 文は、既存の外部ファイルを装置に結合したり、ファイルを作成して装置に結合したり、あるいは結合の指定子を変更します。

OPEN ([UNIT=] *u*, *slist*)

パラメータ	説明
UNIT	装置番号
slist	*slist* の指定子の並びには、以下のうち 1 つ以上が入ります。 • FILE = *fin* または代替形式として NAME = *fin* • ACCESS = *acc* • BLANK = *blnk* • ERR = *s* • FORM = *fm* • IOSTAT = *ios* • RECL = *rl* または代替形式として RECORDSIZE = *rl* • STATUS = *sta* または代替形式として TYPE = *sta* • FILEOPT = *fopt* ♦ • READONLY ♦ • ACTION = *act* ♦

説明

OPEN 文は名前を指定されたファイルの型を決め、指定された結合がそのファイルの型に対して正当か不当かを判断します (たとえば DIRECT アクセスは磁気テープと tty デバイスに対しては不当となります)。ファイルが磁気テープ上にある場合、またはサブパラメータ FILEOPT='BUFFER=*n*' が指定された場合は、その結合にバッファを割り当てます。既存のファイルが開処理で不完全なものになることは決してありません。

注 – テープ入出力の場合、TOPEN() ルーチンを使用します。

以下の表は各 OPEN キーワード指定子について詳しくまとめたものです。

表 4-3 OPEN 文の指定子

形式：**指定子** = **変数**

指定子	変数の値	変数のデータ型
ACCESS	'APPEND' 'DIRECT' 'SEQUENTIAL'	CHARACTER
ACTION	'READ' 'WRITE' 'READWRITE'	CHARACTER
BLANK	'NULL' 'ZERO'	CHARACTER
ERR	文番号	INTEGER
FORM	'FORMATTED' 'UNFORMATTED' 'PRINT' 'BINARY'♦	CHARACTER
FILE	ファイル名	CHARACTER
FILEOPT	'NOPAD' 'BUFFER=n' 'EOF'	CHARACTER
IOSTAT	エラー番号	INTEGER

表 4-3　OPEN 文の指定子

形式：指定子 = 変数		
指定子	**変数の値**	**変数のデータ型**
READONLY	-	-
RECL	記録長	INTEGER
STATUS	'OLD'	CHARACTER
	'NEW'	
	'UNKNOWN'	
	'SCRATCH'	

以上の指定子キーワードの指定順序は自由です。

OPEN 指定子キーワード

以下に、OPEN 指定子キーワードについて詳しく説明します。

[UNIT=] *u*

- *u* は、装置番号を指定する整数式またはアスタリスク (*)。
  u は必要です。*u* がパラメータ並びの最初にある場合は UNIT= を省略できます。

FILE= *fin*

- *fin* は開くファイルを指定する省略可能な文字式。
  OPEN 文では必ずしもファイル名を指定する必要はありません。指定しない場合はデフォルトのファイル名が生成されます。

- ファイルを再び開く

ファイル名を指定せずに (または前のファイル名で)、一度開いた装置を再び開こうとすると、Fortran はユーザーがパラメータ変更のためにファイルを開きなおそうとしているものと解釈します。ファイルの位置は変わりません。
ユーザーが変更できるパラメータは BLANK (NULL または ZERO) と FORM (FORMATTED または PRINT) だけです。その他のパラメータを変更する場合はファイルを一度閉じてから開きなおします。

- ファイル切り替え

すでに開かれている装置を再び開く際に前と異なるファイル名を指定した場合は、開く前に旧ファイル名で閉じたのと同じ状態になります。

- 装置交換

すでに開かれているファイルを再び開く際に前と異なる装置を指定した場合は誤りとなります。ただし、この誤りは ERR= オプションでは検出されないため、プログラムは異常終了しません。

- 一時ファイル

ファイルが STATUS='SCRATCH' で開かれると、一時ファイルが作成および開かれます。STATUS=*sta* を参照してください。

ACCESS= *acc*

- ACCESS=*acc* 句 は省略可能。*acc* は文字式。可能な値は APPEND、DIRECT、および SEQUENTIAL です。デフォルト値は SEQUENTIALです。
- ACCESS='APPEND' の場合
  SEQUENTIAL および FILEOPT='EOF'とみなされます。ファイルを開くためのものであり、既存の順番探査ファイルに記録を追加します。ファイルには WRITE 操作だけができますが、エラー・メッセージは出力されません。これは拡張機能で、ディスクファイルのみに適用することができます。©
- ACCESS='DIRECT' の場合
  入出力の転送はすべて固定長記録の倍数で実行されるため、RECL も指定しなければなりません。
- 直接探査可能なファイルだけに限られるので、 tty、パイプ、磁気テープには使用できません。ファイルを順番探査ファイルとして作成した場合は直接探査できません。
- FORM を指定しない場合は書式なし転送とみなされます。
- FORM='UNFORMATTED' の場合、転送サイズは転送されるデータによって異なります。
- ACCESS='SEQUENTIAL' の場合、RECL は無視されます。♦FORTRAN 77 規格では順番探査で RECL は指定できません。
- 記録はパディングされません。

- ファイルを直接探査ファイルとして作成した場合は、順番探査ファイルとしてアクセスすることはできません。
- tty、パイプ、磁気テープが使用されるという意味では、ファイルがランダムアクセスできなくてもかまいません。しかしテープの場合には TOPEN() ルーチンを使用したほうが確実です。
- FORM を指定しない場合は書式付き転送とみなされます。
- FORM='FORMATTED' の場合、各記録は改行文字 (\n) で終了します。これによって、各記録の文字数は、実際は 1 文字増分されます。
- FORM='PRINT' の場合、ファイルは出力時の 1 桁目の文字の解釈 (空白 = 1 行送り、0 = 2 行送り、1 = 用紙送り、+ = 行送りなし) を除き、FORM='FORMATTED' ファイルと同様に処理されます。
- FORM='UNFORMATTED' の場合、各記録の前後にそれぞれ INTEGER*4、すなわち 4 バイトが付けられ、通常の長さより 8 文字長くなります。この規定は他の言語には用いられていないため、Fortran 77 プログラム間でのデータの受け渡しに限り有用です。

FORM= *fm*

- FORM=*fm* 句は省略可能。*fm* は文字式。可能な値は、'FORMATTED'、'UNFORMATTED'、'BINARY' 、および 'PRINT' です。♦ デフォルトは 'FORMATTED' です。
- このオプションは ACCESS と相互に関連しています。
- 'PRINT' の場合は ACCESS の対象がプリントファイルになります。
- 'BINARY' では、ファイルはレコードマークがない順番の書式なしファイルとして処理されます。♦ FORM = 'BINARY' で ACCESS = 'DIRECT' や RECL = *n* を指定すると、エラーになります。WRITE 文は、出力並びのデータにあるバイト数と同じバイト数をバイナリで書き込みます。READ 文は、入力並びで要求されるバイト数を、入力ファイルから読み取ります。レコードマークは認識されないため、「レコード外」を読み取ることはできません。異常なシステムエラー以外で発生する可能性がある入力エラーは、エンドオブファイルの読取りだけです。FORM = 'BINARY' のファイルには BACKSPACE は指定できず、実行時にエラーとなります。

RECL= *rl*

- RECL=*rl* 句は ACCESS='DIRECT' の場合は必須。その他の場合は無視されます。
- *rl* はファイルの各記録の長さを文字数で指定する整数式です。*rl* は正である必要があります。
- 記録の長さが分からない場合は RECL=1 を使用します。詳細については、255 ページの「直接探査入出力」を参照してください。
- -xl[d] オプションが設定されていない場合、 *rl* は文字数で記録長は *rl* になります。
- -xl[d]が 設定されている場合、 *rl* はワード数で記録長は *rl**4 になります。♦
- 詳細については前記の ACCESS='SEQUENTIAL' の項を参照のこと。
- 1 回の WRITE 操作につき 1 記録が定義され、1 回の READ 操作ごとに 1 記録が読み取られます (読み取られない文字はフラッシュされる)。
- テープのデフォルトのバッファサイズは 64K 文字です。テープの場合は TOPEN() ルーチンを使用したほうが確実です。

ERR= *s*

- ERR=*s* 句は省略可能。*s* は OPEN 文の実行時にエラーが生じた場合に分岐する文の文番号です。

IOSTAT= *ios*

- IOSTAT=*ios* 句は省略可能。*ios* は OPEN 文から誤り状態を受け取る整変数です。OPEN 実行後、誤り状態がなければゼロ、あれば正となります。
- OPEN でエラーが生じてもプログラムを異常終了したくない場合は、ERR=*s* または IOSTAT=*ios* を指定します。

BLANK= *blnk*

- BLANK=*blnk* 句は省略可能。この句は書式化された入力専用句です。*blnk* は空白の扱われ方を指定する文字式です。可能な値は ZERO および NULL です。
- 'ZERO' ― 空白はゼロとして扱われます。

- 'NULL' — 数値変換時に空白は無視されます。デフォルトは 'NULL'です。

STATUS= *sta*

- STATUS=*sta* 句は省略可能。*sta* は文字式です。可能な値は 'OLD'、'NEW'、'UNKNOWN'、および 'SCRATCH'です。
- 'OLD'
  ファイルはすでに存在していることを示します (存在しない場合は誤り)。
  例 : STATUS='OLD'
- 'NEW'
  ファイルは存在しないことを示します (存在している場合は誤り)
  'FILE=*name*' を指定しない場合は 'fort.*n*' と名付けられたファイルが開きます (ここで *n* は指定された論理装置番号)。
- 'UNKNOWN' — ファイルの存在が不明なことを示します (デフォルト)。
- 'SCRATCH'
  STATUS='SCRATCH' でファイルが開かれると、tmp.*FAAAAxnnnnn* という名前の一時ファイルが開きます。関連するファイル名がない他の STATUS 識別子を使用すると、fort.*n* (*n* は指定した論理ユニット番号) という名前のファイルが開かれます。デフォルトでは、一時ファイルは閉じる時または通常終了時に削除される。プログラムが異常終了した場合、このファイルは削除されないことがあります。削除したくない場合は、STATUS='KEEP' オプションで閉じます。
- FORTRAN 77 規格では、名前を指定したファイルを一時的に開くことはできません。OPEN に FILE=*name* オプションを指定する場合、STATUS='SCRATCH' オプションは指定できません。この Fortran 77 では、名前を指定したファイルを一時的に開くように拡張されています。♦ 通常、このようなファイルは閉じる時または通常終了時に削除されます。
- TMPDIR: 通常、scratch ファイルは現在の作業ディレクトリに作成されます。TMPDIR 環境変数が書き込み可能なディレクトリに設定されている場合、scratch ファイルはそのディレクトリに作成されます。♦

FILEOPT= *fopt* ♦

- FILEOPT=*fopt* 句は省略可能。*fopt* は文字式です。可能な値は次のとおりです。

- 'NOPAD' — 記録の終わりを超えて読み取る場合、空白を追加して記録を拡張することはありません (書式付き入力の場合に限る)。つまり、短い記録の読み取りは後ろに空白を加えて続行されることがなく、エラーメッセージが出されて異常終了します。

- 'BUFFER' = *n* — 入出力装置が使用するバッファサイズを *n* バイトに設定します。このサブオプションは、ディスクファイルなどの通常ファイルで使用するためのものです。効率を上げるには、バッファサイズをページサイズの倍数にします。バッファサイズを大きくすると、通常は、順次入出力の効率が向上します。バッファサイズが記録長の長さと等しい場合、通常は直接入出力の効率が最高に達します。ただし、例外として、記録長が 1 のときは、バッファサイズは最低 1 ページでなければなりません。

  BUFFER サブオプションは、通常はテープファイルには使用しません。テープファイルでは、テープのハードウェア、制御装置、ファイルシステムによって適切なバッファサイズが決まります。テープドライブによっては、機能が限定されているものがあり、そのために、通常の Fortran 入出力が不安定になることがあります。そうしたドライブの代わりとして、Fortran ライブラリのテープ入出力ルーチン (topen(3F)参照) を推奨します。

- 'EOF' — テープの始めからではなく、ファイル終了マークでファイルが開きます (ファイルにデータを追加するために使用する)。
  例 : FILEOPT='EOF'
  ACCESS='APPEND' と異なり、 READ と BACKSPACE の両方の操作が可能です。

READONLY ♦

- ファイルを読み取り専用として開きます。

ACTION = *act*

- *act* はファイルのアクセス権を示します。可能な値は READ、WRITE、READWRITE です。

- ACTION = READ の場合、開かれるファイルは読み取り専用になります。

- ACTION = WRITE の場合、開かれるファイルは書き込み専用になります。書き込み専用のファイルで BACKSPACE 文を実行することはできません。

- ACTION = READWRITE の場合、開かれるファイルは読み取りも書き込みも可能です。

例

以下に例を示します。

例 1 : ファイルを開き、それを装置 8 に結合します。

次の OPEN 文のどちらもファイル projectA/data.test を開き、それを FORTRAN 77 装置 8 に結合します。

```
OPEN( UNIT=8, FILE='projectA/data.test' )
OPEN( 8, FILE='projectA/data.test' )
```

上記の例ではデフォルトで、次の特性が確立されます。デフォルトの設定は順番探査、書式付きファイル、およびファイルを開く時のエラーを想定しません。

例 2 : 特性を明示的に指定します。

```
OPEN( UNIT=8, FILE='projectA/data.test', &
ACCESS='SEQUENTIAL', FORM='FORMATTED' )
```

例 3 : 次のどちらもファイル fort.8 を開き、それを装置 8 に結合します。

```
OPEN( UNIT=8 )
OPEN( 8 )
```

上記の例では順番探査、書式付きファイルで、ファイルを開く時に誤りを想定しません。実行するにあたって、ファイル fort.8 が存在しない場合は fort.8 が作成されます。このファイルは終了後も残ります。

例 4 : 開く時の誤りを想定します。

```
OPEN( UNIT=8, FILE='projectA/data.test', ERR=99 )
```

上記の文では、OPEN 文の実行時に誤りが生じた場合は文番号 99 に分岐します。

例 5 : 変数長記録を許可します。

```
OPEN( 1, ACCESS='DIRECT', recl=1 )
```

可変長記録に関する詳細については、255 ページの「直接探査入出力」を参照してください。

例 6 : 一時ファイル

```
    OPEN( 1, STATUS='SCRATCH' )
```

この文は、tmp.FAAAa003zU などの名前の一時ファイルを開きます。このファイルは通常、現在の作業ディレクトリか、(設定されていれば) TMPDIR 変数の指すディレクトリに作成されます。

OPTIONS

OPTIONS ♦ 文はコンパイラのコマンド行オプションを無効にします。

```
OPTIONS /修飾子   [/修飾子 …]
```

説明

以下の**表に、**OPTIONS 文の修飾子を示します。

表 4-4　OPTIONS 文の修飾子

修飾子	処置
/[NO]G_FLOATING	何もしない （機能なし）
/[NO]I4	-i2 オプションの指定を許可または禁止する
/[NO]F77	何もしない （機能なし）
/CHECK=ALL	-C オプションの指定を許可する
/CHECK=[NO]OVERFLOW	何もしない （機能なし）
/CHECK=[NO]BOUNDS	-C オプションの指定を許可または禁止する
/CHECK=[NO]UNDERFLOW	何もしない （機能なし）

表 4-4　OPTIONS 文の修飾子 (続き)

修飾子	処置
/CHECK=NONE	-C オプションの指定を禁止する
/NOCHECK	-C オプションの指定を禁止する
/[NO]EXTEND_SOURCE	-e オプションの指定を許可または禁止する

制限事項

OPTIONS 文はプログラム単位の先頭に置かなくてはなりません。つまり BLOCK DATA 文、FUNCTION 文、PROGRAM 文、SUBROUTINE 文の前ということです。

OPTIONS 文により設定されたオプションは、コマンド行に指定したオプションを無効にします。

OPTIONS 文により設定されたオプションは、そのプログラム単位内でのみ有効です。

修飾子は 4 文字以上の省略形にすることができます。

大文字/小文字の区別はありません。

例

次の例ではサイズを明示されずに宣言された整変数は、コマンド行の -i2 オプションの有無にかかわらず、2 バイトではなく 4 バイトを占めます。これによって整定数のサイズは変更されず、変数のサイズだけが変更されます。

```
OPTIONS /I4
PROGRAM FFT
…
END
```

これに対し /NOI4 を使用した場合は、サイズを明示されずに宣言されたすべての整変数は、コマンド行の -i2 オプションの有無にかかわらず、4 バイトではなく 2 バイトを占めます。整定数は、-i2 オプションがある場合は 2 バイト、ない場合は 4 バイトとなります。

PARAMETER

PARAMETER 文は定数に英字名を割り当てます。

```
PARAMETER ( p =e [, p = e ] … )
```

パラメータ	説明
p	英字名
e	定数式

-xl フラグを設定すると、代替構文を用いることができます。♦

```
PARAMETER p = e [, p = e ] …
```

この代替形式では定数式の型により名前の型が決まりますが、変換は行われません。

説明

e は任意の型にすることができます。英字名とそれに対応する式の型は同じでなければなりません。

英字名を用いて、複素定数の実部、虚部、またはその両方を表すことができます。

定数式は、明示的な定数、パラメータ、および Fortran 演算子で構成されています。詳細については、80 ページの「定数式」を参照してください。

定数式は構造体の記録と記録欄は使用できません。

浮動小数点数によるべき乗は使用できません。使用しようとすると警告が出されます。

英字の型がデータ式の型と同じでない場合は、英字名が PARAMETER 文に初めて現れる前に、型宣言文または IMPLICIT 文で英字名の型を指定しなければなりません。指定しないと変換されます。

CHARACTER 文で英字名の長さが明示的に指定されている場合は、PARAMETER 文の定数がそれより長くてはいけません。長い場合はその分が切り捨てられ、警告が出されます。CHARACTER 文は PARAMETER 文の前に書かなければなりません。

CHARACTER 文が *(*) を用いて英字名の長さを指定した場合、その定数名の長さを決定するのに PARAMETER 文のデータが使用されます。CHARACTER 文は PARAMETER 文の前に書かなければなりません。

式 *e* の中で使用する定数の英字名はすべて、同一プログラム単位内の同じ、または別の PARAMETER 文であらかじめ定義しておかなければなりません。

♦ Sun WorkShop FORTRAN 77 コンパイラでは PARAMETER 文が拡張され、非定数式を含むあらゆる式に対応できます。この文では、規格外であるという警告メッセージが常に表示され、シンボルが参照されたところでは、実行時に値が決められます。ただし、非定数式のある PARAMETER 文で定義したシンボルが、定数が必要となる文 (たとえば DATA 文など) に使われると、エラーになります。

制限事項

定数名は 1 つのプログラム単位内で 1 回しか定義できません。

PARAMETER 文で定義した英字名は、そのプログラム単位内で別のものを表現するために使用することはできません。

英字名は定数書式仕様で使用できませんが、変数書式仕様では使用することができます。

パラメータを引数として渡し、副プログラムがそのパラメータを変更しようとした場合、実行時エラーが発生します。

例

例 1 : 実数型のパラメータ、文字型のパラメータ、論理型のパラメータ

```
      CHARACTER HEADING*10
      LOGICAL T
      PARAMETER (EPSILON=1.0E-6, PI=3.141593,
     &   HEADING=' IO Error #',
     &   T=.TRUE. )
      …
```

例 2 : コンパイラに文字数をカウントさせます。

```
CHARACTER HEADING*(*)
PARAMETER ( HEADING='入出力エラー番号' )
…
```

例 3 : -xl コンパイルフラグを指定した場合の代替構文

```
PARAMETER FLAG1 = .TRUE.
```

上記の文は次のように取り扱われます。

```
LOGICAL FLAG1
PARAMETER (FLAG1 = .TRUE.)
```

PARAMETER 文とも代入文とも解釈できるあいまいな文は、-xl または -xld のどちらかのオプションが指定されていれば、常に PARAMETER 文とみなされます。

例 : あいまいな文

```
PARAMETER S = .TRUE.
```

-xl オプションが設定されていれば、上記の文は変数 S に関する PARAMETER 文となります。

```
PARAMETER S = .TRUE.
```

次のような変数 PARAMETERS に関する代入文とはみなされません。

```
PARAMETERS = .TRUE.
```

PAUSE

PAUSE 文は実行を中断し、go が入力されるのを待ちます。

```
PAUSE [str ]
```

パラメータ	説明
str	5 桁以下の文字列または文字定数

説明

PAUSE 文はプログラムの実行を一時中断し、再開の指示を待ちます。再開の指示があると実行が継続されます。

引数文字列がある場合はそれが画面に表示されます (stdout に書き込まれます)。続いて次のメッセージが表示されます。

```
    PAUSE: 実行を再開するには、go と入力します。
その他の入力では、プログラムを中止します。
```

go を入力すると CONTINUE 文を実行した場合と同様に実行が継続されます。以下はその例です。

```
demo% cat p.f
    PRINT *, "Start"
    PAUSE 1
    PRINT *, "Ok"
    END
demo% f77 p.f -silent
demo% a.out
Start
PAUSE: 1
実行を再開するには、go と入力します。
その他の入力では、プログラムを中止します。
go
PAUSE の後に実行を再開しました。
Ok
demo%
```

stdin が tty 入出力装置ではない場合、PAUSE は次の形式のメッセージを表示します。

```
PAUSE: 実行を再開するには、kill -15 pid と入力します。
```

ここで *pid* はプロセス識別子です。

例 : stdin が tty 入出力装置ではない場合

```
demo% a.out < mydatafile
PAUSE: 実行を再開するには、kill -15 20537 と入力します。
demo%
```

メッセージを表示するウィンドウはコマンド入力を受け取ることができないため、上記の例では他のウィンドウのシェルプロンプトに以下のコマンド行を入力します 。

```
demo% kill -15 20537
```

POINTER

POINTER ♦ 文は変数とポインタの対を確立します。

POINTER (*p1, v1*) [, (*p2, v2*) ...]

パラメータ	説明
v1, v2	ポインタ基底付き変数、指示先とも言う
p1, p2	対応するポインタ

説明

各ポインタの内容は対応する変数のアドレスです。

ポインタ基底付き変数、すなわち指示先は POINTER 文によりポインタと対になった変数です。通常はポインタ基底付き変数のことを単に基底付き変数と言います。ポインタはアドレスを内容とする整変数です。(POINTER 文で指定する変数名は、コンパイラでは VOLATILE と見なされます)

ポインタの使用法については、58 ページの「ポインタ」を参照してください。

例

例 1 : 単純な POINTER 文

```
    POINTER ( P, V )
```

ここで V はポインタ基底付き変数で、P はその関連付けられたポインタです。

例 2 : LOC() 関数によるアドレスの獲得

```
    * ptr1.f: LOC() を介してアドレスを割り当てる
    POINTER ( P, V )
    CHARACTER A*12, V*12
    DATA A / 'ABCDEFGHIJKL' /
    P = LOC( A )
    PRINT *, V(5:5)
    END
```

上記の例では CHARACTER 文は A に 12 バイトの記憶領域を割り当てますが、V には記憶領域を割り当てません。V はポインタ基底付き変数なので、型だけを指定します。次に A のアドレスを P に割り当てます。それによって、V を使用するとポインタ P によって A が引用されます。プログラムは E を出力します。

例 3 : MALLOC によるポインタへの記憶域割り当て

```
    POINTER ( P1, X ), ( P2, Y ), ( P3, Z )
    …
    P1 = MALLOC ( 36 )
    …
    CALL FREE ( P1 )
    …
```

上記の例では MALLOC() で 36 バイトの記憶領域を獲得し、他の命令をいくつか実行し、おそらくこの一連の記憶領域を使用する命令をいくつか実行した後、この 36 バイトをメモリーマネージャに戻すように FREE() に指示します。

例 4 : 記憶領域とそのアドレスの獲得

```
POINTER ( P, V )
CHARACTER V*12, Z*1
P = MALLOC( 12 )
…
END
```

上記の例では関数 MALLOC() で 12 バイトの記憶領域を獲得し、その記憶領域のアドレスをポインタ P に割り当てます。

例 5 : 配列の動的割り当て

```
PROGRAM UsePointers
REAL X
POINTER ( P, X )
…
READ ( *,* ) Nsize ! サイズを取得
P = MALLOC( Nsize )! メモリーを割り当てる
…
CALL CALC ( X, Nsize )
…
END
SUBROUTINE CALC ( A, N )
REAL A(N)
…   ! どのようなサイズの配列でも使用する
RETURN
END
```

この例は今までの例と異なり、やや現実的な形式をとっています。大きさは、たとえば
10,000 のように非常に大きな数値をとることができます。サブルーチンは、一度配列が割り当てられてしまうと、それが動的に割り当てられたことを知らずに動作します。

例 6 : ポインタを使用して、f77 でリンクリストを作成する

```
demo% cat Linked.f
    STRUCTURE /NodeType/
    INTEGER recnum
    CHARACTER*3 label
    INTEGER  next
    END STRUCTURE
    RECORD /NodeType/ r, b
    POINTER (pr,r), (pb,b)
    pb = malloc(12) 基準記録 b を作成
    pr = pb pr は b を指示
    NodeNum = 1
    DO WHILE (NodeNum .LE. 4)記録の初期化と作成
    IF (NodeNum .NE. 1) pr = r.next
    CALL struct_creat(pr,NodeNum)
    NodeNum = NodeNum + 1
    END DO
    r.next = 0
    pr = pb    全記録を表示
    DO WHILE (pr .NE. 0)
    PRINT *, r.recnum, " ", r.label
    pr = r.next
    END DO
    END
    SUBROUTINE struct_creat(pr,Num)
    STRUCTURE /NodeType/
    INTEGER recnum
    CHARACTER*3 label
    INTEGER next
    END STRUCTURE

    RECORD /NodeType/ r
    POINTER (pr,r), (pb,b)
    CHARACTER v*3(4)/'aaa', 'bbb', 'ccc', 'ddd'/

    r.recnum = Num 現在の記録を初期化
    r.label = v(Num)
    pb = malloc(12)   次の記録を作成
    r.next = pb
    RETURN
    END
```

```
demo% f77 -silent Linked.f
"Linked.f", line 6: 警告:  局所変数"b" が使用されていません。
"Linked.f", line 31: 警告:  局所変数"b" が使用されていません。
demo% a.out
   1 aaa
   2 bbb
   3 ccc
   4 ddd
demo%
```

以下の点に注意してください。

- このように -03 または -04 にポインタを使用して、プログラムを最適化しないでください。
- 警告は無視してください。
- これは、この節の始めに説明したポインタの通常の使用法ではありません。

PRINT

PRINT 文は並びから標準出力への書き込みを実行します。

PRINT *f* [, *iolist*]

PRINT *grname*

パラメータ	説明
f	書式識別子
iolist	変数、部分列、配列、記録の並び
grname	変数群の名前

説明

PRINT 文は以下の引数をとります。

書式識別子

f は書式識別子で、以下のものが使用できます。

- 並びによる入出力を示すアスタリスク(*)。詳細については、297 ページの「並びによる入出力」を参照してください。
- 同一プログラム単位内に出現する FORMAT 文の文番号。
- 同一プログラム単位内に出現する FORMAT 文の文番号を割り当てられた整変数名。
- 書式文字列を指定する文字式または整数配列。整数配列は規格外です。♦

出力並び

iolist は、空の場合と、出力並びと DO 型並びの両方または一方を含む場合があります。出力並びは以下のいずれかです。

- 変数
- 部分列
- 配列
- 配列要素
- 記録欄
- その他の式

添字なしで単に配列名だけを書けば、記憶領域へ格納されている順に (左端の添字が他より速く増加する) 配列の全要素が指定されます。

DO 型並びの詳細については、112 ページの「DO 型並び」を参照してください。

変数群 PRINT

PRINT 文の第 2 の形式は指定された変数群の項目を出力するために使用されます。ここで *grname* は NAMELIST 文で先に定義されている変数群の名前です。

実行手順は以下のとおりです。

1. **書式が指定されていればそれが確立されます。**
2. **出力並びが空でなければその並びから標準出力へデータが転送されます。**

   書式が指定されているとそれに応じてデータが編集されます。

3. PRINT **文の第 2 の形式では、指定された変数群の項目から標準出力へデータが転送されます。**

制限事項

例外ハンドラからの出力は予測できません。ユーザー自身の例外ハンドラを作成する場合、そのハンドラから Fortran 77 出力を行わないでください。どうしても必要な場合は、出力直後に異常終了を呼び出します。こうすることによって、システムが凍結する危険を減らすことができます。例外ハンドラからの Fortran 77 入出力は再帰的な入出力です。次の段落を参照してください。

再帰的な入出力の動作は信頼できません。入出力リスト中に関数をリストして、その関数が 入出力を行う場合、実行時に実行が凍結するか、その他にも予測できない問題が発生する可能性があります。この危険は並列化に関わらず存在します。

例 : 再帰的な入出力が断続的に失敗する。

```
    PRINT *, x, f(x)        f() が入出力を行なうため、できない。
    END
    FUNCTION F(X)
    PRINT *, X
    RETURN
    END
```

例

例 1 : 書式付きスカラー

```
    CHARACTER TEXT*16
    PRINT 1, NODE, TEXT
1   FORMAT ( I2, A16 )
```

例 2 : 並びによる配列

```
    PRINT *, I, J, ( VECTOR(I), I = 1, 5 )
```

例 3 : 書式付き配列

```
INTEGER VECTOR(10)
PRINT '( 12 I2 )', I, J, VECTOR
```

例 4 : 変数群

```
CHARACTER LABEL*16
REAL QUANTITY
INTEGER NODE
NAMELIST /SUMMARY/ LABEL, QUANTITY, NODE
PRINT SUMMARY
```

PROGRAM

PROGRAM 文は、プログラム単位を主プログラムとして識別します。

PROGRAM *pgm*

パラメータ	説明
pgm	主プログラムの英字名

説明

ローダーにとって主プログラムの名前は常に MAIN です。PROGRAM 文はプログラムを読む人だけに役立つものです。

制限事項

PROGRAM 文は主プログラムの最初の文としてのみ使用できます。

プログラムに次のような名前を付けることはできません。

- 外部手続きまたは共通ブロックと同じ名前
- MAIN (すべて大文字)。この名前にすると実行時誤りとなります。

プログラムの名前は主プログラムでの局所名と同じにすることができます。♦ FORTRAN 77 規格ではこれができません。

例

例： PROGRAM 文

```
PROGRAM US_ECONOMY
NVARS = 2
NEQS = 2
…
```

READ

READ 文はファイルまたはキーボードから並びの項目へデータを読み取ります。

注 – テープ装置からデータを読み取る場合は、TOPEN() ルーチンを使用します。『Fortran ライブラリ・リファレンス』を参照してください。

```
READ([UNIT=] u [,[FMT=]f] [, IOSTAT=ios] [, REC=rn] [, END= s]
     [, ERR=s]) iolist

READ f[, iolist]

READ([UNIT=] u, [NML=] grname [,IOSTAT=ios] [,END=s] [,ERR=s] )

READ grname
```

パラメータ	説明
u	ファイルに結合された装置の装置識別子
f	書式識別
ios	入出力状態指定子
rn	読み取られる記録番号

s	ファイル処理終了のための文番号
iolist	変数の並び
grname	変数群の名前

UNIT=*u*,REC=*rn* の代替形式は以下のとおりです。♦

READ(*u* ' *rn* ...) *iolist*

オプションはどのような順番で指定してもかまいません。

説明

READ 文は以下の引数をとります。

装置識別子

u は外部装置識別子または内部ファイル識別子です。

外部装置識別子には以下のどちらかを使用してください。

- 負でない整数式。
- アスタリスク (*)。*stdin* を識別し、通常はキーボードに結合されます。

装置指定子で省略可能な文字 UNIT= を省略する場合は、*u* を指定子の並びの最初の項目としなければなりません。

書式識別子

f は書式識別子で以下のどれかを使用します。

- アスタリスク (*) ― 並びによる入出力を示します。詳細については、297 ページの「並びによる入出力」を参照してください。
- 同一プログラム単位内に出現する FORMAT 文の文番号
- 同一プログラム単位内に出現する FORMAT 文の文番号を割り当てられている整変数名
- 書式文字列を指定する文字式または整数配列。実行時書式または変数書式といいます。整数配列は規格外です。♦

実行時に評価される書式の詳細については、290 ページの「実行時書式」を参照してください。

書式指定子で省略可能な文字 FMT= を省略する場合、*f* は書式付き読み取りの第 2 の引数として現れるか、全く現れないかのどちらかでなければなりません。

内部ファイルと端末ファイルから書式なしデータを転送することはできません。したがってこのようなファイルにとっては *f* が不可欠です。

直接探査ファイルと内部ファイルから並びによるデータを転送することができます。したがってこのようなファイルの場合、*f* をアスタリスクにします。♦

ファイルが書式付き入出力に結合される場合、書式なしデータを転送することはできません。同様に、ファイルが書式なし入出力に結合されている場合は書式付きデータを転送することはできません。

入出力状態指定子

ios は整変数または整数の配列要素です。

記録番号

rn は正の整数式で、直接探査ファイルに限り使用できます。*rn* は内部ファイルに指定することもできます。♦

ファイル終了指定子

s は READ 文のあるプログラム単位内の実行文の文番号です。

END=*s* と REC=*rn* の各指定子は READ 文で同時に指定できます。♦

誤り指定子

s は READ 文のあるプログラム単位内の実行文の文番号です。

入力並び

iolist は空の場合と、入力並びと DO 型並びの両方または一方を含む場合があります。入力並びは以下のいずれかです。

- 変数
- 部分列
- 配列

- 配列要素
- 記録
- 記録欄

添字なしで単に配列名だけを書くと、記憶領域に格納されている順に (左端の添字が他より速く増加する) 配列の全要素が指定されます。

DO 型並びの詳細については、112 ページの「DO 型並び」を参照してください。

変数群 READ

READ 文の第 3 と第 4 の形式は、指定された変数群の項目を読み取るために使用されます。*grname* は先に NAMELIST 文で定義されている変数群の名前です。

実行

実行順序は以下のとおりです。

1. **指定された装置にファイルが結合されます。**

   書式が指定されていればそれが確立されます。データの転送に先立ちファイルが適切に位置付けされます。

2. **入力並びが空でない場合、ファイルから対応する並びの項目へデータが転送されます。**

   入力並びがなくなるまで項目が順に処理されます。次の指定項目が決定され、読み取られた値がそれに転送されます。書式付き READ でのデータ編集は指定された書式に従って行われます。

3. **変数群 READ の第 3 と第 4 の形式では、指定された変数群の項目が変数群入力の規則に従って処理されます。**

4. **データ転送後、ファイルが適切に再び位置付けされます。**

5. ***ios* が指定されている場合、誤りが生じなければ *ios* はゼロに設定されます。**

   誤りまたはファイルの終了になった場合は正の値に設定されます。

6. ***s* が指定されている場合、ファイルが終了すると制御が *s* に移されます。**

7. ***s* が指定されている場合、誤りが生じると制御が *s* に移されます。**

READ には次の 2 つの形式があります。

```
READ f [, iolist ]

READ [ NML= ] grname
```

上記の 2 つの形式は、キーボードから読み取られることを除き、他と同じように動作します。

上記の形式には、次のような違いがあります。

- 入力並びがなくなると入力した行の次行の先頭にカーソルが移動します。空の入力並びの場合は、入力した行の次行の先頭にカーソルが移動します。
- 入力並びが満たされる前に、行の終わり、CR、NL のいずれかに到達した場合は次行へ入力が続行します。
- 入力並びが満たされる前にファイル終了 (Control-D) を受け取った場合は入力を中止し、入力並びの未処理の項目はそのままとなります。

u がファイルに結合されていない外部装置を指定した場合は暗黙の OPEN 操作が実行され、下記の例のオプションでファイルを開いた場合と同じ状態になります。

```
    OPEN( u, FILE='FORT.u', STATUS='OLD', ACCESS='SEQUENTIAL',
&             FORM=fmt )
```

次の点にも注意してください。

- *fmt* の値は、読み取りが書式付きか否かにより 'FORMATTED' または 'UNFORMATTED' となります。
- 添字なしで単に配列名だけを書くと、記憶領域に格納されている順 (左端の添字が他より速く増加する) 配列の全要素が指定されます。
- まだ書き込まれていない直接探査ファイルの記録を読み取ろうとすると、入力並び項目がすべて未定義になります。
- 記録番号は 1 から始まります。
- 変数群入力ができるのは順番探査ファイルだけです。

例

例 1 : 書式付き READ、入出力誤りのトラップ、ファイル終了、入出力状態

```
      READ( 1, 2, ERR=8, END=9, IOSTAT=N ) X, Y
      …
8     WRITE( *, * ) '1 での入出力エラー番号:', N
      STOP
9     WRITE( *, * ) '1 で EoF'
      RETURN
      END
```

例 2 : 直接、書式なし READ、入出力エラートラップ、入出力状態

```
      READ( 1, REC=3, IOSTAT=N, ERR=8 ) V
      …
4     CONTINUE
      RETURN
8     WRITE( *, * ) '1 での入出力エラー番号:', N
      END
```

例 3 : キーボードからの並びによる READ

```
      READ( *, * ) A, V
または
      READ *, A, V
```

例 4 : 内部ファイルからの書式付き READ

```
      CHARACTER CA*16 / 'abcdefghijklmnop' /, L*8, R*8
      READ( CA, 1 ) L, R
1     FORMAT( 2 A8 )
```

例 5 : 配列全体の読み取り

```
      DIMENSION V(5)
      READ( 3, '(5F4.1)') V
```

例 6 : 変数群 READ

```
CHARACTER SAMPLE*16
LOGICAL NEW*4
REAL DELTA*4
NAMELIST /G/ SAMPLE, NEW, DELTA
…
READ( 1, G )
または
READ( UNIT=1, NML=G )
または
READ( 1, NML=G )
```

REAL

REAL 文は定数名、変数、配列、関数、仮関数の型を実数に指定します。また、REAL 文で配列の次元と大きさを指定したり、初期値を与えることもできます。

REAL [*len[,]] *v*[* *len* [/*c*/]] [, *v* [* *len* [/*c*/]] …

パラメータ	説明
v	変数、定数名、配列、配列宣言子、関数、仮関数の名前
len	4、8、16 (SPARC のみ) のどれかで、定数名、変数、配列要素または関数の長さをバイト数で表したもの
c	この前にくる名前の定数の並び

説明

REAL、REAL*4、REAL*8、REAL*16 について説明します。

REAL

REAL W のように宣言すると、通常、変数 W はメモリーの REAL*4 型の要素で、実数と解釈されます。 サイズの指定は規格外になります。 ♦

REAL H などの宣言のデフォルトサイズを変更するには、オプション -dbl、-r8、または -xtypemap のいずれかを指定してコンパイルします。詳細については、第 2 章「データ型とデータ項目」を参照してください。

REAL*4 ♦

REAL*4 W のように宣言すると、変数 W は常にメモリーの REAL*4 型の要素で、単長の実数と解釈されます。

REAL*8 ♦

REAL*8 W のように宣言すると、変数 W は常にメモリーの REAL*8 型の要素で、倍長の実数と解釈されます。

REAL*16 ♦

(SPARC のみ) REAL*16 W のように宣言すると、変数 W は常にメモリー内の REAL*16 型の要素で、4 倍長の実数と解釈されます。

例

例 1 : 単純な実数変数 — 以下の宣言は、すべて等価です。

```
    REAL U, V(9)
    REAL*4 U, V(9)
    REAL U*4, V(9)*4
```

例 2 : 変数の初期化 (REAL*16 は SPARC のみ)

```
    REAL U/ 1.0 /, V/ 4.3 /, D*8/ 1.0 /, Q*16/ 4.5 /
```

例 3 : 実数の配列の寸法を指定

```
    REAL A(10,100), V(10)
    REAL X*4(10), Y(10)*4
```

例 4 : 配列の初期化

```
    REAL A(10,100) / 1000 * 0.0 /, B(2,2) /  1.0, 2.0, 3.0, 4.0 /
```

例 5 : 倍精度と 4 倍精度 (REAL*16 は SPARC のみ)

```
    REAL*8  R
    REAL*16 Q
    DOUBLE PRECISION D
```

D と R は倍精度です。Q は 4 倍精度です。

RECORD

RECORD ♦ 文は指定された構造体を持つ変数を定義したり、指定された構造体を持つ変数の配列を定義します。

RECORD /構造体名/ 記録の並び [,/構造体名/ 記録の並び]…

パラメータ	説明
構造体名	先に宣言されている構造体の名前
記録の並び	変数、配列、または配列宣言子

説明

構造体は記録のひな形です。構造体の名前は STRUCTURE 文にあり、STRUCTURE 文で一度定義され命名された構造体は、RECORD 文で使用することができます。

記録は変数または配列を一般化したものです。変数や配列が型を持つように、記録は構造体を持ちます。配列の要素はすべて同じ型でなければなりませんが、記録欄の型は異なっていてもかまいません。

RECORD 行は本来複数行に渡る文の集合の一部であり、RECORD 行と END RECORD 行のどちらも継続行はありません。6 桁目に空白以外の文字を入れたり、1 桁目に & を入れたりしないでください。

構造体、フィールド、記録については、49 ページの「構造体」を参照してください。

制限事項

- 各記録はメモリー内に別々に割り当てられます。
- 最初、記録の値は定義されていません。
- 記録、記録欄、記録配列、記録配列の要素は、引数または仮引数として使用することができます。記録を引数として渡す場合、その記録欄の型、順序、寸法は同じでなければなりません。呼び出し先と呼び出し元の記録の宣言は一致していなければなりません。
- 共用体の宣言においては、マップ欄はどのような順番でもかまいません。
- 記録欄は COMMON 文で指定できます。
- 記録と記録欄は DATA 文、EQUIVALENCE、NAMELIST、PARAMETER、AUTOMATIC、STATIC、SAVE の文では指定できません。記録と記録欄の初期化には STRUCTURE 文を使用します。詳細については、224 ページの「STRUCTURE」を参照してください。

例

例 1 : 指定した構造体の記録として複数の項目を宣言

```
    STRUCTURE /PRODUCT/
    INTEGER*4 ID
    CHARACTER*16 NAME
    CHARACTER*8 MODEL
    REAL*4 COST
    REAL*4 PRICE
    END STRUCTURE
    RECORD /PRODUCT/ CURRENT, PRIOR, NEXT, LINE(10)
    …
```

CURRENT、PRIOR、NEXT の各変数は PRODUCT という構造体を持つ記録です。LINE は PRODUCT という構造体を持つ記録を 10 個含む配列です。

例 2 : 記録の複数の欄を定義し、使用する

```
    STRUCTURE /PRODUCT/
    INTEGER*4     ID
    CHARACTER*16 NAME
    CHARACTER*8   MODEL
    REAL*4        COST
    REAL*4        PRICE
    END STRUCTURE
    RECORD /PRODUCT/  CURRENT, PRIOR, NEXT, LINE(10)
    CURRENT.ID = 82
    PRIOR.NAME = "CacheBoard"
    NEXT.PRICE = 1000.00
    LINE(2).MODEL = "96K"
    PRINT 1, CURRENT.ID, PRIOR.NAME, NEXT.PRICE, LINE(2).MODEL
1   FORMAT(1X I5/1X A16/1X F8.2/1X A8)
    END
```

上記のプログラムの出力は以下のようになります。

```
82
CacheBoard
1000.00
96K
```

RETURN

RETURN 文は呼び出し元のプログラム単位に制御を返します。

RETURN [*e*]

パラメータ	説明
e	INTEGER 型または REAL 型の式

説明

RETURN 文を実行すると関数またはサブルーチンの引用が終了します。

関数またはサブルーチンで END 文を実行すると RETURN 文を実行するのと同じ結果になります。♦

e という式は評価され、必要であれば整数に変換されます。*e* は使用する選択戻り文番号の順序を表します。選択戻り文番号は SUBROUTINE 文ではアスタリスク (またはアンパサンド) ♦ で指定されます。

e が指定されていない場合、また *e* が 1 より小さいか RETURN 文を含む SUBROUTINE 文で指定されたアスタリスクまたはアンパサンドの数より大きい場合、制御はサブルーチンを呼び出した CALL 文の次の文に戻ります。

e の値が 1 以上で SUBROUTINE 文で指定されたアスタリスク (またはアンパサンド) の数以下である場合、制御は *e* 番目の選択戻り文番号で識別される文に戻ります。
RETURN 文は関数の副プログラムまたはサブルーチンにしか使用できません。

例

例 1 : 標準的な戻り

```
CHARACTER*25 TEXT
TEXT = "Some kind of minor catastrophe"
…
CALL OOPS ( TEXT )
STOP
END
SUBROUTINE OOPS ( S )
CHARACTER S* 32
WRITE (*,*) S
RETURN
END
```

例 2 : 選択戻り

```
      CALL RANK ( N, *8, *9 )
      WRITE (*,*) 'OK - 正常戻り'
      STOP
8     WRITE (*,*) '非重要 - 最初の代替戻り'
      STOP
9     WRITE (*,*) '重要 - 2 番目の代替戻り'
      END
      SUBROUTINE RANK (N, *,*)
      IF ( N .EQ. 0 ) RETURN
      IF ( N .EQ. 1 ) RETURN 1
      RETURN 2
      END
```

REWIND

REWIND 文は指定された装置に結合されたファイルをその始点に位置付けます。

注 – テープ装置をまき戻す場合は、TOPEN() ルーチンを使用します。『Fortran ライブラリ・リファレンス』を参照してください。

```
REWIND u

REWIND ( [ UNIT=] u [, IOSTAT=ios ] [, ERR= s ])
```

パラメータ	説明
u	ファイルに結合された外部装置の装置識別子。順番探査または追加探査の際には *u* が結合されていること。
ios	入出力状態指定子。整変数または整数配列要素。
s	誤り指定子。*s* は REWIND 文のあるプログラム内の実行文の文番号であること。REWIND 文の実行中にエラーが生じた場合、プログラムの制御はこの文番号に移される。

説明

オプションはどのような順番で指定してもかまいません。

ファイルが結合されていない装置に REWIND 文を実行しても何も起こりません。

端末ファイルで REWIND 文を実行しても何も起こりません。

直接探査ファイル上での REWIND 文の使用は FORTRAN 77 規格では定義されておらず、その動作は予測できません。

例

例 1 : 単純な形式の装置指定子

```
    ENDFILE 3
    REWIND 3
    READ (3,'(I2)') I
    REWIND 3
    READ (3,'(I2)')I
```

例 2 : UNIT=*u* という形式の装置指定子とエラートラップを持つ REWIND 文

```
    INTEGER  CODE
    …
    REWIND (UNIT = 3)
    REWIND (UNIT = 3, IOSTAT = CODE, ERR = 100)
    …
100 WRITE (*,*) '巻き戻し中のエラー'
    STOP
```

SAVE

SAVE 文では、RETURN 文または END 文の実行後に副プログラム内の項目を保存することによって、未定義にならないようにします。

SAVE [*v* [, *v*] …]

パラメータ	説明
v	副プログラム内の配列、変数、(スラッシュで囲まれた) 共通ブロックの名前

説明

保存する変数は内部の静的領域に格納されます。すべての共通ブロックは、静的領域に割り当てられているため、すでに保存されています。したがって、SAVE 文で共通ブロック名を指定することは可能ですが、無視されます。

SAVE 文は主プログラムに指定する必要はなく、指定しても何の効果もありません。

SAVE 文に何も指定しない場合、ルーチン内の局所変数と配列がすべて保存されます。

局所変数と配列はデフォルトですでに静的であるため、SAVE する必要はありません。しかし、SAVE を使用すれば、特に RETURN 以外の方法で副プログラムを出るルーチンでは、移植性が保証されます。

制限事項

以下のものを SAVE 文に入れることはできません。

- 共通ブロック内の変数または配列
- 仮引数の名前
- 記録名
- 手続き名
- 自動変数または自動配列

例

例 : SAVE

```
    SUBROUTINE FFA(N)
    DIMENSION A(1000,1000), V(1000)
    SAVE A
    …
    RETURN
    END
```

文関数

文関数の定義文は関数に似た、1 つの文だけからなる宣言です。

fun ([*d* [, *d*] ...]) = *e*

パラメータ	説明
fun	定義する文関数の名前
d	文関数の仮引数
e	式。*e* の型は算術、論理、文字のどれでもよい

説明

文関数を引用すると定義された計算が挿入されます。

例 : 次の文は文関数です。

```
    ROOT( A, B, C ) = (-B + SQRT(B**2-4.0*A*C))/(2.0*A)
```

文関数の引数並びは文関数の引数の順序、個数、型を示します。

文関数を引用するには、引数を持つ文関数の名前を式のオペランドとして使用します。

文関数は以下のように実行されます。

1. **実引数が式である場合、実引数が評価されます。**
2. **実引数が対応する仮引数に関連付けられます。**
3. **文関数の本体である式 *e* が評価されます。**
4. 3. **の結果の型が関数名の型と異なる場合、結果の型が変換されます。**
5. **値が返されます。**

   これにより、関数を引用した式が結果の値を使用できるようになります。

制限事項

制限事項は次のとおりです。

- 文関数は宣言文の後で、その文関数が引用されるプログラム単位の最初の実行文の前になければなりません。
- 文関数は指定された箇所では実行されません。他の関数と同じように、式の中で関数の引用が実行されるときに実行されます。
- *fun* と *e* 間の一致に関する規則は、代入文に関する規則と同じです。*fun* と *e* の型は同じである必要はありませんが、同じでない場合には *e* が *fun* と同じ型に変換されます。
- 実引数は対応する仮引数と順序、個数、型が一致しなければなりません。
- 仮引数を構造体として定義した場合は、対応する実引数も同じ構造体として定義する必要があります。
- 仮引数を配列名または関数名にすることはできません。関数と同じ名前にすることもできません。
- 同じ引数を複数回指定することはできません。
- 文関数はその文関数を含むプログラム単位内でしか引用できません。
- 文関数の名前は実引数としては指定できません。また、`EXTERNAL` 文で指定することもできません。
- 引数の型は文関数が 1 つのプログラム単位であるかのように決められます。
- 文関数の引数の名前と同じ名前の局所変数があっても、その名前の引用は文関数の仮引数の引用として扱われます。局所変数の引用としては扱われません。
- 文字型の文関数の長さ指定や、その `CHARACTER` 型の仮引数は整定数式でなければなりません。
- 文関数は再帰的に呼び出すことはできません。

例

例 1 : 算術文関数

```
    PARAMETER ( PI=3.14159 )
    REAL RADIUS, VOLUME
    SPHERE ( R ) = 4.0 * PI * (R**3) / 3.0
    READ *, RADIUS
    VOLUME = SPHERE( RADIUS )
    …
```

例 2 : 論理文関数

```
    LOGICAL OKFILE
    INTEGER STATUS
    OKFILE ( I ) = I .LT. 1
    READ( *, *, IOSTAT=STATUS ) X, Y
    IF ( OK FILE(STATUS) ) CALL CALC ( X, Y, A )
    …
```

例 3 : 文字文関数

```
    CHARACTER FIRST*1, STR*16, S*1
    FIRST(S) = S(1:1)
    READ( *, * ) STR
    IF ( FIRST(STR) .LT. " " ) CALL CONTROL ( S, A )
    …
```

STATIC

STATIC ♦ 文は指定した項目が必ず静的領域に格納されるようにします。

STATIC *list*

パラメータ	説明
list	変数と配列の並び

説明

すべての局所変数と配列は、デフォルトでは静的であると分類されます。各データのコピーは 1 つだけ存在し、呼び出しと呼び出しの間でもその値は保持されます。STATIC 文または AUTOMATIC 文を使用して変数を明示的に静的変数または自動変数に定義することもできます。また型宣言文または IMPLICIT 文でも定義できます。

しかし、移植性を保証するために STATIC を使用することができます。特に、RETURN 以外の方法で副プログラムを出るルーチンでは STATIC 文が有効です。

さらに、次の点に注意してください。

- 引数と関数値は自動です。
- STATIC 文と型宣言文を組み合わせて STATIC 型宣言文にすることはできません。たとえば、STATIC REAL X という文は変数 X が STATIC であり REAL であると宣言しているのではなく、変数 REALX が STATIC であると宣言しています。

例

```
    STATIC A, B, C
    REAL P, D, Q
    STATIC P, D, Q
    IMPLICIT STATIC (X-Z)
```

STOP

STOP 文はプログラムの実行を終了します。

STOP [*str*]

パラメータ	説明
str	5 桁以下の文字列または文字定数

説明

プログラム終了時には引数 *str* が表示されます。

str が指定されない場合にはメッセージは表示されません。

例

例 1 : 整数

```
    stop 9
```

上記の文は次を表示します。

```
    STOP: 9
```

例 2 : 文字

```
    stop 'エラー'
```

上記の文は次を表示します。

```
    STOP: エラー
```

STRUCTURE

STRUCTURE ♦ 文はデータを構造体に編成します。

```
STRUCTURE   [/構造体名/]   [ 欄の並び ]
    欄宣言
    欄宣言
    …
END STRUCTURE
```

欄宣言は以下のうちのいずれかです。

- 部分構造体 (別の構造体宣言、またはすでに定義されている記録)

- 共用体宣言
- 型宣言 (初期値を含むこともできます)

説明

STRUCTURE 文は、記録を構成する欄の名前、型、サイズ、順序を指定することによって記録の形式を定義します。また、必要に応じて初期値を指定することもできます。

構造体は記録用のテンプレートです。構造体の名前は STRUCTURE 文で定義され、このように一度 STRUCTURE 文で定義されて名付けられた構造体は RECORD 文で使用することができます。

記録は変数または配列を一般化したものです。変数や配列が型を持つように、記録は構造体を持ちます。配列の要素はすべて同じ型でなければなりませんが、記録の欄の型は異なっていてもかまいません。

構造体、欄、および記録については、49 ページの「構造体」を参照してください。

制限事項

構造体の名前はスラッシュで囲みます。名前を省略できるのは入れ子にされた構造体だけです。

スラッシュがある場合は、名前もなければなりません。

欄の並びが指定できるのは入れ子にされた構造体内だけです。

欄宣言は最低 1 つはなければなりません。

各構造体名は構造体間で一意でなければなりません。ただし、構造体名を他の構造体内の欄の名前として使用したり、変数名として使用することはできます。

STRUCTURE 文と END STRUCTURE 文の間に入れることのできる文は欄宣言文と PARAMETER 文だけです。構造体宣言ブロック内の PARAMETER 文はその外側にある PARAMETER 文と同じです。

欄に関する制限事項

欄宣言が型宣言の場合、通常の Fortran 77 の型宣言文と同じ構文を使用します。ここでは以下のような規則と制限に従えば、f77 のすべての型が許されます。

- 寸法の指定が必要であれば、それを型宣言文の中に含めなければなりません。DIMENSION 文は欄名には影響しません。
- 欄名には %FILL という疑似名を指定することができます。%FILL はFortran 77 の他バージョンとの互換性を保つために提供されています。f77 では整列の問題が解消されているため、%FILL は必要ありません。%FILL は、1 つまたは複数の欄を特定のサブルーチンで引用できないようにするときに便利です。%FILL を指定した場合、指定された大きさと型の欄が提供されるだけで、それを引用することはできません。
- 欄名はすべて明示的に型宣言しなければなりません。STRUCTURE 宣言内の文には、IMPLICIT 文も暗黙の I、J、K、L、M、N の規則も適用されません。
- 欄宣言では整合配列や大きさ引き継ぎ配列を使用することはできません。また、長さが引数として渡される CHARACTER 宣言も使用することはできません。

構造体宣言では、欄のオフセット *n* はその前の欄のオフセットに前の欄の長さを足したもので、整列させるために調整されることもあります。

変数、配列、部分列、部分構造体、共用体である欄は初期化することができます。

例

例 1： 5 つの欄からなる構造体

```
STRUCTURE /PRODUCT/
INTEGER*4 ID/ 99 /
CHARACTER*16 NAME
CHARACTER*8 MODEL/ 'Z' /
REAL*4 COST
REAL*4 PRICE
END STRUCTURE
RECORD /PRODUCT/ CURRENT, PRIOR, NEXT, LINE(10)
```

上記の例では、5 つの欄 ID、NAME、MODEL、COST、PRICE からなる PRODUCT という名前の構造体が定義されています。変数 CURRENT、PRIOR、NEXT はそれぞれ PRODUCT 構造体を持つ記録で、LINE はこのような記録を 10 個含む配列です。これらの記録の ID には 99、MODEL には Z という初期値が設定されます。

例 2 : 2 つの欄からなる構造体

```
STRUCTURE /VARLENSTR/
INTEGER*4 NBYTES
CHARACTER A*25
END STRUCTURE
RECORD /VARLENSTR/ VLS
VLS.NBYTES = 0
```

SUBROUTINE

SUBROUTINE 文は指定したプログラム単位をサブルーチンと識別し、その引数を指定します。

SUBROUTINE *sub* [([*d* [, *d*] …])]

パラメータ	説明
sub	副プログラムの名前
d	変数名、配列名、記録名、仮手続き名、アスタリスク、アンパサンドのいずれか

説明

副プログラムの最初の文は SUBROUTINE 文でなければなりません。サブルーチンには、BLOCK DATA、FUNCTION、PROGRAM、および別の SUBROUTINE 文を書くことはできません。

sub はサブルーチン名で、大域名ですが、共通ブロック名や関数名といった他の大域名と同じであってはなりません。また、同じサブルーチン内の局所名と同じであってもなりません。

d は仮引数です。仮引数が複数あるときにはそれぞれをコンマで区切ります。*d* は以下のうちのどれかです。

- 変数名
- 配列名
- 仮手続き名

- 記録名
- アスタリスク (*) またはアンパサンド (&) ♦

仮引数はサブルーチンに対し局所的です。また、共通ブロック名以外として以下の文に現れることはできません。

- EQUIVALENCE
- PARAMETER
- SAVE
- STATIC
- AUTOMATIC
- INTRINSIC
- DATA
- COMMON

サブルーチンを引用する CALL 文の実引数は、SUBROUTINE 文の対応する仮引数の順序、数、型と一致しなければなりません。

仮引数の並びのアスタリスク (またはアンパサンド) は選択戻り文番号を示します。どの選択戻り文を取るかはサブルーチン内の RETURN 文で指定できます。

例

例 1 : 仮引数としての変数と配列

```
      SUBROUTINE SHR ( A, B )
      CHARACTER A*8
      REAL B(10,10)
      …
      RETURN
      END
```

例 2 : 規格に従った選択戻り

```
      PROGRAM TESTALT
      CALL RANK ( N, *8, *9 )
      WRITE (*,*)'OK - 正常戻り [n=0]'
      STOP
8     WRITE (*,*)'非重要 - 最初の代替戻り [n=1]'
      STOP
9     WRITE (*,*)'重要 - 2 番目の代替戻り [n=2]'
      END
      SUBROUTINE RANK ( N, *, * )
      IF ( N .EQ. 0 ) RETURN
      IF ( N .EQ. 1 ) RETURN 1
      RETURN 2
      END
```

この例の RETURN 1 文は 最初の選択戻り文番号 (最初の *) を引用し、RETURN 2 文は SUBROUTINE 文で指定された 2 番目の選択戻り文番号 (2 番目の *) を引用します。

TYPE

TYPE ♦ 文は標準出力に出力します。

TYPE *f* [, *iolist*]

TYPE *grname*

パラメータ	説明
f	書式識別子
iolist	変数、部分列、配列、記録の並び
grname	変数群の名前

説明

TYPE 文は、互換性を保つために提供されており、以下は同じ働きをします。

- PRINT *f* [,*iolist*]
- PRINT grname
- WRITE(*,*f*) [*iolist*]
- WRITE(*,*grname*)

例

例 1 : 書式付き出力と変数群出力

```
    INTEGER V(5)
    REAL X(9), Y
    NAMELIST /GNAM/ X, Y
    ...
    TYPE 1, V
1   FORMAT( 5 I3 )
    ...
    TYPE GNAM
    ...
```

型宣言文

型宣言文は並びの各項目のデータ型を指定します。また、型宣言文で配列の形状を指定したり、初期値を与えることもできます。

型 *v* [*/ clist /*] [,*v* [*/ clist /*]...

パラメータ	説明
型	以下のいずれかが入ります BYTE ♦ CHARACTER CHARACTER*n（n は 1 以上であること） CHARACTER*(*) COMPLEX COMPLEX*8 ♦ COMPLEX*16 ♦ COMPLEX*32 ♦（*SPARC* のみ） DOUBLE COMPLEX ♦ INTEGER INTEGER*2 ♦ INTEGER*4 ♦ INTEGER*8 ♦ LOGICAL LOGICAL*1 ♦ LOGICAL*2 ♦ LOGICAL*4 ♦ LOGICAL*8 ♦ REAL REAL*4 ♦ REAL*8 ♦ REAL*16 ♦（*SPARC* のみ） DOUBLE PRECISION
v	変数名、配列名、配列宣言子、定数の英字名、文関数、関数の副プログラム名のどれか
clist	定数の並び (*clist* の詳細については、DATA 文の説明を参照してください)

型の前に AUTOMATIC 文または STATIC 文が入ることもあります。

説明

型宣言文を使用して以下のことが行えます。

- デフォルト、または IMPLICIT 文によって設定された型を変更したり、その型と同じ型を宣言することができる。
- 配列の形状を指定することができる。また、型宣言文は組み込み関数の型を確定するのにも使用できる。
- データ型の長さを別の有効な長さに変えることができる。

DATA 文で行うように型宣言に定数の並び (*clist*) を指定すると、変数、配列、記録欄に初期値を設定することができます。♦

以下に型宣言による初期化の一般的な形式を示します。

型　変数名 / 定数 / ...

または

型 配列名 / 定数, ... /

または

型 配列名 / *r****定数** /

例を示します。

```
    CHARACTER LABEL*12 / 'Standard' /
    COMPLEX STRESSPT / ( 0.0, 1.0 ) /
    INTEGER COUNT / 99 /, Z / 1 /
    REAL PRICE / 0.0 /, COST / 0.0 /
    REAL LIST(8) / 0.0, 6*1.0, 0.0 /
```

データ型を初期化する場合、以下の制限事項があります。

- 単純な変数には定数を 1 つだけ指定できます。
- 配列の要素を 1 つでも初期化する場合は、残り全部も初期化しなければなりません。
- 整数の後にアスタリスク (*) と定数を続けて、反復因子として指定することができます (上記の例では 1.0 という値が 6 個、LIST の配列要素 2、3、4、5、6、7 にそれぞれ格納されます)。

- AUTOMATIC と宣言された変数、または配列を初期化することはできません。
- ポインタ基底付き変数またはポインタ基底付き配列を初期化することはできません。次に例を示します。

```
INTEGER Z / 4 /
POINTER ( x, Z )     警告が出され、初期化は行われない
```

上記の場合、コンパイラは警告メッセージを発し、Z は初期化されません。

変数または配列が初期化されない場合、その値は未定義になります。

初期化の文が共通ブロックの変数に指定されている場合、-ansi コンパイラフラグが設定されていると、警告メッセージが出力されます。

注 – オプション -dbl、-r8、-i2、または -xtypemap のいずれかを指定してコンパイルすると、明示的なサイズなしで型宣言された名前のデフォルトサイズを変更できます。第 2 章「データ型とデータ項目」を参照してください。

制限事項

1 つのプログラム単位内にある型宣言文で同じ英字名を複数回指定することはできません。

型宣言文はすべての実行文の前になければなりません。

例

例 : 型宣言文

```
INTEGER*2 I, J/0/
REAL*4 PI/3.141592654/,ARRAY(10)/5*0.0,5*1.0/
CHARACTER*10 NAME
CHARACTER*10 TITLE/'Heading'/
```

上記の例では、

- J は 0 に初期化されます。
- PI は 3.141592654 に初期化されます。

- ARRAY の最初の 5 つの要素は 0.0 に初期化されます。
- 次の 5 つの要素は 1.0 に初期化されます。
- TITLE は 'Heading' に初期化されます。

UNION と MAP

UNION ♦ 文は実行時にメモリーを共有する欄のグループを定義します。

共用体宣言の構文は以下のとおりです。

```
UNION
    MAP
                欄宣言
                欄宣言
                …
    ENDMAP
    MAP
                欄宣言
                欄宣言
                ノ
    END MAP

END UNION
```

説明

MAP 文は共用体内に代替の欄グループを定義します。実行時には 1 つのマップだけが共有の記憶場所と関連付けられます。マップ内の欄を引用すると、それまでのマップの欄グループは未定義になり、新しく引用された欄のマップ内の欄グループが記憶場所を引き継ぎます。

- UNION 宣言は、STRUCTURE 宣言の中だけで行うことができます。
- 共用体が使用するメモリー量は、その最大マップで使用するメモリー量です。
- UNION 宣言内の MAP 文は、どのような順番でもかまいません。

UNION 行は本来複数行の文の集合の一部であり、UNION 行または END UNION 行のどちらも継続行はありません。6 桁目に空白以外の文字を入れたり、1 桁目に & を入れたりしないでください。

MAP 宣言内の各欄宣言は以下のいずれかです。

- 構造体宣言
- 記録
- 共用体宣言
- データの型宣言

例

/STUDENT/ という構造体が NAME、CLASS、MAJOR または NAME、CLASS、CREDIT、GRAD_DATE のどちらかを含むように宣言しています。

```
STRUCTURE /STUDENT/
CHARACTER*32  NAME
INTEGER*2  CLASS
UNION
MAP
CHARACTER*16 MAJOR
END MAP
MAP
INTEGER*2  CREDITS
CHARACTER*8  GRAD_DATE
END MAP
END UNION
END STRUCTURE
RECORD /STUDENT/ PERSON
```

上記の例では、変数 PERSON は構造体 /STUDENT/ を持つため下記のことが言えます。

- PERSON.MAJOR は最初のマップの欄を引用し、PERSON.CREDITS は 2 番目のマップの欄を引用します。
- 最初のマップの欄の変数が初期化された後にプログラムが変数 PERSON.MAJOR を引用した場合、最初のマップが有効になり、2 番目のマップの各変数は未定義になります。

VIRTUAL

VIRTUAL ♦ 文は DIMENSION 文と同じように扱われます。

```
VIRTUAL  a ( d  ) [, a ( d ) ] …
```

パラメータ	説明
a	配列名
a(d)	配列の次元。1 個から 7 個までの宣言子をコンマで区切って指定する。

説明

VIRTUAL 文は形式も働きも DIMENSION 文と同じです。VIRTUAL 文は Fortran 77 の旧バージョンとの互換性を保つためにあります。

例

```
VIRTUAL M(4,4), V(1000)
…
END
```

VOLATILE

VOLATILE ♦ 文は特定の項目に対して最適化が行われないようにします。

```
VOLATILE  nlist
```

パラメータ	説明
nlist	変数、配列、共通ブロックの並び

説明

VOLATILE 文は指定された並びの項目に対して最適化が行われないようにします。VOLATILE 文に動作が左右されるプログラムは、普通は移植できません。

例

例： VOLATILE ♦

```
PROGRAM FFT
INTEGER NODE*2, NSTEPS*2
REAL DELTA, MAT(10,10), V(1000), X, Z
COMMON /INI/ NODE, DELTA, V
…
VOLATILE V, Z, MAT, /INI/
…
EQUIVALENCE ( X, V )
…
```

上記の例では、配列 V、変数 Z、共通ブロック /INI/ は VOLATILE であると明示的に指定されます。変数 X は、EQUIVALENCE を通して VOLATILE であると指定されます。

WRITE

WRITE 文は出力並びからファイルにデータを書き込みます。

注 – テープ入出力の場合、TOPEN() ルーチンを使用します。

WRITE([UNIT=] *u* [, [FMT=] *f*] [, IOSTAT=*ios*] [, REC=*rn*] [, ERR=*s*]) *iolist*

WRITE([UNIT=] *u*, [NML=] *grname* [, IOSTAT= *ios*] [, ERR= *s*])

パラメータ	説明
u	ファイルに結合された装置の装置識別子
f	書式識別子
ios	入出力状態識別子
rn	記録番号

s	誤り指定子 (文番号)
iolist	変数の並び
grname	変数群の名前

オプションはどのような順番で指定してもかまいません。

REC=*rn* という形式の代わりに次のように書くこともできます。♦

```
WRITE( u ' rn ... )  iolist  ♦
```

242 ページの「例」の例 3 は、この形式の例も兼ねています。

説明

テープ上のファイルに対しては、TOPEN() ルーチンを使用する方が確実です。

装置識別子

u は外部装置識別子または内部ファイル識別子です。

外部装置識別子は次のどちらかです。

- 負でない整数式。
- 標準出力を指定するアスタリスク。通常はコンソールに結合されている。

装置指定子から 省略可能な UNIT= を省略する場合、指定子の並びの中では *u* を最初の項目に指定しなければなりません。

書式識別子

f は書式識別子で、次のものがあります。

- アスタリスク (*)。並びによる入出力を示します。詳細については、297 ページの「並びによる入出力」を参照してください。
- 同じプログラム単位内にある FORMAT 文の文番号。
- 同じプログラム単位内にある FORMAT 文の文番号が割り当てられている整変数の名前。

- 書式文字列を指定する文字式または整数の配列。これは実行時書式または変数書式と呼ばれます。整数の配列は規格外です。♦

実行時に評価される書式の詳細については、290 ページの「実行時書式」を参照してください。

書式識別子から省略可能な FMT= を省略する場合、*f* は書式付き WRITE 文の 2 番目の引数に指定しなければなりません。他の場合に *f* を指定することはできません。

f は直接探査のときは、アスタリスクにすることはできません。

f は内部ファイルのとき、アスタリスクにすることができます。♦

ファイルが書式付き入出力に結合されている場合、書式なしのデータは転送できません。逆に、ファイルが書式なし入出力に結合されている場合は、書式付きのデータは転送できません。

入出力状態指定子

ios は整変数、整数配列要素、整数の記録欄のいずれかでなければなりません。

記録番号

rn は正の整数式でなければなりません。この引数は直接探査ファイルにだけ使用できます。*rn* は内部ファイルに対しても指定できます。♦

エラー指定子

s は WRITE 文があるプログラム単位内の実行文の文番号でなければなりません。

出力並び

iolist には、何も指定しないことも、出力並びの項目と DO 型並びのどちらかまたは両方を指定することもできます。出力並びの項目には以下のものがあります。

- 変数
- 部分列
- 配列
- 配列要素
- 記録

- 記録欄
- その他の式

添字が指定されていない単純な配列名は、配列のすべての要素をメモリーに格納されている順に指定します。このとき添字は左の方が他より速く増加します。

DO 型並びの詳細については、112 ページの「DO 型並び」を参照してください。

出力並び項目が連結演算子を使用した文字式である場合、そのオペランドの長さ指定子にアスタリスク (*) を指定することもできます。これは規格外です。♦

出力並びにある関数は、入出力文を実行するものであってはなりません。

変数群を指定した WRITE

WRITE 文のもう 1 つの形式は、指定された変数群の項目を出力するために使用されます。この *grname* は NAMELIST 文で、すでに定義された変数群の名前です。

実行

この形式は以下の手順で実行されます。

1. **指定された装置に結合するファイルを決めます。**

   書式が指定されている場合はその書式が設定されます。ファイルはデータ転送前に適切な場所に位置付けられます。

2. **出力並びが空でない場合、データは出力並びからファイルに転送されます。**

   書式が指定されている場合は書式に合わせてデータが編集されます。

3. **変数群が指定された WRITE の 2 つ目の構文では、データは変数群出力に関する規則に従って指定された変数群の項目から転送されます。**

4. **ファイルはデータ転送後、再び適切な場所に位置付けられます。**

5. ***ios* が指定されている場合、エラーが生じなければ *ios* はゼロに設定されます。エラーが生じれば、*ios* は正の値に設定されます。**

6. ***s* が指定されている場合にエラーが生じると制御は *s* に渡されます。**

制限事項

次の制限事項があります。

- 例外ハンドラからの出力は予測できません。

ユーザー自身の例外ハンドラを作成する場合、そのハンドラから Fortran 77 出力を行わないでください。どうしても必要な場合は、出力直後に ABORT を呼び出します。こうすることによって、システムが凍結する危険を減らすことができます。例外ハンドラからの Fortran 77 入出力は再帰的な入出力です。次の段落を参照してください。

- 再帰的な入出力の動作は信頼できません。

入出力リスト中に関数をリストして、その関数が入出力を行う場合、実行時に実行が凍結するか、その他にも予測できない問題が発生する可能性があります。この危険は並列化に関わらず存在します。

例 : 再帰的な入出力は時として失敗する。

```
PRINT *, x, f(x)        f() が入出力を行なうため、使用できない。
END
FUNCTION F(X)
WRITE(*,*) X
RETURN
END
```

注釈

u がファイルに結合されていない外部装置を指定する場合、ファイルを開くための OPEN 操作が暗黙的に実行されます。この操作は以下のようにオプションを指定してファイルを開くのと同等です。

```
OPEN( u, FILE ='FORT.u', STATUS ='UNKNOWN',
& ACCESS='SEQUENTIAL', FORM= fmt )
```

fmt の値は書式付きの場合は 'FORMATTED' で、指定なしの場合は 'UNFORMATTED' です。

添字が指定されていない単純な配列名は、配列のすべての要素をメモリーに格納されている順に指定します。このとき添字は左の方が他より速く増加します。

直接探査ファイルの記録番号は 1 から始まります。

変数群出力は、順番探査ファイルでのみ行うことができます。

例

例 1 : 書式付き WRITE、入出力エラートラップと入出力状態あり

```
    WRITE( 1, 2, ERR=8, IOSTAT=N ) X, Y
    RETURN
    …
8   WRITE( *, * ) '1 における入出力エラー番号: ', N
    STOP
    END
```

例 2 : 直接、書式なし WRITE、入出力エラートラップと入出力状態あり

```
    …
    WRITE( 1, REC=3, IOSTAT=N, ERR=8 ) V
    …
4   CONTINUE
    RETURN
8   WRITE( *, * ) '1 における入出力エラー番号: ', N
    END
```

例 3 : 直接、代替構文 (上記の例と同等) ♦

```
    …
    WRITE( 1 ' 3, IOSTAT=N, ERR=8 ) V ♦
    …
4   CONTINUE
    RETURN
8   WRITE( *, * ) '1 における入出力エラー番号: ', N
    END
```

例 4 : 画面への並びによる WRITE

```
    WRITE( *, * ) A, V
または
    PRINT *, A, V
```

例 5 : 内部ファイルへの書式付き WRITE

```
    CHARACTER CA*16, L*8 /'abcdefgh'/, R*8 /'ijklmnop'/
    WRITE( CA, 1 ) L, R
1   FORMAT( 2 A8 )
```

例 6 : 配列全体の書き込み

```
    DIMENSION V(5)
    WRITE( 3, '(5F4.1)') V
```

例 7 : 変数群出力を指定した WRITE

```
    CHARACTER SAMPLE*16
    LOGICAL NEW*4
    REAL DELTA*4
    NAMELIST /G/ SAMPLE, NEW, DELTA
    …
    WRITE( 1, G )
または
    WRITE( UNIT=1, NML=G )
または
    WRITE( 1, NML=G )
```

第5章

入出力

この章では、Fortran 入出力の一般的な概念について説明します。また、さまざまな入出力の種類についても説明しています。入出力については、『Fortran プログラミングガイド』も参照してください。

FORTRAN 77 入出力の基本概念

UNIX オペレーティングシステムにもとづいたオペレーティングシステムは、Fortran 77 ほど記録処理向きではありません。UNIX オペレーティングシステムは、ファイルを記録の集まりとしてではなく文字の列として扱います。Fortran 77 実行時システムは、実行時にファイル書式と探査モードをチェックします。また、Fortran 77 ライブラリと標準入出力ライブラリを含むファイル機能を提供します。

論理ユニット

プログラムが 1 度に開くことができる論理ユニットの最大数は、Fortran 77 のデフォルトでは 64 です。Solaris では、この制限は 256 です。Fortran 77 プログラムは、`setrlim()` 関数を呼び出すことによって、この制限を 65 以上に増加することができます。`setrlim`(2) のマニュアルページを参照してください。`csh` を実行している場合は、`limit` または `unlimit` コマンドでも同じことができます。*csh*(1) を参照してください。

標準論理ユニット 0、5、および 6 は、それぞれ 標準エラー、標準入力、標準出力として事前に結合されています。これらの名前は、実際のファイル名ではないので各装置を開くためには使用できません。`INQUIRE` 文はこれらの名前を戻さず、上記の装置には、実ファイルに対して開かれていないかぎり名前がないことを示します。しかし `OPEN` 文を使用してこれらの装置を再定義することができます。

以前に結合された論理ユニットのデフォルトのファイル名でファイルを開きたいときは、その装置をまず閉じてください。標準装置を再定義すると通常のコンソール入出力が損なわれることがあります。代替方法はシェルによる切り替えを使用して、上記の装置を外部的に再定義することです。

デフォルト空白制御や標準入出力ファイルの書式を再定義する場合は、OPEN 文で装置番号を指定し、ファイル名は指定しないでください。また希望する空白制御の種類をオプションで設定してください。

入出力エラー

入出力処理中にエラーが検出されると、代わりとなる動作がプログラムの中で特に与えられていなければ、プログラムは処理を異常終了します。入出力文に ERR= (および IOSTAT=) 項目を含めて、エラーを発見したときにとる代替分岐先 (および特定のエラーコードを戻すこと) を指定することができます。READ 文では、ファイルの最後に達した所で分岐するために END=*n* を含めることができます。ファイル位置と入出力並び項目の値は、エラーの後では未定義です。END= は EOF と誤り条件の両方をとらえます。ERR= は誤り条件だけをとらえます。

ユーザーのプログラムが入出力エラーを捕捉しない場合、処理を異常終了する前に、論理ユニット、入出力状態、および大括弧 ([]) で囲まれたエラー番号を含むエラーメッセージが stderr に書き込まれます。処理を異常終了させるシグナルは IOT です。

1000 未満のエラー番号は、オペレーティングシステムのエラーを示します。*intro*(2) を参照してください。1000 以上のエラー番号は、入出力ライブラリに原因があります。

外部入出力では、バックスペースが可能なファイルからの読み取り中にエラーが起きた場合、現在の記録の一部が表示されます。内部入出力では、文字列中の現在位置に垂直バー (|) を付けて文字列の一部が出力されます。

一般的制限事項

入出力並びでは、実行された際に入出力文の実行を伴うような関数を引用しないでください。例を次に示します。

```
WRITE( 1, 10)  Y, A + 2.0 * F(X)    !  F() が入出力を行うのは間違い
```

入出力の種類

f77 でサポートされている入出力の種類には、書式付き、書式なし、バイナリ ♦、並び、および NAMELIST があります。

ファイルへの探査には順番と直接という 2 つのモードがあります。ファイルを開くと、探査モードは順番または直接に設定されます。ユーザーが明示的に設定しなければ、デフォルトで順番に設定されます。

ファイルには外部と内部の 2 つの型があります。外部ファイルは、たとえばディスクやテープのような物理的な周辺装置上に常駐します。内部ファイルは主記憶装置内にあります。内部ファイルは文字型で、変数、部分列、配列、配列要素または構造体記録の欄のいずれかです。内部ファイルは周辺装置のように読み取ったり、書き込んだりすることができます。

入出力の組み合わせ

外部ファイルの入出力組み合わせを以下の表に示します。

許可される組み合わせ	許可されない組み合わせ
順番探査、書式なし	直接探査、並びによる入出力
順番探査、書式付き	直接探査、変数群入出力
順番探査、並びによる	内部ファイルの変数群入出力
順番探査、変数群	書式なし、内部入出力
直接探査、書式なし	バイナリ、直接探査 ♦
直接探査、書式付き	バイナリ、書式付き ♦
バイナリ、順番探査、書式なし ♦	

以下の表は入出力書式、探査モード、および物理ファイルの型の組み合わせを示します。

表 5-1 f77 入出力の要約

入出力の種類		探査モード	
書式	**ファイルの型**	**順番**	**直接**
書式付き	内部	ファイルは文字変数、部分列、配列、または配列要素。♦	ファイルは文字の配列である。各記録は 1 つの配列要素である。
	外部	同一長または可変長の書式付き記録	書式付き記録だけ。すべて同一長。
書式なし	内部	(許可されない)	(許可されない)
	外部	書式なし記録だけを含む。	READ: 一度に 1 つの論理記録を獲得する。 WRITE: 記録の満たされない部分は未定義。
並び	内部	READ: EOF まで、または入出力並びが満たされるまで読み取る。 WRITE: 入出力並びが満たされるまで書き込む。♦	(許可されない)
	外部	変数の型と要素のサイズにもとづいた標準書式を使用する。 空白またはコンマは分離文字である。任意の桁数。	(許可されない)
変数群	内部	(許可されない)	(許可されない)
	外部	READ: 桁 2-80 の中に $*groupname* を見つけるまで記録を読み取る。その後記録を読み取ってそのグループ内の名前を探し出し、それらの変数にデータを格納する。$ またはファイル終了マークで読み取りを停止する。 WRITE: グループ名と値を持つ各変数名を表している記録を書き込む。	(許可されない)

並びによる内部ファイルの書き込みは避けてください。行数と行毎の項目数は項目の値で変わります。

プリントファイル

OPEN 文の中で規格外の FORM='PRINT' を使用するとプリントファイルが得られます。♦

```
    OPEN ( ..., FORM='PRINT', ... )
```

この指定子は、順番探査ファイルに対してだけ有効です。

定義

プリントファイルには以下の機能があります。

- 書式付き出力の場合、その論理ユニットに対して垂直書式制御が可能です。
  - 桁 1 は出力されません。
  - 桁 1 が空白、0、または 1 のときは、それぞれ 1 行、2 行またはページの先頭へ行送りします。
  - 桁 1 が + のときは、制御処理によって前の行の先頭へ戻るような制御シーケンスに置き換えられます。
- 並びによる出力の場合、その論理ユニットに対して以下のものが得られます。桁 1 は出力されません。

一般に FORM='PRINT' でファイルを開くと、並びによる出力は桁 1 に Fortran 77 規格の空白を与えず、FORM='PRINT' 以外で開いた場合はその空白を与えます。
FORM='PRINT' は、1 回の呼び出しにつき 1 ファイルに対応します。

FORM='PRINT' で開いたファイルの内容は、FORM='FORMATTED'で開き、出力フィルタ asa でフィルタされたものと同じになります。

-oldldo オプション (旧の並びによる出力) を設定してコンパイルすると、プログラムによって書き込まれるすべてのファイルは、桁 1 に FORTRAN 77 規格の空白を与えずに並びによる出力を行いますが、それ以外のときのすべてのファイルは、空白を与えられます。
-oldldo オプションは大域的です。

INQUIRE 文

INQUIRE 文は、'PRINT' ファイルとして開かれた論理ユニットに対しては、FORM 変数の中で 'PRINT' を戻します。INQUIRE 文は、開かれていないファイルの装置番号に対しては -1 を返します。

OPEN の特殊用途

論理ユニットがすでに開かれていると、BLANK オプションを使用した OPEN 文は何も実行せず、そのオプションの再定義だけを行います。

規格外の拡張として、論理ユニットがすでに開かれていると、FORM='PRINT' と FORM='FORMATTED' を切り換えるために FORM オプションを指定した OPEN 文は、何も実行せず、そのオプションの再定義だけを行います。♦

これら OPEN 文の書式には、ファイル名を含める必要はありません。さらに UNIT が標準入力、標準出力、または標準エラーを引用しているときは、ファイル名を含めてはなりません。

OPEN 文を使用して装置を結合したときにファイル名パラメータを使用しないと、デフォルトファイル名 fort.*nn* が与えられます。ここで *nn* は装置番号です。したがって、標準出力をプリントファイルとして再定義するには次の書式を使用してください。

```
    OPEN( UNIT=6, FORM='PRINT')
```

一時 (スクラッチ) ファイル

一時 (スクラッチ) ファイルを作成すると、一般に実行後に消滅します。

例: 一時ファイルの作成

```
    OPEN( UNIT=7, STATUS='SCRATCH' )
```

実行終了後に一時ファイルが消失するのを防ぐには、STATUS='KEEP' を設定して CLOSE 文を実行しなければなりません (一時ファイル以外のファイルでは、KEEP がデフォルトです)。

例: 後で探査したい一時ファイルを閉じる

```
    CLOSE( UNIT=7, STATUS='KEEP' )
```

後で再び開く場合は、INQUIRE 文を使用して一時ファイルの実名を取得することを忘れないでください。

IOINIT による入出力初期設定の変更

従来の Fortran 77 環境は、通常すべての論理ユニットについてキャリッジ制御を仮定していました。その環境では通常は入力時の空白スペースをゼロと解釈し、実行時に論理ユニットに大域的ファイル名を結合することもよくありました。ルーチン IOINIT(3F) を呼び出して、以下のような入出力制御パラメータを指定することができます。

- すべての書式付きファイルのキャリッジ制御を認識します。
- 入力ファイルの後続の空白と埋め込みの空白を無視します。
- ファイルを開く時に、ファイルの最初または最後に位置付けます。
- 指定されたパターンのファイル名を事前に論理ユニットに結合します。

例: IOINIT と論理ユニットの事前結合

```
    CALL IOINIT ( .TRUE., .FALSE., .FALSE., 'FORT', .FALSE.)
```

ユーザーのプログラムが上記のように呼び出しを行うと、Fortran 77 実行時システムは環境を調べて形式 FORT*nn* の名前を探してから、対応する論理ユニットを順番書式付き入出力として開きます。

この例では、次のようにプログラムが装置 7 を開いたとします。

```
    OPEN( UNIT=07, FORM='FORMATTED' )
```

Fortran 77 実行時システムが環境を調べて FORT07 ファイルを探し、それを装置 7 に結合します。

一般に名前は *PREFIXnn* の形式でなければなりません。個々の *PREFIX* は、IOINIT への呼び出しの中で指定されます。*nn* は開く論理ユニットです。9 以下の装置番号は前に '0' を書かなければなりません。詳細は、IOINIT(3F) および『Fortran ライブラリ・リファレンス』を参照してください。

例: 外部ファイル ini1.inp と ini1.out を装置 1 と 2 に結合する

sh:

```
demo$ TST01=ini1.inp
demo$ TST02=ini1.out
demo$ export TST01 TST02
```

csh:

```
demo% setenv TST01 ini1.inp
demo% setenv TST02 ini1.out
```

例: ファイル ini1.inp と ini1.outを装置 1 と 2 に結合します

```
demo% cat ini1.f
    CHARACTER PRFX*8
    LOGICAL CCTL, BZRO, APND, VRBOSE
    DATA CCTL, BZRO, APND, PRFX, VRBOSE
&   /.TRUE., .FALSE., .FALSE., 'TST', .FALSE. /
C
    CALL IOINIT( CCTL, BZRO, APND, PRFX, VRBOSE )
    READ( 1, *) I, B, N
    WRITE( *, *) 'I = ', I, ' B = ', B, ' N = ', N
    WRITE( 2, *) I, B, N
    END
demo% cat $TST01
 12 3.14159012 6
demo% f77 ini1.f
ini1.f:
 MAIN:
demo% a.out
 I =   12 B =     3.14159 N =   6
demo% cat $TST02
   12    3.14159  6
```

IOINIT は作成されたときに、多くのプログラムにとって十分であることを立証しなければなりません。Fortran で書かれているため同様のユーザー作成ルーチンの例として役立ちます。IOINIT のソースコードは次から取り出すことができます。

/opt/SUNWspro/<release>/src/ioinit.f

*<release>*パスは、コンパイラのリリースごとに変わります。

直接探査

直接探査ファイルは、記録番号を指定して読み取り、または書き込みを行う多数の記録で構成されています。直接探査はランダム探査とも呼ばれます。

直接探査の特長および、制限事項は次のとおりです。

- 記録はすべて同じ長さでなければなりません。
- 通常、記録はすべて同じ型です。
- 直接探査外部ファイルの論理記録は、ファイルが開かれたときに指定された長さのバイト数を持つ文字列です。
- READ 文または WRITE 文は、もとになる記録サイズ定義よりも長い論理記録を指定してはなりません。
- 短い論理記録は許可されます。
  - 書式なし直接書き込みでは、記録の満たされない部分を未定義のまま残します。
  - 書式付き直接書き込みは、満たされない記録を空白にして転送します。
- 直接書式なし入出力の使用に際しては、ユーザーのプログラムが読み取ることを予期している値の数に十分注意しなければなりません。
- 直接探査 READ 文および WRITE 文は、引数 REC=*n* を持ちます。この引数は、読み取る記録または書き込む記録の記録番号を与えます (代替の規格外の書式は '*n* です)。

書式なし入出力

例: 直接探査、書式なし

```
    OPEN( 2, FILE='data.db', ACCESS='DIRECT', RECL=20,
&   FORM='UNFORMATTED', ERR=90 )
    READ( 2, REC=13, ERR=30 ) X, Y
    READ( 2 ' 13, ERR=30 ) X, Y     別の書式 ♦
```

このコードは直接探査、記録長 20 文字の書式なし入出力のファイルを開き、13 番目の記録をそのまま読み取ります。

書式付き入出力

例: 直接探査、書式付き

```
    OPEN( 2, FILE='inven.db', ACCESS='DIRECT', RECL=20,
&   FORM='FORMATTED', ERR=90 )
    READ( 2, FMT='(I10,F10.3)', REC=13, ERR=30 ) A, B
```

このコードは直接探査、記録長 20 文字の書式付き入出力のファイルを開き、13 番目の記録を読み取って、それを `(I10,F10.3)` 書式に従って変換します。

内部ファイル

内部ファイルは、定数、変数、部分列、配列、配列の要素、または構造体記録欄のような文字列オブジェクトです。これらはすべて文字型です。変数または部分列の場合には、ファイル中には 1 つの記録しかありません。配列の場合には、各配列要素が記録です。

順番書式付き入出力

FORTRAN 77 規格では、内部ファイルは順番書式付き入出力だけを含みます (ここでは入出力という用語は正確ではありませんが、内部ファイルは READ 文と WRITE 文を使用して処理されます)。内部ファイルは、装置番号の代わりに文字オブジェクトの名前を与えて使用します。順番探査内部ファイルからの読み取りは、常に内部ファイルの最初から開始されます。書き込みのときも同様です。

例：順番書式付き READ

```
    CHARACTER X*80
    READ( 5, '(A)' ) X
    READ( X, '(I3,I4)' ) N1, N2
```

これは出力行イメージを X に読み込んで、その後 X から 2 個の整数を読み取ります。

直接探査入出力

f77 は、直接入出力を内部ファイルにまで拡張します。♦

これはファイル内の記録数を変更できないことを除いて、外部ファイルの直接入出力に類似しています。この場合 1 記録は文字列の配列の単一要素です。

例: 内部ファイル LINE の 3 番目の記録の直接探査読み取り

```
demo% cat intern.f
    CHARACTER LINE(3)*14
    DATA LINE(1) / ' 81 81 ' /
    DATA LINE(2) / ' 82 82 ' /
    DATA LINE(3) / ' 83 83 ' /
    READ ( LINE, FMT='(2I4)', REC=3 ) M, N
    PRINT *, M, N
    END
demo% f77 -silent intern.f
demo% a.out
  83 83
demo%
```

書式付き入出力

書式付き入出力では、以下のことが行われます。

- 並びの項目は並びで現れる順に処理されます。
- どの並びの項目も次の項目が開始される前に完全に処理されます。
- 各順番探査は、 1 つまたは複数の論理記録を読み取るか、または書き込みます。

入力探査

一般に、書式付き READ 文は以下のことを行います。

- 外部記録 (または内部ファイル) から文字データを読み取ります。
- 関連付けられた書式の命令に従って、並びの項目を文字形式からバイナリ形式に変換します。
- 変換したデータを並びの各項目に対応する内部記憶領域に格納します。

例: 書式付き READ

```
    READ( 6, 10 ) A, B
10  FORMAT( F8.3, F6.2 )
```

出力探査

一般に、書式付き WRITE 文は以下のことを行います。

- 並びで指定された各並びの項目に対応する内部記憶領域からデータを獲得します。
- 関連付けられた書式の命令に従って、項目をバイナリ形式から文字形式に変換します。
- 項目を外部記録または内部ファイルに転送します。
- 書式付き出力記録は改行 (newline) で終わります。

例: 書式付き WRITE

```
    REAL  A / 1.0 /,  B / 9.0 /
    WRITE( 6, 10 ) A, B
10  FORMAT( F8.3, F6.2 )
```

書式付き WRITE 文の場合、論理記録長は実行時に入力変数、または出力変数の並び (入出力並び) と相互に作用する FORMAT 文によって決定されます。

書式付き WRITE 文の場合、外部データ表現が指定された欄の幅に対して大きすぎるときは、指定された欄はアスタリスク (*) で満たされます。

書式付き READ 文の場合、並びの項目数がデータの欄数より少ないとき、余分の欄は無視されます。

書式指定子

表 5-2　書式指定子

用途	FORTRAN 77	`f77` **の拡張**
空白制御	*BN*, *BZ*	*B*
キャリッジ制御	*/*, *space*, *0*, *1*	*$*
文字編集	*n*H, A*w* , `'aaa'`	*"aaa"*, A
浮動小数点編集	D*w.dEe*, E*w.dEe*, F*w.dEe*, G*w.dEe*	E*w.d.e*, D*w.d.e*, G*w.d.e*
16 進編集		Z*w.m*
整数編集	I*w.m*	
論理編集	L*w*	
8 進編集		O*w.m*
位置制御	*n*X, T*n*, TL*n*, TR*n*	*n*T, T, X
指数制御		*n*R, R
残存文字		Q
スケール制御	*n*P	P
符号制御	S, SP, SS	SU
書式終了	:	
変数式		< *e* >

FORMAT 文の中、および入出力ライブラリルーチンへのすべての英字引数の中の書式指定子には、小文字だけでなく大文字も使用できます。

w、*m*、*d*、*e* パラメータ (G*w*.*d*E*e* の場合)

w、*m*、*d*、*e* パラメータの定義は次のとおりです。

- *w* と *e* は 0 以外の符号なし整定数です。
- *d* と *m* は 符号なし整定数です。
- *w* は欄が *w* 桁を占めることを指定します。
- *m* は、*m* 桁になるように先行のゼロを挿入することを指定します。

- *d* は小数点以下が何桁かを指定します。
- *e* は指数欄の幅を指定します。

w、*d*、*e* のデフォルト値

欄指示子 *w*、*d*、または *e* なしで欄記述子 A、D、E、F、G、I、L、O、または Z を書くことができます。♦ これらが指定されないままのとき、入出力並び要素のデータの型にもとづいて適切なデフォルトが使用されます。表 5-3 を参照してください。

w、*d*、または *e* を使用する一般的な書式欄記述子を次に示します。

A*w*, I*w*, L*w*, O*w*, Z*w*, D*w*.*d*, E*w*.*d*, G*w*.*d*, E*w*.*d*E*e*, G*w*.*d*E*e*

例: INTEGER*2 に対してデフォルト *w*=7 の場合 (10進数 161=16進数 A1)

```
    INTEGER*2 M
    M = 161
    WRITE ( *, 8 ) M
8   FORMAT ( Z )
    END
```

上記の例は、次のように表示されます。

```
demo% f77 def1.f
def1.f:
 MAIN:
demo% a.out
ΔΔΔΔΔa1
demo%
```

Δ は、空白文字の位置を示します。*w*、*d*、および *e* のデフォルト値を以下にまとめます。

表 5-3　書式欄記述子のデフォルト *w*、*d*、*e* の値

欄記述子	並びの要素	*w*	*d*	*e*
I,O,Z	BYTE	7	-	-
I,O,Z	INTEGER*2, LOGICAL*2	7	-	-
I,O,Z	INTEGER*4, LOGICAL*4	12	-	-
O,Z	REAL*4	12	-	-

表 5-3　書式欄記述子のデフォルト *w*、*d*、*e* の値

欄記述子	並びの要素	*w*	*d*	*e*
O,Z	REAL*8	23	-	-
O,Z	REAL*16, COMPLEX*32	44	-	-
L	LOGICAL	2	-	-
F,E,D,G	REAL, COMPLEX*8	15	7	2
F,E,D,G	REAL*8, COMPLEX*16	25	16	2
F,E,D,G	REAL*16, COMPLEX*32	42	33	3
A	LOGICAL*1	1	-	-
A	LOGICAL*2, INTEGER*2	2	-	-
A	LOGICAL*4, INTEGER*4	4	-	-
A	REAL*4, COMPLEX*8	4	-	-
A	REAL*8, COMPLEX*16	8	-	-
A	REAL*16, COMPLEX*32	16	-	-
A	CHARACTER*n	*n*	-	-

複素数項目の *w* の値は各実数成分に対する値です。文字データを持つ A 記述子のデフォルトは、対応する入出力並びの要素の宣言された長さです。REAL*16 と COMPLEX*32 は *SPARC* のみです。

アポストロフィ編集 ('*aaa*')

アポストロフィ編集指定子は文字定数の書式を使用します。このため編集指定子自身に囲まれた文字 (空白を含みます) が書き込まれます。アポストロフィ編集指定子を入力に使用してはなりません。欄の幅は区切り用のアポストロフィの間に含まれる文字数です (区切り用のアポストロフィは含みません)。欄の中にあって間に空白がない 2 個の連続するアポストロフィは、1 個のアポストロフィとして数えられます。同様の方法で引用符も使用できます。

例: apos.f、アポストロフィ編集 (等価な 2 つの方法)

```
      WRITE( *, 1 )
1     FORMAT( 'これはアポストロフィ''です。')
      WRITE( *, 2 )
2     FORMAT( "これはアポストロフィ'です。")
      END
```

上記の例は「これはアポストロフィ'です。」を 2 回書き込みます。

空白編集 (B、BN、BZ)

B、BN、および BZ 編集指定子は、数値入力によって埋め込まれた空白と後に続く空白の解釈を制御します。

以下に示す空白指定子が使用可能です。

- **BN**

BN が指定の前にあるときは、入力データの空白 (先行の空白を除く) は 空文字と見なされ、無視されます。

- **BZ**

BZ が指定の前にあるときは、入力データの空白 (先行の空白を除く) はゼロと見なされます。

- **B**

B が指定の前にあるときは、解釈を空白解釈のデフォルトモードに戻します。これはデフォルトの符号制御へ戻す S と一致しています。♦

数値入力欄の空白 (先行の空白を除く) は、書式に特定の空白指定子がない場合、現在装置に対して有効な OPEN 文の BLANK= サブオプションの値に応じてゼロと解釈されるか、または無視されます。そのサブオプションのデフォルト値は ignore (無視する) なので、BN/BZ/B と BLANK= の両方にデフォルトを使用すれば ignore (無視する) となります。

例: 同じデータを BZ と BN で一度ずつ読み取って出力する例

```
demo% cat bz1.f
*       12341234
    CHARACTER LINE*18 / ' 82 82 ' /
    READ ( LINE, '( I4, BZ, I4 ) ') M, N
    PRINT *, M, N
    READ ( LINE, '( I4, BN, I4 ) ') M, N
    PRINT *, M, N
    END
demo% f77 -silent  bz1.f
demo% a.out
  82 8200
  82 82
demo%
```

空白制御の以下の規則に注意してください。

- 空白制御指定子は入力だけに適用されます。
- 空白制御指定子は別の空白制御指定子に出会うか、または書式解釈が完了するまでは、効果を持ちます。
- B、BN、および BZ 指定子は、I 形、F 形、E 形、D 形、および G 形編集に対してだけ効果を与えます。

キャリッジ制御 ($、スペース、0、1)

キャリッジ制御として、編集記述子の $ およびスペース、0、または 1 を使用します。

ドル記号 $

特殊編集記述子 $ は、キャリッジリターンを抑止します。♦

この処理は書式の第 1 文字に依存しません。$ は一般にコンソールプロンプトに使用されます。たとえば入力応答が出力されたプロンプトと同じ行に続くようにするために、この記述子を使用することができます。この編集記述子は、コロン (:) と同じ規則によって制約されます。

例: $ 記号のキャリッジ制御

```
* dol1.f   スペースを伴う $ 編集記述子
    WRITE ( *, 2 )
2   FORMAT (' ノード番号を入力してください: ', $ )
    READ ( *, * ) NODENUM
    END
```

上記のプログラムにより、次のようなプロンプトとユーザーの入力応答が表示されます。

```
ノード番号を入力してください:  82
```

書式の第 1 文字には、この場合空白が出力されます。入力文の場合 $ 記述子は無視されます。

スペース、0、1、+

以下に改行制御とその動作を示します。

表 5-4　空白、0、1、+ によるキャリッジ制御

文字	出力に先立つ行送り
Δ（スペース）	1 行
0	2 行
1	次ページの最初の行へ行送り
+	改行なし (標準出力のみ、ファイルでは無効)

書式の第 1 文字が スペース、0、1、+ 以外のときは、スペースとして扱われ出力されません。

最初の桁の文字が + の場合、前の行の先頭に戻るような制御シーケンスに置き換えられます。

asa を通じてパイプされていると、スペース、0 、1、および + は標準出力に対して働きます。

例：第 1 文字の書式化、標準出力が asa を介してパイプされます。

```
demo% cat slew1.f
    WRITE( *, '("abcd")')
    WRITE( *, '(" efg")')  空白はスペース 1 つ
    WRITE( *, '("0hij")')  "0" はダブルスペース
    WRITE( *, '("1klm")')  "1" はこれを新しいページで始める
    WRITE( *, '("+", T5, "nop")')   "+" は最終行の 1 桁目でこれを始める
    END
demo% f77 -silent slew1.f
demo% a.out | asa | lpr
demo%
```

次のようにプログラム slew1.f は、lp で印刷するとファイル slew1.out を作成します。

```
 bcd
 efg

 hij
```

```
klmnop            これは新しいページに始まる。+nop の + のため
```

画面では結果が違います。タブがスペースを挿入します。

```
demo% cat slew1.out
bcd
efg

hij
```

```
   nop            これは新しいページに始まる。+nop の + のため
demo%
```

asa(1) を参照してください。

スペース、0、 1 および + は、次のような条件で開いたファイルに対して動作します。

- 順番探査
- FORM='PRINT'

例： 第 1 文字書式、ファイル出力

```
demo% cat slew2.f
    OPEN( 1,FILE='slew.out',FORM='PRINT' )
    WRITE( 1, '("abcd")')
    WRITE( 1, '("efg")')
    WRITE( 1, '("0hij")')
    WRITE( 1, '("1klm")')
    WRITE( 1, '("+", T5, "nop")')
    CLOSE( 1, STATUS='KEEP')
    END
demo% f77 -silent slew2.f
demo% a.out
```

次のようにプログラム slew2.f は、lp で印刷するとファイル slew2.out を作成します。ファイル slew2.out は、前述したファイル slew1.out と同じです。

桁 1 の 改行制御コード '0'、'1' と '+' は、出力ファイルの中でそれぞれ ' \n'、' \f' と ' \r' に置換されます。

文字編集 (A)

A 形指定子は文字型データ項目に使用されます。一般書式を次に示します。

```
A[w]
```

入力時、文字データは対応する並びの項目内に格納されます。

出力時、対応する並びの項目が文字データとして表示されます。

w が省略されると、次のようになります。

- 文字データ型変数の場合、その変数のサイズが仮定されます。
- 文字以外のデータ型変数の場合、そのデータ型の変数に適合する文字の最大数が仮定されます。これは規格外の動作です。♦

さまざまな *n* の値についてサイズ変数 *n* (CHARACTER*n) を読み取る例を以下に示します。たとえば、*n*=9 のときは次のとおりです。

```
    CHARACTER C*9
    READ '( A7 )', C
```

CHARACTER C*n のさまざまな *n* の値を以下の表に示します。

サイズ *n*	**9**	**7**	**4**	**1**
データ	NodeΔId	NodeΔId	NodeΔId	NodeΔId
書式	A7	A7	A7	A7
メモリー	NodeΔIdΔΔ	NodeΔId	eΔId	d

Δ は空白のスペースを示します。

出力例: 3 文字、5 文字、および 7 文字の文字列の出力 (それぞれ 5 文字の欄に出力)

```
    PRINT 1, 'The', 'whole', 'shebang'
1   FORMAT( A5 / A5 / A5 )
    END
```

上記のプログラムは次のように表示します。

```
ΔΔThe
whole
sheba
```

文字でない型の最大文字数を次にまとめます。

表 5-5　文字以外の型ホレリス (*n*H*aaa*) の最大文字数

並び項目の型	最大文字数
BYTE	1
LOGICAL*1	1
LOGICAL*2	2
LOGICAL*4	4
LOGICAL*8	8
INTEGER*2	2
INTEGER*4	4
INTEGER*8	8
REAL	4
REAL*4	4
REAL*8	8
REAL*16 （SPARC のみ）	16
DOUBLE PRECISION	8
COMPLEX	8
COMPLEX*8	8
COMPLEX*16	16
COMPLEX*32 （SPARC のみ）	32
DOUBLE COMPLEX	16

f77 では FORMAT 文、代入文、および DATA 文の中で文字定数が使用できるときは、どのような場合でもホレリス定数を使用できます。♦ これら定数はお勧めできません。Fortran 77 規格は、旧プログラムとの互換性を改善するためホレリス機能を備えることを勧めていますが、Fortran 77 はこれら旧ホレリス (*n*H) 表記法を持っていません。このような定数は並びによる入力または変数群入力の入力データ要素として使用できません。

たとえば、次の 2 つの書式は等価です。

```
10  FORMAT( 8H Code = , A6 )
20  FORMAT( ' Code = ', A6 )
```

f77 では、編集記述子間のコンマは一般に省略可能です。

```
10  FORMAT( 5H flex 4Hible )
```

ホレリス編集記述子への読み込み

旧プログラムとの互換性のため、f77 もホレリス編集記述子へ読み込むことができます。♦

例: ホレリス編集記述子への読み込み。READ 文に並びがないことに注意してください。

```
demo% cat hol1.f
    WRITE( *, 1 )
1   FORMAT( 6Holder )
    READ( *, 1 )
    WRITE( *, 1 )
    END
demo% f77 hol1.f
hol1.f:
 MAIN
demo% a.out
older
newer
newer
demo%
```

上記の例で書式が実行時書式 (変数書式) のときは、上記の実際の書式への読み込みは行われず、書式は変更されないままです。つまり、次のような読み込みは失敗します。

```
    CHARACTER F*18 / '(A8)' /
    READ(*,F)! <- 読み込みは行われない
    …
```

どちらにしても、実際の書式へ読み込むためのより良い方法があることは明らかです。

整数編集 (I)

I 形指定子は 10 進整数データ項目に使用されます。一般書式を次に示します。

```
I[w[.m]]
```

I*w* と I*w.m* 編集指定子は、編集する欄が *w* 桁を占めることを示します。指定された入出力並びの項目は整数型でなければなりません。入力時に指定される並びの項目は、整数データで定義されることになります。出力時に指定される並びの項目は、整数データとして定義されなければなりません。

入力時には、I*w.m* 編集指定子は I*w* 編集指定子と同等に扱われます。

I*w* 編集指定子の出力欄は以下のものから構成されます。

- ゼロまたは 1 個以上の先行する空白
- 次にマイナス (値が負のとき) または省略可能なプラス
- 符号なし整定数 (先行のゼロは付けない) の書式内の値の大きさ

整定数は少なくとも 1 桁を持ちます。

I*w.m* 編集指定子の出力欄は、符号なし整定数が少なくとも *m* 桁で構成され、必要に応じて先行のゼロを持つこと以外は、I*w* 編集指定子と同じです。*m* の値は *w* の値を超えてはなりません。*m* がゼロで項目の値もゼロのときは、有効な符号制御に関係なく、出力欄は空白文字だけで構成されます。

例: int1.f、整数入力

```
    CHARACTER LINE*8 / '12345678' /
    READ( LINE, '(I2, I3, I2 )') I, J, K
    PRINT *, I, J, K
    END
```

上記のプログラムは次のように表示されます。

```
     12 345 67
```

例: int2.f、整数出力

```
    N = 1234
    PRINT 1, N, N, N, N
1   FORMAT( I6 / I4 / I2 / I6.5 )
    END
```

上のプログラムは次のように表示します。

```
  1234
1234
**
01234
```

論理編集 (L)

L 形指定子は論理データ項目に使用されます。一般書式を次に示します。

```
Lw
```

L*w* 編集指定子は欄が *w* 桁を占めることを示します。指定された入出力並びの項目は論理型でなければなりません。入力時には並びの項目は論理データで定義されることになります。出力時には並びの項目は論理データとして定義されなければなりません。

入力欄は省略可能な空白、その後に省略可能な小数点、さらにその後に真のときは T、偽のときは F が続きます。T または F の後の欄内に文字を付けることが可能です。論理定数 .TRUE. と .FALSE. は入力として受け入れられます。出力欄は *w*-1 個の空白の後に真のときは T、偽のときは F が続きます。

例: log1.f、論理出力

```
    LOGICAL A*1 /.TRUE./, B*2 /.TRUE./, C*4 /.FALSE./
    PRINT '( L1 / L2 / L4 )', A, B, C
    END
```

上記のプログラムは次のように表示します。

```
T
ΔT
ΔΔΔF
```

例: log2.f 、**論理入力**

```
    LOGICAL*4 A
1   READ '(L8)', A
    PRINT *, A
    GO TO 1
    END
```

上記のプログラムは次の入力データを有効なデータとして受け入れます。

```
t true T TRUE .t .t. .T .T. .TRUE. TooTrue
f false F FALSE .f .F .F. .FALSE. Flakey
```

8 進および 16 進編集 (O, Z)

FORMAT 文の O 形と Z 形欄記述子はそれぞれ 8 進または 16 進整数用ですが、それらはどのデータの型でも使用できます。♦

一般書式は次のとおりです。

O*w*[.*m*]
Z*w*[.*m*]

ここで *w* は外部欄内の文字数であり、出力の場合 *m* が指定されている時は、外部欄内の合計桁数が決定されます (つまり *m* 個に満たないゼロでない桁がある場合、その欄は合計で *m* 桁になるまで左側にゼロが満たされます)。*m* は入力には影響を与えません。

8 進および 16 進入力

FORMAT に O 形または Z 形欄記述子を持つ READ は、それぞれ 8 進または 16 進整数として *w* 文字を読み込み、対応する入出力並びのメンバーに値を割り当てます。

例: 8 進入力時の外部データ欄

```
654321
```

上記の最初の桁は、1 桁目から入力します。

入力を行うプログラムは次のとおりです。

```
    READ ( *, 2 ) M
2   FORMAT ( O6 )
```

上記のようなデータとプログラムでは、8 進値 654321 が変数 M に読み取られます。さらに詳しい例を次の表に示します。

表 5-6　8 進 / 16 進入力値の例

書式	外部欄	内部 (8 進または 16 進) 値
O4	1234Δ	1234
O4	16234	1623
O3	97ΔΔΔ	エラー: 9 は許可されない
Z5	A23DEΔ	A23DE
Z5	A23DEF	A23DE
Z4	95.AF2	エラー: ピリオド (.) は許可されない

8 進および 16 進入力の一般規則は、次のとおりです。

- 8 進値の場合、外部欄は数値 0〜7 だけを含むことができます。
- 16 進値の場合、外部欄は数値 0〜9 および文字 A〜F、または a〜f だけを含むことができます。
- 符号、小数点、および指数欄は許可されません。
- 全て空白の欄はゼロの値を持つものとして扱われます。
- データ項目が対応する変数に対して長すぎるときは、エラーメッセージが表示されます。

8 進および 16 進出力

FORMAT に O 形または Z 形欄記述子を持つ WRITE は、それぞれ 8 進または 16 進整数として値を書き出します。*w* 文字の幅の欄に右詰めで書き込まれます。

例: 16 進出力

```
    M = 161
    WRITE ( *, 8 ) M
8   FORMAT ( Z3 )
    END
```

上記のプログラムは A1 (10 進数 161=16 進数 A1) を表示します。

```
ΔA1
```

文字 A が出力桁 2 に書き込まれます。

さらに詳しい例を次の表に示します。

表 5-7　8 進 / 16 進出力値の例

書式	内部 (10 進) 値	外部 (8 進または 16進) 表現
O6	32767	Δ77777
O2	14251	**
O4.3	27	Δ033
O4.4	27	0033
O6	-32767	*****
Z4	32767	7FFF
Z3.3	2708	A94
Z6.4	2708	ΔΔ0A94
Z5	-32767	******

8 進および 16 進出力の一般規則は、次のとおりです。

- 負の値は符号なしのように書き込まれ、マイナス符号は出力されません。
- 外部欄は必要であれば、最大 *w* の幅まで先行するスペースで満たされます。
- 欄が狭すぎる場合はアスタリスクで満たされます。
- *m* が指定されると、欄は *m* の幅になるまで先行するゼロで左側から満たされます。

位置付け編集

f77 は出力行に沿った水平位置付けに関しては、書式 T*n*、TL*n*、TR*n*、*n*T、および *n*X をサポートします。ここで *n* は正の整数でなければなりません。書式指定子 T は単独に、または正のゼロでない数の前後に置くことができます。

T*n* — 絶対値桁

タブは *n* 番目の桁から読み取るか、または *n* 番目の桁へ書き込みます。*n* がないときは、T1 として処理されます。

TL*n* — 相対桁

タブは左側の *n* 番目の桁から読み取るか、または左側の *n* 番目の桁へ書き込みます *n* がないときは、TL0 として処理されます。♦

TR*n* — 相対桁

タブは右側の *n* 番目の桁から読み取るか、または右側の *n* 番目の桁へ書き込みます *n* がないときは、TR0 として処理されます。♦

*n*T — 相対タブストップ ♦

タブにより読み取りのときも書き込みのときも、*n* 番目のタブストップに移動します。*n* が省略されると *n*=1 が使用され、次のタブストップに移動します。(この編集は、FORTRAN 77 の規格外です。)

タブ操作の規則と制限事項は、次のとおりです。

- 入力論理記録の終わりを越えて右へタブ操作するとエラーになります。
- 入力論理記録の始まりを越えて左へタブ操作すると、入力ポインタは記録の始めに置かれます。
- 内部書式付き入出力と外部書式付き入出力の両方について、非破壊的なタブ操作が可能です。非破壊的なタブ操作とは、出力の左または右へのタブ操作が記録の先に書かれた部分を破壊しないということです。
- 出力の右へのタブ操作により記録の書き込まれていない部分は空白で満たされます。
- 出力の左へのタブ操作は論理ユニットがシークを許可することを必要とします。したがって端末との入出力つまりパイプは許可されません。
- 同様にどちらの方向にせよ非破壊的なタブ操作はシークできる装置でのみ可能です。それ以外の場合には、右へのタブ操作または X 編集指定子による空白処理は出力に空白を書き込みます。
- タブストップは、8 桁ごとに固定されています。

*n*X — 位置

*n*X 編集指定子は記録への、または記録からの次の文字の転送を現在の位置から *n* 文字先の位置で行うことを指示します。

入力時、*n*X 編集指定子は記録ポインタを *n* 桁進めるので、*n* 文字スキップします。

記録の最終文字を越えた位置は、この位置から文字が転送されない場合は指定することができます。

出力時、*n*X 指定子は *n* 個の空白を書き込みます。

n のデフォルト値は 1 です。

例: 入力 T*n* (絶対タブ)

```
demo% cat rtab.f
    CHARACTER C*2, S*2
    OPEN( 1, FILE='mytab.data')
    DO I = 1, 2
    READ( 1, 2 ) C, S
2   FORMAT( T5, A2, T1, A2 )
    PRINT *, C, S
    END DO
    END
demo%
```

2 行のデータファイル

```
demo% cat mytab.data
defguvwx
12345678
demo%
```

実行および出力

```
demo% a.out
uvde
5612
demo%
```

上記の例では、まず桁 5 と 6 を、次に桁 1 と 2 を読み取ります。

例: 出力 Tn (絶対タブ) — このプログラムは出力ファイルを書き込みます。

```
demo% cat otab.f
    CHARACTER C*20 / "12345678901234567890" /
    OPEN( 1, FILE='mytab.rep')
    WRITE( 1, 2 ) C, ":", ":"
2   FORMAT( A20, T10, A1, T20, A1 )
    END
demo%
```

出力ファイル

```
demo% cat mytab.rep
123456789:123456789:
demo%
```

上記の例は 20 文字を書き込み、桁 10 と 20 を変更しました。

例: 入力 TRn および TLn (相対タブ) — プログラム

```
demo% cat rtabi.f
    CHARACTER C, S, T
    OPEN( 1, FILE='mytab.data')
    DO I = 1, 2
    READ( 1, 2 ) C, S, T
2   FORMAT( A1, TR5, A1, TL4, A1 )
    PRINT *, C, S, T
    END DO
    END
demo%
```

2 行のデータファイル

```
demo% cat mytab.data
defguvwx
12345678
demo%
```

実行と出力

```
demo% a.out
dwg
174
demo%
```

上記の例は、桁 1 を読み取ってから桁 7 まで 5 桁右に移動し、その後桁 4 まで 4 桁左に移動します。

例: TR*n* と TL*n* の出力 (相対タブ) — このプログラムは出力ファイルを書き込みます。

```
demo% cat rtabo.f
    CHARACTER C*20 / "12345678901234567890" /
    OPEN( 1, FILE='rtabo.rep')
    WRITE( 1, 2 ) C, ":", ":"
2   FORMAT( A20, TL11, A1, TR9, A1 )
    END
demo%
```

実行しても何も表示しませんが、mytab.rep 出力ファイルを見ることができます。

```
demo% cat rtabo.rep
123456789:123456789:
demo%
```

上記のプログラムは 20 文字を書き込んでから桁 10 まで 11 桁左に移動し、その後桁 20 まで 9 桁右に移動します。

引用符編集 (“aaa”)

引用符編集指定子は文字定数の書式を使用します。♦ 引用符編集指定子は編集指定子自身が囲んだ文字 (空白を含みます) を書き込みます。引用符編集指定子は入力に使用してはなりません。

欄の幅は区切り用引用符の間に含まれる文字数ですが、区切り用の引用符は含みません。欄内にあって間に空白がない 2 個の連続する引用符は、1 個の引用符として数えられます。同様の方法でアポストロフィを使用することができます。

例: quote.f (等価な 2 つの方法)

```
    WRITE( *, 1 )
1   FORMAT( 'これは引用符 " です。' )
    WRITE( *, 2 )
2   FORMAT( "これは引用符 "" です。" )
    END
```

これはメッセージ「これは引用符 " です。」を 2 回書き込みます。

基数制御 (R)

書式指定子は R または *n*R です。ただし 2≦*n*≦36 です。*n* が省略されると、デフォルトの 10 進基数が復元されます。

基数制御により書式付き整数入出力変換に 10 以外の基数を指定することができます。この指定子は浮動小数点変換の桁移動 P にならっています。他の基数が指定されるか、または書式解釈が終了するまでは効果を持ち続けます。入出力項目は 32 ビット整数として扱われます。

例: 基数が 16 で、符号なし、16 進の整数、10 の幅を持ち、8 桁までゼロで満たす書式は (su,16r,I10.8) となります。

```
demo% cat radix.f
    integer i / 110 /
    write( *, 1 ) i
1   format( su, 16r, I10.8 )
    end
demo% f77 -silent radix.f
demo% a.out
 ΔΔ0000006E
demo%
```

SU については、288 ページの「符号編集 (SU、SP、SS、S)」で説明します。

実数編集 (D、E、F、G)

D 形、E 形、F 形、および G 形指定子は、10 進の実数データ項目に使用されます。

D 形編集

D 形指定子は 10 進倍精度項目の指数書式に使用されます。一般書式を次に示します。

```
D[w[.d]]
```

D*w* および D*w.d* 形編集指定子は、編集する欄が *w* 桁を占めることを示します。*d* は数の小数部 (小数点の右側の部分) が *d* 桁であることを示します。しかし入力データが小数点を持つ場合は、その小数点が *d* の値を無効にします。

入力時に指定される並びの項目は、実数データで定義されることになります。出力時に指定される並びの項目は、実数データとして定義されなければなりません。

出力文では、E の代わりに D が使用されること以外は、D 形編集記述子が E 形編集記述子と同じことを行います。D*w.d* 形編集指定子に対する出力欄は幅 *w* を持ちます。値は欄に右詰めされます。欄はゼロまたは 1 個以上の先行する空白の次にマイナス (値が負のとき) または省略可能なプラスがあり、さらに *d* 桁の 10 進数に丸められた並び項目の値の大きさが続きます。

w は、マイナス符号、小数点の左側の 1 桁以上の数字、小数点、および小数点の右側の
d 桁までの数字を許可しなければなりません。したがって、この場合 *w*≧*d*+3 でなければなりません。

例: D 形編集を設定した実入力。プログラム Dinp.f

```
      CHARACTER LINE*24 / '12345678 23.5678 .345678' /
      READ( LINE, '( D8.3, D8.3, D8.3 )') R, S, T
      PRINT '( D10.3, D11.4, D13.6 )', R, S, T
      END
```

上記のプログラムは次のように表示されます。

```
 0.123D+05 0.2357D+02 0.345678D+00
```

上記の例で最初の入力データ項目は小数点を持たないので D8.3 が小数点を決定します。その他の入力データ項目は小数点を持つので、小数点に関する限りこれらの小数点が
D 形編集記述子を無効にします。

例: D 形編集を設定した実出力。プログラム Dout.f

```
    R = 1234.678
    PRINT 1, R, R, R
1   FORMAT( D9.3 / D8.4 / D13.4 )
    END
```

上記のプログラムは次のように表示します。

```
0.123D+04
********
ΔΔΔ0.1235D+04
```

上記の例で 2 番目の出力行は D8.4 が符号を許可しないのでアスタリスクになり、3 番目の出力行は D13.4 の結果として先頭に 3 個の空白がきます。

E 形編集

E 形指定子は 10 進の実数データ項目の指数書式に使用されます。一般書式を次に示します。

```
E[w[.d][Ee]]
```

w は編集する欄が *w* 桁を占めることを示します。

d は数の小数部 (小数点の右側の部分) が *d* 桁であることを示します。しかし入力データが小数点を持つ場合は、その小数点が *d* の値を無効にします。

e は指数欄の桁の数を示します。デフォルトは 2 になります。

指定される入出力並びの項目は実数型の項目でなければなりません。入力時に指定される並びの項目は実数データで定義されることになります。出力時に指定される並びの項目は実数データとして定義されなければなりません。

E*w.d* 形編集指定子に対する出力欄は幅 *w* を持ちます。値はその欄に右詰めされます。欄はゼロまたは 1 個以上の先行する空白のあとに、マイナス (値が負のとき) または省略可能なプラスが続き、さらにゼロ、小数点、*d* 桁の 10 進数に丸められた並びの項目の値の大きさおよび指数が続きます。

書式 E*w.d* は指数の絶対値によって以下のようになります。

- ｜指数｜≤ 99 ならば、書式 E*nn* または 0*nn* になります。
- 99 ≤ ｜指数｜≤ 999 ならば、書式 *nnn* になります。

書式 E*w.d*E*e* は、｜指数｜ ≤ (10e) -1 ならば、指数は書式 *nnn* になります。

書式 D*w.d* は指数の絶対値によって以下のようになります。

- ｜指数｜ ≤ 99 ならば、書式 D*nn* または E*nn* または 0*nn* になります。
- 99 ≤ ｜指数｜ ≤ 999 ならば、書式 *nnn* になります。

ここで *n* は任意の数字です。

指数には符号が必要です。

w はマイナス符号を許可する必要はありませんが、ゼロ、小数点、および小数点の右側の *d* 桁の数字、および指数を許可しなければなりません。したがって、負数でないときは *w* ≥ *d*+6 ですが、 *e* があれば *w* ≥ *d*+*e*+4 です。負数のときは *w* ≥ *d*+7 ですが、 *e* があれば *w* ≥ *d*+*e*+5 です。

例: E 形編集を設定した実入力。プログラム Einp.f

```
    CHARACTER L*40/'1234567E2 1234.67E-3 12.4567 '/
    READ( L, '( E9.3, E12.3, E12.6 )') R, S, T
    PRINT '( E15.6, E15.6, E15.7 )', R, S, T
    END
```

上記のプログラムは次のように表示します。

```
ΔΔΔ0.123457E+06ΔΔΔ0.123467E+01ΔΔ0.1245670E+02
```

上記の例で最初の入力データ項目は小数点を持たないので E9.3 が小数点を決定します。その他の入力データ項目は小数点を持つので、小数点に関する限りこれら小数点が D 形編集記述子を無効にします。

例: E 形編集を設定した実出力。プログラム Eout.f

```
    R = 1234.678
    PRINT 1, R, R, R
1   FORMAT( E9.3 / E8.4 / E13.4 )
    END
```

上記のプログラムは次のように表示します。

```
0.123E+04
********
ΔΔΔ0.1235E+04
```

上記の例で E8.4 は符号を許可しないのでアスタリスクが出力されます。また E13.4 の余分な幅の欄は 3 個の先行する空白になります。

例: E*w.d*E*e* 形編集を設定した実出力。プログラム EwdEe.f

```
    REAL   X / 0.000789 /
    WRITE(*,'( E13.3)') X
    WRITE(*,'( E13.3E4)') X
    WRITE(*,'( E13.3E5)') X
    END
```

このプログラムは次のように表示します。

```
ΔΔΔΔ0.789E-03
ΔΔ0.789E-0003
Δ0.789E-00003
```

F 形編集

F 指定子は 10 進の実数データ項目に使用されます。一般書式を次に示します。

```
    F[w[.d]]
```

F*w* と F*w.d* 形編集指定子は編集する欄が *w* 桁を占めることを示します。

d は数の小数部 (小数点の右側の部分) が *d* 桁であることを示します。しかし入力データが小数点を持つ場合は、その小数点が *d* の値を無効にします。

指定される入出力並びの項目は、実数型の項目でなければなりません。入力時に指定される並びの項目は、実数データで定義されることになります。出力時に指定される並びの項目は、実数データとして定義されなければなりません。

F*w.d* 形編集指定子に対する出力欄は幅 *w* を持ちます。値はその欄に右詰めされます。欄はゼロまたは 1 個以上の先行する空白のあとにマイナス (値が負のとき) または省略可能なプラスが続き、さらにその次に *d* 桁の 10 進数に丸められた並びの項目の値の大きさが続きます。

w はマイナス符号、小数点の左側の 1 桁以上の数字、小数点、および小数点の右側の *d* 桁の数字を許可しなければなりません。したがってこの場合は、*w* ≥ *d*+3 でなければなりません。

例: F 形編集を設定した実入力。プログラム Finp.f

```
    CHARACTER LINE*24 / '12345678 23.5678 .345678' /
    READ( LINE, '( F8.3, F8.3, F8.3 )') R, S, T
    PRINT '( F9.3, F9.4, F9.6 )', R, S, T
    END
```

上記のプログラムは次のように表示します。

```
12345.678DD23.5678D0.345678
```

上記の例で、最初の入力データ項目は小数点を持たないので、F8.3 が小数点を決定します。その他の入力データ項目は小数点を持つので、小数点に関する限りこれら小数点が F 形編集記述子を無効にします。

例: F 形編集を設定した実出力。プログラム Fout.f

```
    R = 1234.678
    PRINT 1, R, R, R
1   FORMAT( F9.3 / F8.4 / F13.4 )
    END
```

上記のプログラムは次のように表示します。

```
Δ1234.678
********
ΔΔΔΔ1234.6780
```

上記の例で F8.4 は符号を許可しません。F13.4 により 先頭に 4 個の空白がきて、後ろに 1 個のゼロがきます。

G 形編集

G 形指定子は 10 進の実数データ項目に使用されます。一般書式を次に示します。

G[*w*[.*d*]]

あるいは

G*w*.*d*E*e*

D、E、F、および G 形編集指定子は同じ方法でデータを解釈します。

G 形編集記述子による出力の表現は内部データの大きさに左右されます。次の表で *N* は
内部データの大きさを示します。

範囲	書式
0.1 ≤ *N* < 1.0	F(*w*-4).*d*
1.0 ≤ *N* < 10.0	F(*w*-4).(*d*-1)
...	...
$10^{(d-2)} \le N \le 10^{(d-1)}$	F(*w*-4).1
$10^{(d-1)} \le N < 10^{d}$	F(*w*-4).0

書式付き入力のコンマ

固定桁書式で制御される数値データを入力するとき、コンマを使用して桁の制限を無効にすることができます。

例: 書式

```
    (I10, F20.10, I4)
```

上記の書式を使用すれば次の記録を正しく読み取ることができます。

```
-345,.05e-3,12
```

入出力システムは FORTRAN 77 規格に記述されているよりも寛大になっています。一般に文字以外の変数の書式付き読み取りを行う場合、コンマは欄の長さを変更します。正確に言うと Iw、Fw.d、Fw.d[Ee]、および Gw.d の入力欄の場合、w 個の文字が走査されるか、またはコンマがあると欄は終了します。コンマで終了したとき欄はコンマまでの文字を含みますが、コンマは含みません。次の欄はコンマの次の文字で始まります。

残存文字 (Q)

Q 形編集記述子は入力記録の長さ、または入力記録の読み取られていない残存部分の長さを知るために使用されます。♦

現在の記録から読み取るべき残り文字数を獲得します。

例: 実数と文字列からの実数、文字列長、および文字列の獲得

```
demo% cat qed1.f
* qed1.f Q edit descriptor (real & string)
    CHARACTER CVECT(80)*1
    OPEN ( UNIT=4, FILE='qed1.data' )
    READ ( 4, 1 ) R, L, ( CVECT(I), I=1,L )
1   FORMAT ( F4.2, Q, 80 A1 )
    WRITE ( *, 2 ) R, L, '"', (CVECT(I),I=1,L), '"'
2   FORMAT ( 1X, F7.2, 1X, I2, 1X, 80A1 )
    END
demo% cat qed1.data
8.10qwerty
demo% f77 qed1.f -o qed1
qed1.f:
 MAIN:
demo% qed1
   8.10 6 "qwerty"
demo%
```

上記の例は、変数 R に欄を読み込んでからその欄の後に残っている文字数を L に読み込み、それから L の文字を CVECT に読み込みます。*n* 番目の編集記述子としての Q は、READ 並び内の *n* 番目の要素としての L と一致します。

例: 入力記録の長さの獲得。Q 記述子を最初に設定。

```
demo% cat qed2.f
    CHARACTER CVECT(80)*1
    OPEN ( UNIT=4, FILE='qed2.data' )
    READ ( 4, 1 ) L, ( CVECT(I), I=1,L )
1   FORMAT ( Q, 80A1 )
    WRITE ( *, 2 ) L, '"', (CVECT(I),I=1,L), '"'
2   FORMAT ( 1X, I2, 1X, 80A1 )
    END
demo% cat qed2.data
qwerty
demo% f77 qed2.f -o qed2
qed2.f:
 MAIN:
demo% qed2
  6 "qwerty"
demo%
```

上記の例は入力記録の長さを獲得します。これで入力文字列全体とその長さについてはユーザー自身が解析することができます。

Q 編集記述子の制限事項は、次のとおりです。

- 対応する並びの要素は、INTEGER または LOGICAL データ型でなければなりません。
- Q は厳密に文字を数えます。入力記録中に残っている文字の数を獲得します。整数、実数またはその他の数は獲得しません。
- Q 編集記述子はパイプファイルに適用することはできません。ファイルは再読み取り可能であることが必要です。
- この記述子はファイルと stdin **入力** (端末) 入力に対して有効です。
- この記述子は出力には使えず、実行時にエラーが発生します。

桁移動 (P)

P 形編集記述子は実数の入力値を 10 のべき乗だけ桁移動します。出力値に対しては、表示される有効数字を制御します。

一般書式を次に示します。

[*k*] P

パラメータ	説明
k	符号なしまたは符号付きの整定数

k は桁移動数と呼ばれ、そのデフォルト値はゼロです。

例: 桁移動数を設定した入出力文

```
READ ( 1, '( 3P E8.2 )' ) X
WRITE ( 1, '( 1P E8.2 )' ) X
```

P 自身だけのときは 0P と等価です。P は桁移動数をデフォルト値 0P にリセットします。P 単独は規格外です。♦

有効範囲

桁移動数は、各入出力文の実行を開始する時にゼロにリセットされます。桁移動数は D、E、F、および G 形編集記述子に適用されます。

入力

入力時、指数欄を持たない外部データは、10*k* で除算されてから内部的に格納されます。

入力例: データ、桁移動数、および結果として記憶される値

データ	18.63	18.63	18.63E2	18.63
書式	E8.2	3P E8.2	3P E8.2	-3P E8.2
メモリー	18.63	.01863	18.63E2	18630.

出力

D 形、E 形記述子を使用する出力、または E 形編集に対応する G 形記述子を使用する出力時に、内部項目は基本の実定数部に 10*k* を乗じられ、指数部は *k* だけ減らされて、書き出されます。

F 形記述子を使用する出力、および F 形編集に対応する G 形記述子を使用する出力時に、内部項目は基本の実定数部分に 10*k* を乗じられた後に書き出されます。

出力例: 格納される値、桁移動数、および結果の出力

メモリー	290.0	290.0	290.0	290.0
書式	2P E9.3	1P E9.3	-1P E9.3	F9.3
表示	29.00E+01	2.900E+02	0.029E+04	0.290E+03

符号編集 (SU、SP、SS、S)

SU 形、SP 形、および S 形編集記述子は先行する符号の出力を制御します。特殊な符号指定子がない通常の出力の場合、値が負のときはマイナス符号が左端の桁の 1 桁左に出力されます。値が正のときは処理系によってはプラス符号が出力されますが、f77 はプラス符号を省略します。

以下の符号指定子が使用できます。

- SP — SP が指定の前にあるときは符号が出力されます。
- SS — SS が指定の前にあるときはプラス符号の出力は抑止されます。
- S — S が指定の前にあるときはシステムのデフォルトが有効となります。デフォルトは SS です。
- SU — SU が指定の前にあるときは整数値は符号なしと解釈されます。これは規格外です。♦

たとえば符号なし指定子は、次に示すように基数指定子とともに使用して、16 進数ダンプを書式化することができます。

```
2000 FORMAT( SU, 16R, 8I10.8 )
```

符号制御の規約と制限事項は次のとおりです。

- 符号制御指定子は出力だけに適用されます。
- 符号制御指定子は別の符号制御指定子に出会うか、または書式解釈が終了するまでは有効です。

- S 形、SP 形、および SS 形指定子は I 形、F 形、E 形、D 形、および G 形編集だけに影響を与えます。
- SU 形指定子は I 形編集だけに影響を与えます。

スラッシュ編集 (/)

スラッシュ (/) 編集指定子は現在の記録におけるデータ転送の終了を指示します。

順番探査

入力時は現在の記録の残りの部分を無視し、ファイルは次の記録の先頭に位置付けられます。2 個の連続するスラッシュ (//) は記録全体を無視します。

出力時は記録終了が書き込まれ、新しい記録が開始されます。2 個の連続するスラッシュ (//) は文字なしの記録を作成します。ファイルが内部ファイルのとき、その記録は空白で満たされます。

直接探査

各スラッシュは記録番号を 1 つずつ増やし、ファイルをその番号の記録の先頭に位置付けます。

出力時、2 個の連続するスラッシュ (//) は文字なしの記録を作成します。その記録は空白で満たされます。

終了制御 (:)

コロン (:) 編集記述子により書式を条件付きで終了することができます。入出力並びが書式の途中で尽きると書式はコロンで終了します。

例: 終了制御

```
* col1.f コロン (:) 編集記述子
      DATA INIT / 3 /, LAST / 8 /
      WRITE ( *, 2 ) INIT
      WRITE ( *, 2 ) INIT, LAST
2     FORMAT ( 1X 'INIT = ', I2, :, 3X, 'LAST = ', I2 )
      END
```

上記のプログラムは、次のような出力を生成します。

```
    INIT = 3
    INIT = 3 LAST = 8
```

コロンがないと出力は次のようになります。

```
    INIT = 3 LAST =
    INIT = 3 LAST = 8
```

実行時書式

実行中に変更可能な書式指定子をオブジェクトの中に設定することができます。これによって柔軟性が向上します。この種類の書式指定子は、入出力文が実行されるたびに解析されるので、実行時間は増加します。これらは変数書式とも呼ばれます。

オブジェクトは以下の種類のいずれかでなければなりません。

- 文字式

文字式はスカラー、配列、配列の要素、部分列、構造体記録の欄♦、またはこれらを連結したものなどです。

- 整数配列♦

整数配列は DATA 文、代入文、READ 文などでその文字値を獲得することができます。

区切るのに左右の括弧が必要です。ただしキーワード FORMAT と文番号は除きます。

オブジェクトは書式全体を設定するのに十分な大きさで宣言しなければなりません。たとえば、'(8X,12I)' は INTEGER*4 または CHARACTER*4 オブジェクトには適しません。

例: 文字式と整数配列の実行時書式

```
demo% cat runtim.f
    CHARACTER CS*8
    CHARACTER CA(1:7)*1 /'(','1','X',',','I','2',')'/
    CHARACTER S(1:7)*6
    INTEGER*4 IA(2)
    STRUCTURE / STR /
    CHARACTER*4 A
    INTEGER*4 K
    END STRUCTURE
    CHARACTER*8 LEFT, RIGHT
    RECORD /STR/ R
    N = 9
    CS = '(I8)'
    WRITE( *, CS ) N! 文字スカラー
    CA(2) = '6'
    WRITE( *, CA ) N! 文字配列
    S(2) = '(I8)'
    WRITE( *, S(2) ) N! 文字配列の要素
    IA(1) = '(I8)'
    WRITE(*, IA ) N! ! 整数配列
    R.A = '(I8)'
    WRITE( *, R.A ) N! 記録の欄
    LEFT = '(I'
    RIGHT = '8)'
    WRITE(*, LEFT // RIGHT ) N ! つなげる
    END
demo% f77 -silent runtim.f
demo% a.out
        9
        9
        9
        9
        9
        9
demo%
```

書式内の変数式 (<e>)

一般に FORMAT 文内の整定数は任意の式で置換することができます。♦

式自身は山括弧で囲まなければなりません。

例を次に示します。

```
1    FORMAT( 3F6.1 )
```

上記の文の 6 は、次のように変数 N で置換可能です。

```
1    FORMAT( 3F<N>.1 )
```

または次のように、さらに複雑な式 2*N+M でも置換可能です。

```
1    FORMAT( 3F<2*N+M>.1 )
```

同様に、3 または 1 を任意の式で置換することができます。

ただ 1 つの例外は *n*H... 編集記述子の *n* です。

書式内の変数式に対する規則と制限事項は、以下のとおりです。

- 式は書式走査中に出会うたびに再評価されます。
- 必要であれば、式は整数型に変換されます。
- 関数呼び出しを含めて正しい Fortran 77 式はすべて許可されます。
- 変数式は、実行時に生成される書式では使用できません。
- *n*H... 編集記述子の *n* は変数式にすることができません。

書式なし入出力

書式なし入出力は、2 進情報をその内部表現を変更することなく、メモリーとの間で転送するために使用されます。書式なし入出力文の実行のたびに単一の論理記録の読み取り、または書き込みが行われます。内部表現はアーキテクチャにより異なるので、書式なし入出力は移植性に限界があります。

書式なし入出力を使用して一時的にデータを書き出したり、または迅速にデータを書き出して同一アーキテクチャを持つマシン上で実行される別の Fortran 77 プログラムに入力することができます。

順番探査入出力

書式なし順番探査ファイルの論理記録長は、入出力並び中の項目が必要とするバイト数により異なります。この入出力書式の要求条件により、外部物理記録サイズは論理記録サイズよりいくらか大きくなります。

例:

```
    WRITE( 8 ) A, B
```

Fortran 77 実行時システムは、書式なし順番 WRITE 中にデータ中の記録境界として、各書式なし順番記録の始めと終わりに INTEGER*4 でバイト数を挿入します。記録の終わりのバイト数により、記録の BACKSPACE 操作が可能になります。結果として Fortran 77 プログラムは、書式なし順番 WRITE 操作で書き込まれたデータについてだけ書式なし順番 READ を使用することができます。このような記録を書式付きとして読み取ろうとすると、その結果は予測できません。

以下にガイドラインを示します。

- ファイルがその方法で作成されているとき以外では、書式なし順番 READ を使用しないでください。
- 各書式なし順番記録の始めと終わりには余分なデータがあるので、その余分なデータが目立つ場合、ユーザーは書式なし直接入出力の使用を試みたいと思うかもしれません。きわめて長い記録よりも、短い記録の時にこれが顕著になります。

直接探査入出力

複数の入出力並びが異なった長さのとき、RECL=1 オプションを設定してファイルを開くことができます。このオプションは読み取り、または書き込みを行う項目の数を決定するために入出力並びを使用することを Fortran 77 に知らせます。

ユーザーはさらに各読み取りごとに、読み取りを開始する最初の記録 (この場合どのバイトからか) を Fortran 77 に知らせる必要があるため、各項目のサイズを知っておかなければなりません。♦

簡単な例を以下に示します。

例: 直接探査 — それぞれ 2 つの整数からなる記録を 3 個作成する

```
demo% cat Direct1.f
    integer u/4/, v /5/, w /6/, x /7/, y /8/, z /9/
    open( 1, access='DIRECT', recl=8 )
    write( 1, rec=1 ) u, v
    write( 1, rec=2 ) w, x
    write( 1, rec=3 ) y, z
    end
demo% f77 -silent Direct1.f
demo% a.out
demo%
```

例: 直接探査 — 3 つの記録を読み取る

```
demo% cat Direct2.f
    integer u, v, w, x, y, z
    open( 1, access='DIRECT', recl=8 )
    read( 1, rec=1 ) u, v
    read( 1, rec=2 ) w, x
    read( 1, rec=3 ) y, z
    write(*,*) u, v, w, x, y, z
    end
demo% f77 -silent Direct2.f
demo% a.out
  4  5  6  7  8  9
demo%
```

ファイル上の記録長がわかります。この例では、書き込まれたとおりにファイルを読み込むことができます。

ただし、ファイル上の記録長ではなく各項目のサイズがわかれば、OPEN 文で 、`recl=1` を使用して、入出力並びによって読み込む項目の数を確定することができます。

例: 直接探査読み取り、recl=1 を使用する

```
demo% cat Direct3.f
    integer u, v, w, x, y, z
    open( 1, access='DIRECT', recl=1 )
    read( 1, rec=1 ) u, v, w
    read( 1, rec=13 ) x, y, z
    write(*,*) u, v, w, x, y, z
    end
demo% f77 -silent Direct3.f
demo% a.out
  4  5  6  7  8  9
demo%
```

上記の例のように 3 整数 (12バイト) を読み取った後、ユーザーは記録 13 から次の読み取りを開始します。

♦ バイナリ入出力

バイナリ入出力でファイルを開くと、プログラムはレコード境界のないバイナリデータのストリームとして、データを読み書きできます。(この機能は Fortran 77 の規格外です)。♦

OPEN 文における FORM = 'BINARY' オプションは次のように、そのユニットがレコードマークのない順番の書式なしファイルであると宣言します。

```
    OPEN(1, FORM='BINARY')
```

バイナリファイルは、直接探査や書式付きとして宣言することもできません。

WRITE 文では、バイナリデータは出力並びにあるバイト数と同じバイト数が、バイトのストリームとしてファイルに書き込まれます。READ 文では、入力並びで変数が要求したデータバイト数分が読み取られます。ファイルにはレコードマーク (レコードの終わり) がないので、レコードマーク以降が読取られる可能性はありません。プロ

グラムが検出できる状態は、異常なシステムエラーを除くとファイルの終わり (EOF) での読取りだけです。各 READ 文は以下の例のように、ファイルの次の順番のバイトを読み込むだけです。

```
demo>cat bin.f
                program bin
                character *25 string
                character *5 word

                open(1,FORM='BINARY')

                string = 'alphabetagammaepsilon'

                write(1) string
                rewind 1

                do 1 i=1,6
                word = '     '
                read(1) word
  1             print*, word

                end
demo>f77 -o binf bin.f
bin.f:
 MAIN bin:
demo>binf
 alpha
 betag
 ammae
 psilo
 n
uio: [-1] end of file
logical unit 1, named 'fort.1'
lately: reading sequential unformatted external IO
Abort
```

バイナリファイルに対する INQUIRE は、FORM = パラメータの場合は 'BINARY'、ACCESS = の場合は 'SEQUENTIAL'、UNFORMATTED = の場合は 'YES'、SEQUENTIAL の場合は 'YES' を返します。

バイナリファイルには BACKSPACE は使用できず、実行時にはエラーメッセージが出力されます。

並びによる入出力

並びによる入出力は、順番探査デバイスのための自由形式の入出力です。これを使用するには、次に示すように書式識別子としてアスタリスクを使用してください。

```
READ( 6, * ) A, B, C
```

並びによる入力には、次のような規則があります。

- 入力時、値は空白文字列と (場合によっては) コンマで分離します。
- 文字列の場合を除いて、値には空白を入れることはできません。
- 文字列は、引用符 (") かアポストロフィ (') で囲んだ文字列、または引用符の付かない文字列です (300 ページの「引用符の付かない文字列」を参照してください)。ホレリス (*n*Hxyz) 文字列は使用できません。
- 文字列の場合を除いて、記録の終わりは空白と見なされます。文字列では無視されます。
- コンマで分離し、括弧で囲んだ 2 個の実定数が複素定数として与えられます。
- コンマとコンマの間の NULL 入力欄などは、入出力並びの中のこれに対応する変数が変わらないことを示します。
- 次に示すように、入力データ項目の前に繰り返し回数を置くことができます。

```
4*(3.,2.) 2*, 4*'hello'
```

これは、4 個の複素定数、2 個の NULL 入力欄、4 個の文字列定数を意味します。

- 入力並びの中のスラッシュ (/) は、並びによる入力で入力並びへの値の割り当ての終わりを示し、現在の入力行にある残りをスキップします。スラッシュに続くテキストは無視され、データ行の注釈に使用されます。

出力形式

並びによる出力により、細かい形式に煩わされずに高速で簡単に出力できます。厳密な形式が必要な場合は、書式付き入出力を使用してください。各項目に適した形式が選択され、正確な出力と簡素な出力形式が矛盾する場合は、簡素な形式が選択されます。

並びによる出力には、次のような規則があります。

- 一般に、各記録は空白から始まります。プリントファイルでは、この空白は出力されません。詳細については、249 ページの「プリントファイル」を参照してください。♦
- 文字列はそのまま出力されます。文字列は引用符で囲まれないため、並びによる入力を使用して読み戻すことができるのは特定の形式の文字列に限られます。これについては次節で説明します。
- 正確な 2 進表記になっていない数は四捨五入されます。

例： 不正確なバイナリの表現

```
demo% cat lis5.f
    READ ( 5, * ) X
    WRITE( 6, * ) X, '     美'
    WRITE( 6, 1 ) X
1   FORMAT( 1X, F13.8, ' 真実' )
    END
demo% f77 lis5.f
lis5.f:
 MAIN:
demo% a.out
1.4
    1.40000000 美
    1.39999998 真実
demo%
```

上記の例で、正確な数値が必要な場合は、書式を指定してください。

その他に、次の点に注意してください。

- 80 文字より長い出力行はできるだけ避けてください。
- 複素数値および倍精度複素数値には、適切な位置にコンマが含まれます。

- 実数、倍精度値、および 4 倍精度値は、書式がそれぞれ異なります。
- 文字列の中のバックスラッシュ n (\n) はキャリッジリターンとして出力されます。ただし、-xl オプションが設定されている場合は、バックスラッシュ n として出力されます。

例: -xl オプション有無による並びによる入出力とバックスラッシュ n

```
demo% cat f77 bslash.f
    CHARACTER S*8 / '12\n3' /
    PRINT *, S
    END
demo%
```

-xl なしで、キャリッジリターンとして \n を出力します。

```
demo% f77 -silent bslash.f
demo% a.out
12
3
demo%
```

-xl を付けて、文字列として \n を出力します。

```
demo% f77 -xl -silent bslash.f
demo% a.out
12\n3
demo%
```

表 5-8　並びによる出力のデフォルトの書式

型	書式
BYTE	数字の前に 2 個の空白
CHARACTER*n	A n {n= 文字式の長さ}
COMPLEX	'ΔΔ(', 1PE14.5E2, ',', 1PE14.5E2, ')''ΔΔ(',
COMPLEX*16	1PE22.13.E2' ',', 1PE22.13.E2,')' 'ΔΔ(',
COMPLEX*32	1PE44.34E3, ',', 1PE44.34E3, ')'
INTEGER*2	数字の前に 2 個の空白
INTEGER*4	数字の前に 2 個の空白
INTEGER*8	数字の前に 2 個の空白
LOGICAL*1	数字の前に 2 個の空白
LOGICAL*2	L3
LOGICAL*4	L3
LOGICAL*8	L3
REAL	1PE14.5E2
REAL*8	1PE22.13.E2
REAL*16	1PE44.34E4

COMPLEX*32 と REAL*16 は、SPARC のみ使用できます。

引用符の付かない文字列

f77 の並びによる入出力を使用すると、引用符に囲まれない文字列の読み込みを行うことができます。♦

文字列を数字で始めることはできません。また、分離文字 (コンマまたはスラッシュ (/))、 または空白 (スペースまたはタブ) は含まれません。バックスラッシュ (\) でエスケープしないかぎり、改行 (newline) は文字列の終わりを示します。上記の制限を満たさない文字列は、単一引用符または二重引用符で囲まなければなりません。

例: 引用符の付かない文字列の並びによる入力

```
    CHARACTER C*6, S*8
    READ *, I, C, N, S
    PRINT *, I, C, N, S
    END
```

上記のプログラムで unquoted.f を入力すると以下のように表示されます。

```
demo% a.out
23 label 82 locked
  23label 82locked
demo%
```

内部入出力

f77 は、内部入出力ができるように並びによる入出力を拡張しています。♦

並びによる内部読み込みでは、入力並びが完成するかファイルの終わりになるまで文字が詰められています。並びによる内部書き込みでは、出力並びが完成するまで記録が埋められます。倍精度値の書き込みの際の論理記録オーバーフローを避けるために、内部配列要素の長さは 20 文字以上にするとよいでしょう。並びによる内部読み込みは、コマンド行の解読を簡単にするために実現されました。並びによる内部出力は避けることをお勧めします。

NAMELIST 入出力

NAMELIST 入出力を使用すると、変数の全グループの自由形式入出力、または変数のグループ内の選択された項目の入力が実行できます。♦

NAMELIST 文は、変数または配列のグループを定義します。グループ名を宣言し、そのグループに属する変数および配列を指定します。

構文規則

NAMELIST 文の構文は以下のとおりです。

NAMELIST */group-name/namelist[[,] /group-name/namelist]…*

パラメータ	説明
group-name	グループ名
namelist	変数または配列をコンマで分離した並び

詳細については、180 ページの「NAMELIST」を参照してください。

例: NAMELIST 文

```
    CHARACTER*18 SAMPLE
    LOGICAL*4 NEW
    REAL*4 DELTA
    NAMELIST /CASE/ SAMPLE, NEW, DELTA
```

変数または配列は、複数の NAMELIST にリストできます。

入力データには、配列要素および文字列を入れることができます。また、入力定数データ文字列は宣言された変数の長さより短くできるため、部分列も入れることができます。

制限事項

グループ名は、NAMELIST、READ、または WRITE の各文だけに置くことができ、プログラムに対して一意でなければなりません。

並びには、定数、配列要素、仮大きさ引き継ぎ配列、構造体、部分列、記録、記録欄、ポインタ、またはポインタ基底付き変数を入れることはできません。

例: 2 個の NAMELIST 内にある変数

```
    REAL ARRAY(4,4)
    CHARACTER*18 SAMPLE
    LOGICAL*4 NEW
    REAL*4 DELTA
    NAMELIST /CASE/ SAMPLE, NEW, DELTA
    NAMELIST /GRID/ ARRAY, DELTA
```

上記の例で、DELTA は CASE グループと GRID グループの両方にあります。

出力動作

NAMELIST 出力は、WRITE 文の特殊な書式を使用します。この出力はグループ名を含むレポートを作成します。グループの各変数の名前とメモリー上の現在の値も含めます。それぞれの変数の型に従って値を書式化し、NAMELIST の入力が読み込める形式のレポートを作成します。

NAMELIST WRITE の構文は次のとおりです。

```
WRITE ( extu,  変数群指定子 [, iostat] [, err])
```

ここで、変数群指定子は次の書式を使用します。

```
[NML=]グループ名
```

また、グループ名はすでに NAMELIST 文の中で定義されたものです。

NAMELIST WRITE 文は、グループ内のすべての変数の値を NAMELIST 文と同じ順番で書き込みます。

例: NAMELIST 出力

```
demo% cat nam1.f
* nam1.f Namelist output
    CHARACTER*8 SAMPLE
    LOGICAL*4 NEW
    REAL*4 DELTA
    NAMELIST /CASE/ SAMPLE, NEW, DELTA
    DATA SAMPLE /'Demo'/, NEW /.TRUE./, DELTA /0.1/
    WRITE ( *, CASE )
    END
demo% f77 nam1.f
f77 nam1.f
nam1.f:
 MAIN:
demo% a.out
Δ&case sample= Demo , new= T, delta= 0.100000
Δ&end
demo%
```

キーワード NML を省略する場合は、先頭に装置パラメータ (UNIT) を、2 番目に変数群指定子を置く必要があることに注意してください。書式指定子は置かないでください。

WRITE には次の書式も使用できます。

```
    WRITE ( UNIT=6, NML=CASE )
```

入力動作

NAMELIST 入力文は、第 1 桁を飛ばして第 2 桁以降で記号 $ と、指定されたグループ名を探し、次の外部記録を読み込みます。

$ グループ名が見つからない場合、入力グループはファイルの最後まで読み込まれます。

記録は入力されて、グループの中の名前と一致するデータ中の名前により値が割り当てられ、グループの中の変数のデータ型が使用されます。

入力データの中に見つからないグループの変数は、変更されません。

NAMELIST READ の構文は次のとおりです。

```
READ ( extu, 変数群指定子 [, iostat] [, err] [, end])
```

変数群指定子は次の書式を使用します。

```
[NML=]グループ名
```

また、グループ名はすでに NAMELIST 文の中で定義されたものです。

例: NAMELIST 入力

```
    CHARACTER*16 SAMPLE
    LOGICAL*4 NEW
    REAL*4 DELTA, MAT(2,2)
    NAMELIST /CASE/ SAMPLE, NEW, DELTA, MAT
    READ ( 1, CASE )
```

この例で、グループ CASE は SAMPLE、NEW、DELTA の 3 個の変数と MAT で構成されます。キーワード NML を省略する場合は、キーワードの UNIT も省略しなければなりません。装置パラメータは先頭に、変数群指定子は 2 番目に置きます。書式指定子は置きません。

READ には次の書式も使用できます。

```
    READ ( UNIT=1, NML=CASE )
```

データ構文

NAMELIST 入力データの最初の記録は、第 2 桁以降に特殊記号の $ (ドル記号) を持ち、その後に NAMELIST グループ名が続きます。この後に、同じ記録かこれに続く記録の第 2 桁以降から始まる一連の代入文が続きます。各記録は、指定されたグループの変数に値を代入します (または 1 個以上の値を配列要素に代入します)。次に示すように、第 2 桁以降に入っているもう 1 つの $ が入力データの終わりを示します。

```
Δ$グループ名変数 = 値 [,変数 = 値,…] $[END]
```

このドル記号の代りにアンパサンド (&) を使用できます。ただし、開始と終了の区切り文字は同じものでなければなりません。END は最後の区切り文字のオプションの部分です。

入力データ代入文の書式は、以下のいずれかでなければなりません。

変数 = 値

配列 = 値 1[, 値 2,]…

配列(添え字)= 値 1[, 値 2,]…

配列(添え字,添え字)= 値 1[, 値 2,]…

変数 =文字定数

変数 (索引:索引)=文字定数

配列に添字が付いている場合、適切な添字の個数がなければなりません。

文字定数を区切るには、引用符 (" または ') を使用してください。文字定数の詳細については、次の項を参照してください。

次のデータ例は、上記のプログラム部分で読み込むためのものです。

```
Δ$case delta=0.05, mat( 2, 2 ) = 2.2, sample='デモ' $
```

データは複数の行にまたがることが可能です。ここで、NEW は入力されませんでした。また、順番は NAMELIST 文の中の順番と異なります。

```
Δ$case
Δdelta=0.05
Δmat( 2, 2 ) = 2.2
Δsample='Demo'
Δ$
```

構文規則

NAMELIST で読み込まれる入力データには、次の構文規則が適用されます。

- 名前が指定されたグループの変数の順番は任意です。またどれでも省略できます。
- データは第 2 桁以降から始まります。第 1 桁から始めることもできますが、規格外です。♦
- 変数の間には、コンマ、スペース、またはタブが少なくとも 1 つ必要です。また、1 個以上のスペースまたはタブは 1 個のスペースに相当します。変数名の前に連続するコンマを置くことはできません。コンマの前後のスペースには意味がありません。
- グループ名および変数名の中にスペース、またはタブを置くことはできません。ただし、添字のコンマの前後、部分列のコロンの前後、および「(」の後と「)」の前のスペース、またはタブは認められます。1 つの名前が 2 つの記録に渡ることはできません。
- 記録の終わりはスペース文字のように処理されます。

例外: 文字定数では、記録の終わりは無視され、文字定数は次の記録へ続きます。現在の記録の最後の文字には、次の記録の 2 番目の文字が続きます。どの記録も最初の文字は無視されます。

- 代入文の等号の両側には、空白またはタブが使用できます。
- 添字、部分列の範囲指示子、および変数または配列に代入される値に使用できるのは定数値だけです。実際の入力データに定数名 (パラメータ) は使用できません。

ホレリス定数、8 進定数、16 進定数は使用できません。

代入された定数は、対応する Fortran 77 定数と同じ書式です。

定数と定数の間には、少なくとも 1 個のコンマ、スペース、またはタブが必要です。スペースまたはタブが無い場合も、1 個以上ある場合もスペース 1 個に相当します。1,2,3,1 2 3 または 1, 2, 3 などのように入力します。

文字定数の中の連続したスペースまたはタブは、圧縮されずに保持されます。

文字定数はアポストロフィ (') または引用符 (") で区切ります。ただし、どちらかを使用して文字定数を始めた場合、同じもので終了しなければなりません。アポストロフィを区切り文字として使用する場合、文字列の中にアポストロフィを入れるにはアポストロフィを 2 個続けて使用します。

例: 文字定数

```
Δsample='use "$" in 2'(use "$" in 2 と解釈される)
Δsample='don''t'(don't と解釈される)
Δsample="don''t"(don''t と解釈される)
Δsample="don't"(don't と解釈される)
```

複素定数は、実定数または整定数を 2 個組み合わせ、それをコンマで区切り、括弧で囲みます。スペースはコンマの前後にだけ入れることができます。

論理定数は、真または偽の値の書式です。たとえば、.TRUE. または .FALSE.、もしくは .T、.F などで始まる値です。

NULL データ項目は連続する 2 個のコンマで表され、これに対応する配列要素または複素数変数の値が変更されないことを示します。 NULL データ項目は、配列要素または複素数変数についてだけ使用できます。1 個の NULL データ項目が 1 個の複素定数全体を表します。複素定数の一方の部 (虚部または実部) としては使用できません。

例: NULL データの入った変数群読み込み

```
* nam2.f 連続したコンマのある変数群入力
    REAL ARRAY(4,4)
    NAMELIST /GRID/ ARRAY
    WRITE ( *, * ) '入力 ?'
    READ ( *, GRID )
    WRITE ( *, GRID )
    END
```

nam2.f のデータ

```
Δ$GRID ARRAY = 9,9,9,9,,,,,8,8,8,8 $
```

この結果、9 が配列 ARRAY の第 1 行にロードされ、4 個の要素を飛ばして 8 が第 3 行に読み込まれます。

配列専用

書式 *r***c* および *r** は配列だけで使用できます。

書式 *r***c* では、定数 *c* を *r* 個複写したものが配列に格納されます。ここで、*r* はゼロ以外の符号なし整定数で、*c* は任意の定数です。

例: データに反復因子がある変数群

```
* nam3.f 変数群 "r*c" と "r* "
    REAL PSI(10)
    NAMELIST /GRID/ PSI
    WRITE ( *, * ) '入力 ?'
    READ ( *, GRID )
    WRITE ( *, GRID )
    END
```

nam3.f の入力

```
Δ$GRID PSI = 5*980 $
```

上記の nam3.f プログラムでは、上記の入力を読み込み、980.0 を配列 PSI の最初の 5 個の要素に読み込みます。

- 書式 *r** は、配列の *r* 個の要素を飛ばします (つまりこれらは変更されません)。ここで、*r* は符号なし整定数です。

例: 読み飛ばされるデータのある変数群読み込み

その他の入力

```
Δ$GRID PSI = 3* 5*980 $
```

nam3.f プログラムは、上記の入力では最初の 3 個の要素を飛ばし、980.0 を PSI の要素
4、5、6、7、8 に読み込みます。

名前の要求

端末からの変数群入力の実行中に、使用するグループ名と変数群名を要求できます。これには、第 2 桁に疑問符 (?) を入力し、Return キーを押します。グループ名とそのグループの変数名が表示され、次の入力を待ちます。

例: 名前の要求

```
demo% cat nam4.f
* nam4.f 変数群： 名前の要求
    CHARACTER*14 SAMPLE
    LOGICAL*4 NEW
    REAL*4 DELTA
    NAMELIST /CASE/ SAMPLE, NEW, DELTA
    WRITE ( *, * ) '入力 ?'
    READ ( *, CASE )
    END
demo% f77 -silent nam4.f
demo% a.out
  入力 ?
Δ?          ← キーボード入力
Δ$case
Δsample
Δnew
Δdelta
D$end

Δ$case sample="Test 2", delta=0.03 $ ← キーボード入力
demo%
```

第6章

組み込み関数

この章では、Sun WorkShop FORTRAN 77 の一部である組み込み関数を表に示します。Fortran のライブラリルーチンについての詳細は、『Fortran ライブラリ・リファレンス』を参照してください。

ANSI FORTRAN 77 規格以外の組み込み関数には、♦ 印を付けています。

組み込み関数が複数のデータ型の引数を受け取る場合、組み込み関数には個別名と総称名があります。通常、個別名を使用した場合の戻り値は、引数と同じデータ型になりますが、型変換関数 (表 6-2) や照会関数 (表 6-7) などの例外もあります。特定のデータ型の引数を扱う場合には、個別名によって関数を呼び出します。

複数のデータ項目 (たとえば `sign(a1,a2)`) を扱う関数では、すべての引数が同じデータ型である必要があります。

それぞれの組み込み関数について、以下の項目が示されています。

- 組み込み関数 – 関数の説明
- 定義 – 数学的な定義
- 引数の個数 – 関数が受け取る引数の個数
- 総称名 – 関数の総称名
- 個別名 – 関数の個別名
- 引数のデータ型 – 各個別名に対応するデータ型
- 関数のデータ型 – 特定のデータ型の引数に対する戻り値のデータ型

注 – `-dbl`、`-i2`、`-r8`、`-xtypemap` オプションを指定すると、変数のデフォルトのサイズが変わり、組み込み関数の引用に影響があります。324 ページの「注意」、および 22 ページの「データ型のサイズと整列条件」に記述されたデフォルトサイズと整列条件を参照してください。

算術関数と数学関数

算術関数、型変換関数、三角関数、その他の数学関数について説明します。

a は、関数の 1 つの引数を表わします。a1 および a2 はそれぞれ、関数の 1 つ目の引数、2 つ目の引数を表わしています。ar および ai はそれぞれ、関数の複素の引数の実部と虚部を表わしています。

`REAL*16` および `COMPLEX*12` は、SPARC のみ有効です。

算術関数

表 6-1　算術関数

組み込み関数	定義	引数の数	総称名	個別名	引数の型	関数の型
絶対値 注 (6) 参照	|a|= $(ar^2 + ai^2)^{1/2}$	1	ABS	IABS ABS DABS CABS QABS ✠ ZABS ✠ CDABS ✠ CQABS ✠	INTEGER REAL DOUBLE COMPLEX REAL*16 DOUBLE COMPLEX DOUBLE COMPLEX COMPLEX*32	INTEGER REAL DOUBLE REAL REAL*16 DOUBLE DOUBLE REAL*16
切り捨て 注 (1) 参照	int(a)	1	AINT	AINT DINT QINT ✠	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16
四捨五入	a 0 の場合 int(a+.5) a < 0 の場合 nt(a-.5)	1	AINT	ANINT DNINT QNINT ✠	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16
四捨五入の 整数化	a 0 の場合 int(a+.5) a < 0 の場合 nt(a-.5)	1	NINT	NINT IDNINT IQNINT ✠	REAL DOUBLE REAL*16	INTEGER INTEGER INTEGER
剰余 注 (1) 参照	a1-int(a1 / a2)*a2	2	MOD	MOD AMOD DMOD QMOD ✠	INTEGER REAL DOUBLE REAL*16	INTEGER REAL DOUBLE REAL*16
符号の 付け替え	a2 0 の場合 |a1| a2 < 0 の場合 -|a1|	2	SIGN	ISIGN SIGN DSIGN QSIGN ✠	INTEGER REAL DOUBLE REAL*16	INTEGER REAL DOUBLE REAL*16
超過分	a1 > a2 の場合 a1 ≤ a2 の場合 a1-a20	2	DIM	IDIM DIM DDIM QDIM ✠	INTEGER REAL DOUBLE REAL*16	INTEGER REAL DOUBLE REAL*16
倍精度化 または 4 倍精度化 乗算	a1 * a2	2	-	DPROD QPROD ✠	REAL DOUBLE	DOUBLE REAL*16

表 6-1　算術関数

組み込み関数	定義	引数の数	総称名	個別名	引数の型	関数の型
			AMAX0	AMAX0	INTEGER	REAL
			MAX1	MAX1	REAL	INTEGER
			AMIN0	AMIN0	INTEGER	REAL
			MIN1	MIN1	REAL	INTEGER

型変換関数

表 6-2　型変換関数

変換型 (以下の型への変換)	引数の数	総称名	個別名	型 引数	型 関数
整数 注 (1) 参照	1	INT	-	INTEGER	INTEGER
			INT	REAL	INTEGER
			IFIX	REAL	INTEGER
			IDINT	DOUBLE	INTEGER
			-	COMPLEX	INTEGER
			-	COMPLEX*16	INTEGER
			-	COMPLEX*32	INTEGER
			IQINT ♦	REAL*16	INTEGER
実数 注 (2) 参照	1	REAL	REAL	INTEGER	REAL
			FLOAT	INTEGER	REAL
			-	REAL	REAL
			SNGL	DOUBLE	REAL
			SNGLQ ♦	REAL*16	REAL
			-	COMPLEX	REAL
			-	COMPLEX*16	REAL
			-	COMPLEX*32	REAL
倍精度 注 (3) 参照	1	DBLE	DBLE	INTEGER	DOUBLE PRECISION
			DFLOAT	INTEGER	DOUBLE PRECISION
			DREAL ♦	REAL	DOUBLE PRECISION
			-	DOUBLE	DOUBLE PRECISION
			-	COMPLEX	DOUBLE PRECISION
			-	COMPLEX*16	DOUBLE PRECISION
			DBLEQ ♦	REAL*16	DOUBLE PRECISION
			-	COMPLEX*32	DOUBLE PRECISION
				REAL*16	DOUBLE PRECISION
				COMPLEX*32	DOUBLE PRECISION
4 倍精度実数 注 (3') 参照	1	QREAL ♦	QREAL ♦	INTEGER	REAL*16
		QEXT ♦	QFLOAT ♦	INTEGER	REAL*16
			-	REAL	REAL*16
			QEXT ♦	INTEGER	REAL*16
			QEXTD ♦	DOUBLE	REAL*16
			-	REAL*16	REAL*16
			-	COMPLEX	REAL*16
			-	COMPLEX*16	REAL*16
			-	COMPLEX*32	REAL*16

表 6-2　型変換関数 (続き)

変換型 (以下の型への変換)	引数の数	総称名	個別名	型 引数	型 関数
複素数 注 (4) と (8) 参照	1	CMPLX	-	INTEGER	COMPLEX
			-	REAL	COMPLEX
	または		-	DOUBLE	COMPLEX
			-	REAL*16	COMPLEX
	2		-	COMPLEX	COMPLEX
			-	COMPLEX*16	COMPLEX
			-	COMPLEX*32	COMPLEX
倍精度複素数 注 (8) 参照	1	DCMPLX♦	-	INTEGER	DOUBLE COMPLEX
			-	REAL	DOUBLE COMPLEX
	または		-	DOUBLE	DOUBLE COMPLEX
			-	REAL*16	DOUBLE COMPLEX
	2		-	COMPLEX	DOUBLE COMPLEX
			-	COMPLEX*16	DOUBLE COMPLEX
			-	COMPLEX*32	DOUBLE COMPLEX
4 倍精度複素数 注 (8) 参照	1	QCMPLX♦	-	INTEGER	COMPLEX*32
			-	REAL	COMPLEX*32
	または		-	DOUBLE	COMPLEX*32
			-	REAL*16	COMPLEX*32
	2		-	COMPLEX	COMPLEX*32
			-	COMPLEX*16	COMPLEX*32
			-	COMPLEX*32	COMPLEX*32
整数 注 (5) 参照	1	-	ICHAR	CHARACTER	INTEGER
		-	IACHAR ♦		
文字 注 (5) 参照	1	-	CHAR	INTEGER	CHARACTER
		-	ACHAR ♦		

ASCII プラットフォーム (Sun システムも含む) では、次のようになります。

- ACHAR は CHAR の規格外の同義語です。
- IACHAR は ICHAR の規格外の同義語です。

ACHAR と IACHAR は、非 ASCII プラットフォーム用に ASCII を直接処理するための目的で提供されていました。

三角関数

表 6-3　三角関数

組み込み関数	定義	引数の数	総称名	個別名	型 引数	型 関数
正弦 注 (7) 参照	sin(a)	1	SIN	SIN DSIN QSIN ♦ CSIN ZSIN ♦ CDSIN ♦ CQSIN ♦	REAL DOUBLE REAL*16 COMPLEX DOUBLE COMPLEX DOUBLE COMPLEX COMPLEX*32	REAL DOUBLE REAL*16 COMPLEX DOUBLE COMPLEX DOUBLE COMPLEX COMPLEX*32
正弦 (度) 注 (7) 参照	sin(a)	1	SIND ♦	SIND ♦ DSIND ♦ QSIND ♦	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16
余弦 注 (7) 参照	cos(a)	1	COS	COS DCOS QCOS ♦ CCOS ZCOS ♦ CDCOS ♦ CQCOS ♦	REAL DOUBLE REAL*16 COMPLEX DOUBLE COMPLEX DOUBLE COMPLEX COMPLEX*32	REAL DOUBLE REAL*16 COMPLEX DOUBLE COMPLEX DOUBLE COMPLEX COMPLEX*32
余弦 (度) 注 (7) 参照	cos(a)	1	COSD ♦	COSD ♦ DCOSD ♦ QCOSD ♦	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16
正接 注 (7) 参照	tan(a)	1	TAN	TAN DTAN QTAN ♦	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16
正接 (度) 注 (7) 参照	tan(a)	1	TAND ♦	TAND ♦ DTAND ♦ QTAND ♦	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16
逆正弦 注 (7) 参照	arcsin(a)	1	ASIN	ASIN DASIN QASIN ♦	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16
逆正弦 (度) 注 (7) 参照	arcsin(a)	1	ASIND ♦	ASIND ♦ DASIND ♦ QASIND ♦	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16
逆余弦 注 (7) 参照	arccos(a)	1	ACOS	ACOS DACOS QACOS ♦	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16

表 6-3　三角関数 (続き)

組み込み関数	定義	引数の数	総称名	個別名	型 引数	型 関数
逆余弦 (度) 注 (7) 参照	arccos(a)	1	ACOSD ♦	ACOSD ♦ DACOSD ♦ QACOSD ♦	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16
逆正接 注 (7) 参照	arctan(a)	1	ATAN	ATAN DATAN QATAN ♦	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16
	arctan (a1/a2)	2	ATAN2	ATAN2 DATAN2 QATAN2 ♦	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16
逆正接 (度) 注 (7) 参照	arctan(a)	1	ATAND ♦	ATAND ♦ DATAND ♦ QATAND ♦	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16
	arctan (a1/a2)	2	ATAN2D ♦	ATAN2D ♦ DATAN2D♦ QATAN2D♦	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16
双曲線正弦 注 (7) 参照	sinh(a)	1	SINH	SINH DSINH QSINH ♦	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16
双曲線余弦 注 (7) 参照	cosh(a)	1	COSH	COSH DCOSH QCOSH ♦	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16
双曲線正接 注 (7) 参照	tanh(a)	1	TANH	TANH DTANH QTANH ♦	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16

その他の数学関数

表 6-4　その他の数学関数

組み込み関数	定義	引数の数	総称名	個別名	型 引数	型 関数
複素数の虚部 注 (6) 参照	ai	1	IMAG	AIMAG	COMPLEX	REAL
				DIMAG ♦	DOUBLE COMPLEX	DOUBLE
				QIMAG ♦	COMPLEX*32	REAL*16
複素数の共役 注 (6)参照	(ar,-ai)	1	CONJG	CONJG	COMPLEX	COMPLEX
				DCONJG ♦	DOUBLE COMPLEX	DOUBLE COMPLEX
				QCONJG ♦	COMPLEX*32	COMPLEX*32
平方根	a**(1/2)	1	SQRT	SQRT	REAL	REAL
				DSQRT	DOUBLE	DOUBLE
				QSQRT ♦	REAL*16	REAL*16
				CSQRT	COMPLEX	COMPLEX
				ZSQRT ♦	DOUBLE COMPLEX	DOUBLE COMPLEX
				CDSQRT ♦	DOUBLE COMPLEX	DOUBLE COMPLEX
				CQSQRT ♦	COMPLEX*32	COMPLEX*32
立方根 注 (8') 参照	a**(1/3)	1	CBRT	CBRT ♦	REAL	REAL
				DCBRT ♦	DOUBLE	DOUBLE
				QCBRT ♦	REAL*16	REAL*16
				CCBRT ♦	COMPLEX	COMPLEX
				ZCBRT ♦	DOUBLE COMPLEX	DOUBLE COMPLEX
				CDCBRT ♦	DOUBLE COMPLEX	DOUBLE COMPLEX
				CQCBRT ♦	COMPLEX*32	COMPLEX*32
指数関数	e**a	1	EXP	EXP	REAL	REAL
				DEXP	DOUBLE	DOUBLE
				QEXP ♦	REAL*16	REAL*16
				CEXP	COMPLEX	COMPLEX
				ZEXP ♦	DOUBLE COMPLEX	DOUBLE COMPLEX
				CDEXP ♦	DOUBLE COMPLEX	DOUBLE COMPLEX
				CQEXP ♦	COMPLEX*32	COMPLEX*32
自然対数	log(a)	1	LOG	ALOG	REAL	REAL
				DLOG	DOUBLE	DOUBLE
				QLOG ♦	REAL*16	REAL*16
				CLOG	COMPLEX	COMPLEX
				ZLOG ♦	DOUBLE COMPLEX	DOUBLE COMPLEX
				CDLOG ♦	DOUBLE COMPLEX	DOUBLE COMPLEX
				CQLOG ♦	COMPLEX*32	COMPLEX*32

表 6-4　その他の数学関数 (続き)

組み込み関数	定義	引数の数	総称名	個別名	型 引数	型 関数
常用対数	log10(a)	1	LOG10	ALOG10 DLOG10 QLOG10 ♦	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16
誤差関数 (以下参照)	erf(a)	1	ERR	ERF ♦ DERF ♦	REAL DOUBLE	REAL DOUBLE
誤差関数	1.0 - erf(a)	1	ERR	ERFC ♦ DERFC ♦	REAL DOUBLE	REAL DOUBLE

- 誤差関数 : 2/sqrt (pi)* x $\int_{o}^{a} \exp(-t*t)dt$

文字関数

表 6-5　文字関数

組み込み関数	定義	引数の数	個別名	引数の型	関数の型
変換 注 (5) 参照	文字への変換	1	CHAR ACHAR ✠	INTEGER	CHARACTER
	整数への変換 (表 6-2 参照)	1	ICHAR IACHAR ✠	CHARACTER	INTEGER
部分列の索引	文字列 a1 の中の部分列 a2 の位置 注 (10) 参照	2	INDEX	CHARACTER	INTEGER
長さ	文字本体の長さ 注 (11) 参照	1	LEN	CHARACTER	INTEGER
字句的に等しいか大きい	a1 a2 注 (12) 参照	2	LGE	CHARACTER	LOGICAL
字句的に大きい	a1 > a2 注 (12) 参照	2	LGT	CHARACTER	LOGICAL
字句的に等しいか小さい	a1 ≤ a2 注 (12) 参照	2	LLE	CHARACTER	LOGICAL
字句的に小さい	a1 < a2 注 (12) 参照	2	LLT	CHARACTER	LOGICAL

ASCII プラットフォーム (Sun システムも含む) では、次のようになります。

- ACHAR は CHAR の規格外の同義語です。
- IACHAR は ICHAR の規格外の同義語です。

ACHAR と IACHAR は、非 ASCII プラットフォーム用に ASCII を直接処理するための目的で提供されていました。

その他の関数

ビット単位関数、環境照会関数、記憶領域の割り当ておよび割り当て解除関数について説明します。

ビット操作 ♦

以下の関数はすべて FOTRAN 77 の規格外です。

表 6-6　ビット単位関数

			型	
ビット単位操作	**引数の数**	**個別名**	**引数**	**関数**
補数	1	NOT	INTEGER	INTEGER
論理積	2	AND	INTEGER	INTEGER
	2	IAND	INTEGER	INTEGER
内包的論理和	2	OR	INTEGER	INTEGER
	2	IOR	INTEGER	INTEGER
排他的論理和	2	XOR	INTEGER	INTEGER
	2	IEOR	INTEGER	INTEGER
シフト 注 (14) 参照	2	ISHFT	INTEGER	INTEGER
左シフト 注 (14) 参照	2	LSHIFT	INTEGER	INTEGER
右シフト 注 (14) 参照	2	RSHIFT	INTEGER	INTEGER
論理的右シフト 注 (14) 参照	2	LRSHFT	INTEGER	INTEGER
循環シフト	3	ISHFTC	INTEGER	INTEGER
ビット抽出	3	IBITS	INTEGER	INTEGER
ビットセット	2	IBSET	INTEGER	INTEGER
ビットテスト	2	BTEST	INTEGER	LOGICAL
ビットクリアー	2	IBCLR	INTEGER	INTEGER

上記の関数は、組み込み関数または外部関数として使用可能です。 ライブラリのビット単位操作ルーチンについての詳細は、『Fortran ライブラリ・リファレンス』を参照してください。

環境照会関数 ♦

以下の関数はすべて FOTRAN 77 の規格外です。

表 6-7　環境照会関数

			型	
定義	**引数の数**	**総称**	**引数**	**関数**
進法の基底	1	EPBASE	INTEGER REAL DOUBLE REAL*16	INTEGER INTEGER INTEGER INTEGER
有効ビット数	1	EPPREC	INTEGER REAL DOUBLE REAL*16	INTEGER INTEGER INTEGER INTEGER
最小指数	1	EPEMIN	REAL DOUBLE REAL*16	INTEGER INTEGER INTEGER
最大指数	1	EPEMAX	REAL DOUBLE REAL*16	INTEGER INTEGER INTEGER
最小非ゼロ数	1	EPTINY	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16
表現可能の最大	1	EPHUGE	INTEGER REAL DOUBLE REAL*16	INTEGER REAL DOUBLE REAL*16
イプシロン 注 (16) 参照	1	EPMRSP	REAL DOUBLE REAL*16	REAL DOUBLE REAL*16

メモリー ♦

以下の関数はすべて FOTRAN 77 の規格外です。

表 6-8　メモリー関数

				型	
組み込み関数	**定義**	**引数の数**	**個別名**	**引数**	**関数**
位置	アドレス 注 （17） 参照	1	LOC	任意	INTEGER*4 INTEGER*8
割り当て	記憶領域の割り当てアドレスを戻す。 注 （17） 参照	1	MALLOC MALLOC64	INTEGER*4 INTEGER*8	INTEGER INTEGER*8
割り当て解除	MALLOC で割り当てられた記憶領域の割り当て解除 注 （17） 参照	1	FREE	任意	-
サイズ	引数のサイズをバイト数で戻す。 注 （18） 参照	1	SIZEOF	任意の式	INTEGER

正確には、malloc (3F) と free (3F) は組み込み関数ではありません。これらについては、『Fortran ライブラリ・リファレンス』でも説明しています。さらに、isetjmp(3F)、longjmp(3F)、date_and_time(3F) などの規格外の組込み関数については、『Fortran ライブラリ・リファレンス』やマニュアルページで説明しています。

注意

以下の注意は、本章のすべての組み込み関数表に適用されます。

- DOUBLE は倍精度を意味します。
- INTEGER 引数を取る組み込み関数は、INTEGER*2、INTEGER*4 または INTEGER*8 も使用できます。

- INTEGER 値を戻す組み込み関数は、以下のような条件で INTEGER 型を返します。-i2、-dbl、-xtypemap オプションを指定すると、実引数のデフォルトのサイズが変わります。
  - mod、sign、dim、max、min、and、iand、or、ior、xor、および ieor の場合、戻り値のサイズは、引数の最大サイズになります。
  - abs、ishift、lshift、rshift、lrshft、ibset、btest、ivclr、ishftc、および ibits の場合、戻り値のサイズは、最初の引数のサイズになります。
  - int、epbase、epprec、および irshift、の場合、戻り値のサイズは、デフォルトの INTEGER のサイズになります。
  - ephuge の場合、戻り値のサイズは、INTEGER または引数の大きい方のサイズになります。
- デフォルトのデータサイズを変更するオプション (22 ページの「データ型のサイズと整列条件」を参照) の場合、一部の組み込み関数の使用方法も変わります。たとえば、-dbl オプションを指定している場合、DOUBLE COMPLEX の引数による ZCOS への呼び出しは、自動的に CQCOS への呼び出しになります。引数は COMPLEX*32 にならないためです。以下の関数にも、上述のような機能があります。

aimag	alog	amod	cabs	ccbrt	ccos	cdabs	cdcbrt	cdcos
cdexp	cdlog	cdsin	cdsqrt	cexp	clog	csin	csqrt	dabs
dacos	dacosd	dasin	dasind	datan	datand	dcbrt	dconjg	dcos
dcosd	dcosh	ddim	derf	derfc	dexp	dimag	dint	dlog
dmod	dnint	dprod	dsign	dsin	dsind	dsinh	dsqrt	dtan
dtand	dtanh	idnint	iidnnt	jidnnt	zabs	zcbrt	zcos	zexp
zlog	zsin	zsqrt						

以下の関数は、整数型または論理型の引数を扱うことができます。引数のサイズには制限はありません。

and	iand	ieor	iiand	iieor	iior	inot	ior	jiand
jieor	jior	jnot	lrshft	lshift	not	or	rshift	xor

- (SPARC のみ) デフォルトの REAL、DOUBLE PRECISION、COMPLEX、または DOUBLE COMPLEX 値を戻すよう指定された組み込み関数は、特定のコンパイル オプションに応じて型を返します (22 ページの「データ型のサイズと整列条件」を参照)。たとえば、-xtypemap=real:64,double:64 と指定してコンパイルした場合、結果は以下のようになります。
  - REAL 関数への呼び出しは REAL*8 を返します。
  - DOUBLE PRECISION 関数への呼び出しは REAL*8 を返します。
  - COMPLEX 関数への呼び出しは COMPLEX*16 を返します。
  - DOUBLE COMPLEX 関数への呼び出しは COMPLEX*16 を返します。

  そのほか、データ型のデフォルト データサイズを変更するオプションには -r8 と -db1 があります。これらのオプションも DOUBLE を QUAD に拡張します。ただし、-xtypemap= オプションの方が順応性があるため、こちらを使用することをお勧めします。
- 総称名を持つ関数は、引数と同じ型の値を返します。ただし、型変換関数、最も近い整数関数、複素数引数の絶対値などについては例外です。引数が複数ある場合、すべて同じ型でなければなりません。
- 関数名が実引数として使用される場合、この名前は個別名でなければなりません。
- 関数名が仮引数として使用される場合、これは副プログラムの中の組み込み関数を識別せず、そのデータ型は変数および配列の規則と同じ規則に従います。

関数の注記

表および、以下の注記 (1) 〜 (12) は、ANSI X3.9-1978 『Programming Language Fortran 』の「組み込み関数の表」に Fortran 拡張機能を追加したものにもとづいています。

(1) INT

A が整数型ならば、INT(A) は A です。

A が実数型または倍精度ならば、次のようになります。

A < 1 ならば、INT(A) はゼロ。
A ≧ 1 ならば、INT(A) は A の範囲を超えない最大整数で、A と同じ符号です (このような数学的整数値は、大きすぎてこのコンピュータの整数型に合わない場合があります)。

A が複素数型または倍精度複素数型ならば、上記の規則が A の実部に適用されます。

A が実数型ならば、IFIX(A) は INT(A) と同じです。

(2) REAL

A が実数ならば、REAL(A) は A です。

A が整数型または倍精度型ならば、REAL(A) は実数データが持ち得るのと同じ精度の、A の有効部分です。

A が複素数型ならば、REAL(A) は A の実部です。

A が倍精度複素数型ならば、REAL(A) は実数データが持ち得るのと同じ精度の、A の実部の有効部分です。

(3) DBLE

A が倍精度型ならば、DBLE(A) は A です。

A が整数型または実数型ならば、DBLE(A) は倍精度データが持ち得るのと同じ精度の、A の有効部分です。

A が複素数型ならば、DBLE(A) は倍精度データが持ち得るのと同じ精度の、A の実部の有効部分です。

A が複素数 *16 型ならば、DBLE(A) は A の実部です。

(3') QREAL

A が REAL*16 型ならば、QREAL(A) は A です。

A が整数型、実数型、または倍精度型ならば、QREAL(A) は REAL*16 データが持ち得るのと同じ精度の、A の有効部分です。

A が複素数型または倍精度複素数型ならば、QREAL(A) は REAL*16 データが持ち得るのと同じ精度の、A の実部の有効部分です。

A が CONPLEX*16 型または COMPLEX*32 型ならば、QREAL(A) は A の実部です。

(4) CMPLX

A が複素数型ならば、CMPLX(A) は A です。

A が整数型、実数型、または倍精度型ならば、CMPLX(A) は REAL(A) + 0i です。

A1 と A2 が整数型、実数型、または倍精度型ならば、CMPLX(A1,A2) は REAL(A1) + REAL(A2)*i です。

A が倍精度複素数型ならば CMPLX(A) は REAL(DBLE(A))+ i*REAL(DIMAG(A)) です。

CMPLX に引数が 2 個ある場合、同じ型でなければなりません。また、型は整数、実数、または倍精度のいずれかです。

CMPLX の引数が 1 個の場合、整数、実数、倍精度、複素数、CONPLEX*16 または COMPLEX*32 のいずれかです。

(4') DCMPLX

A が CONPLEX*16 型ならば、DCMPLX(A) は A です。

A が整数型、実数型、または倍精度型ならば、DCMPLX(A) は DBLE(A) + 0i です。

A1 と A2 が整数型、実数型、または倍精度型ならば、DCMPLX(A1,A2) は DBLE(A1) + DBLE(A2)*i です。

DCMPLX に引数が 2 個ある場合、同じ型でなければなりません。また、型は整数、実数、または倍精度のいずれかです。

DCMPLX の引数が 1 個の場合、整数、実数、倍精度、複素数、CONPLEX*16 型または COMPLEX*32 のいずれかです。

(5) ICHAR

ICHAR(A) は照合シーケンスの中の A の位置です。

先頭の位置は 0 で、最後は N-1、0≦ICHAR(A)≦N-1 です。ここで、N は照合シーケンスの中の文字数で、A は長さが 1 の文字型です。

CHAR および ICHAR は次に示すように逆の関係です。

- ICHAR(CHAR(I)) = I、このとき 0≦I≦N-1 。
- CHAR(ICHAR(C)) = C、ここで C はその CPU で表現できる任意の文字。

(6) 複素数

複素数値は順に並べた実数の組み合わせ (ar,ai) で表します。ここで、ar は実部で、ai は虚部です。

(7) ラジアン

角度はすべてラジアンで表します。ただし、"組み込み関数"の列に"(度)"の表記がある場合は除きます。

(8) 複素数の関数

複素数型の関数の結果は、主値です。

(8')CBRT

a が COMPLEX 型ならば、CBRT の結果は COMPLEX RT1=(A, B) となります。このとき、 A ≥ 0.0、 -60 度 ≤ arctan (B/A) < +60 度 です。

以下のようになる場合もあります。

- RT2 = RT1 * (-0.5, $\sqrt{0.75}$)
- RT3 = RT1 * (-0.5, $\sqrt{0.75}$)

(9) 引数型

組み込み関数引用の中のすべての引数は、同じ型でなければなりません。

(10)INDEX

INDEX(X,Y) は、X の中の Y が始まる場所です。つまり、文字列 X の中で文字列 Y が最初に始まる位置です。

Y が X の中にない場合は、INDEX(X,Y) は 0 です。

LEN(X) < LEN(Y) ならば、INDEX(X,Y) は 0 です。

INDEX はデフォルトの INTEGER*4 のデータを返します。64 ビット環境のコンパイルでは、結果が INTEGER*4 のデータ範囲を超えると警告が出されます。64 ビット環境で、INDEX 文に INTEGER*4 の上限 (2 G バイト) を超える文字列を使用する場合は、INDEX 関数と結果を受け取る変数が INTEGER*8 に宣言されていなければなりません。

(11)LEN の引数

LEN は、引数の CHARACTER 変数の宣言された長さを返します。引数の実際の値には重要性はありません。

LEN はデフォルトの INTEGER*4 のデータを返します。64 ビット環境のコンパイルでは、結果が INTEGER*4 のデータ範囲を超えると警告が出されます。64 ビット環境で、LEN 文に INTEGER*4 の上限 (2 Gバイト) を超える文字変数を使用する場合は、LEN 関数と結果を受け取る変数が INTEGER*8 に宣言されていなければなりません。

(12)字句比較

LGE(X, Y) は、X=Y または照合シーケンスの中で X が Y に続くならば真です。その他の場合は偽です。

LGT(X, Y) は、照合シーケンスの中で X が Y に続くならば真です。その他の場合は偽です。

LLE(X, Y) は、X=Y または照合シーケンスの中で、X が Y の前にあるならば真です。その他の場合は偽です。

LLT(X, Y) は、照合シーケンスの中で X が Y の前にあるならば真です。その他の場合は偽です。

LGE、LGT、LLE、および LLT のオペランドの長さが違うと、短い方のオペランドの右を空白で拡張したように見なされます。

(13)ビット関数

ビット単位操作の詳細については、361 ページの「VMS 言語拡張」を参照してください。

(14)シフト

LSHIFT は、*a1* を *a2* ビットだけ論理的に左にシフトします (インラインコード)。
LRSHFT は、*a1* を *a2* ビットだけ論理的に右にシフトします (インラインコード)。
RSHIFT は、*a1* を *a2* ビットだけ算術的に右にシフトします。
ISHIFT は、*a1* を *a2*>0 ならば論理的に左に、*a2*<0 ならば論理的に右にシフトします。

LSHIFT と RSHIFT 関数は、C の << および >> 演算子の Fortran の類似機能です。C と同様、その意味はハードウェアにより異なります。

範囲外のシフトカウントによるシフト関数の動作は、ハードウェアによって異なり、通常は予測できません。このリリースの Fortran では、31 を超えるシフトカウントは、ハードウェアによってことなります。

(15)環境照合

引数の型だけに意味があります。

(16)イプシロン

イプシロンは、1.0 + e ≠ 1.0 であるような最小の e です。

(17)LOC、MALLOC、FREE

LOC 関数は変数または外部手続きのアドレスを返します。MALLOC(*n*) 関数呼び出しは、少なくとも *n* バイトのブロックを割り当て、そのブロックのアドレスを返します。

LOC は、32 ビット環境ではデフォルトの INTEGER*4 を返し、64 ビット環境では INTEGER*8 を返します。

MALLOC はライブラリ関数であり、組み込み関数ではありません。MALLOC も同様に 32 ビット環境ではデフォルトの INTEGER*4 を返し、64 ビット環境では INTEGER*8 を返します。ただし、64 ビット環境用にコンパイルする場合は、MALLOC は明示的に INTEGER*8 と宣言されていなければなりません。

LOC または MALLOC から戻される値は、POINTER、INTEGER*4、または 64 ビット環境では INTEGER8* の型の変数 に格納されます。FREE に渡す引数は、その前の MALLOC への呼び出しによって戻された値でなければなりません。したがって、データ型は POINTER、INTEGER*4、または INTEGER8* になります。

MALLOC64 は、常に INTEGER*8 の引数 (バイト単位のメモリー要求のサイズ) を受け取り、常に INTEGER*8 の値を返します。32 ビット環境と 64 ビット環境の両方で稼働するプログラムをコンパイルしなければならない場合は、MALLOC ではなくこのルーチンを使用します。受け取る変数は POINTER または INTEGER*8 に宣言されていなければなりません。

(18)SIZEOF

SIZEOF 組み込み関数は、大きさ引き継ぎ配列、引き渡された文字の長さ、サブルーチン呼び出しや名前には適用できません。SIZEOF はデフォルトの INTEGER*4 のデータを返します。64 ビット環境用にコンパイルする場合は、結果が INTEGER*4 のデータ範囲を超えると警告が出されます。64 ビット環境で、SIZEOF 文に INTEGER*4 の上限 (2 Gバイト) を超える配列を使用する場合は、SIZEOF 関数と結果を受け取る変数が INTEGER*8 に宣言されていなければなりません。

VMS 組み込み関数 ♦

本節では、`f77` が識別する VMS Fortran 組み込みルーチンを表にして示します。これらは規格外です。♦

VMS 倍精度複素数

表 6-9 VMS 倍精度複素数関数

総称名	個別名	関数	引数の型	結果の型
	CDABS	絶対値	COMPLEX*16	REAL*8
	CDEXP	指数、e**a	COMPLEX*16	COMPLEX*16
	CDLOG	自然対数	COMPLEX*16	COMPLEX*16
	CDSQRT	平方根	COMPLEX*16	COMPLEX*16
	CDSIN	正弦	COMPLEX*16	COMPLEX*16
	CDCOS	余弦	COMPLEX*16	COMPLEX*16
DCMPLX		倍精度複素数への変換	任意の数字	COMPLEX*16
	DCONJG	共役複素数	COMPLEX*16	COMPLEX*16
	DIMAG	複素数の虚部	COMPLEX*16	REAL*8
	DREAL	複素数の実部	COMPLEX*16	REAL*8

VMS 度単位を用いる三角関数

表 6-10 vms 度単単位を用いる三角関数

総称名	個別名	関数	引数の型	結果の型
SIND		正弦	-	-
	SIND		REAL*4	REAL*4
	DSIND		REAL*8	REAL*8
	QSIND		REAL*16	REAL*16
COSD		余弦	-	-
	COSD		REAL*4	REAL*4
	DCOSD		REAL*8	REAL*8
	QCOSD		REAL*16	REAL*16
TAND		正接	-	-
	TAND		REAL*4	REAL*4
	DTAND		REAL*8	REAL*8
	QTAND		REAL*16	REAL*16
ASIND		逆正弦	-	-
	ASIND		REAL*4	REAL*4
	DASIND		REAL*8	REAL*8
	QASIND		REAL*16	REAL*16

表 6-10 vms 度単単位を用いる三角関数

総称名	個別名	関数	引数の型	結果の型
ACOSD		逆余弦	-	-
	ACOSD		REAL*4	REAL*4
	DACOSD		REAL*8	REAL*8
	QACOSD		REAL*16	REAL*16
ATAND		逆正接	-	-
	ATAND		REAL*4	REAL*4
	DATAND		REAL*8	REAL*8
	QATAND		REAL*16	REAL*16
ATAN2D		a1/a2 の 逆正接	-	-
	ATAN2D		REAL*4	REAL*4
	DATAN2D		REAL*8	REAL*8
	QATAN2D		REAL*16	REAL*16

VMS ビット操作

表 6-11 vms ビット操作関数

総称名	個別名	関数	引数の型	結果の型
IBITS		a1 から、初期ビット a2、a3 ビット抽出	-	-
	IIBITS		INTEGER*2	INTEGER*2
	JIBITS		INTEGER*4	INTEGER*4
ISHFT		a1 を論理的に a2 ビットシフト。a2 が正ならば左へ、負ならば右へシフト。	-	-
	IISHFT	a1 を論理的に左に a2 ビットシフト	INTEGER*2	INTEGER*2
	JISHFT	a1 を論理的に左に a2 ビットシフト	INTEGER*4	INTEGER*4
ISHFTC		a1 の右 a3 ビット、 a2 桁だけ循環シフト	-	-
	IISHFTC		INTEGER*2	INTEGER*2
	JISHFTC		INTEGER*4	INTEGER*4
IAND		a1 と a2 のビット単位論理積	-	-
	IIAND		INTEGER*2	INTEGER*2
	JIAND		INTEGER*4	INTEGER*4
IOR		a1 と a2 のビット単位論理和	-	-
	IIOR		INTEGER*2	INTEGER*2
	JIOR		INTEGER*4	INTEGER*4
IEOR		a1 と a2 のビット単位排他的論理積	-	-
	IIEOR		INTEGER*2	INTEGER*2
	JIEOR		INTEGER*4	INTEGER*4
NOT		ビット単位補数	-	-
	INOT		INTEGER*2	INTEGER*2
	JNOT		INTEGER*4	INTEGER*4
IBSET		a1 で、ビット a2 を 1 に設定	-	-
	IIBSET	a1 で、ビット a2 を 1 に設定し、新しい a1 を戻す	INTEGER*2	INTEGER*2
	JIBSET	a1 で、ビット a2 を 1 に設定し、新しい a1 を戻す	INTEGER*4	INTEGER*4
BTEST	BITEST	a1 のビット a2 が 1 ならば、.TURE. を戻す	-	-
	BJTEST		INTEGER*2	INTEGER*2
			INTEGER*4	INTEGER*4
IBCLR	IIBCLR	a1 で、ビット a2 を 0 に設定し、新しい a1 を戻す	-	-
	JIBCLR		INTEGER*2	INTEGER*2
			INTEGER*4	INTEGER*4

VMS 多重整数型

FORTRAN 77 規格では多重整数型を扱えるかどうかは表明されていません。f77 では特定の INTEGER から INTEGER 関数名 (IABS 等) を総称名の特別な種類として扱うことによって、複数の整数型に対処します。引数型を使用して適当な実行時ルーチン名が選択されますが、プログラマはこの名前を関知できません。

VMS Fortran は同じような方法を取りますが、個別名が使用できます。

表 6-12 VMS 整数関数

個別名	関数	引数の型	結果の型
IIABS	絶対値	INTEGER*2	INTEGER*2
JIABS	絶対値	INTEGER*4	INTEGER*4
IMAX0	最大 [1]	INTEGER*2	INTEGER*2
JMAX0	最大 [1]	INTEGER*4	INTEGER*4
IMIN0	最小 [1]	INTEGER*2	INTEGER*2
JMIN0	最小 [1]	INTEGER*4	INTEGER*4
IIDIM	超過分 [2]	INTEGER*2	INTEGER*2
JIDIM	超過分 [2]	INTEGER*4	INTEGER*4
IMOD	a1/a2 の剰余	INTEGER*2	INTEGER*2
JMOD	a1/a2 の剰余	INTEGER*4	INTEGER*4
IISIGN	符号の付け替え、|a1|* sign(a2)	INTEGER*2	INTEGER*2
JISIGN	符号の付け替え、|a1|* sign(a2)	INTEGER*4	INTEGER*4

1.引数は少なくても 2 個 でなければならない
2.超過分 a1-min(a1,a2)

特定型に強制的に解釈される関数

VMS Fortran 関数の中には、強制的に特定の INTEGER 型にする関数があります。

表 6-13 VMS が特定型へ強制的に変換する関数

個別名	関数	引数の型	結果の型
IINT	ゼロへ切り捨て	REAL*4	INTEGER*2
JINT	ゼロへ切り捨て	REAL*4	INTEGER*4
LINT	ゼロへ切り捨て	REAL*4	INTEGER*8
IIDINT	ゼロへ切り捨て	REAL*8	INTEGER*2
JIDINT	ゼロへ切り捨て	REAL*8	INTEGER*4
IIQINT	ゼロへ切り捨て	REAL*16	INTEGER*2
JIQINT	ゼロへ切り捨て	REAL*16	INTEGER*4
ININT	四捨五入、INT(a+.5*sign(a))	REAL*4	INTEGER*2
JNINT	四捨五入、INT(a+.5*sign(a))	REAL*4	INTEGER*4
IIDNNT	四捨五入、INT(a+.5*sign(a))	REAL*8	INTEGER*2
JIDNNT	四捨五入、INT(a+.5*sign(a))	REAL*8	INTEGER*4
IIQNNT	四捨五入、INT(a+.5*sign(a))	REAL*16	INTEGER*2
JIQNNT	四捨五入、INT(a+.5*sign(a))	REAL*16	INTEGER*4
IIFIX	固定	REAL*4	INTEGER*2
JIFIX	固定	REAL*4	INTEGER*4
IMAX1(*a,a2,...*)	2 つ以上の引数の最大値	REAL*4	INTEGER*2
JMAX1(*a,a2,...*)	2 つ以上の引数の最大値	REAL*4	INTEGER*4
IMIN1(*a,a2,...*)	2 つ以上の引数の最小値	READ*4	INTEGER*2
JMIN1(*a,a2,...*)	2 つ以上の引数の最小値	READ*4	INTEGER*4

総称名へ変換される関数

場合によっては、VMS 個別名は f77 総称名へ変換されます。

表 6-14 f77 総称名へ変換される VMS 関数

Specific Names	Function	Argument Type	Result Type
FLOATI	REAL*4 へ変換	INTEGER*2	REAL*4
FLOATJ	REAL*4 へ変換	INTEGER*4	REAL*4
DFLOTI	REAL*8 へ変換	INTEGER*2	REAL*8
DFLOTJ	REAL*8 へ変換	INTEGER*4	REAL*8
AIMAX0	最大	INTEGER*2	REAL*4
AJMAX0	最大	INTEGER*4	REAL*4
AIMIN0	最小	INTEGER*2	REAL*4
AJMIN0	最小	INTEGER*4	REAL*4

ゼロ拡張

次のゼロ拡張関数は、f77 により識別されます。最初使用されない上位ビットはゼロに設定され、表で示された幅まで最上位ビットの方向へ拡張されます。

表 6-15 ゼロ拡張変換

総称名	個別名	関数	引数の型	結果の型
ZEXT		ゼロ拡張	-	-
	IZEXT	ゼロ拡張	BYTE LOGICAL*1 LOGICAL*2 INTEGER*2	INTEGER*2
	JZEXT	ゼロ拡張	BYTE LOGICAL*1 LOGICAL*2 LOGICAL*4 INTEGER INTEGER*2 INTEGER*4	INTEGER*4

付録 A

ASCII 文字セット

この付録では、ASCII 文字セットおよび制御文字に関する表を示します。

表 A-1 ASCII 文字セット

10進	8 進	16進	名称	10進	8 進	16進	名称	10進	8 進	16進	名称	10進	8 進	16進	名称
0	000	00	NUL	32	040	20	SP	64	100	40	@	96	140	60	`
1	001	01	SOH	33	041	21	!	65	101	41	A	97	141	61	a
2	002	02	STX	34	042	22	"	66	102	42	B	98	142	62	b
3	003	03	ETX	35	043	23	#	67	103	43	C	99	143	63	c
4	004	04	EOT	36	044	24	$	68	104	44	D	100	144	64	d
5	005	05	ENQ	37	045	25	%	69	105	45	E	101	145	65	e
6	006	06	ACK	38	046	26	&	70	106	46	F	102	146	66	f
7	007	07	BEL	39	047	27	'	71	107	47	G	103	147	67	g
8	010	08	BS	40	050	28	(	72	110	48	H	104	150	68	h
9	011	09	HT	41	051	29	)	73	111	49	I	105	151	69	i
10	012	0A	LF	42	052	2A	*	74	112	4A	J	106	152	6A	j
11	013	0B	VT	43	053	2B	+	75	113	4B	K	107	153	6B	k
12	014	0C	FF	44	054	2C	,	76	114	4C	L	108	154	6C	l
13	015	0D	CR	45	055	2D	-	77	115	4D	M	109	155	6D	m
14	016	0E	SO	46	056	2E	.	78	116	4E	N	110	156	6E	n
15	017	0F	SI	47	057	2F	/	79	117	4F	O	111	157	6F	o
16	020	10	DLE	48	060	30	0	80	120	50	P	112	160	70	p
17	021	11	DC1	49	061	31	1	81	121	51	Q	113	161	71	q
18	022	12	DC2	50	062	32	2	82	122	52	R	114	162	72	r
19	023	13	DC3	51	063	33	3	83	123	53	S	115	163	73	s
20	024	14	DC4	52	064	34	4	84	124	54	T	116	164	74	t
21	025	15	NAK	53	065	35	5	85	125	55	U	117	165	75	u
22	026	16	SYN	54	066	36	6	86	126	56	V	118	166	76	v
23	027	17	ETB	55	067	37	7	87	127	57	W	119	167	77	w
24	030	18	CAN	56	070	38	8	88	130	58	X	120	170	78	x
25	031	19	EM	57	071	39	9	89	131	59	Y	121	171	79	y
26	032	1A	SUB	58	072	3A	:	90	132	5A	Z	122	172	7A	z
27	033	1B	ESC	59	073	3B	;	91	133	5B	[	123	173	7B	{
28	034	1C	FS	60	074	3C	<	92	134	5C	\	124	174	7C	|
29	035	1D	GS	61	075	3D	=	93	135	5D	]	125	175	7D	}
30	036	1E	RS	62	076	3E	>	94	136	5E	^	126	176	7E	~
31	037	1F	US	63	077	3F	?	95	137	5F	_	127	177	7F	DEL

表 A-2　制御文字　　^=Control **キー** 、s^=Shift **キーと** Control **キー**

10 進	8 進	16 進	名称	キー	意味
0	000	00	NUL	s^P	NULL または時間の空きを埋める
1	001	01	SOH	^A	ヘッディングの始め
2	002	02	STX	^B	テキストの始め
3	003	03	ETX	^C	テキストの終り (EOM)
4	004	04	EOT	^D	伝送終了
5	005	05	ENQ	^E	問合せ (WRU)
6	006	06	ACK	^F	肯定応答 (RU)
7	007	07	BEL	^G	ベル
8	010	08	BS	^H	バックスペース
9	011	09	HT	^I	水平タブ
10	012	0A	LF	^J	改行 (Newline)
11	013	0B	VT	^K	垂直タブ
12	014	0C	FF	^L	書式送り
13	015	0D	CR	^M	キャリッジリターン
14	016	0E	SO	^N	シフトアウト
15	017	0F	SI	^O	シフトイン
16	020	10	DLE	^P	伝送制御拡張
17	021	11	DC1	^Q	デバイス制御 1 (X-ON)
18	022	12	DC2	^R	デバイス制御 2 (TAPE)
19	023	13	DC3	^S	デバイス制御 3 (X-OFF)
20	024	14	DC4	^T	デバイス制御 4 (TAPE)
21	025	15	NAK	^U	否定応答
22	026	16	SYN	^V	同期アイドル
23	027	17	ETB	^W	伝送ブロックの終結
24	030	18	CAN	^X	取り消し
25	031	19	EM	^Y	媒体の終端
26	032	1A	SS	^Z	特殊シーケンス
27	033	1B	ESC	s^K	エスケープ (^[ )
28	034	1C	FS	s^L	ファイル分離文字 (^\ )
29	035	1D	GS	s^M	グループ分離文字 (^])
30	036	1E	RS	s^N	記録分文字 (^`)
31	037	1F	US	s^O	装置分離文字 (^/)
127	177	7F	DEL	s^0	削除または抹消 (^_)

付録 B

文の例

この付録では、f77 構文の型の選ばれた例を表に示します。この表は、各文の型の一般的なバリエーションに関する構文の早見表になっています。

規格外の機能には、普通小さなシンボル記号 (♦) を付けます。

この表では、次の規約を使用します。

表 B-1 Fortran 77 文の例

名称	例	注釈
ACCEPT ♦	`ACCEPT *, A, I`	READ を参照
ASSIGN	`ASSIGN 9 TO I`	

ASSIGNMENT	`C = 'abc'`
	`C = "abc"`
	`C = S // 'abc'`
	`C = S(I:M)`
		文字 ♦

	`L = L1 .OR. L2` `L = I .LE. 80`	論理
	`N = N+1` `X = '7FF00000'x`	算術 16 進 ♦
	`CURR = NEXT` `NEXT.ID = 82`	RECORD を参照

AUTOMATIC ♦	`AUTOMATIC A, B, C`
	`AUTOMATIC REAL P, D, Q`
	`IMPLICIT AUTOMATIC REAL (X-Z)`	

BACKSPACE	`BACKSPACE U` `BACKSPACE( UNIT=U, IOSTAT=I, ERR=9 )`	
BLOCK DATA	`BLOCK DATA` `BLOCK DATA COEFFS`	

BYTE ♦	`BYTE A, B, C`
	`BYTE A, B, C(10)`
	`BYTE A /'x'/, B /255/, C(10)`
		A と B を初期化

表 B-1 Fortran 77 文の例 (続き)

名称	例	注釈
CALL	CALL P(A, B) CALL P(A, B, *9) CALL P(A, B, &9) CALL P	 選択戻り 選択戻り ♦
CHARACTER	CHARACTER C*80, D*1(4) CHARACTER*18 A, B, C CHARACTER A, B*3 /'xyz'/, C /'z'/	 B と C を初期化 ♦
CLOSE	CLOSE (UNIT=I) CLOSE(UNIT=U, ERR=90, IOSTAT=I)	
COMMON	COMMON / DELTAS / H, P, T COMMON X, Y, Z COMMON P, D, Q(10,100)	
COMPLEX	COMPLEX U, V, U(3,6) COMPLEX U*16 COMPLEX U*32 *SPARC* のみ COMPLEX U / (1.0,1.0) /, V /(1.0,10.0) /	 倍精度複素数 ♦ 4 倍精度複素数 ♦ U と V を初期化 ♦
CONTINUE	100 CONTINUE	
DATA	DATA A, C / 4.01, 'z' / DATA (V(I),I=1,3) /.7, .8, .9/ DATA ARRAY(4,4) / 1.0 / DATA B,O,X,Y /B'0011111', O'37', X'1f', Z'1f'/	♦
DECODE ♦	DECODE (4, 1, S) V	
DIMENSION	DIMENSION ARRAY(4, 4) DIMENSION V(1000), W(3)	
DO	DO 100 I = INIT, LAST, INCR … 100 CONTINUE	
	DO I = INIT, LAST … END DO	文番号なし DO ♦
	DO WHILE (DIFF .LE. DELTA) … END DO	DO WHILE ♦
	DO 100 WHILE (DIFF .LE. DELTA) … 100 CONTINUE	♦

表 B-1　Fortran 77 文の例 (続き)

名称	例	注釈
DOUBLE COMPLEX ♦	DOUBLE COMPLEX U, V DOUBLE COMPLEX U, V COMPLEX U / (1.0,1.0D0) /, V / (1.0,1.0D0) /	Complex*16 ♦ Complex ♦ U と V を初期化
DOUBLE PRECISION	DOUBLE PRECISION A, D, Y(2) DOUBLE PRECISION A, D / 1.2D3 /, Y(2)	REAL*8 ♦ D を初期化 ♦
ELSE	ELSE	IF (ブロック) を参照
ELSE IF	ELSE IF	
ENCODE ♦	ENCODE(4, 1, T) A, B, C	
END	END	
END DO ♦	END DO	DO を参照
ENDFILE	ENDFILE (UNIT=I) ENDFILE I ENDFILE(UNIT=U, IOSTAT=I, ERR=9)	
END IF	END IF	
END MAP ♦	END MAP	MAP を参照
END STRUCTURE	END STRUCTURE	STRUCTURE を参照
END UNION ♦	END UNION	UNION を参照
ENTRY	ENTRY SCHLEP(X, Y) ENTRY SCHLEP(A1, A2, *4) ENTRY SCHLEP	
EQUIVALENCE	EQUIVALENCE (V(1), A(1,1)) EQUIVALENCE (V, A) EQUIVALENCE (X,V(10)), (P,D,Q)	
EXTERNAL	EXTERNAL RNGKTA, FIT	

表 B-1　Fortran 77 文の例 (続き)

名称	例	注釈
FORMAT	10 FORMAT(// 2X, 2I3, 3F6.1, 4E12.2, 2A6,3L2) 10 FORMAT(// 2D6.1, 3G12.2) 10 FORMAT(2I3.3, 3G6.1E3, 4E12.2E3)	
	10 FORMAT('a quoted string', " another", I2) 10 FORMAT(18Ha hollerith string, I2) 10 FORMAT(1X, T10, A1, T20, A1)	文字列 ♦ ホレリス タブ
	10 FORMAT(5X, TR10, A1, TR10, A1, TL5, A1) 10 FORMAT(" Init=", I2, :, 3X, "Last=", I2) 10 FORMAT(1X, "Enter path name ", $)	タブ右、左 : $
	10 FORMAT(F4.2, Q, 80 A1) 10 FORMAT('Octal ', O6, ', Hex ' Z6) 10 FORMAT(3F<N>.2)	Q ♦ 8 進、16 進 ♦ 変数式 ♦
FUNCTION	FUNCTION Z(A, B) FUNCTION W(P,D, *9) CHARACTER FUNCTION R*4(P,D,*9) INTEGER*2 FUNCTION M(I, J)	短整数 ♦
GO TO	GO TO 99	条件なし
	GO TO I, (10, 50, 99) GO TO I	代入
	GO TO (10, 50, 99), I	計算
IF	IF (I -K) 10, 50, 90	算術 IF
	IF (L) RETURN	論理 IF

	IF (L) THEN
	  N=N+1
	  CALL CALC
	ELSE
	  K=K+1
	  CALL DISP
	ENDIF
		ブロック IF

	IF (C .EQ. 'a') THEN
	  NA=NA+1
	  CALL APPEND
	ELSE IF (C .EQ. 'b') THEN
	  NB=NB+1
	  CALL BEFORE
	ELSE IF (C .EQ. 'c') THEN
	  NC=NC+1
	  CALL CENTER
	END IF
		ELSE IF 付きブロック IF

表 B-1　Fortran 77 文の例 (続き)

名称	例	注釈
IMPLICIT	IMPLICIT COMPLEX (U-W,Z) IMPLICIT UNDEFINED (A-Z)	
INCLUDE ♦	INCLUDE 'project02/header'	
INQUIRE	INQUIRE(UNIT=3, OPENED=OK) INQUIRE(FILE='mydata', EXIST=OK) INQUIRE(UNIT=3, OPENED=OK, IOSTAT=ERRNO)	
INTEGER	INTEGER C, D(4) INTEGER C*2 INTEGER*4 A, B, C	短整数 ♦
	INTEGER A/ 100 /, B, C / 9 /	A と C を初期化 ♦
INTRINSIC	INTRINSIC SQRT, EXP	
LOGICAL	LOGICAL C LOGICAL B*1, C*1 LOGICAL*1 B, C LOGICAL*4 A, B, C	♦ ♦ ♦
	LOGICAL B / .FALSE. /, C	B を初期化 ♦
MAP ♦	MAP CHARACTER *18 MAJOR END MAP MAP INTEGER*2 CREDITS CHARACTER*8 GRAD_DATE END MAP	構造体 (STRUCTURE) と共用体 (UNION) を参照
NAMELIST ♦	NAMELIST /CASE/ S, N, D	
OPEN	OPEN(UNIT=3, FILE="data.test") OPEN(UNIT=3, IOSTAT=ERRNO)	
OPTIONS ♦	OPTIONS /CHECK /EXTEND_SOURCE	
PARAMETER	PARAMETER (A="xyz"), (PI=3.14) PARAMETER (A="z", PI=3.14) PARAMETER X=11, Y=X/3	♦
PAUSE	PAUSE	
POINTER ♦	POINTER (P, V), (I, X)	
PRAGMA ♦	EXTERNAL RNG, !$PRAGMA C(RNG)	C() 指令
PROGRAM	PROGRAM FIDDLE	
PRINT	PRINT *, A, I	並びによる
	PRINT 10, A, I	書式付き
	PRINT 10, M	配列 M

表 B-1　Fortran 77 文の例 (続き)

名称	例	注釈
	`PRINT 10, (M(I),I=J,K)`	DO 型並び
	`PRINT 10, C(I:K)`	部分列
	`PRINT '(A6,I3)', A, I` `PRINT FMT='(A6,I3)', A, I`	文字定数書式
	`PRINT S, I` `PRINT FMT=S, I`	スイッチ変数は書式番号
	`PRINT G`	変数群 ♦
READ	`READ *, A, I`	並びによる
	`READ 1, A, I`	書式付き
	`READ 10, M`	配列 M
	`READ 10, (M(I),I=J,K)`	DO 型並び
	`READ 10, C(I:K)`	部分列
	`READ '(A6,I3)', A, I`	文字定数書式

表 B-1　Fortran 77 文の例 (続き)

名称	例	注釈
	READ(1, 2) X, Y READ(UNIT=1, FMT=2) X,Y READ(1, 2, ERR=8,END=9) X,Y READ(UNIT=1, FMT=2, ERR=8,END=9) X,Y	ファイルからの書式付き READ
	READ(*, 2) X, Y	標準入力からの書式付き READ
	READ(*, 10) M	配列 M
	READ(*, 10) (M(I),I=J,K)	DO 型並び
	READ(*, 10) C(I:K)	部分列
	READ(1, *) X, Y READ(*, *) X, Y	ファイルからの並びによる入力 — 標準入力から
	READ(1, '(A6,I3)') X, Y READ(1, FMT='(A6,I3)') X, Y	文字定数書式
	READ(1, C) X, Y READ(1, FMT=C) X, Y	
	READ(1, S) X, Y READ(1, FMT=S) X, Y	スイッチ変数は書式番号付き
	READ(*, G) READ(1, G)	変数群 READ ♦ ファイルからの変数群 READ ♦
	READ(1, END=8, ERR=9) X, Y	書式なし直接探査
	READ(1, REC=3) V READ(1 ' 3) V	書式なし直接探査
	READ(1, 2, REC=3) V	書式つき直接探査
	READ(CA, 1, END=8, ERR=9) X, Y	内部書式付き順次
	READ(CA, *, END=8, ERR=9) X, Y	内部リスト指示順番探査 ♦
	READ(CA, REC=4, END=8, ERR=9) X, Y	内部直接探査 ♦
REAL	REAL R, M(4) REAL R*4 REAL*8 A, B, C REAL*16 A, B, C　SPARCのみ	♦ 倍精度 ♦ 4 倍精度 ♦
	REAL A / 3.14 /, B, C / 100.0 /	A と C を初期化 ♦
RECORD ♦	RECORD /PROD/ CURR,PRIOR,NEXT	

表 B-1　Fortran 77 文の例 (続き)

名称	例	注釈
RETURN	RETURN RETURN 2	標準戻り 選択戻り
REWIND	REWIND 1 REWIND I REWIND (UNIT=U, IOSTAT=I, ERR=9)	
SAVE	SAVE A, /B/, C SAVE	
STATIC ♦	STATIC A, B, C STATIC REAL P, D, Q IMPLICIT STATIC REAL (X-Z)	
STOP	STOP STOP "all gone"	
STRUCTURE	STRUCTURE /PROD/ INTEGER*4 ID / 99 / CHARACTER*18 NAME CHARACTER*8 MODEL / 'XL' / REAL*4 COST REAL*4 PRICE END STRUCTURE	
SUBROUTINE	SUBROUTINE SHR(A, B, *9) SUBROUTINE SHR(A, B, &9) SUBROUTINE SHR(A, B) SUBROUTINE SHR	選択戻り ♦
TYPE ♦	TYPE *, A, I	PRINT を参照
UNION ♦	UNION MAP CHARACTER*18 MAJOR END MAP MAP INTEGER*2 CREDITS CHARACTER*8 GRAD_DATE END MAP END UNION	構造体 (STRUCTURE) を参照
VIRTUAL ♦	VIRTUAL M(10,10), Y(100)	
VOLATILE ♦	VOLATILE V, Z, MAT, /INI/	
WRITE	WRITE(1, 2) X, Y } WRITE(UNIT=1, FMT=2) X, Y WRITE(1, 2, ERR=8, END=9) X, Y WRITE(UNIT=1, FMT=2, ERR=8, END=9) X, Y	ファイルへの書式付き WRITE

表 B-1　Fortran 77 文の例 (続き)

名称	例	注釈
	`WRITE( *, 2 ) X, Y` `WRITE( *, 10 ) M`	標準出力への書式付き WRITE 配列 M
	`WRITE( *, 10 ) (M(I),I=J,K)`	DO 型並び
	`WRITE( *, 10) C(I:K)`	部分列
	`WRITE( 1, * ) X, Y` `WRITE( *, * ) X, Y`	ファイルへの並びによる WRITE 標準出力への並びによる WRITE
	`WRITE( 1, '(A6,I3)') X, Y` `WRITE( 1, FMT='(A6,I3)') X, Y`	文字定数書式
	`WRITE( 1, C ) X, Y` `WRITE( 1, FMT=C ) X, Y`	文字変数書式
	`WRITE( 1, S ) X, Y` `WRITE( 1, FMT=S ) X, Y`	スイッチ変数は書式番号
	`WRITE( *, CASE )` `WRITE( 1, CASE )`	変数群 WRITE ♦ ファイルへの変数群 WRITE ♦
	`WRITE( 1, END=8, ERR=9 ) X, Y`	書式なし順番探査
	`WRITE( 1, REC=3 ) V` `WRITE( 1 ' 3 ) V`	書式なし直接探査
	`WRITE( 1, 2, REC=3 ) V`	書式付き直接探査
	`WRITE( CA, 1, END=8, ERR=9 ) X, Y`	内部書式付き順番探査
	`WRITE( CA, *, END=8, ERR=9 ) X, Y`	並びによる内部順番探査 ♦
	`WRITE( CA, REC=4, END=8, ERR=9 ) X, Y`	内部直接探査 ♦

付録C

データの表現

問題のデータ要素の大きさに関わらず、データ要素の最上位ビットは、そのオブジェクトを表わすために必要なバイト列の最も小さい番号のバイトに格納されます。

この付録ではデータ表現を簡単に説明します。詳細は、『Fortran プログラミングガイド』および『数値計算ガイド』を参照してください。

実数、倍精度、4 倍精度

実数、倍精度、4 倍精度の数値データ要素は IEEE 規格に従って次の形式によって表わされます。ここで *f* というのは小数部におけるビットです。4 倍精度は SPARC のみです。

$$(-1)^{\text{符号}} * 2^{\text{指数} - \text{バイアス}} * 1.f$$

表 C-1　浮動小数点表現

	単精度	倍精度	符号
符号	ビット 31	ビット 63	ビット 127
指数部	ビット 30–23 バイアス 127	ビット 62–52 バイアス 1023	ビット 126-112 バイアス 16583
小数部	ビット 22–0	ビット 51–0	ビット 111-0
範囲の近似値	3.402823e+38 1.175494e-38	1.797693e+308 2.225074e-308	3.362E-4932 1.20E+4932

極端な指数

ここでは、「極端な指数」について説明します。

ゼロ (符号付き)

ゼロ (符号付き) はゼロの指数部とゼロの小数部により表わされます。

非正規数値

非正規数値の形式は次のとおりです。

(-1) ^{符号} * 2 *1-*^{バイアス} *0.f

ここで *f* は有効数字を示すビットです。

符号付き無限

符号付き無限 (すなわち疑似無限) は、指数部がとることのできる最も大きな値 (すべて 1) とゼロの小数部により表わされます。

非数 (NaN)

非数 (NaN) は、指数がとることのできる最も大きな値 (すべて 1) とゼロ以外の小数部により表わされます。

正規化された `REAL` および `DOUBLE PRECISION` 数値は、メモリーに格納されているよりビットの精度が 1 つだけ多い暗黙的先行ビットを持っています。たとえば、IEEE の倍精度は小数部に格納された 52 ビットに暗黙的な先行をする 1 ビットを加えた 53 ビットの精度になります。

選択数値の IEEE 表現

以下の数値は dbx により示された 16 進表現です。

表 C-2　選択数値の IEEE 表記

値	単精度	倍精度
+0	00000000	0000000000000000
-0	80000000	8000000000000000
+1.0	3F800000	3FF0000000000000
-1.0	BF800000	BFF0000000000000
+2.0	40000000	4000000000000000
+3.0	40400000	4008000000000000
+ 無限	7F800000	7FF0000000000000
- 無限	FF800000	FFF0000000000000
非数	7Fxxxxxx	7FFxxxxxxxxxxxx

極端な数値に対する演算

本節では、極端な数値と通常値による基本算術演算の結果について説明します。すべての入力は正であり、トラップ、オーバーフロー、アンダーフロー、またはその他の例外は起こらないものとします。

表 C-3　極端な値の省略形

省略形	意味
Sub	非正規数
Num	正規化された数
Inf	無限 （正または負）
NaN	非数
Uno	順序付けられない

表 C-4　極端な数値: 加算と減算

	右オペランド				
左オペランド	**0**	**Sub**	**Num**	**Inf**	**NaN**
0	0	Sub	Num	Inf	NaN
Sub	Sub	Sub	Num	Inf	NaN
Num	Num	Num	Num	Inf	NaN
Inf	Inf	Inf	Inf	注意を参照	NaN
NaN	NaN	NaN	NaN	NaN	NaN

注意: Inf±Inf の場合: Inf + Inf = Inf、Inf - Inf = NaN

表 C-5　極端な数値: 乗算

	右オペランド				
左オペランド	**0**	**Sub**	**Num**	**Inf**	**NaN**
0	0	0	0	NaN	NaN
Sub	0	0	NS	Inf	NaN
Num	0	NS	Num	Inf	NaN
Inf	NaN	Inf	Inf	Inf	NaN
NaN	NaN	NaN	NaN	NaN	NaN

上記において、NS は Num か Sub のどちらかの結果が可能であることを意味します。

表 C-6　極端な数値: 除算

左オペランド	右オペランド				
	0	**Sub**	**Num**	**Inf**	**NaN**
0	NaN	0	0	0	NaN
Sub	Inf	Num	Num	0	NaN
Num	Inf	Num	Num	0	NaN
Inf	Inf	Inf	Inf	NaN	NaN
NaN	NaN	NaN	NaN	NaN	NaN

表 C-7　極端な数値: 比較

左オペランド	右オペランド				
	0	**Sub**	**Num**	**Inf**	**NaN**
0	=	<	<	<	Uno
Sub	>		<	<	Uno
Num	>	>		<	Uno
Inf	>	>	>	=	Uno
NaN	Uno	Uno	Uno	Uno	Uno

以下の点に注意してください。

- `X` か `Y` のどちらかが `NaN` であれば、`X.NE.Y` は `.TRUE.` であり、その他 (`.EQ.`、 `.GT.`、 `.GE.`、 `.LT.`、 `.LE.`) は `.FALSE.` です。
- +0 比較 は -0 に等しくなります。
- いずれかの引数が `NaN` であれば、`MAX` または `MIN` の結果は未定義です。

アーキテクチャによるビットとバイト

データ、すなわちビットとバイトが並べられる順序は、VAX コンピュータと SPARC コンピュータ間では異なっています。

32 ビット整数のバイトは、アドレス n から読み取られると、以下に示すようにレジスタに入ります。

表 C-8 Intel と VAX コンピュータの場合のビットとバイト

バイト *n*+3	**バイト *n*+2**	**バイト *n*+1**	**バイト *n***
31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05 04 03 02 01 00
最上位			最下位

表 C-9 680x0と SPARC コンピュータの場合のビットとバイト

バイト *n*	**バイト *n*+1**	**バイト *n*+2**	**バイト *n*+3**
31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 09 08	07 06 05 04 03 02 01 00
最上位			最下位

これらのシステムにおいては、各バイトの番号の付け方が異なっていても、各ビットには同じように番号が付けられています。

注意すべき点は次のとおりです。

- ネットワーク上でのバイナリデータのやりとり。外部データ表現 (XDR) 形式などの標準ネットワーク形式を使用すれば、問題は起きません。
- アーキテクチャ間におけるラスタグラフィックスイメージの移植。プログラムがバイナリ形式のグラフィクスイメージを使用しており、それらのデータが SPARC システムルーチンにより生成されるものとは異なるバイト順序付けになっていれば、変換する必要があります。
- アーキテクチャ間において、文字から整数または整数から文字へ変換を行う場合は、XDR を使用しなければなりません。

- 異なるバイト順序のアーキテクチャで生成されたバイナリデータを読み取る場合は、そのデータをフィルタにかけて、バイト順序を正しくする必要があります。

マニュアルページの xdr(3N) も参照してください。

付録 D

VMS 言語拡張

この付録では Fortran 77 がサポートする VMS 言語拡張について説明します。これらの拡張はすべて規格外です。 ♦

背景

この Fortran コンパイラには、VMS から Solaris 環境への Fortran 77 プログラムの移植を簡単にするために VMS 拡張が含まれています。このコンパイラは VMS Fortran とほぼ完全な互換性を持っています。これらの拡張は `dbx` と `f77` に含まれています。

サンの Fortran の VMS 言語機能

以下のリストは、`f77` に含まれる VMS 機能の要約です。詳細は本書の他の部分で説明します。

- 変数群入出力
- ラベルなし `DO` ... `END DO`
- 確定していない `DO WHILE` ... `END DO`
- `BYTE` データ型
- 整数についての論理演算および論理についての算術演算
- FORMAT 文のための追加の欄と編集記述子
  - 残りの文字 (`Q`)
  - キャリッジ制御 (`$`)
  - 8 進 (`O`)
  - 16 進 (`X`)

- 16 進 (Z)

- FORMAT 文における *w*、*d*、*e* の各欄のデフォルト欄記述子
- ホレリス編集記述子への読み取り
- OPEN のための APPEND オプション
- 長い名前 (32 文字)
- 名前における “_” と “$”
- -e オプションが設定されている長いソース行 (132 文字)
- 記録、構造体、共用体、マップ
- %LOC 関数によるアドレスの獲得
- %VAL 関数による引数渡し
- 行末注釈
- OPTIONS 文
- VMS タブフォーマットソース行は有効です。

- 共通の初期化

BLOCK DATA 副プログラム外で共通ブロックの変数を初期化することができます。共通ブロックのいくつかの部分を初期化することができますが、複数の副プログラムにある 1 つの共通ブロックの部分を初期化することはできません。

- 50 進数

50 進数の定数は f77 ではビット列定数として扱われます。つまり、型は想定されていません。

- IMPLICIT NONE は IMPLICIT UNDEFINED (A-Z) として扱われます。
- VIRTUAL は DIMENSION として扱われます。
- 宣言で初期化を行います。

宣言文における変数の初期化は許可されます。例を次に示します。

```
    CHARACTER*10 NAME /'Nell'/
```

- 文字型でない書式指定子

実行時書式指定子が CHARACTER 型でない場合も、コンパイラは受け付けます。Fortran 77 規格では CHARACTER 型を必要とします。

- 副プログラム呼び出しにおける省略された引数

コンパイラはサブルーチン呼び出しにおける省略された実引数を受け付けます。つまり、2 つの連続するコンマは NULL ポインタにコンパイルされます。その仮引数を参照すると、セグメンテーション違反が生じます。

- REAL*16

(SPARC のみ) コンパイラは REAL*16 型の変数を 4 倍精度として扱います。

- 非文字変数

Fortran 77 規格では、OPEN および INQUIRE のときの FILE= 指定子は CHARACTER 型の式でなければなりません。しかし、f77 は数値変数または配列要素の引用を受け付けます。

- 連続演算子

f77 では、2 番目の演算子が単項 + または - の場合、2 つの連続する算術演算子が許可されます。

```
    X = A ** -B
```

上記の文は次のように扱われます。

```
    X = A ** (-B)
```

- 不正な実数式

コンパイラは、整数式を予想している場所に REAL 式を見つけるとそれを切り捨て、INTEGER への型変換を行います。

例 : f77 が不正な実数式を整数に変換する場合

- 選択 RETURN
- 次元宣言子および配列記述子
- 部分列選択子
- 計算形 GO TO
- 論理装置番号、記録番号、記録長

- 型なし数値定数

2 進、16 進および 8 進の定数は VMS 形式で受け付けられます。

例 : 定数 — 2 進 (B)、8 進 (O)、16 進 (X または Z)

```
    DATA N1 /B'0011111'/, N2/O'37'/, N3/X'1f'/, N4/Z'1f'/
```

■ FUNCTION にではなく、関数名に長さ指定

コンパイラは、関数宣言における規格外の長さ指定子を受け付けます。

例： FUNCTION というワードよりも関数名としてのサイズ

```
    INTEGER FUNCTION FCN*2 ( A, B, C )
```

■ TYPE 文と ACCEPT 文は許可されます。

■ 選択戻り

選択戻りの実引数のための規格外の & 構文は Fortran 77 規格の * 構文として扱われます。次に例を示します。

```
    CALL SUBX ( I, *100, Z)  ! 標準
    CALL SUBX ( I, &100, Z ) ! 非標準の代替構文
```

■ ENCODE 文と DECODE 文は受け付けられます。

■ 'N 記録指定子を持つ直接入出力

直接入出力文のための規格外の標準記録指定子 'N は受け付けられます。

例 : 記録指定子のための規格外の形式

```
    READ ( K ' N ) LIST
```

上記の形式は次のように扱われます。

```
    READ ( UNIT=K, REC=N ) LIST
```

論理装置番号は K であり、記録数は N です。

■ NAME、RECORDSIZE、TYPE の各オプション — OPEN は次の代替オプションを持っています。

- NAME は FILE として扱われます。
- RECORDSIZE は RECL として扱われます。
- TYPE は STATUS として扱われます。

- DISPOSE=*p*

CLOSE 文における DISPOSE=*p* 句は、STATUS=*p* として扱われます。

- 特殊組込み関数

コンパイラは特定の特殊組込み関数を備えています。

- %VAL は受け付けられます。
- %LOC は LOC として扱われます。
- %REF (*expr*) は *expr* として (*expr* が CHARACTER 型の場合は警告付きで) 扱われます。
- %DESCR は翻訳不能の関数として報告されます。

- FORMAT 文における変数式

一般に、FORMAT 文の中では、整定数を任意の式に置き換えることができます。
唯一の例外は、*n*H... 編集記述子における *n* です。式そのものは角括弧で囲まなければなりません。
例 : 次の文における 6 は定数です。

```
1   FORMAT( 3F6.1 )
```

6 は次のように変数 N に置き換えることができます。

```
1   FORMAT( 3F<N>.1 )
```

-xl または -vax=spec を必要とする VMS 機能

特殊なオプションを使用しなくても、ほとんどの VMS 機能は自動的に得られますが、f77 コマンド行に -xl オプションを追加しなければならないものもあります。

一般に、ソース文が VMS か f77 のどちらの動作様式としても解釈可能で、ユーザーが VMS の動作様式を希望する場合、この -xl オプションが必要です。この場合、-xl オプションはコンパイラにそれを VMS Fortran として解釈するよう強制します。

これら VMS 拡張機能を個々に指定することができる -vax=*spec* オプションもあります。詳細は、『Fortran ユーザーズガイド』を参照してください。

-xl[d] オプションを必要とする機能の要約

以下の機能を使用するためには `-xl[d]` オプションが必要です。

- 書式なし記録のサイズ指定の単位がバイトでなくワード (`-xl`)
- VMS スタイルの論理ファイル名 (`-xl`)
- 8 進定数を指定する引用符 (") 文字 (`-xl`)
- 文字定数内の通常文字としてのバックスラッシュ (\) 文字 (`-xl`)
- `PARAMETER` 文の規格外の形式 (`-xl`)
- 注釈行、あるいは Fortran 77 文としてのデバッグ行 (`-xld`)
- VMS Fortran の構造体にする (`-xl`)

-xl[d] オプションを必要とする機能の詳細

詳細は次のとおりです。

- バイト単位の書式なし記録サイズ

`f77` を使用すると、書式なし直接探査ファイルは常にバイト単位の論理記録サイズで開きます。

`-xl[d]` オプションを設定しなかった場合、`OPEN` オプション `RECL=`*n* の引数 *n* は、記録サイズのために使用するバイト数であると想定されます。

`-xl[d]` オプションを設定した場合、`OPEN` オプション `RECL=`*n* の引数 *n* は、ワード数であると想定されるので、コンパイラは *n*`*4` を記録サイズのためのバイト数として使用します。

`-xl[d]` オプションを設定した場合、ファイルが書式付きか、書式付きでないかをコンパイラが判断できないと、記録サイズを調整する必要があるかもしれないという警告メッセージが出されます。これは、情報が可変文字列で渡される場合に起こる可能性があります。

`INQUIRE` 文により返される記録サイズはコンパイラにより調整されません。つまり、`INQUIRE` は常にバイト数を返します。

これらの記録サイズは、書式なし直接探査ファイルにだけ適用されるという点に注意してください。

- VMS スタイルの論理ファイル名

-xl[d] オプションを設定した場合、コンパイラは、環境変数 LOGICALNAMEMAPPINGを見つけ、論理名と UNIX パス名との間のマッピングを定義していることが分かれば、INCLUDE 文において VMS 論理ファイル名を解釈します。

環境変数を次の形式の文字列に設定します。

```
"lname1=path1; lname2=path2; ..."
```

VMS スタイルの論理ファイル名の規則は、以下のとおりです。

- 各々の *lname* は論理名であり、各々の *path1*、*path2* などはディレクトリのパス名です (後続 / はなし)。
- コンパイラはこのような文字列を構文解析するとき、すべての空白を無視します。
- コンパイラは INCLUDE 文におけるファイル名から後続の /[no]list を除去します。
- ファイル名における論理名は VMS ファイル名の最初の : で区切られます。
- コンパイラは *lname1:file* から *path1/file* 形式へファイル名を変換します。
- 論理名については、大文字 / 小文字が区別されます。INCLUDE 文で LOGICALNAMEMAPPING に指定されていない論理名は、そのファイル名が変更されずにそのまま使用されます。

- 8 進定数を指定する引用符 (") 文字

-xl[d] コンパイラオプションを設定した場合、VMS Fortran の 8 進整定数はその 10 進形式として扱われます。

例 : VMS の 8 進整定数

```
    JCOUNT = ICOUNT + "703
```

上記の文は次のように扱われます。

```
    JCOUNT = ICOUNT + 451
```

-xl[d] オプションが設定されていない場合、"703 はエラーになります。

-xl[d] が設定されている場合、VMS Fortran の "703 という表記法は、整数の 8 進定数からその整数の 10 進定数 (この場合は 451) に変換するよう f77 にシグナルを送ります。VMS Fortran では "703 は文字定数の最初に使用することはできないという点に注意してください。VMS Fortran 文字定数は引用符ではなく、アポストロフィで区切られるからです。

- 文字定数内のバックスラッシュ (\) 文字

-xl[d] オプションが設定されている場合、文字列におけるバックスラッシュは通常の文字として、設定されていない場合は、エスケープ文字として扱われます。

- PARAMETER 文の規格外の形式

-xl[d] オプションが設定されていれば、代替 PARAMETER 文の構文は許可されます。
例 : PARAMETER 文の VMS 代替形式は括弧を省略します。

```
    PARAMETER FLAG1 = .TRUE.
```

- 注釈行、あるいは Fortran 文としてのデバッグ行 (-xld)

-xld オプションが設定されているかどうかによって、コンパイラは、注釈行あるいは Fortran 文としてデバッグ行を解釈します。-xld オプションが設定されている場合は、コンパイルされます。設定されていなければ、それらは注釈になります。

例 : デバッグ行

```
    REAL A(5) / 5.0, 6.0, 7.0, 8.0, 9.0 /
    DO I = 1, 5
    X = A(I)**2
D   PRINT *, I, X
    END DO
    PRINT *, '終了'
    END
```

-xld を付けると、上記の例では I と X が出力されます。-xld がなければ、それらは出力されません。

- VMS Fortran の構造体にする

ユーザーのプログラムで VMS 構造体を使用する方法が詳しくわかっている場合は、この機能を使用してください。C と構造体を共有する必要がある場合は、デフォルトを使用します。`-xl` は使用しません。

サポートされていない VMS Fortran

ほとんどの VMS Fortran 拡張は `f77` コンパイラに組み込まれています。コンパイラはソースファイルのサポートされていない文について標準エラーにメッセージを出力します。

以下に、サポートされていない VMS 文を列挙します。

- `DEFINE FILE` 文
- `DELETE` 文
- `UNLOCK` 文
- `FIND` 文
- `REWRITE` 文
- `READ` 文の `KEYID` とキー指定子
- 規格外の `INQUIRE` 指定子
  - `CARRIAGECONTROL`
  - `DEFAULTFILE`
  - `KEYED`
  - `ORGANIZATION`
  - `RECORDTYPE`
- 規格外の `OPEN` 指定子
  - `ASSOCIATEVARIABLE`
  - `BLOCKSIZE`
  - `BUFFERCOUNT`
  - `CARRIAGECONTROL`
  - `DEFAULTFILE`
  - `DISP[OSE]`
  - `EXTENDSIZE`
  - `INITIALSIZE`

- KEY
- MAXREC
- NOSPANBLOCKS
- ORGANIZATION
- RECORDTYPE
- SHARED
- USEROPEN

- 組み込み関数 %DESCR

- OPTIONS 文における次のパラメータ
  - [NO]G_FLOATING
  - [NO]F77
  - CHECK=[NO]OVERFLOW
  - CHECK=[NO]UNDERFLOW

- INCLUDE 文の一部

INCLUDE 文は部分的に変換されます。INCLUDE 文はオペレーティングシステムに依存しているので、完全な自動変換はできません。VMS バージョンでは UNIX ファイル名の延長と区別できないモジュール名および LIST 制御指令が許可されます。また、VMS は英文字の大文字小文字を区別しないので、プログラマが大文字化について一貫していなければ、意図に反して異なる結果になります。

- 短整数を期待している場所での 長整数の利用

VMS Fortran では、短整数を予想しているサブルーチンに 長整数の引数を渡すことができます。VAX は下位バイトに整数をアドレス指定するため、長整数が 16 ビットに納まる場合はこれでうまくいきます。しかし SPARC コンピュータでは無効です。

- オペレーティングシステムに直接関係する VMS システムコール

- 2 つ以上の副プログラムにおける共通ブロックの初期化

- ブロックをロードする順序に依存するように共通ブロックをアルファベット化します。-M *mapfile* オプションのついた古いブロックを ld に指定することができます。

- 次の両方に対してデフォルトを使用する場合、
  - OPEN オプションの BLANK=
  - BN/BZ/B 書式編集指定子

書式付き数値入力では、埋め込み空白および後続空白が無視されます。対応する VMS デフォルトは、それらの空白をゼロとして扱います。

索引

A

B

C

D

E

F

G

I

L

T

U

V

W

X

Z

あ

い

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

ち

て

と

な

に

は

ひ

ふ

り

れ

ろ