Fortran ライブラリ・リファレンス

Sun™ Studio 8

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054 U.S.A.
650-960-1300

Part No. 817-5797-10
2004 年 3 月, Revision A

原典： Fortran Library Reference
Part No: 817-5065-10
Revision A

Adobe PostScript

目次

表目次

はじめに

このマニュアルでは、Sun™ ONE Studio Compiler Collection Fortran ライブラリの組み込み関数およびルーチンについて説明しています。Fortran 言語と Solaris™ オペレーティング環境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。

このマニュアルは、Fortran に関する実用的な知識を持ち、Sun Fortran コンパイラの効率的な使用法を学ぼうとしている、科学者、技術者、プログラマを対象に書かれています。また、Solaris™ オペレーティング環境や UNIX® の一般的な知識を持つ読者を対象としています。

入出力、アプリケーション開発、ライブラリの作成とその使用、プログラム解析、移植、最適化、並列化といった Solaris オペレーティング環境での Fortran プログラミングについては、『Fortran プログラミングガイド』 を参照してください。

書体と記号について

次の表と記述は、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ています。

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」

- 小さい三角(△)は意味のある空白を示します。

```
ΔΔ36.001
```

- 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。現在の表記規則では、「Fortran 95」と小文字を使用しています。
- オンラインマニュアル (man) ページへの参照は、トピック名とセクション番号とともに表示されます。たとえば、GETENV への参照は、`getenv(3F)` と表示されます。`getenv(3F)` とは、このページにアクセスするためのコマンドが `man -s 3F getenv` であるという意味です。

コードの記号	意味	記法	コード例
[ ]	角括弧にはオプションの引数が含まれます。	`O[n]`	`-O4, -O`
{ }	中括弧には、必須オプションの選択肢が含まれます。	`d{y|n}`	`-dy`
|	「パイプ」または「バー」と呼ばれる記号は、その中から 1 つだけを選択可能な複数の引数を区切ります。	`B{dynamic|static}`	`-Bstatic`
:	コロンは、コンマ同様に複数の引数を区切るために使用されることがあります。	`Rdir[:dir]`	`-R/local/libs:/U/a`
...	省略記号は、連続するものの一部が省略されていることを示します。	`-xinline=f1[,...fn]`	`-xinline=alpha,dos`

シェルプロンプトについて

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)	#

コンパイラとツールへのアクセス方法

PATH 環境変数を変更して、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るかどうか判断するには以下を実行します。

▼ 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

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 変数はコンパイラと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、次の指示に従って、PATH 環境変数を設定してください。

▼ PATH 環境変数を設定してコンパイラとツールにアクセスする

1. C シェルを使用している場合は、ホームの .cshrc ファイルを編集します。Bourne シェルまたは Korn シェルを使用している場合は、ホームの .profile ファイルを編集します。

2. 次のパスを PATH 環境変数に追加します。

   /opt/SUNWspro/bin

マニュアルページへのアクセス方法

マニュアルページにアクセスするために MANPATH 変数を変更する必要があるかどうかを判断するには以下を実行します。

▼ 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

1. 次のように入力して、dbx マニュアルページを表示します。

```
% man dbx
```

2. 出力された場合、内容を確認します。

   dbx(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。

▼ MANPATH 変数を設定してマニュアルページにアクセスする

1. C シェルを使用している場合は、ホームの .cshrc ファイルを編集します。Bourne シェルまたは Korn シェルを使用している場合は、ホームの .profile ファイルを編集します。

2. 次のパスを PATH 環境変数に追加します。

   ```
   /opt/SUNWspro/man
   ```

コンパイラコレクションのマニュアルへのアクセス

マニュアルには、以下からアクセスできます。

- 製品マニュアルは、ご使用のローカルシステムまたはネットワークの製品にインストールされているマニュアルの索引から入手できます。
  /opt/SUNWspro/docs/ja/index.html

製品ソフトウェアが /opt 以外のディレクトリにインストールされている場合は、システム管理者に実際のパスをお尋ねください。

- マニュアルは、docs.sun.com の Web サイトで入手できます。以下に示すマニュアルは、インストールされている製品のマニュアルの索引から入手できます (docs.sun.com Web サイトでは入手できません)。
  - 『Standard C++ Library Class Reference』
  - 『標準 C++ ライブラリ・ユーザーズガイド』
  - 『Tools.h++ クラスライブラリ・リファレンスマニュアル』
  - 『Tools.h++ ユーザーズガイド』
- リリースノートは、docs.sun.com で入手できます。

インターネットの docs.sun.com Web サイト (http://docs.sun.com) から、サンのマニュアルを参照したり、印刷したり、購入することができます。マニュアルが見つからない場合はローカルシステムまたはネットワークの製品とともにインストールされているマニュアルの索引を参照してください。

注 - Sun では、本マニュアルに掲載した第三者の Web サイトのご利用に関しましては責任はなく、保証するものでもありません。また、これらのサイトあるいはリソースに関する、あるいはこれらのサイト、リソースから利用可能であるコンテンツ、広告、製品、あるいは資料に関して一切の責任を負いません。Sun は、これらのサイトあるいはリソースに関する、あるいはこれらのサイトから利用可能であるコンテンツ、製品、サービスのご利用あるいは信頼によって、あるいはそれに関連して発生するいかなる損害、損失、申し立てに対する一切の責任を負いません。

アクセシブルな製品マニュアル

マニュアルは、技術的な補足をすることで、ご不自由なユーザーの方々にとって読みやすい形式のマニュアルを提供しております。アクセシブルなマニュアルは以下の表に示す場所から参照することができます。製品ソフトウェアが /opt 以外のディレクトリにインストールされている場合は、システム管理者に実際のパスをお尋ねください。

マニュアルの種類	アクセシブルな形式と格納場所
マニュアル (サードパーティ製マニュアルは除く)	形式 : HTML (日本語版は PDF のみ) 場所 : http://docs.sun.com
サードパーティ製マニュアル: 『Standard C++ Library Class Reference』 『標準 C++ ライブラリ・ユーザーズガイド』 『Tools.h++ クラスライブラリ・リファレンスマニュアル』 『Tools.h++ ユーザーズガイド』	形式 : HTML 場所 : file:/opt/SUNWspro/docs/ja/index.html のマニュアル索引
Readme および マニュアルページ	形式 : HTML 場所 : file:/opt/SUNWspro/docs/ja/index.html のマニュアル索引
リリースノート	製品 CD 内の HTML ファイル

関連するコンパイラコレクションマニュアル

以下の表は、file:/opt/SUNWspro/docs/ja/index.html および http://docs.sun.com から参照できるマニュアルの一覧です。製品ソフトウェアが /opt 以外のディレクトリにインストールされている場合は、システム管理者に実際のパスをお尋ねください。

マニュアルタイトル	内容の説明
Fortran ユーザーズガイド	f95 コンパイラのコンパイル時環境とコマンド行オプションについて説明しています。従来の f77 のプログラムを f95 に移行するためのガイドラインも記載されています。
Fortran ライブラリ・リファレンス	Fortran ライブラリと組み込みルーチンについて詳しく説明しています。
OpenMP API ユーザーズガイド	OpenMP 多重処理 API の概要とその Sun™ ONE Studio Compiler Collection 実装の詳細について説明します。
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。

関連する Solaris マニュアル

次の表では、docs.sun.com の Web サイトで参照できる関連マニュアルについて説明します。

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 8 Reference Manual Collection	マニュアルページの節を参照。	Solaris のオペレーティング環境に関する情報を提供しています。
Solaris 8 Software Developer Collection	リンカーとライブラリ	Solaris のリンクエディタと実行時リンカーの操作について説明しています。
Solaris 8 Software Developer Collection	マルチスレッドのプログラミング	POSIX と Solaris スレッド API、同期オブジェクトのプログラミング、マルチスレッド化したプログラムのコンパイル、およびマルチスレッド化したプログラムのツール検索について説明します。

開発者向けのリソース

`http://www.sun.com/developers/studio` にアクセスし、Compiler Collection というリンクをクリックして、以下のようなリソースを利用できます。リソースは頻繁に更新されます。

- プログラミング技術と最適な演習に関する技術文書
- プログラミングに関する簡単なヒントを集めた知識ベース
- Compiler Collection のコンポーネントのマニュアル、ソフトウェアと共にインストールされるマニュアルの訂正
- サポートレベルに関する情報
- ユーザーフォーラム

- ダウンロード可能なサンプルコード
- 新しい技術の紹介
- `http://www.sun.co.jp/developers/` でも開発者向けのリソースが提供されています。

技術サポートへの問い合わせ

製品についての技術的なご質問がございましたら、以下のサイトからお問い合わせください (このマニュアルで回答されていないものに限ります)。

http://sun.co.jp/service/contacting

第1章

Fortran ライブラリルーチン

この章では、Fortran のライブラリルーチンについて説明します。

本章で説明するルーチンにはすべて、対応するマニュアルページがマニュアルライブラリのセクション 3F に用意されています。たとえば、**man -s 3F access** を実行すると、`access` といううライブラリルーチンに関するマニュアルページの内容が表示されます。

この章では、標準の Fortran 95 組み込み関数については説明しません。組み込み関数の詳細は、関連する Fortran 95 の標準ドキュメントを参照してください。

Fortran や C から呼び出しが可能な上記以外の数学ルーチンについては、『数値計算ガイド』も参照してください。呼び出し可能な数学ルーチンには、`libm` や `libsunmath` の標準数学ライブラリルーチン (`Intro`(3M)参照)、これらのライブラリの最適化バージョン、SPARC ベクトル数学ライブラリ `libmvec` などがあります。

Fortran 77 および `f95` コンパイラによって実装される VMS の組み込み関数については、第 3 章 を参照してください。

データ型について

特に指示がない限り、本章に記載する関数ルーチンは、組み込みルーチンではありません。したがって、関数から返されるデータ型が、関数名だけを指定した場合に仮定されるデータ型と食い違う可能性がある場合は、ユーザーが明示的にデータ型を宣言する必要があります。たとえば、`getpid()` で `INTEGER*4` を戻す場合は、`INTEGER*4 getpid` と宣言しないと、結果の正しい処理が保証されません (データ型

を明示的に指定しないと、関数名が g で開始するため、REAL (実数) 型の結果が仮定される)。なお、こういったルーチンについては、その機能要約で明示的な型宣言文が覚え書きの目的で記載されています。

引数および戻り値のデータ指定は、IMPLICIT 文、-dbl、-xtypemap といった各コンパイラオプションによっても変更されることに注意してください。これらのライブラリルーチンを呼び出す際に、期待するデータ型と実際のデータ型が一致していないと、プログラムは予期しない動きをします。コンパイラオプション -xtypemap、-dbl を指定すると、INTEGER 関数のデータ型は INTEGER*8 に、REAL 関数は REAL*8 に、DOUBLE 関数は DOUBLE*16 にそれぞれ変更されます。こういった問題を回避するには、ライブラリ呼び出しで指定する関数名と変数について、期待するそれらのサイズを明示的に指定する必要があります。次の例を参考にしてください。

```
        integer*4 seed, getuid
        real*4 ran
        ...
        seed = 70198
        val = getuid() + ran(seed)
        ...
```

上記の例のようにサイズを明示的に指定しておくと、コンパイラオプションとして -xtypemap と -dbl を指定しても、ライブラリ呼び出しの際にデータ型の変更が行われません。明示的な指定が行われない場合は、これらのコンパイラオプションによって、予期しない結果を招く可能性があります。これらのオプションの詳細については、『Fortran ユーザーズガイド』および f95(1) マニュアルページを参照してください。

Fortran 95 コンパイラの f95 は、ほとんどの非組み込みライブラリルーチンのインタフェースを定義するインクルードファイルsystem.inc を提供します。特にデフォルトのデータ型が-xtypemapを指定して変更された場合、このファイルをインクルードして、呼び出した関数やその引数が正しく入力されているか確認してください。

```
        include 'system.inc'
        integer(4) mypid
        mypid = getpid()
        print *, mypid
```

Fortran コンパイラの広域プログラムチェックオプション `-Xlist` を使用すると、ライブラリ呼び出し全体のデータ型のミスマッチに関連した多数の問題を把握できます。`f95` コンパイラによる広域プログラムチェックについては、『Fortran ユーザーズガイド』、『Fortran プログラミングガイド』、およびマニュアルページの `f95` (1) で説明しています。

64 ビット環境

プログラムを 64 ビットのオペレーティング環境で動作するようにコンパイルすると (つまり、`-xarch=v9`、`v9a`、または`v9b`を使ってコンパイルし、64 ビット Solaris オペレーティング環境を実行する SPARC プラットフォーム上で実行可能プログラムを実行すると)、特定の関数の戻り値が変更されます。この特定の関数は、通常、`malloc`(3F) (62 ページの「`malloc`、`malloc64`、`realloc`、`free`：記憶領域の割り当て/再割り当て/割り当て解除」参照) などの標準システムレベルのルーチンとのインタフェースとなり、その環境に応じて 32 ビット値または 64 ビット値をとったり、戻したりできます。32 ビットと 64 ビット環境間でコードに互換性を持たせるために、これらのルーチンの 64 ビットバージョンは、必ず 64 ビット値をとるまたは戻す (あるいはこの両方を行う) ように規定されています。次の表に、64 ビット環境で使用するために提供されたライブラリルーチンを表示します。

表 1-1　64 ビット環境向けライブラリルーチン

関数	内容の説明
`malloc64`	メモリーを割り当て、ポインタを戻す
`fseeko64`	大規模ファイルの再位置付け
`ftello64`	大規模ファイルの位置付け
`stat64, fstat64, lstat64`	ファイルの状態を決定する
`time64,ctime64,gmtime64,ltime64`	システム時間を取得し、文字に変換するか月、日などに分解する
`qsort64`	配列の要素をソートする

Fortran 数学関数

次の関数とサブルーチンは、Fortran 数学ライブラリの一部です。これらの関数とサブルーチンは、`f95` でコンパイルしたすべてのプログラムで使用することができます。ルーチンには、引数として特定のデータ型をとり、それと同じデータ型を戻す非組み込み関数があります。非組み込み関数は、これを参照するルーチン内で宣言する必要があります。

こうしたルーチンの大半は、C 言語ライブラリのルーチンに対する Fortran のインタフェースである「ラッパー」であり、したがって、標準の Fortran ではありません。この中には、IEEE 推奨のサポート関数や特殊な乱数発生関数があります。これらのライブラリの詳細については、『数値計算ガイド』やマニュアルページ `libm_single`(3F)、`libm_double`(3F)、`libm_quadruple`(3F) を参照してください。

単精度関数

これらの副プログラムは、単精度の数学関数およびサブルーチンです。

通常、以下の数学単精度関数にアクセスする関数は、Fortran 規格の総称組み込み関数とは対応していません。データ型は通常の型決定規則によって決定されます。

デフォルトの型決定を保持している限り、`REAL` 文でこれらの関数の型を明示的に指定する必要はありません。(`r` で始まる名前は `REAL` 型、`i` で始まる名前は `INTEGER` 型になります)。

これらのルーチンの詳細については、C 数学ライブラリのマニュアルページ (3M) を参照してください。たとえば、r_acos(x) の場合は、マニュアルページの acos(3M) を参照します。

表 1-2 数学単精度関数

関数名	結果の型	内容の説明
r_acos(x)	REAL	逆余弦
r_acosd(x)	REAL	--
r_acosh(x)	REAL	逆双曲余弦
r_acosp(x)	REAL	--
r_acospi(x)	REAL	--
r_atan(x)	REAL	逆正接
r_atand(x)	REAL	--
r_atanh(x)	REAL	逆双曲正接
r_atanp(x)	REAL	--
r_atanpi(x)	REAL	--
r_asin(x)	REAL	逆正弦
r_asind(x)	REAL	--
r_asinh(x)	REAL	逆双曲正弦
r_asinp(x)	REAL	--
r_asinpi(x)	REAL	--
r_atan2((y, x)	REAL	逆正接
r_atan2d(y, x)	REAL	--
r_atan2pi(y, x)	REAL	--
r_cbrt(x)	REAL	立方根
r_ceil(x)	REAL	小数点以下切り上げ
r_copysign(x, y)	REAL	--
r_cos(x)	REAL	余弦
r_cosd(x)	REAL	--
r_cosh(x)	REAL	双曲余弦
r_cosp(x)	REAL	--
r_cospi(x)	REAL	--
r_erf(x)	REAL	誤差関数
r_erfc(x)	REAL	--

表 1-2　数学単精度関数 (続き)

関数名	結果の型	内容の説明
r_expm1(x)	REAL	(e**x)-1
r_floor(x)	REAL	小数点以下切り捨て
r_hypot(x, y)	REAL	斜辺
r_infinity( )	REAL	--
r_j0(x)	REAL	ベッセル関数
r_j1(x)	REAL	--
r_jn(x)	REAL	
ir_finite(x)	INTEGER	--
ir_fp_class(x)	INTEGER	--
ir_ilogb(x)	INTEGER	--
ir_irint(x)	INTEGER	--
ir_isinf(x)	INTEGER	--
ir_isnan(x)	INTEGER	--
ir_isnormal(x)	INTEGER	--
ir_issubnormal(x)	INTEGER	--
ir_iszero(x)	INTEGER	--
ir_signbit(x)	INTEGER	--
r_addran()	REAL	乱数発生関数
r_addrans(x, p, l, u)	subroutine	
r_lcran()	REAL	
r_lcrans(x, p, l, u)	サブルーチン	
r_shufrans(x, p, l, u)	サブルーチン	
r_lgamma(x)	REAL	ガンマの対数
r_logb(x)	REAL	--
r_log1p(x)	REAL	--
r_log2(x)	REAL	--

表 1-2　数学単精度関数 (続き)

関数名	結果の型	内容の説明
r_max_normal()	REAL	
r_max_subnormal()	REAL	
r_min_normal()	REAL	
r_min_subnormal()	REAL	
r_nextafter(x, y)	REAL	
r_quiet_nan(n)	REAL	
r_remainder(x, y)	REAL	
r_rint(x)	REAL	
r_scalb(x, y)	REAL	
r_scalbn(x, n)	REAL	
r_signaling_nan(n)	REAL	
r_significand(x)	REAL	
r_sin(x)	REAL	正弦
r_sind(x)	REAL	--
r_sinh(x)	REAL	双曲正弦
r_sinp(x)	REAL	--
r_sinpi(x)	REAL	--
r_sincos(x, s, c)	サブルーチン	正弦と余弦
r_sincosd(x, s, c)	サブルーチン	--
r_sincosp(x, s, c)	サブルーチン	--
r_sincospi(x, s, c)	サブルーチン	--
r_tan(x)	REAL	正接
r_tand(x)	REAL	--
r_tanh(x)	REAL	双曲正接
r_tanp(x)	REAL	--
r_tanpi(x)	REAL	--
r_y0(x)	REAL	ベッセル関数
r_y1(x)	REAL	--
r_yn(n, x)	REAL	--

- 変数 c、l、p、s、u、x、y は REAL 型です。
- IMPLICIT 文が有効で、r で始まる名前を別のデータ型に対して指定する場合、これらの関数を REAL として明示的に指定します。
- sind(x) や asind(x) などでは、ラジアンではなく度が使用されます。

参照: intro (3M)、『数値計算ガイド』

倍精度関数

次の副プログラムは、倍精度の数学関数およびサブルーチンです。

通常、これらの関数は Fortran 規格の総称的な組み込み関数とは対応していません。データ型は、通常のデータ型決定規則によって決定されます。

これらの DOUBLEPRECISION 関数は DOUBLEPRECISION文に指定する必要があります。

詳細については、C ライブラリのマニュアルページを参照してください。d_acos(x) のマニュアルページは acos(3M) です。

表 1-3 数学倍精度関数

関数名	結果の型	内容の説明
d_acos(x)	DOUBLE PRECISION	逆余弦
d_acosd(x)	DOUBLE PRECISION	--
d_acosh(x)	DOUBLE PRECISION	逆双曲余弦
d_acosp(x)	DOUBLE PRECISION	--
d_acospi(x)	DOUBLE PRECISION	--
d_atan(x)	DOUBLE PRECISION	逆正接
d_atand(x)	DOUBLE PRECISION	--
d_atanh(x)	DOUBLE PRECISION	逆双曲正接
d_atanp(x)	DOUBLE PRECISION	--
d_atanpi(x)	DOUBLE PRECISION	--
d_asin(x)	DOUBLE PRECISION	逆正弦
d_asind(x)	DOUBLE PRECISION	--
d_asinh(x)	DOUBLE PRECISION	逆双曲正弦
d_asinp(x)	DOUBLE PRECISION	--
d_asinpi(x)	DOUBLE PRECISION	--
d_atan2((y, x)	DOUBLE PRECISION	逆正接
d_atan2d(y, x)	DOUBLE PRECISION	--
d_atan2pi(y, x)	DOUBLE PRECISION	--
d_cbrt(x)	DOUBLE PRECISION	立方根
d_ceil(x)	DOUBLE PRECISION	小数点以下切り上げ
d_copysign(x, x)	DOUBLE PRECISION	--

表 1-3　数学倍精度関数 (続き)

関数名	結果の型	内容の説明
d_cos(x)	DOUBLE PRECISION	余弦
d_cosd(x)	DOUBLE PRECISION	--
d_cosh(x)	DOUBLE PRECISION	双曲余弦
d_cosp(x)	DOUBLE PRECISION	--
d_cospi(x)	DOUBLE PRECISION	--
d_erf(x)	DOUBLE PRECISION	誤差関数
d_erfc(x)	DOUBLE PRECISION	--
d_expm1(x)	DOUBLE PRECISION	(e**x)-1
d_floor(x)	DOUBLE PRECISION	小数点以下切り捨て
d_hypot(x, y)	DOUBLE PRECISION	斜辺
d_infinity( )	DOUBLE PRECISION	--
d_j0(x)	DOUBLE PRECISION	ベッセル関数
d_j1(x)	DOUBLE PRECISION	--
d_jn(x)	DOUBLE PRECISION	--
id_finite(x)	INTEGER	
id_fp_class(x)	INTEGER	
id_ilogb(x)	INTEGER	
id_irint(x)	INTEGER	
id_isinf(x)	INTEGER	
id_isnan(x)	INTEGER	
id_isnormal(x)	INTEGER	
id_issubnormal(x)	INTEGER	
id_iszero(x)	INTEGER	
id_signbit(x)	INTEGER	
d_addran()	DOUBLE PRECISION	乱数発生関数
d_addrans(x, p, l, u)	サブルーチン	
d_lcran()	DOUBLE PRECISION	
d_lcrans(x, p, l, u)	サブルーチン	
d_shufrans(x, p, l,u)	サブルーチン	
d_lgamma(x)	DOUBLE PRECISION	ガンマの対数
d_logb(x)	DOUBLE PRECISION	--
d_log1p(x)	DOUBLE PRECISION	--
d_log2(x)	DOUBLE PRECISION	--

表 1-3　数学倍精度関数 (続き)

関数名	結果の型	内容の説明
d_max_normal()	DOUBLE PRECISION	
d_max_subnormal()	DOUBLE PRECISION	
d_min_normal()	DOUBLE PRECISION	
d_min_subnormal()	DOUBLE PRECISION	
d_nextafter(x, y)	DOUBLE PRECISION	
d_quiet_nan(n)	DOUBLE PRECISION	
d_remainder(x, y)	DOUBLE PRECISION	
d_rint(x)	DOUBLE PRECISION	
d_scalb(x, y)	DOUBLE PRECISION	
d_scalbn(x, n)	DOUBLE PRECISION	
d_signaling_nan(n)	DOUBLE PRECISION	
d_significand(x)	DOUBLE PRECISION	
d_sin(x)	DOUBLE PRECISION	正弦
d_sind(x)	DOUBLE PRECISION	--
d_sinh(x)	DOUBLE PRECISION	双曲正弦
d_sinp(x)	DOUBLE PRECISION	--
d_sinpi(x)	DOUBLE PRECISION	--
d_sincos(x, s, c)	サブルーチン	正弦と余弦
d_sincosd(x, s, c)	サブルーチン	--
d_sincosp(x, s, c)	サブルーチン	--
d_sincospi(x, s, c)	サブルーチン	
d_tan(x)	DOUBLE PRECISION	正接
d_tand(x)	DOUBLE PRECISION	--
d_tanh(x)	DOUBLE PRECISION	双曲正接
d_tanp(x)	DOUBLE PRECISION	--
d_tanpi(x)	DOUBLE PRECISION	--
d_y0(x)	DOUBLE PRECISION	ベッセル関数
d_y1(x)	DOUBLE PRECISION	--
d_yn(n, x)	DOUBLE PRECISION	--

- 変数 c, l、p、s、u、x、y は DOUBLE PRECISION 型です。
- DOUBLEPRECISION 文に、または適当な IMPLICIT 文でこれらの関数の型を明示的に指定します。
- sind(x) や asind(x) などでは、ラジアンではなく度が使用されます。

参照 : intro (3M)、『数値計算ガイド』

4 倍精度関数

これらの副プログラムは、4 倍精度 (REAL*16) の数学関数およびサブルーチンです 。

通常、これらの関数は Fortran 規格の総称組み込み関数とは対応していません。データ型は通常の型決定規則によって決定されます。

4 倍精度関数は REAL*16 文に指定しなくてはなりません。

表 1-4 4 倍精度 libm 関数

関数名	結果の型
q_copysign(x, y)	REAL*16
q_fabs(x)	REAL*16
q_fmod(x)	REAL*16
q_infinity( )	REAL*16
iq_finite(x)	INTEGER
iq_fp_class(x)	INTEGER
iq_ilogb(x)	INTEGER
iq_isinf(x)	INTEGER
iq_isnan(x)	INTEGER
iq_isnormal(x)	INTEGER
iq_issubnormal(x)	INTEGER
iq_iszero(x)	INTEGER
iq_signbit(x)	INTEGER
q_max_normal()	REAL*16
q_max_subnormal()	REAL*16
q_min_normal()	REAL*16
q_min_subnormal()	REAL*16
q_nextafter(x, y)	REAL*16
q_quiet_nan(n)	REAL*16
q_remainder(x, y)	REAL*16
q_scalbn(x, n)	REAL*16
q_signaling_nan(n)	REAL*16

- 変数 c、l、p、s、u、x、y は 4 倍精度です。

- REAL*16 文または適当な IMPLICIT 文でこれらの関数の型を明示的に指定します。
- sind(x) や asind(x) などでは、ラジアンではなく度が使用されます。

その他の 4 倍精度 libm 関数を使用する必要がある場合、その呼び出しの前に $PRAGMA C <関数名> を使用してください。詳細については、『Fortran プログラミングガイド』の 第 11 章「C と Fortran のインタフェース」を参照してください。

abort：終了とコアファイルへの書き込み

サブルーチンは、次のように呼び出されます。

```
call abort
```

abort は、入出力バッファをフラッシュ (バッファ内のデータを実際にファイルに書き込むこと) し、現在のディレクトリにコアファイルのメモリーダンプを作成して、処理を異常終了させます。コアダンプを制限、または行わないようにする方法については、limit(1) を参照してください。

access：ファイルのアクセス権または有無の検査

関数は、次のように呼び出します。

INTEGER*4 access status = access (*name, mode*)			
name	character	入力	ファイル名
mode	character	入力	アクセス権
戻り値	INTEGER*4	出力	*status*=0: 正常, *status*>0: エラーコード

access は、nameで指定したファイルに mode で指定したアクセス権でアクセスできるかどうかを決定します。mode で指定したアクセスが正常終了した場合は、ゼロが返されます。エラーコードを解釈する場合は、gerror(3F) も参照してください。

mode には、r、w、x を単独で指定することも、任意の順序で 2 つ以上組み合わせて指定することも、あるいは空白を指定することもできます。r、w、x の意味はそれぞれ以下のとおりです。

'r'	読み取りアクセス権をテストする
'w'	書き込みアクセス権をテストする
'x'	実行アクセス権をテストする
' '	ファイルの有無をテストする

例 1 : 読み取りおよび書き込みに関するアクセス権のテスト

```
      INTEGER*4  access, status
      status = access ( 'taccess.data', 'rw' )
      if ( status .eq. 0 ) write(*,*) "ok"
      if ( status .ne. 0 ) write(*,*) '読み取り/書き込み不可', status
```

例 2 : ファイルの有無のテスト

```
      INTEGER*4  access, status
      status = access ( 'taccess.data', ' ' )    ! 空白モード
      if ( status .eq. 0 ) write(*,*) "ファイル存在"
      if ( status .ne. 0 ) write(*,*) 'ファイルはない', status
```

alarm : 指定時間後のサブルーチンの呼び出し

関数は、次のように呼び出します。

INTEGER*4 alarm n = alarm (*time*, *sbrtn*)			
time	INTEGER*4	入力	待ち時間の秒数 (0 の場合は呼び出さない)
sbrtn	ルーチン名	入力	実行する副プログラムは EXTERNAL 文で宣言しなければならない
戻り値	INTEGER*4	出力	前回呼び出した alarm の残り時間

例 : alarm の使用例 : 9 秒待機してから sbrtn を呼び出します。

```
        integer*4 alarm, time / 1 /
        common / alarmcom / i
        external sbrtn
        i = 9
        write(*,*) i
        nseconds =  alarm ( time, sbrtn )
        do n = 1,100000          !alarm が sbrtn をアクティブにするまで待機
         r = n                 !(時間がかかる計算)
         x=sqrt(r)
        end do
        write(*,*) i
        end
        subroutine sbrtn
        common / alarmcom / i
        i = 3                         !このルーチンでは I/O を行わない
        return
        end
```

参照 : alarm(3C)、sleep(3F)、signal(3F) 以下の制限事項に注意してください。

- サブルーチンは自分自身の名前を alarm に渡すことはできません。
- alarm ルーチンは、入出力に干渉する可能性のあるシグナルを発生させます。呼び出されたサブルーチン (*sbrtn*) では、いっさい入出力を実行してはなりません。

- Fortran の並列プログラムまたはマルチスレッドプログラムから `alarm()` を呼び出すと、予期しない結果を招くことがあります。

`bit`：ビット関数: `and`、`or`、`bit`、`setbit`

定義は以下のとおりです。

`and(` *word1, word2* `)`	引数のビット単位の論理積を計算する
`or(` *word1, word2* `)`	引数のビット単位の論理和を計算する
`xor` (*word1, word2*)	引数のビット単位の排他的論理和を計算する
`not(` *word* `)`	引数のビット単位の補数を戻す
`lshift(` *word, nbits* `)`	循環桁上げなしで左へ論理シフトする
`rshift(` *word, nbits* `)`	符号拡張を行い右へ算術シフトする
`call bis(` *bitnum*, *word* `)`	*word* の第 *bitnum* ビットを 1 に設定する
`call bic(` *bitnum*, *word* `)`	*word* の第 *bitnum* ビットを 0 にクリアする
`bit(` *bitnum*, *word* `)`	*word* の第 *bitnum* ビットを検査し、ビットが 1 であれば `true` を返し、ビットが 0 であれば `false` を戻す
`call setbit(`*bitnum*,*word*,*state*`)`	*state* がゼロ以外であれば *word* の第 *bitnum* ビットを 1 に設定し、*state* がゼロであれば 0 にクリアする

MIL-STD-1753 の代替外部バージョンは以下のとおりです。

`iand(` *m*, *n* `)`	引数のビット単位の論理積を計算する
`ior(` *m*, *n* `)`	引数のビット単位の論理和を計算する
`ieor(` *m*, *n* `)`	引数のビット単位の排他的論理和を計算する
`ishft(` *m*, *k* `)`	循環桁上げなしで論理シフトする (*k*>0 のときは左、*k*<0 のときは右へ)
`ishftc(` *m*, *k*, *ic* `)`	循環シフト: *m* の、右から *ic* ビットを左へ *k* ビット循環シフトする
`ibits(` *m*, *i*, *len* `)`	ビットの切り出し: *i* ビット目から始まる *len* ビット分を *m* から切り出す

`ibset(` *m*, *i* `)`	ビットをセットする: ビット *i* が 1 であれば戻り値は m と同じ
`ibclr(` *m*, *i* `)`	ビットをクリアーする: ビット *i* が 0 であれば戻り値は *m* と同じ
`btest(` *m*, *i* `)`	ビットのテスト: *m* の *i* 番目のビットをテストする。ビットが 1 のときは `.true.` を返し、ビットが 0 のときは `.false.` を戻す

ビットフィールドを操作するその他の関数については、65 ページの「`mvbits` : ビットフィールドの移動」、および第 2 章と 3 章を参照してください。

and、or、xor、not、rshift、lshift の使用法

組み込み関数の場合は、次のように使います。

```
x = and( word1, word2 )
x = or( word1, word2 )
x = xor( word1, word2 )
x = not( word )
x = rshift( word, nbits )
x = lshift( word, nbits )
```

word、*word1*、*word2*、および *nbits* は、整数型の入力引数です。これらは組み込み関数で、コンパイラによりインライン展開されます。戻されるデータの型は、第 1 引数のデータ型です。

nbits の値が正当かどうかの検査は行われません。

例: and、or、xor、not

```
demo% cat tandornot.f
        print 1, and(7,4), or(7,4), xor(7,4), not(4)
 1      format(4x 'and(7,4)', 5x 'or(7,4)', 4x 'xor(7,4)',
     1         6x 'not(4)'/4o12.11)
        end
demo% f95 tandornot.f
demo% a.out
    and(7,4)     or(7,4)    xor(7,4)      not(4)
 00000000004 00000000007 00000000003 37777777773
demo%
```

例: lshift, rshift:

```
demo% cat tlrshift.f
       integer*4 lshift, rshift
       print 1, lshift(7,1), rshift(4,1)
 1     format(1x 'lshift(7,1)', 1x 'rshift(4,1)'/2o12.11)
       end
demo% f95 tlrshift.f
demo% a.out
 lshift(7,1) rshift(4,1)
 00000000016 00000000002
demo%
```

bic、bis、bit、setbit の使用法

サブルーチンと関数については、以下を参照してください。

---

call bic(*bitnum*, *word*)

call bis(*bitnum*, *word*)

call setbit(*bitnum*, *word*, *state*)

LOGICAL bit
x = bit(*bitnum*, *word*)

---

bitnum、*state*、および *word* は、INTEGER*4 型の入力引数です。bit() 関数では、論理値が返されます。

ビットは、ビット 0 が最下位ビット、ビット 31 が最上位ビットになるように番号が付けられます。

bic、bis、および setbit は外部サブルーチン、bit は外部関数です。

例 3 : bic、bis、setbit、bit

```
        integer*4 bitnum/2/, state/0/, word/7/
        logical bit
        print 1, word
 1      format(13x 'word', o12.11)
        call bic( bitnum, word )
        print 2, word
 2      format('bic(2,word)の後', o12.11)
        call bis( bitnum, word )
        print 3, word
 3      format('bis(2,word)の後', o12.11)
        call setbit( bitnum, word, state )
        print 4, word
 4      format('setbit(2,word,0)の後', o12.11)
        print 5, bit(bitnum, word)
 5      format('bit(2,word)', L )
        end
<出力>
             word 00000000007
bic(2,word)の後 00000000003
bis(2,word)の後 00000000007
setbit(2,word,0)の後 00000000003
bit(2,word) F
```

chdir : デフォルトディレクトリの変更

関数は、次のように呼び出します。

INTEGER*4 chdir *n* = chdir(*dirname*)			
dirname	character	入力	ディレクトリ名
戻り値	INTEGER*4	出力	*n*=0: 正常、*n*>0: エラーコード

例: chdir - 現在の作業ディレクトリを MyDir に変更します。

```
      INTEGER*4  chdir, n
      n =  chdir ( 'MyDir' )
      if ( n .ne. 0 ) stop 'chdir: error'
      end
```

参照: chdir(2)、cd(1)、gerror(3F) (エラーコードの解釈)

パス名は、<sys/param.h> で定義されている MAXPATHLEN より長くすることはできません。相対パス名でも、絶対パス名でもかまいません。

この関数を使用すると、装置による照会が失敗する場合があります。

いくつかの Fortran のファイル操作は、ファイルを名前で再オープンします。入出力動作中に chdir を使用すると、実行時システムが相対パス名で作成されたファイル (ファイル名を指定せずに open 文で作成されたファイルを含む) を見失ってしまうことがあります。

chmod : ファイルモードの変更

関数は、次のように呼び出します。

INTEGER*4 chmod *n* = chmod(*name*, *mode*)			
name	character	入力	パス名
mode	character	入力	chmod(1) に認識されるモード (o-w、444 など)
戻り値	INTEGER*4	出力	*n* = 0: 正常。*n*>0: システムエラー番号

例: chmod - 書き込み権を MyFile に追加します。

```
        character*18 name, mode
        INTEGER*4 chmod, n
        name = 'MyFile'
        mode = '+w'
        n =  chmod( name, mode )
        if ( n .ne. 0 ) stop 'chmod: error'
        end
```

参照：chmod(1)、gerror(3F) (エラーコードの解釈)

パス名を <sys/param.h> で定義されている MAXPATHLEN より長くすることはできません。相対パス名でも、絶対パス名でもかまいません。

date：文字列として現在の日付を取得

注 – このルーチンは年を示す場合に 2 桁の値しか返さないので、「2000 年には無効」になります。このルーチンの出力を使用して日付間の差を計算するプログラムは、1999 年12 月 31 日以降は正しく機能しなくなります。この date( ) ルーチンを使用しているプログラムは、ルーチンの初期呼び出し時に実行時警告メッセージを表示してユーザーに警告します。このルーチンの代わりに呼び出すことのできるルーチンとして、data_and_time() を参照して下さい。

サブルーチンは、次のように呼び出されます。

call date(*c*)			
c	CHARACTER*9	出力	変数、配列、配列要素、あるいは部分列

戻される文字列 c の形式は、*dd-mmm-yy* です。ここで、*dd* は 2 桁の数値で表した日、*mmm* は 3 文字に省略した英語の月名、*yy* は 2 桁の数値で表した年 (2000年には対応していない) です。

例: date:

```
demo% cat dat1.f
* dat1.f ー 日付けを文字列として取得
        character c*9
        call date ( c )
        write(*,"(' 本日の日付けは、', A9 )" ) c
        end
demo% f95 dat1.f
demo% a.out
Computing time differences using the 2 digit year from subroutine
date is not safe after year 2000.
本日の日付けは、9-Jan-02
demo%
```

idate() とdate_and_time()も参照してください。

date_and_time：日付と時刻の取得

これは、Fortran 95 組み込みルーチンで、2000 年以降も有効です。

date_and_time サブルーチンはリアルタイムクロックと日付のデータを返します。現地時間の他に、現地時間と世界標準時 (UTC: Universal Coordinated Time)(グリニッジ平均時 (GMT: Greenwich Mean Time) とも呼ぶ) の時差も返します。

date_and_time( ) サブルーチンは、次のように呼び出します。

call date_and_time(*date*, *time*, *zone*, *values*)			
date	CHARACTER*8	出力	日付。書式は CCYYMMDD。CCYY は 4 桁の年、MM は 2 桁の月、DD は 2 桁の日。次に例を示します。19980709
time	CHARACTER*10	出力	現在の時刻。書式は hhmmss.sss。hh は時、mm は分、ss.sss は秒とミリ秒
zone	CHARACTER*5	出力	UTC を使用した場合の時差。時分で示す。書式は hhmm
values	INTEGER*4 VALUES(8)	出力	以下で説明する 8 要素の整数配列

INTEGER*4 values に返される 8 つの値は次のとおりです。

VALUES(1)	4 桁の整数の年。たとえば、1998。
VALUES(2)	1〜12 の整数の月。
VALUES(3)	1〜31 の整数の日。
VALUES(4)	UTC を使用した場合の時差 (分)。
VALUES(5)	1〜23 の整数の時。
VALUES(6)	1〜59 の整数の分。
VALUES(7)	0〜60 の整数の秒。
VALUES(8)	0〜999 の範囲のミリ秒。

date_and_time の使用例 :

```
demo% cat dtm.f
       integer date_time(8)
       character*10 b(3)
       call date_and_time(b(1), b(2), b(3), date_time)
       print *,'date_timearray 配列の値'
       print *,'年=',date_time(1)
       print *,'month_of_year=',date_time(2)
       print *,'day_of_month=',date_time(3)
       print *,'時差(分)=',date_time(4)
       print *,'時=',date_time(5)
       print *,'分=',date_time(6)
       print *,'秒=',date_time(7)
       print *,'ミリ秒=',date_time(8)
       print *,'DATE=',b(1)
       print *,'TIME=',b(2)
       print *,'ZONE=',b(3)
       end
```

2000 年 2 月 16 日にカリフォルニアで実行した場合の出力は次のとおりです。

```
date_time 配列の値:
年= 2000
month_of_year= 2
day_of_month= 16
時差（分）= -420
時= 11
分= 49
秒= 29
ミリ秒= 236
DATE=20000216
TIME=114929.236
ZONE=-0700
```

dtime、etime：経過実行時間

これらの 2 つの関数は、経過実行時間 (あるいはエラー指示子として -1.0) を返します。返される時間は秒単位です。

Fortran 95 が使用する dtime と etime のバージョンは、デフォルトではシステムの低分解能クロックを使用します。分解能は 100 分の 1 秒です。ただし、プログラムが Sun OS™ オペレーティングシステムのユーティリティ ptime(1)、(/usr/proc/bin/ptime) の下で実行された場合は、高分解能クロックが使用されます。

dtime：前回の dtime 呼び出しからの経過時間

dtime の場合、経過時間は次のとおりです。

- 最初の呼び出し：実行開始からの経過時間
- 2 回目以降の呼び出し：前回の dtime の呼び出しからの経過時間
- シングルプロセッサ：CPU の使用時間
- マルチプロセッサ：すべての CPU 使用合計時間 (あまり便利ではないので、etime を使用してください)

注 – 並列化ループ内から dtime を呼び出すと、決定性のない結果になります。経過時間カウンタが、ループに関与しているすべてのスレッドに対してグローバルであるためです。

関数は、次のように呼び出します。

e = dtime(*tarray*)			
tarray	real(2)	出力	*e*= -1.0: エラー: *tarray* 値は未定義 *e*≠ -1.0: *tarray*(1) にユーザー時間 (エラーがない場合) *tarray*(2) にシステム時間 (エラーがない場合
戻り値	real	出力	*e*= -1.0: エラー *e*≠ -1.0: *tarray*(1) と *tarray*(2) の合計時間

例: dtime()、シングルプロセッサ

```
demo% cat tdtime.f
       real e, dtime, t(2)
       print *, 'elapsed:', e, ', user:', t(1), ', sys:', t(2)
       do i = 1, 10000
        k=k+1
       end do
       e = dtime( t )
       print *, 'elapsed:', e, ', user:', t(1), ', sys:', t(2)
       end
demo% f95 tdtime.f
demo% a.out
elapsed: 0.0E+0 , user: 0.0E+0 , sys: 0.0E+0
 elapsed: 0.03 , user: 0.01 , sys: 0.02
demo%
```

etime：実行開始からの経過時間

etime の場合、経過時間は次のとおりです。

- シングルプロセッサ実行：呼び出したプロセスの CPU 時間
- マルチプロセッサ実行：プログラムを処理している間の実時間

実行時ライブラリは、PARALLEL またはOMP_NUM_THREADS 環境変数が 1 より大きい整数に定義される場合に、プログラムのマルチプロセッサモードでの実行を決定します。

関数は、次のように呼び出します。

e = etime(*tarray*)			
tarray	real(2)	出力	*e*= -1.0: エラー: *tarray* の値は未定義 *e*≠ -1.0: シングルプロセッサ: *tarray*(1) にユーザー時間 (エラーがない場合) *tarray*(2) にシステム時間 (エラーがない場合 マルチプロセッサ: *tarray*(1) に実時間、*tarray*(2) に 0.0
戻り値	real	出力	*e*= -1.0: エラー *e*≠ -1.0: *tarray*(1) と *tarray*(2) の合計時間

etime の初期呼び出しで返される結果は不正確です。初期呼び出しでは、単にシステムクロックを稼働させるだけなので、etime の初期呼び出しで返された値は使用しないでください。

例: etime() ー シングルプロセッサ

```
demo% cat tetime.f
       real e, etime, t(2)
       e = etime(t)          !Startup etime - do not use result
       do i = 1, 10000
        k=k+1
       end do
       e = etime( t )
       print *, 'elapsed:', e, ', user:', t(1), ', sys:', t(2)
       end
demo% f95 tetime.f
demo% a.out
elapsed:0.02 , user:0.01 , sys: 0.01
demo%
```

times(2) のマニュアルページ、および『Fortran プログラミングガイド』も参照してください。

exit：プロセスの終了および状態の設定

サブルーチンは、次のように呼び出されます。

call exit(*status*)		
status	INTEGER*4	入力

例: exit():

```
      ...
        if(dx .lt. 0.) call exit( 0 )
     ...
        end
```

exit はフラッシュしてからプロセスのすべてのファイルを閉じ、その親プロセスが wait を実行している場合は親プロセスに通知します。

親プロセスは *status* の下位 8 ビットを使用できます。この 8 ビットは左に 8 ビットシフトされ、他のビットはすべてゼロになります (したがって *status* は 256 ～ 65280 の範囲になります)。この呼び出しは復帰しません。

C の関数である exit は、最終的なシステム終了動作が実行される前に整理の処理を行うことがあります。

引数なしで exit を呼び出すとコンパイル時警告メッセージが出され、自動的に引数にゼロが与えられます。参照 : exit(2)、fork(2)、fork(3F)、wait(2)、wait(3F)

fdate：ASCII 文字列で日付および時刻を返す

サブルーチンまたは関数は、次のように呼び出されます。

call fdate(*string*)		
string	character*24	出力

または

CHARACTER fdate*24 *string* = fdate()			fdate を関数として使用する場合、それを呼び出すルーチンは fdate の型と長さを定義する必要がある
戻り値	character*24	出力	

例 1 : サブルーチンとしての使用

```
        character*24 string
        call fdate( string )
        write(*,*) string
        end
```

上記の例の出力は次のようになります。

```
  Wed Aug 3 15:30:23 1994
```

例 2 : 関数としての使用。出力は上記の例と同じ

```
        character*24 fdate
        write(*,*)  fdate()
        end
```

参照 : ctime(3)、time(3F)、および idate(3F)

flush : 論理装置への出力のフラッシュ

関数は、次のように呼び出します。

INTEGER*4 flush *n* = flush(*lunit*)			
lunit	INTEGER*4	入力	論理装置
戻り値	INTEGER*4	出力	*n* = 0 エラーなし *n* > 0 エラー番号

flush 関数は、論理装置 lunitに対するバッファの内容を結合されているファイルにフラッシュします。このサブルーチンがもっとも役に立つのは、論理装置 0 と 6 がどちらもコンソールに結合されていて、それらの装置に対してこのサブルーチンを使用する場合です。関数はエラーが発生すると、正のエラー番号を返し、エラーが発生しないとゼロを返します。

参照 : fclose(3S)

fork : 現プロセスのコピーの生成

関数は、次のように呼び出します。

INTEGER*4 fork *n* = fork()			
戻り値	INTEGER*4	出力	*n*>0: *n*= コピーのプロセス識別子 *n*<0, *n*= (システムエラーコード)

fork 関数はそれを呼び出したプロセスのコピーを生成します。元のプロセスとコピーとの違いは、元のプロセス (親プロセスと呼ばれる) に返される値がコピーのプロセス識別子であるということだけです。コピーは一般に子プロセスと呼ばれます。子プロセスに返される値はゼロです。

書き込み用に開いているすべての論理装置は、fork が実行される前にフラッシュされます。これは入出力バッファの内容が外部ファイルに重複して書き込まれるのを防ぎます。

例: fork():

```
        INTEGER*4 fork, pid
        pid = fork()
        if(pid.lt.0) stop 'フォーク失敗'
        if(pid.gt.0) then
               print *, '親プロセス'
        else
               print *, '子プロセス'
        endif
```

fork ルーチンと対をなす exec ルーチンは提供されていません。これは論理装置を開いたままで exec ルーチンに渡せる良い方法がないためです。ただし、system(3F) を使用すれば fork/exec の通常の機能を実行することができます。参照 : fork(2)、wait(3F)、kill(3F)、system(3F)、および perror(3F)

fseek、ftell：ファイルのポインタの位置付けと再位置付け

fseek および ftell は、ファイルの再位置付けを可能にするルーチンです。ftell は、ファイルの現在位置をファイルの先頭からのオフセットを示すバイト数で返します。プログラムの後方で、この値を使用して fseek を呼ぶことにより、ファイルの読み込み位置を元に戻すことができます。

fseek：論理装置上のファイルのポインタの再位置付け

関数は、次のように呼び出します。

INTEGER*4 fseek *n* = fseek(*lunit*, *offset*, *from*)			
lunit	INTEGER*4	入力	開いている論理装置
offset	INTEGER*4 または INTEGER*8	入力	*from*で指定された位置からのオフセットを示すバイト数
	-xarch=v9 を使って、Solaris 7 または 8 などの 64 ビット環境用にコンパイルする場合は、INTEGER*8 オフセット値が必要。定数を入力する場合は、それを 64 ビット定数にする必要がある。たとえば、100_8		
from	INTEGER*4	入力	0=ファイルの先頭 1=現在の位置 2=ファイルの終了
戻り値	INTEGER*4	出力	*n*=0: 正常。*n*>0: システムエラーコード

注 – 順編成ファイルでは、`fseek 64` に続く呼び出しの後の出力操作 (WRITE など) は、`fseek` の位置に続くすべてのデータレコードの削除、新しいデータレコード (とファイルの終わりのマーク) での書き換えの原因となります。正しい位置へのレコードの書き換えは、直接アクセスファイルでのみ実行可能です。

例: `fseek()- MyFile` のポインタを先頭から 2 バイトの位置に再位置付けします。

```
        INTEGER*4 fseek, lunit/1/, offset/2/, from/0/, n
        open( UNIT=lunit, FILE='MyFile' )
        n = fseek( lunit, offset, from )
        if ( n .gt. 0 ) stop 'fseek エラー'
        end
```

例: 上記の例を 64 ビット環境で、`-xarch=v9` を使ってコンパイルすると次のようになります。

```
        INTEGER*4 fseek, lunit/1/,  from/0/, n
        INTEGER*8 offset/2/
        open( UNIT=lunit, FILE='MyFile' )
        n = fseek( lunit, offset, from )
        if ( n .gt. 0 ) stop 'fseek エラー'
        end
```

`ftell`：ファイルの現在位置を戻す

関数は、次のように呼び出します。

`INTEGER*4 ftell` *n* = `ftell(` *lunit* `)`			
lunit	`INTEGER*4`	入力	開いている論理装置
戻り値	`INTEGER*4`	出力	*n*>=0: *n*=ファイルの先頭からのオフセットを示すバイト数 *n*<0: *n*=システムエラーコード

例: `ftell()`

```
INTEGER*4 ftell, lunit/1/, n
open( UNIT=lunit, FILE='MyFile' )
...
n = ftell( lunit )
if ( n .lt. 0 ) stop 'ftell エラー'
...
```

例: 上記の例を 64 ビット環境で、`-xarch=v9` を使ってコンパイルすると次のようになります。

```
INTEGER*4 lunit/1/
INTEGER*8 ftell, n
open( UNIT=lunit, FILE='MyFile' )
...
n = ftell( lunit )
if ( n .lt. 0 ) stop 'ftell エラー'
...
```

参照 : `fseek`(3S)、`perror`(3F)、`fseeko64`(3f)、`ftello64`(3f)

`fseeko64`、`ftello64`：大規模ファイルのポインタの位置付けと再位置付け

`fseeko64` と `ftello64` は、それぞれ `fseek` と `ftell` の 「大規模ファイル」 バージョンです。`fseeko64` と `ftello64` は、`INTEGER*8` ファイル位置のオフセットを入出力します。(「大規模ファイル」 とは 2G バイトを超えるファイルのことで、バイト位置は 64 ビットの整数で示します。) これらのバージョンを使用して、大規模ファイルのポインタの位置付けや再位置付けを行います。

fseeko64：論理装置上のファイルのポインタの再位置付け

関数は、次のように呼び出します。

INTEGER fseeko64 *n* = fseeko64(*lunit*, *offset64*, *from*)			
lunit	INTEGER*4	入力	開いている論理装置
offset64	INTEGER*8	入力	*from* で指定された位置からの 64 ビット オフセットを示すバイト数
from	INTEGER*4	入力	0=ファイルの先頭 1=現在の位置 2=ファイルの終了
戻り値	INTEGER*4	出力	*n*=0: 正常。*n*>0: システムエラーコード

注 – 順編成ファイルでは、fseek 64 に続く呼び出しの後の出力操作 (WRITE など) は、fseek の位置に続くすべてのデータレコードの削除、新しいデータレコード (とファイルの終わりのマーク) での書き換えの原因となります。正しい位置へのレコードの書き換えは、直接アクセスファイルでのみ実行可能です。

例: fseeko64( )- MyFile のポインタを先頭から 2 バイトの位置に再位置付けします。

```
      INTEGER fseeko64, lunit/1/, from/0/, n
      INTEGER*8 offset/200/
      open( UNIT=lunit, FILE='MyFile' )
      n = fseeko64( lunit, offset, from )
      if ( n .gt. 0 ) stop 'fseek エラー'
      end
```

ftello64：ファイルの現在位置を戻す

関数は、次のように呼び出します。

INTEGER*8 ftello64 *n* = ftello64(*lunit*)			
lunit	INTEGER*4	入力	開いている論理装置
戻り値	INTEGER*8	出力	*n*≥0: *n*=ファイルの先頭からのオフセットを示すバイト数 *n*<0: *n*=システムエラーコード

例: ftello64( ):

```
INTEGER*8 ftello64, lunit/1/, n
open( UNIT=lunit, FILE='MyFile' )
...
n = ftello64( lunit )
if ( n .lt. 0 ) stop 'ftell エラー'
...
```

getarg、iargc：コマンド行の引数の取得

getarg と iargc は、コマンド行プリプロセッサによって展開されたコマンド行引数にアクセスします。

getarg：コマンド行の引数の取得

サブルーチンは、次のように呼び出されます。

call getarg(*k*, *arg*)			
k	INTEGER*4	入力	引数の索引 (0 = 最初の引数 = コマンド名)
arg	character**n*	出力	*k* 番目の引数
n	INTEGER*4	引数のサイズ	もっとも長い引数が入るだけの大きさ

iargc：コマンド行の引数の個数の取得

関数は、次のように呼び出します。

m = iargc()			
戻り値	INTEGER*4	出力	コマンド行の引数の個数

例: iargc と getarg: 引数の個数を調べ、各引数を読み取ります。

```
demo% cat yarg.f
       character argv*10
       INTEGER*4 i, iargc, n
       n = iargc()
       do 1 i = 1, n
         call getarg( i, argv )
 1       write( *, '( i2, 1x, a )' ) i, argv
       end
demo% f95 yarg.f
demo% a.out *.f
1 first.f
2 yarg.f
```

参照 : execve(2)、getenv(3F)

getc、fgetc：次の文字の取得

getc と fgetc は、入力ストリームから次の文字を読み取ります。同じ論理装置上では、これらのルーチンの呼び出しを通常の Fortran の入出力と混合して使用しないでください。

getc：標準入力からの次の文字の取得

関数は、次のように呼び出します。

INTEGER*4 getc *status* = getc(*char*)			
char	character	出力	次の文字
戻り値	INTEGER*4	出力	*status*=0: 正常 *status*=-1: ファイルの終了 *status*>0: システムエラーコードまたはf77/f90 入出力エラーコード

例: getc でキーボードから文字を 1 文字ずつ入力します。Control-D (^D) に注意してください。

```
        character char
        INTEGER*4 getc, status
        status = 0
        do while ( status .eq. 0 )
          status = getc( char )
          write(*, '(i3, o4.3)') status, char
        end do
        end
```

上記のソースプログラムを (コンパイル後に) 実行した例を以下に示します。

```
demo% a.out
ab              プログラムが入力された文字を読み取る。
0 141           プログラムが入力された文字の状態コードと 8 進値を出力する。
0 142                 141 は 'a' を、142 は 'b' を表す。
0 012                 012 はリターンキーを表す。
^D              Control-D キーで終了された。
-1 377          次の読み取りが試行され Control-D が戻された。
demo%
```

どの論理装置に対しても、通常の Fortran の入力と getc() を混在して使用しないでください。

fgetc：指定した論理装置からの次の文字の取得

関数は、次のように呼び出します。

INTEGER*4 fgetc *status* = fgetc(*lunit*, *char*)			
lunit	INTEGER*4	入力	論理装置
char	character	出力	次の文字
戻り値	INTEGER*4	出力	*status*=-1: ファイルの終了 *status*>0: システムエラーコードまたはf77/f90 入出力エラーコード

例: fgetc で tfgetc.data から文字を 1 文字ずつ読み取ります。改行 (8 進の 012) に注意してください。

```
        character char
        INTEGER*4 fgetc, status
        open( unit=1, file='tfgetc.data' )
        status = 0
        do while ( status .eq. 0 )
           status = fgetc( 1, char )
           write(*, '(i3, o4.3)') status, char
        end do
        end
```

上記のソースプログラムを (コンパイル後に) 実行した例を以下に示します。

```
demo% cat tfgetc.data
ab
yz
demo% a.out
0 141       'a' が読み取られる
0 142       'b' が読み取られる
0 012       改行が読み取られる
0 171       'y' が読み取られる
0 172       'z' が読み取られる
0 012       改行が読み取られる
-1 012      CONTROL-D が読み取られる
demo%
```

どの論理装置に対しても、通常の Fortran の入力と fgetc() を混在して使用しないでください。

参照 : getc(3S)、intro(2)、および perror(3F)

getcwd : 現在のディレクトリパスの取得

関数は、次のように呼び出します。

INTEGER*4 getcwd *status* = getcwd(*dirname*)			
dirname	character*n	出力 現在のディレクトリのパスが返される。	現在のディレクトリのパス名。n は、もっとも長いパス名が入るのに十分な大きさであることが必要
戻り値	INTEGER*4	出力	*status*=0: 正常 *status*>0: エラーコード

例: getcwd:

```
        INTEGER*4 getcwd, status
        character*64 dirname
        status = getcwd( dirname )
        if ( status .ne. 0 ) stop 'getcwd: error'
        write(*,*) dirname
        end
```

参照 : chdir(3F)、perror(3F)、および getwd(3)

注意: パス名を <sys/param.h> で定義されている MAXPATHLEN より長くすることはできません。

getenv：環境変数の値の取得

サブルーチンは、次のように呼び出されます。

call getenv(*ename*, *evalue*)			
ename	character*n	入力	検索する環境変数の名前
evalue	character*n	出力	見つかった環境変数の値。見つからなかった場合は空

ename と *evalue* には、それぞれの文字列が十分入るだけの大きさが必要です。

evalue が小さすぎてすべての文字列を格納できない場合は、文字列は *evalue* の長さに合わせて切り捨てされます。

getenv サブルーチンは環境リストから *ename=evalue* の形式の文字列を検索し、その文字列があった場合には *evalue* の値を返し、なかった場合には *evalue* を空白で埋めます。

例: $SHELL の値を印刷するには、getenv() を使用します。

```
      character*18  evalue
      call getenv( 'SHELL', evalue )
      write(*,*) "'", evalue, "'"
      end
```

参照 : *execve*(2) および *environ*(5)

getfd : 外部装置番号に対するファイル記述子の取得

関数は、次のように呼び出します。

INTEGER*4 getfd *fildes* = getfd(*unitn*)			
unitn	INTEGER*4	入力	外部装置番号
戻り値	INTEGER*4- または -INTEGER*8	出力	ファイルが結合されている場合はファイル記述子、結合されていない場合は -1。64 ビット環境用にコンパイルすると、結果として INTEGER*8 が戻る。

例: getfd():

```
      INTEGER*4 fildes, getfd, unitn/1/
      open( unitn, file='tgetfd.data' )
      fildes = getfd( unitn )
     if ( fildes .eq. -1 ) stop 'getfd: ファイルは結合されていません'
      write(*,*) 'ファイル記述子 = ', fildes
      end
```

参照 : open(2)

getfilep：外部装置番号に対するファイルポインタの取得

関数は、次のように呼び出します。

irtn = *c_read*(getfilep(*unitn*), *inbyte*, 1)			
c_read	C 関数	入力	この C 関数はユーザーが書く。下記の例を参照
unitn	INTEGER*4	入力	外部装置番号
getfilep	INTEGER*4-または -INTEGER*8	戻り値	ファイルが結合されている場合はファイルポインタ、結合されていない場合は -1。64 ビット環境用にコンパイルすると、INTEGER*8 の値が戻る。

この関数は標準 Fortran の入出力と C の入出力を混在させるために使用します。このような混在は移植不可能であり、今後リリースされるオペレーティングシステムまたは Fortran で使用できる保証はありません。したがって、この関数の使用は勧められませんし、直接のインタフェースは提供されていません。ユーザーは getfilep が戻す値を使用するために 独自の C ルーチンを作成する必要があります。C ルーチンの例を以下に示します。

例: Fortran は C の関数に渡すのに getfilep を使用します。

```
demo% cat tgetfilepF.f

        character*1  inbyte
        integer*4    c_read,  getfilep, unitn / 5 /
        external     getfilep
        write(*,'(a,$)') '数字は何? '

        irtn = c_read( getfilep( unitn ), inbyte, 1 )

        write(*,9)  inbyte
    9   format('C の読み取った数字は ', a )
        end
```

getfilep を実際に使用する C 関数の例を以下に示します。

```
demo% cat tgetfilepC.c

#include <stdio.h>
int c_read_ ( fd, buf, nbytes, buf_len )
FILE **fd ;
char *buf ;
int *nbytes, buf_len ;
{
          return fread( buf, 1, *nbytes, *fd ) ;
}
```

上記のソースプログラムをコンパイル、リンク、実行した例を以下に示します。

```
demo% cc -c tgetfilepC.c
demo% f95 tgetfilepC.o tgetfilepF.f
demo% a.out
数字は何 ? 3
C の読み取った数字は 3
demo%
```

詳細については、『Fortran プログラミングガイド』の 第 11 章「C と Fortran のインタフェース」を参照してください。参照 : open(2)

getlog : ユーザーのログイン名の取得

サブルーチンは、次のように呼び出されます。

call getlog(*name*)			
name	character*n	出力	ユーザーのログイン名。プロセスが端末から切り離されて実行されている場合はすべて空白。*n* は、もっとも長い名前が入るのに十分な大きさであることが必要

例: getlog:

```
        character*18 name
        call getlog( name )
        write(*,*) "'", name, "'"
        end
```

参照 : getlogin(3)

getpid : プロセス識別子の取得

関数は、次のように呼び出します。

INTEGER*4 getpid *pid* = getpid()			
戻り値	INTEGER*4	出力	現プロセスのプロセス識別子 (ID)

例: getpid:

```
        INTEGER*4 getpid, pid
        pid = getpid()
        write(*,*) 'プロセス ID = ', pid
        end
```

参照 : getpid(2)

getuid、getgid : プロセスのユーザー識別子またはグループ識別子の取得

getuid と getgid はそれぞれ、ユーザー識別子またはグループ識別子を読み取ります。

getuid：プロセスのユーザー識別子の取得

関数は、次のように呼び出します。

INTEGER*4 getuid *uid* = getuid()			
戻り値	INTEGER*4	出力	プロセスのユーザー識別子 (ID)

getgid：プロセスのグループ識別子の取得

関数は、次のように呼び出します。

INTEGER*4 getgid gid = getgid()			
戻り値	INTEGER*4	出力	プロセスのグループ識別子 (ID)

例: getuid() と getpid():

```
        INTEGER*4 getuid, getgid, gid, uid
        uid = getuid()
        gid = getgid()
        write(*,*) uid, gid
        end
```

参照 : getuid(2)

hostnm : 現在のホスト名の獲得

関数は、次のように呼び出します。

INTEGER*4 hostnm *status* = hostnm(*name*)			
name	character*n	出力	現在のホストの名前。*n* は、ホスト名が入るのに十分な大きさであることが必要
戻り値	INTEGER*4	出力	*status*=0: 正常 *status*>0: エラー

例: hostnm():

```
      INTEGER*4 hostnm, status
      character*8 name
      status = hostnm( name )
      write(*,*) 'ホスト名 = "', name, '"'
      end
```

参照 : gethostname(2)

idate : 現在の日付を戻す

idate は、現在のシステム日付を 1 つの整数配列に日、月、年の順で入れます。

サブルーチンは、次のように呼び出されます。

call idate(*iarray*)			標準バージョン
iarray	INTEGER*4	出力	3 要素数の配列。日、月、年

例: idate (標準バージョン)

```
demo% cat tidate.f
       INTEGER*4 iarray(3)
       call idate( iarray )
       write(*, "(' 日付は: ',3i5)" )  iarray
       end
demo% f95 tidate.f
demo% a.out
 日付は: 10 8 1998
demo%
```

ieee_flags、ieee_handler、sigfpe:IEEE 算術演算

これらの副プログラムは、Fortran プログラムで ANSI/IEEE 規格 754-1985 の算術演算機能を十分に利用するために必要なモードと状態を提供します。これらの副プログラムは関数 ieee_flags(3M)、ieee_handler(3M)、および sigfpe(3) と密接に対応しています。

以下に、要約を示します。

表 1-5 IEEE 算術演算サポートルーチン

ieeer = ieee_flags(*action*,*mode*,*in*,*out*)		
ieeer = ieee_handler(*action*,*exception*,*hdl*)		
ieeer = sigfpe(*code*, *hdl*)		
action	character	入力
code	sigfpe_code_type	入力
mode	character	入力
in	character	入力
exception	character	入力
hdl	sigfpe_handler_type	入力
out	character	出力
戻り値	INTEGER*4	出力

これらの関数を効果的に使用する方法については、『数値計算ガイド』を参照してください。

`sigfpe` を使用する場合、浮動小数点状態レジスタ内の対応するトラップ可能マスクビットをユーザーが設定する必要があります。詳細は『SPARC アーキテクチャマニュアルバージョン 8』(トッパン刊) で説明されています。`libm` 関数の `ieee_handler` を呼び出すと、トラップ可能マスクビットが自動的に設定されます。

mode と *exception* が受け付ける文字型のキーワードは、*action* の値によって異なります。

表 1-6 `ieee_flags`(*action,mode,in,out*) パラメータと動作

action = `'clearall'`	*mode*、*in*、*out* は未使用。戻り値は 0	
action = `'clear'` *clear mode, in* *out* は未使用。戻り値は 0	*mode* = `'direction'`	

	mode = `'exception'`	*in* = `'inexact'`
		`'division'`
		`'underflow'`
		`'overflow'`
		`'invalid'`
		`'all'`
		`'common'` のいずれか

表 1-6　ieee_flags(*action*,*mode*,*in*,*out*) パラメータと動作 (続き)

action = 'set' 浮動小数点の *mode* と *in* を設定する。 *out* は未使用。戻り値は 0	*mode* = 'direction'	*in* = 'nearest' 'tozero' 'positive' 'negative'
	mode = 'exception'	*in* = 'inexact' 'division' 'underflow' 'overflow' 'invalid' 'all' 'common' のいずれか
action = 'get' *mode* の設定値を調査する。 *in*、*out* は、空白にするか、テスト対象の設定値の 1 つを設定する。*in*、*out* に設定値を設定すると、*mode* の設定値に従った現在の設定値または not available (無効) が戻る。関数は 0 を戻す。ただし、*mode* = exception の場合は、現在の例外フラグを戻す。	*mode* = 'direction'	*out* = 'nearest' 'tozero' 'positive' 'negative'
	mode = 'exception'	*out* = 'inexact' 'division' 'underflow' 'overflow' 'invalid' 'all' 'common' のいずれか

表 1-7　ieee_handler(action,in,out) パラメータ

action = *'clear'* in に指定したユーザー例外処理をクリアする。 out は未使用。	*in* = 'inexact' 'division' 'underflow' 'overflow' 'invalid' 'all' 'common' のいずれか
action = *'set'* *in* にユーザー例外処理を設定する。*out* は、ルーチンのアドレス、または f77/f77_floating point.h に定義されている SIGFPE_DEFAULT、SIGFPE_ABORT、または SIGFPE_IGNORE。	*in* = 'inexact' 'division' 'underflow' 'overflow' 'invalid' 'all' 'common' のいずれか

例 1 : (ハードウェアが方向をもつ丸めモードをサポートしていない場合を除いて) 丸め方向をゼロの方向に設定します。

```
INTEGER*4 ieeer
character*1 mode, out, in
ieeer = ieee_flags( 'set', 'direction', 'tozero', out )
```

例 2 : 丸め方向をクリアします (デフォルトの方向、つまり四捨五入して丸めます)。

```
character*1 out, in
ieeer = ieee_flags('clear','direction', in, out )
```

例 3 : 設定されている例外発生ビットをすべてクリアします。

```
character*18 out
ieeer = ieee_flags( 'clear', 'exception', 'all', out )
```

例 4 : 例 3 でオーバーフロー例外が発生すると、次のように検出します。

```
    character*18 out
    ieeer = ieee_flags( 'get', 'exception', 'overflow', out )
    if (out .eq. 'overflow' ) stop 'overflow'
```

上記の例は、*out* を overflow にし、ieeer を 25 に設定しています。同様にコーディングすれば、invalid や inexact のような例外を検出できます。

例 5 : handl.f の内容。シグナルハンドラを書き込み、使用しています。

```
      external hand
      real r / 14.2 /,  s / 0.0 /
      i = ieee_handler( 'set', 'division', hand )
      t = r/s
      end

      INTEGER*4 function hand ( sig, sip, uap )
      INTEGER*4 sig, address
      structure /fault/
          INTEGER*4 address
      end structure
      structure /siginfo/
          INTEGER*4 si_signo
          INTEGER*4 si_code
          INTEGER*4 si_errno
          record /fault/ fault
      end structure
      record /siginfo/ sip
      address = sip.fault.address
      write (*,10) address
10    format('例外の起きたアドレス (16進) ', z8 )
      end
```

address と function hand の宣言を INTEGER*8 に変更すると、64 ビットの SPARC V9 環境 (-xarch=v9) で例 5 が実行できます。

『数値計算ガイド』を参照してください。参照 : floatingpoint(3)、signal(3)、sigfpe(3)、floatingpoint(3F)、ieee_flags(3M)、および ieee_handler(3M)

floatingpoint.h：Fortran IEEE 定義

ヘッダーファイル floatingpoint.h は、ANSI/IEEE 規格 754-1985 に従って、標準浮動小数点の実装に使用される定数と型を定義します。

このファイルの Fortran 95 ソースプログラムへのインクルードは、次のように行います。

```
#include "floatingpoint.h"
```

このインクルードファイルを使用するには、Fortran のコンパイル前に前処理が必要になります。このインクルードファイルを参照するソースファイルは、名前の拡張子が F、f90 またはF95の場合に、自動的に前処理が行われます。

IEEE 丸めモード

fp_direction_type	IEEE 丸め方向モードの型。列挙の順序はハードウェアにより異なるので注意すること。

SIGFPE 処理

sigfpe_code_type	SIGFPE コードの型
sigfpe_handler_type	ユーザー定義の SIGFPE 例外ハンドラの型。特定の SIGFPE コードを処理するために呼び出される。
SIGFPE_DEFAULT	デフォルトの SIGFPE 例外処理を指示するマクロ。IEEE 例外。デフォルトの結果で実行を継続させ、他の SIGFPE コードに対しては、実行を異常終了させる。
SIGFPE_IGNORE	代替 SIGFPE 例外処理を指示するマクロ。無視して実行を継続させる。
SIGFPE_ABORT	代替 SIGFPE 例外処理を指示するマクロ。コアダンプを取り、実行を異常終了させる。

IEEE 例外処理

N_IEEE_EXCEPTION	IEEE 浮動小数点例外の数
fp_exception_type	N_IEEE_EXCEPTION 個の例外の型。各例外はビット番号を与えられる。
fp_exception_field_type	fp_exception_type により番号が与えられた IEEE 例外に対応する N_IEEE_EXCEPTION 個のビットだけをとることを目的とした型。たとえば fp_inexact は最下位ビットに対応し、fp_invalid は最下位から 5 番目のビットに対応する。操作によっては 2 つ以上の例外を設定できる。

IEEE クラス分類

fp_class_type	IEEE 浮動小数点の値と記号のクラスの並び

『数値計算ガイド』を参照してください。参照 : ieee_environment(3F)

index、rindex、lnblnk : 部分列のインデックスまたは長さ

これらの関数は、次のように文字列による探索を行います。

index(*a1*,*a2*)	文字列 *a1* の中で最初に出現する文字列 *a2* のインデックス
rindex(*a1*,*a2*)	文字列 *a1* の中で最後に出現する文字列 *a2* のインデックス
lnblnk (*a1*)	文字列 *a1* の中の空白以外の最後の文字のインデックス

index は以下の形式をとります。

index：文字列の中で最初に出現する部分文字列

index は、組み込み関数で次のように呼び出します。

n = index(*a1*, *a2*)			
a1	character	入力	文字列
a2	character	入力	部分列
戻り値	INTEGER	出力	*n*>0: *a1* の中で最初に出現する *a2* のインデックス *n*=0: *a1* の中に *a2* が出現しない

INTEGER*8 と宣言されている場合は、64 ビット環境用にコンパイルされ、さらに文字変数 *a1* が非常に大きな文字列であるときに (2 Gバイトを超えるもの)、index( ) は INTEGER*8 値を戻します。

rindex：文字列の中で最後に出現する部分文字列

関数は、次のように呼び出します。

INTEGER*4 rindex *n* = rindex(*a1*, *a2*)			
a1	character	入力	文字列
a2	character	入力	部分列
戻り値	INTEGER*4 または INTEGER*8	出力	*n*>0: *a1* の中で最後に出現する *a2* のインデックス *n*=0: *a1* の中に *a2* が出現しない。64 ビット環境の場合は、INTEGER*8 が戻る。

lnblnk：文字列の中の空白以外の最後の文字

関数は、次のように呼び出します。

n = lnblnk(*a1*)			
a1	character	入力	文字列
戻り値	INTEGER*4 または INTEGER*8	出力	*n*>0: *a1* の中の空白以外の最後の文字のインデックス *n*=0: *a1* はすべて空白以外の文字。64 ビット環境の場合は、INTEGER*8 が戻る。

例: index()、rindex()、lnblnk():

```
demo% cat tindex.f
*                          123456789012345678901
       character s*24 / 'abcPDQxyz...abcPDQxyz' /
       INTEGER*4 declen, index, first, last, len, lnblnk, rindex
       declen = len( s )
       first = index( s, 'abc' )
       last = rindex( s, 'abc' )
       lastnb = lnblnk( s )
       write(*,*) declen, lastnb
       write(*,*) first, last
       end
demo% f95 tindex.f
demo% a.out
24 21    <- 組み込み関数 len()が宣言されたS の長さを返すため、declen は 24
1 13
```

注 – 64 ビット環境で動作するようコンパイルされたプログラムは、非常に大きな文字列を処理するには index、rindex、および lnblnk (および返される変数) INTEGER*8 を宣言しなければなりません。

inmax：正の整数の最大値の返却

関数は、次のように呼び出します。

m = inmax()			
戻り値	INTEGER*4	出力	正の整数の最大値

例: inmax:

```
demo% cat tinmax.f
        INTEGER*4 inmax, m
        m = inmax()
        write(*,*) m
        end
demo% f95 tinmax.f
demo% a.out
   2147483647
demo%
```

参照：libm_single (3F) およびlibm_double(3F)、第 3 章で記述されている非標準 FORTRAN 77 組み込み関数ephuge()。

itime：現在の時刻

itime は、現在のシステム時刻の時、分、秒を整数配列に入れます。サブルーチンは、次のように呼び出されます。

call itime(*iarray*)			
iarray	INTEGER*4	出力	3 要素の配列: *iarray*(1) = 時 *iarray*(2) = 分 *iarray*(3) = 秒

例: itime:

```
demo% cat titime.f
       INTEGER*4 iarray(3)
       call itime( iarray )
       write(*, "(' 時刻は: ',3i5)" )  iarray
       end
demo% f95 titime.f
demo% a.out
 時刻は: 15 42 35
```

参照 : time(3f)、ctime(3F)、fdate(3F)

kill : プロセスへのシグナルの送信

関数は、次のように呼び出します。

status = kill(*pid*, *signum*)			
pid	INTEGER*4	入力	ユーザーのプロセスのプロセス識別子
signum	INTEGER*4	入力	有効なシグナル番号。*signal*(3) を参照
戻り値	INTEGER*4	出力	*status*=0: 正常 *status*>0: エラーコード

例 (該当部分のみ): kill() を使用してメッセージを送ります。

```
       INTEGER*4 kill, pid, signum
*     ...
       status = kill( pid, signum )
       if ( status .ne. 0 ) stop 'getcwd: error'
       write(*,*) 'シグナル ', signum,'をプロセス ', pid, ' に送付し
ました'
       end
```

関数は、*signum* という整数型の番号で表わされるシグナルを *pid* というプロセスに送ります。有効なシグナル番号は、/usr/include/sys/signal.h という C 言語のインクルードファイル中にリストされています。

参照 : kill(2)、signal(3)、signal(3F)、fork(3F)、および perror(3F)

link、symlnk：既存ファイルへのリンクの作成

link は既存ファイルへのリンクを作成します。symlnk は既存ファイルへのシンボリックリンクを作成します。

関数は、次のように呼び出します。

status = link(*name1*, *name2*)			
INTEGER*4 symlnk *status* = symlnk(*name1*, *name2*)			
name1	character*n	入力	既存ファイルのパス名
name2	character*n	入力	ファイル *name1* にリンクさせるパス名 ファイル *name2* は、既存ファイルであってはならない
戻り値	INTEGER*4	出力	*status*=0: 正常 *status*>0: システムエラーコード

link：既存ファイルへのリンクの作成

例 1 : link ー ファイル tlink.db.data.1 に対して、data1 という名前のリンクを作成します。

```
demo% cat tlink.f
        character*34 name1/'tlink.db.data.1'/, name2/'data1'/
        integer*4 link, status
        status = link( name1, name2 )
        if ( status .ne. 0 ) stop 'link: error'
        end
demo% f95 tlink.f
demo% ls -l data1
data1: ファイルもディレクトリもありません
demo% a.out
demo% ls -l data1
-rw-rw-r-- 2 generic 2 8月 11日 08:50 data1
demo%
```

symlnk：既存ファイルへのシンボリックリンクの作成

例 2 : symlnk ー ファイル tlink.db.data.1 に対して、data1 という名前のシンボリックリンクを作成します。

```
demo% cat tsymlnk.f
       character*34 name1/'tlink.db.data.1'/, name2/'data1'/
       INTEGER*4 status, symlnk
       status = symlnk( name1, name2 )
       if ( status .ne. 0 ) stop 'symlnk: error'
       end
demo% f95 tsymlnk.f
demo% ls -l data1
data1: ファイルもディレクトリもありません
demo% a.out
demo% ls -l data1
lrwxrwxrwx 1 generic 15 8月 11日 11:09 data1 -> tlink.db.data.1
demo%
```

参照 : link(2)、symlink(2)、perror(3F)、および unlink(3F)

注意: パス名を <sys/param.h> で定義されている MAXPATHLEN より長くすることはできません。

loc : オブジェクトのアドレスを戻す

この組み込み関数は、次のように呼び出します。

k = loc(*arg*)			
arg	任意の型	入力	任意の変数、配列、または構造体の名前
戻り値	INTEGER*4 または INTEGER*8	出力	*arg* のアドレス
	-xarch=v9 を使って、64 ビット環境で動作するようにコンパイルした場合は、INTEGER*8 ポインタが戻る。以下の注を参照。		

例: loc:

```
      INTEGER*4 k, loc
      real arg / 9.0 /
      k = loc( arg )
      write(*,*) k
      end
```

注 – 64 ビット Solaris 7 で動作するようコンパイルされたプログラムは、loc( ) 関数から出力を返す変数 INTEGER*8 を宣言しなければなりません。

long、short：整数オブジェクトの変換

long および short は INTEGER*4 と INTEGER*2 間で整数オブジェクトの変換を行います。この変換は、サブプログラム呼び出し一覧では特に有効です。

long：短整数 (INTEGER*2) から長整数 (INTEGER*4) への変換

関数は、次のように呼び出します。

call 長整数をとるサブルーチン (long(*int2*))		
int2	INTEGER*2	入力
戻り値	INTEGER*4	出力

short：長整数から短整数への変換

関数は、次のように呼び出します。

INTEGER*2 short call 短整数をとるサブルーチン (short(*int4*))		
int4	INTEGER*4	入力
戻り値	INTEGER*2	出力

例 (該当部分のみ): `long()` と `short()`

```
      integer*4 int4/8/, long
      integer*2 int2/8/, short
      call ExpecLong( long(int2) )
      call ExpecShort( short(int4) )
      ...
      end
```

ExpecLong はユーザープログラムによって呼び出されるサブルーチンで、長整数 (`INTEGER*4`) の引数をとります。*ExpecShort* は短整数 (`INTEGER*2`) の引数をとります。

`long` はライブラリルーチンの呼び出しに定数が使用され、`-i2` オプションを指定してコードをコンパイルする場合に役立ちます。

`short` は、長い型のオブジェクトを短い型の整数として渡す必要がある場合に役立ちます。短い型に渡す整数が大きすぎた場合、エラーは発生しませんが、プログラムが予期しない動きをします。

longjmp、isetjmp：isetjmp で設定した位置に戻す

`isetjmp` は `longjmp` の位置を設定します。`longjmp` は `isetjump` で設定した位置に戻ります。

isetjmp：longjmp の設定

この組み込み関数は、次のように呼び出します。

ival = isetjmp(*env*)			
env	INTEGER*4	出力	*env* は 12 要素の整数配列。64 ビット環境では、INTEGER*8 で宣言する必要がある。
戻り値	INTEGER*4	出力	*ival* = 0、isetjmp が明示的に呼び出された場合 *ival*≠ 0、isetjmp が longjmp から呼び出された場合

longjmp：isetjmp で設定した位置に戻す

サブルーチンは、次のように呼び出されます。

call longjmp(*env*, *ival*)			
env	INTEGER*4	入力	*env* は isetjmp で初期化された 12 語の整数配列。64 ビット環境では、INTEGER*8 で宣言する必要がある。
ival	INTEGER*4	出力	*ival* = 0、isetjmp が明示的に呼び出された場合 *ival*≠ 0、isetjmp が longjmp から呼び出された場合

説明

isetjmp とlongjmp ルーチンは、プログラムの低レベルルーチンで遭遇するエラーや障害を処置するために使用します。この 2 つは、f95の組み込み関数です。

これらのルーチンは、最後の手段としてのみ使用してください。これらの取り扱いには、十分注意してください。また、移植性はありません。バグやその他の詳細については、setjmp(3V) のマニュアルページを参照してください。

isetjmp は*env* にスタック環境を保存します。またレジスタ環境も保存します。

longjmp は、最後に isetjmp を呼び出して保存した環境を復元し、あたかも isetjmp の呼び出しが値 *ival* を返したかのように戻り、実行を継続します。

isetjmp から返された整数式 *ival* は、longjmp が呼び出されなければゼロです。longjmp が呼び出されれば、ゼロ以外になります。

例: isetjmp と longjmp を使用したコード部分

```
      INTEGER*4  env(12)
      common /jmpblk/ env
      j = isetjmp( env )
      if ( j .eq. 0 ) then   call  sbrtnA
      else
         call error_processor
      end if
      end
      subroutine sbrtnA
      INTEGER*4  env(12)
      common /jmpblk/ env
      call longjmp( env, ival )
      return
      end
```

制限

- longjmp() を呼び出す前に isetjmp を起動しなければなりません。
- isetjmp と longjmp で使用される整数型の配列引数 *env* は、最低 12 個の要素からなる配列でなければなりません。
- isetjmp を呼び出すルーチンから longjmp を呼び出すルーチンへ、共通ブロック経由であるいは引数として*env* 変数を渡さなければなりません。
- longjmp はスタックをクリーンアップしようとします。longjmp は isetjmp よりも低レベルのルーチンから呼び出さなければなりません。
- 手続き名の引数として isetjmp を渡しても作用しません。

参照: setjmp(3V)

malloc、malloc64、realloc、free：記憶領域の割り当て/再割り当て/割り当て解除

malloc()、malloc64()、および realloc() 関数は、記憶領域のブロックを割り当て、ブロックの開始アドレスを戻します。戻り値は INTEGER や *Cray-style* の POINTER 変数の設定に使用できます。realloc() は既存の記憶領域ブロックを新しいサイズで再割り当てします。free () は malloc()、malloc64()、または realloc()により割り当てられた記憶領域ブロックの割り当てを解除します。

注 – これらのルーチンは、f95 の組み込み関数として実装されていますが、f77 の外部関数でした。お手持ちのバージョンを使用しない場合は、Fortran 95 プログラムの型宣言や EXTERNAL 文で使用すべきではありません。realloc() ルーチンは f95 のみに実装されています。

Fortran 95 規格合致プログラムは、ALLOCATE および DEALLOCATE 文を割り当て可能な配列に使用して、動的メモリー管理を実行します。malloc/realloc/free への直接呼び出しは作成されません。

従来の Fortran 77 プログラムは malloc()/malloc64() を使用して、INTEGER 変数として同じデータ表現を持つ *Cray-style* の POINTER 変数に値を代入できました。*Cray-style* のPOINTER 変数は f95 に実装され、Fortran 77 からの可搬性をサポートします。

記憶領域の割り当てmalloc、malloc64

malloc() 関数は、次のように呼び出します。

k = malloc(*n*)			
n	INTEGER	入力	記憶領域のバイト数
戻り値	INTEGER(*Cray* POINTER)	出力	*k*>0: *k* は割り当てられた記憶領域の開始アドレス *k*=0: エラー
	-xarch=v9 を使って、64 ビット環境用にコンパイルした場合は、INTEGER*8 ポインタ値が戻る。以下の注を参照。		

注 – この関数は、Fortran 95では組み込み関数ですが、Fortran 77 では外部関数でした。64 ビット環境で動作するようにコンパイルされたプログラムでは、`malloc()` 関数とその出力を受け取る変数を `INTEGER*8` と宣言する必要があります。関数 `malloc64`(3F)は、プログラムを 32 ビット環境と 64 ビット環境間で可搬性を持たせるために提供された関数です。

k = malloc64(*n*)			
n	INTEGER*8	入力	記憶領域のバイト数
戻り値	INTEGER*8 (*Cray* POINTER)	出力	*k*>0: *k* は割り当てられた記憶領域の開始アドレス *k*=0: エラー

これらの関数は、記憶領域を割り当て、その領域の開始アドレスを返します。64 ビット環境では、この返されたバイトアドレスは、`INTEGER*4` の数値範囲外になる可能性があります。受け取り側の変数では `INTEGER*8` と宣言し、メモリーアドレスが切り捨てられないようにする必要があります。この記憶領域は、初期化できません。そのため記憶領域が、ある値、特にゼロに事前設定されていることを前提としないでください。

例: `malloc` を使用したコード部分

```
      parameter (NX=1000)
      integer ( p2X, X )
      real*4 X(1)
      ...
      p2X = malloc( NX*4 )
      if ( p2X .eq. 0 ) stop 'malloc: 割り当て不可'
      do 11 i=1,NX
11         X(i) = 0.
      ...
      end
```

上記の例では、`p2X` で指定される 4,000 バイトのメモリーを獲得し、この領域を 0 に初期化しています。

記憶領域の再割り当てrealloc

relloc() f95 組み込み関数は、次のように呼び出します。

k = realloc(*ptr*, *n*)			
ptr	INTEGER	入力	既存の記憶領域へのポインタ (前述の malloc() または realloc() 呼び出しからの戻り値)。
n	INTEGER	入力	必要とされるブロックの新しいサイズ。バイト単位。
戻り値	INTEGER (*Cray* POINTER)	出力	*k*>0: *k* は割り当てられた新しい記憶領域の開始アドレス *k*=0: エラー
	-xarch=v9 を使って、64 ビット環境用にコンパイルした場合は、INTEGER*8 ポインタ値が戻る。以下の注を参照。		

realloc() 関数は ptr によって指定される記憶領域のサイズを n バイトに変更し、ポインタを (移動された可能性のある) 新しいブロックに戻します。記憶領域の中身は新規および古いサイズの最小までには変更されません。

ptrが 0 の場合、realloc() は malloc() と同じ処理を行い、新しくn バイトの記憶領域サイズを割り当てます。

n が 0 で ptrが 0 でない場合、指定された記憶領域は将来の割り当てに対して有効にされ、アプリケーションが終了した場合のみ、システムに戻されます。

例: malloc()、realloc()、および *Cray-style* POINTER 変数を使用した例を示します。

```
      PARAMETER (nsize=100001)
      POINTER (p2space,space)
      REAL*4 space(1)

      p2space = malloc(4*nsize)
      if(p2space .eq. 0) STOP 'malloc: 割り当てできません'
      ...
      p2space = realloc(p2space, 9*4*nsize)
      if(p2space .eq. 0) STOP 'realloc: 再割り当てできません'
      ...
      CALL free(p2space)
      ...
```

realloc() は f95 のみに実装されていることに注意してください。

free：Malloc により割り当てられた記憶領域の割り当て解除

サブルーチンは、次のように呼び出されます。

call free (*ptr*)		
ptr	*Cray* POINTER	入力

free は malloc やrealloc() により割り当てられた記憶領域の割り当てを解除します。記憶領域はメモリーマネージャに戻されます。これでユーザーのプログラムでは使用できなくなります。

例: free():

```
      real x
      pointer ( ptr, x )
      ptr = malloc ( 10000 )
      call free ( ptr )
      end
```

mvbits：ビットフィールドの移動

サブルーチンは、次のように呼び出されます。

call mvbits(*src*, *ini1*, *nbits*, *des*, *ini2*)			
src	INTEGER*4	入力	移動元
ini1	INTEGER*4	入力	移動元でのビットの初期位置
nbits	INTEGER*4	入力	移動させるビット数
des	INTEGER*4	出力	移動先
ini2	INTEGER*4	入力	移動先でのビットの初期位置

例: mvbits:

```
demo% cat mvb1.f
* mvb1.f ― 移動元 src の初期ビット位置 0 から 3 ビットを des のビット 3 へ移動
*    src    des
* 543210 543210 ← ビット番号
* 000111 000001 ← 移動前の値
* 000111 111001 ← 移動後の値
       INTEGER*4 src, ini1, nbits, des, ini2
       data src, ini1, nbits, des, ini2
     1    / 7,    0,     3,   1,    3 /
       call mvbits ( src, ini1, nbits, des, ini2 )
       write (*,"(5o3)") src, ini1, nbits, des, ini2
       end
demo% f95 mvb1.f
demo% a.out
  7  0  3 71  3
demo%
```

以下の点に注意してください。

- 各ビットには、最下位ビットから最上位ビットまで、0 から 31 までの番号が付けられます。
- MVBITS は *des* のビット *ini2* か *ini2+nbits*-1までを変更し、*src* のビットは変更しません。
- 制限事項:
  - *ini1* + *nbits*≧32
  - *ini2* + *nbits*≦32

perror、gerror、ierrno：エラーメッセージの取得

これらのルーチンは、以下の関数を実行します。

perror	Fortran 論理装置 0 (stderr) へのメッセージの出力
gerror	システムエラーメッセージ (最後に検出されたシステムエラー) の読み取り
ierrno	最後に検出されたシステムエラーのエラー番号の読み取り

perror：論理装置 0 (stderr) へのメッセージ出力

サブルーチンは、次のように呼び出されます。

call perror(*string*)			
string	character*n	入力	メッセージ。標準エラーメッセージに先だって出力される。最後に検出されたシステムエラーに対するメッセージ

例 1 :

```
    call perror( "ファイルは書式付き入出力用" )
```

gerror：最後に検出されたエラーメッセージの取得

サブルーチンまたは関数は、次のように呼び出されます。

call gerror(*string*)			
string	character*n	出力	最後に検出されたシステムエラーのメッセージ

例 2 : gerror のサブルーチンとしての使用

```
        character string*30
        ...
        call gerror ( string )
        write(*,*) string
```

例 3 : gerror の関数としての使用 (この場合、*string* は使用しません)

```
        character gerror*30, z*30
        ...
        z = gerror( )
        write(*,*) z
```

ierrno：最後に検出されたエラー番号の取得

関数は、次のように呼び出します。

n = ierrno()			
戻り値	INTEGER*4	出力	最後に検出されたシステムエラーの番号

この番号はエラーが実際に起こった時にしか更新されません。このようなエラーを発生させるほとんどのルーチンと入出力文は、呼び出しの後でエラーコードを返します。その値はエラー条件を引き起こした原因を示す信頼度の高いデータです。

例 4 : ierrno():

```
        INTEGER*4 ierrno, n
        ...
        n = ierrno()
        write(*,*) n
```

参照: intro(2)、perror(3)

注意:

- perror を呼び出す際の *string* は、127 文字を超えてはいけません。
- gerror により返される文字列の長さは、それを呼び出すプログラムにより決められます。

- f95 の実行時入出力エラーについては、『Fortran ユーザーズガイド』に記載されています。

putc、fputc：論理装置への 1 文字出力

putc は論理装置 6 に出力します。通常は制御端末への出力になります。

fputc は任意の論理装置に出力します。

これらの関数は、通常の Fortran 入出力をバイパスして、Fortran 論理装置に関連付けられているファイルに 1 文字を出力します。

同じ装置上では、通常の Fortran 出力とこれらの関数の出力を混在させて使用しないでください。

\n などの特殊な \エスケープ文字を記述する場合は、-f77=backslash FORTRAN 77 互換オプションをつけてコンパイルする必要があります。

putc：論理装置 6 への出力

関数は、次のように呼び出します。

INTEGER*4 putc *status* = putc(*char*)			
char	character	入力	装置に出力する文字
戻り値	INTEGER*4	出力	*status*=0: 正常 *status*>0: システムエラーコード

例: putc():

```
demo% cat tputc.f
       character char, s*10 / 'OK by putc' /
       INTEGER*4 putc, status
       do i = 1, 10
        char = s(i:i)
        status = putc( char )
       end do
       status = putc( '\n' )
       end
demo% f95 -f77=backslash tputc.f
demo% a.out
OK by putc
demo%
```

fputc：指定した論理装置への出力

関数は、次のように呼び出します。

INTEGER*4 fputc *status* = fputc(*lunit*,*char*)			
lunit	INTEGER*4	入力	出力先装置
char	character	入力	装置に出力する文字
戻り値	INTEGER*4	出力	*status*=0: 正常 *status*>0: システムエラーコード

例: fputc():

```
demo% cat tfputc.f
       character char, s*11 / 'OK by fputc' /
       INTEGER*4 fputc, status
       open( 1, file='tfputc.data')
       do i = 1, 11
        char = s(i:i)
        status = fputc( 1, char )
       end do
       status = fputc( 1, '\n' )
       end
demo% f95 -f77=backslash tfputc.f
demo% a.out
demo% cat tfputc.data
OK by fputc
demo%
```

参照: putc(3S)、intro(2)、および perror(3F)

qsort、qsort64：1 次元配列の要素のソート

サブルーチンは、次のように呼び出されます。

call qsort(*array*, *len*, *isize*, *compar*) call qsort64(*array*, *len8*, *isize8*, *compar*)			
array	配列	入力	ソートする要素が入っている配列
len	INTEGER*4	入力	配列内の要素の個数
len8	INTEGER*8	入力	配列内の要素の個数

isize	INTEGER*4	入力	要素のサイズ: 4= 整数または実数 8= 倍精度または複素数 16= 倍精度複素数 文字配列の場合は文字オブジェクトの長さ
isize8	INTEGER*8	入力	要素のサイズ: 4_8= 整数または実数 8_8= 倍精度または複素数 16_8= 倍精度複素数 文字配列の場合は文字オブジェクトの長さ
compar	関数名	入力	ユーザーが提供する INTEGER*2 型の関数の名前。 *compar* (*arg1*, *arg2*) と指定して、ソート順を決定する

64 ビット環境においては、2 Gバイトを超える配列には qsort64 を使用します。この場合、INTEGER*8 データとして、配列の長さは *len8*、要素サイズは *isize8* に必ず指定してください。Fortran 95 型の定数を使用して INTEGER*8 定数を明示的に指定します。

compar の引数である *arg1* と *arg2* は、配列の要素であり、次の結果を返します。

負数	*arg1* は *arg2* の前に置かれると見なされる場合
ゼロ	*arg1* と *arg2* が等しい場合
正数	*arg1* は *arg2* の後に置かれると見なされる場合

次に例を示します。

```
demo% cat tqsort.f
       external compar
       integer*2 compar
       INTEGER*4 array(10)/5,1,9,0,8,7,3,4,6,2/,len/10/,
     1           isize/4/
       call qsort( array, len, isize, compar )
       write(*,'(10i3)') array
       end
       integer*2 function compar( a, b )
       INTEGER*4 a, b
       if ( a .lt. b ) compar = -1
       if ( a .eq. b ) compar = 0
       if ( a .gt. b ) compar = 1
       return
       end
demo% f95 tqsort.f
demo% a.out
  0 1 2 3 4 5 6 7 8 9
```

ran : 0 - 1 間の乱数の生成

反復して ranを呼び出すと、均一した分布で一連の乱数を生成します。lcrans(3m) を参照してください。

r = ran(i)			
i	INTEGER*4	入力	変数または配列要素
r	REAL	出力	変数または配列要素

例: ran:

```
demo% cat ran1.f
* ran1.f — 乱数を生成する。
       INTEGER*4 i, n
       real r(10)
       i = 760013
       do n = 1, 10
        r(n) = ran ( i )
       end do
       write ( *, "( 5 f11.6 )" ) r
       end
demo% f95 ran1.f
demo% a.out
   0.222058 0.299851 0.390777 0.607055 0.653188
   0.060174 0.149466 0.444353 0.002982 0.976519
demo%
```

以下の点に注意してください。

- 0.0 は範囲に含まれますが、1.0 は含まれません。
- 乗算合同型の一般乱数発生アルゴリズムを使用しています。
- 一般に、i の値は呼び出し元のプログラム実行中で一度だけ設定されます。
- i の初期値は大きい奇数の整数でなければなりません。
- 続けて ran を呼び出すと、次々に別の乱数が得られます。
- プログラムを実行するたびに異なる乱数の列を得るには、実行ごとに引数を異なる初期値に設定しなければなりません。
- ran は引数を下記のアルゴリズムに従って、次の乱数計算用に値を格納するために使用します。

```
    SEED = 6909 * SEED + 1 (MOD 2**32)
```

- SEED は 32 ビット数値を含み、上位 24 ビットは浮動小数点に変換され、その値が返されます。

rand、drand、irand：乱数を戻す

rand は 0.0 から 1.0 の範囲の実数値の乱数を返します。

drand は 0.0 から 1.0 の範囲の倍精度値の乱数を返します。

irand は 0 から 2147483647 の範囲の正の整数値の乱数を返します。

これらの関数は、random(3) を使用して乱数の列を発生させます。これらの 3 つの関数は、同じ 256 バイトの状態配列を共有します。3 つの関数の唯一の利点は、UNIX システムで幅広く利用できることです。乱数を生成するさらに優れた関数としては、lcrans、addrans、および shufrans があります。random(3) および『数値計算ガイド』も参照して比較してください。

i = irand(*k*) *r* = rand(*k*) *d* = drand(*k*)			
k	INTEGER*4	入力	*k*=0: 次の乱数を乱数列から取り出す *k*=1: 乱数列を再開し、最初の数を戻す *k*>0: 新しい乱列数の種として使用し、最初の数を戻す
rand	REAL*4	出力	
drand	REAL*8	出力	
irand	INTEGER*4	出力	

例: irand():

```
demo% cat trand.f
       integer*4 v(5), iflag/0/
       do i = 1, 5
        v(i) = irand( iflag )
       end do
       write(*,*) v
       end
demo% f95 trand.f
demo% a.out
   2078917053 143302914 1027100827 1953210302 755253631
demo%
```

rename：ファイルの名称変更

関数は、次のように呼び出します。

INTEGER*4 rename *status* = *rename*(*from, to*)			
from	character*n	入力	既存ファイルのパス名
to	character*n	入力	ファイルの新しいパス名
戻り値	INTEGER*4	出力	*status*=0: 正常 *status*>0: システムエラーコード

to で指定されたファイルがすでに存在する場合、*from* と *to* はどちらも同じタイプのファイルでなければならず、かつ同じファイルシステムに存在していなければなりません。*to* がすでに存在する場合、最初にそのファイルが削除されます。

例: rename() ― ファイル trename.old の名前を trename.new に変更します。

```
demo% cat trename.f
       INTEGER*4 rename, status
       character*18 from/'trename.old'/, to/'trename.new'/
       status = rename( from, to )
       if ( status .ne. 0 ) stop 'rename:error'
       end
demo% f95 trename.f
demo% ls trename*
trename.f trename.old
demo% a.out
demo% ls trename*
trename.f trename.new
demo%
```

参照: rename(2)、perror(3F)

注意: パス名を <sys/param.h> で定義されている MAXPATHLEN より長くすることはできません。

secnds：秒単位のシステム時間 (マイナス時間) を取得

t = secnds(*t0*)			
t0	REAL	入力	定数、変数、あるいは配列要素
戻り値	REAL	出力	午前 0 時からの秒数から *t0* を引いた値

例: secnds:

```
demo% cat sec1.f
       real elapsed, t0, t1, x, y
       t0 = 0.0
       t1 = secnds( t0 )
       y = 0.1
       do i = 1, 10000
        x = asin( y )
       end do
       elapsed = secnds( t1 )
       write ( *, 1 ) elapsed
  1    format ( ' 10000 arcsines: ', f12.6, ' sec' )
       end
demo% f95 sec1.f
demo% a.out
 1000 arcsines:6.699141 sec
demo%
```

以下の点に注意してください。

- secnds からの戻り値は、0.01 秒の桁まで正確です。
- 値は真夜中からのシステム秒数で、次の真夜中を越えても正確です。
- 日の終り近くのわずかな期間だけ精度がいくらか失われることがあります。

set_io_err_handler、get_io_err_handler : 入出力エラーハンドラの設定と取得

set_io_err_handler() は、ユーザー定義のルーチンを宣言し、特定の入力論理装置でエラーが検出されたときにそのルーチンを呼び出せるようにします。

get_io_err_handler() は、宣言されているエラー処理ルーチンのアドレスを戻します。

これらのルーチンはモジュール化されたサブルーチンで、呼び出し側のルーチンに USE SUN_IO_HANDLERS が示されている場合のみアクセスすることができます。

USE SUN_IO_HANDLERS call set_io_err_handler(*iu*, *subr_name*, *istat*)			
iu	INTEGER*8	入力	論理装置番号
subr_name	EXTERNAL	入力	ユーザーが提供するエラーハンドラサブルーチンの名前
istat	INTEGER*4	出力	状態を戻す

USE SUN_IO_HANDLERS call get_io_err_handler(*iu*, *subr_pointer*, *istat*)			
iu	INTEGER*8	入力	論理装置番号
subr_pointer	POINTER	出力	宣言されているハンドラルーチンのアドレス
istat	INTEGER*4	出力	状態を戻す

SET_IO_ERR_HANDLER は、論理装置 *iu* で入力エラーが発生したときに入出力エラーハンドラとして使用できるようにユーザー定義のサブルーチン *subr_name* を設定します。*iu* は、フォーマットされたファイル用に接続された Fortran 論理装置である必要があります。エラーの場合は istat に 0以外の値が設定され、エラー以外の場合は 0 が設定されます。

たとえば、論理装置 *iu* をオープンする前に SET_IO_ERR_HANDLER を呼び出すと、*istat* には *1001* ("不正な装置") が設定されます。*subr_name* に何も指定しない場合は、ユーザーのエラー処理がオフになり、プログラムは Fortran のデフォルトエラー処理に戻ります。

対象の論理装置のエラーハンドラとして使用されている関数のアドレスを取得するには、GET_IO_ERR_HANDLER を使用します。たとえば、他のハンドラルーチンに切り替える前に現在の入出力を保存するには、GET_IO_ERR_HANDLER を呼び出します。エラーハンドラは後で、保存されていた値まで戻ることができます。

subr_name は、特定の論理装置 *iu* で入出力エラーを処理するためのユーザー定義ルーチンの名前です。実行時入出力ライブラリは、関連するすべての情報を *subr_name* へ渡し、ルーチンで問題点を診断し、可能であれば問題を修正してから処理を続行します。

ユーザー定義のエラーハンドラルーチンに対するインタフェースは次のとおりです。

SUBROUTINE SUB_NAME(UNIT, SRC_FILE, SRC_LINE, DATA_FILE, FILE_POS, CURR_BUFF, CURR_ITEM, CORR_CHAR, CORR_ACTION) INTENT (IN) UNIT, SRC_FILE, SRC_LINE, DATA_FILE INTENT (IN) FILE_POS, CURR_BUFF, CURR_ITEM INTENT (OUT) CORR_CHAR, CORR_ACTION			
UNIT	INTEGER*8	入力	エラーが検出された入力ファイルの論理装置番号
SRC_FILE	CHARACTER*(*)	入力	入力操作を行なった、Fortran のソースファイル名
SRC_LINE	INTEGER*8	入力	SRC_FILE で入力操作のエラーが検出された行番号
DATA_FILE	CHARACTER*(*)	入力	読み取り中のデータファルの名前。このパラメータは、ファイルが、オープンされた外部ファイルの場合のみ使用できる。名前が無効 (論理装置 5 など) の場合は、DATA_FILE に長さゼロの文字データ項目が設定される。
FILE_POS	INTEGER*8	入力	入力ファイルの現在の位置 (バイト)。DATA_FILE の名前がわかっている場合のみ定義される。

CURR_BUFF	CHARACTER*(*)	入力	入力レコードの残りのデータが含まれている文字列。文字列の先頭が不正な文字となっている。
CURR_ITEM	INTEGER*8	入力	読み取られたレコードの入力項目数 (エラーが検出された場合の項目数も含む)。例: READ(12,10)L,(ARR(I),I=1,L) CURR_ITEM の値が 15 の場合は、ARR の 14 番目の要素の読み取り中にエラーが発生したことを表す。L は最初の項目、ARR(1) は 2 番目の項目、のように順次に表す。
CORR_CHAR	CHARACTER	出力	ハンドラが返すように指定された、ユーザー定義の正しい文字。この値は、CORR_ACTION がゼロ以外の場合のみ使用する。 CORR_CHAR が不正な文字の場合は、正しい文字が返されるまでハンドラが繰り返し呼び出される。これにより無限ループが発生する可能性があるが、無限ループにならないように注意する必要がある。
CORR_ACTION	INTEGER	出力	入出力ライブラリで行う修正措置を指定する。値がゼロの場合は特別な処理は不要で、ライブラリはデフォルトのエラー処理に戻る。値が 1 の場合は、入出力エラー処理ルーチンに CORR_CHAR を戻す。

制限事項

入出力ハンドラでできる処理は、1 文字を他の文字に置き換えることだけです。1 文字を 複数の文字に置き換えることはできません。

エラーリカバリのアルゴリズムでは、読み取り中の不正な文字を置き換えることのみ可能で、別のコンテキストですでに不正な文字として読み取られた文字を置き換えることはできません。たとえば、正しい内容が "1.2345 9.8765" でリストのとおりに読み取った入力が "1.234509.8765" の場合には、入出力ライブラリは (不正な数であるため) 2 つめのピリオドでエラーとなります。ただし、その時点までは、後ろに戻って'0' を空白に戻すことはできません。

現在は、このエラー処理の機能は、*namelist* のとおりに読み取った入力に対しては機能していません。*namelist* のとおりに読み取った入力を処理している場合は、エラーが発生しても、指定した入出力エラーハンドラは呼び出されません。

入出力エラーハンドラは、(内部ファイルではなく) 外部ファイルに対してのみ設定されます。これは、内部ファイルには、関連付けられる論理装置がないためです。

入出力エラーハンドラは、構文エラーに対してのみ呼び出されます。システムエラーやセマンティックエラー (入力値のオーバーフローなど) では呼び出されません。

ユーザー定義の入出力エラーハンドラが、入出力ライブラリに不正な文字を渡すようになっている場合は、ユーザー定義の入出力エラーハンドラを繰り返し呼び出すことになり、無限ループが発生することがあります。ファイルの同じ位置でエラーが発生する場合、エラーハンドラは終了するべきです。このようにするには、CORR_ACTION をゼロに設定する方法があります。このようにすると、入出力ライブラリは通常のエラー処理を続行します。

sh : sh コマンドの高速実行

関数は、次のように呼び出します。

INTEGER*4 sh *status* = sh(*string*)			
string	character*n	入力	実行するコマンドを含む文字列
戻り値	INTEGER*4	出力	実行されたシェルの終了状態。この値の説明については *wait(2)* を参照

例: sh():

```
character*18 string / 'ls > MyOwnFile.names' /
INTEGER*4 status, sh
status = sh( string )
if ( status .ne. 0 ) stop 'sh: error'
...
end
```

関数 sh は、あたかも文字列がコマンドとしてキー入力されたかのように、sh シェルに *string* を渡します。

現プロセスはコマンドが終了するまで待機します。

fork されたプロセスは、開いているファイルをすべてフラッシュします。

- 出力ファイルに関しては、バッファは実ファイルにフラッシュされます。
- 入力ファイルに関しては、ポインタの位置が予測不能になります。

sh() はマルチスレッド対応ではありません。マルチスレッドプログラム、または並列プログラムからは呼び出さないでください。

参照 : execve(2)、wait(2)、system(3)

注意: *string* の長さは 1,024 文字を超えることができません。

signal : シグナルに対する動作の変更

関数は、次のように呼び出します。

INTEGER*4 signal *or* INTEGER*8 signal *n* = signal(*signum*, *proc*, *flag*)			
signum	INTEGER*4	入力	シグナル番号。*signal(3)* を参照
proc	ルーチン名	入力	ユーザーが作成したシグナル処理ルーチンの名前 (EXTERNAL 文で指定する必要がある)

flag	INTEGER*4	入力	*flag* < 0: *proc* をシグナル処理として使用する。 *flag* ≥ 0: *proc* を無視する。*flag* を動作の定義として渡す *flag* = 0: デフォルトの動作を使用する。 *flag* = 1: このシグナルを無効にする。
戻り値	INTEGER*4	出力	*n*=-1: システムエラー *n*>0: 変更前の動作の定義 *n*>1: n は呼び出されたルーチンのアドレス *n*<-1: *signum* が正しいシグナル番号の場合: *n* は呼び出されたルーチンのアドレス *signum* が正しいシグナル番号でない場合: *n* はエラー番号
	INTEGER*8		64 ビット環境では、signal とその出力を受け取る変数は、INTEGER*8 と宣言する必要がある

proc が呼び出される場合、シグナル番号が整数の引数として渡されます。

プロセスがシグナルを受信した場合のデフォルトの動作は、通常はプロセスの終了処理と異常終了です。シグナル処理ルーチンは、特殊な処理を行うために特定の例外やインタラプトを得ることができます。

この関数の戻り値を後で signal を呼び出す時に使用して、処理の定義を変更前に戻すことができます。

エラーがなくても負数の戻り値が返されることがあるので注意してください。有効なシグナル番号を signal() に渡して、戻り値が -1 未満であった場合は問題ありません。

floatingpoint.h では、SIGFPE_DEFAULT、SIGFPE_IGNORE および SIGFPE_ABORTという *proc* 値を定義しています。50 ページの「floatingpoint.h: Fortran IEEE 定義」 を参照してください。

64 ビット環境では、signal は、その出力を受け取る変数とともに INTEGER*8 と宣言し、返される可能性のあるアドレスが切り捨てられるのを防ぐ必要があります。

参照 : kill(1)、signal(3)、kill(3F)、『数値計算ガイド』

sleep : 一定時間の実行中断

サブルーチンは、次のように呼び出されます。

call sleep(*itime*)			
itime	INTEGER*4	入力	スリープする秒数

システム時間記録が粗いので、実際の時間は *itime* よりも最大で 1 秒短くなることがあります。

例: sleep():

```
        INTEGER*4 time / 5 /
        write(*,*) '開始'
        call sleep( time )
        write(*,*) '終了'
        end
```

参照 : sleep(3)

stat、lstat、fstat：ファイルの状態の取得

これらの関数が戻す情報は次のとおりです。

- デバイス
- i ノード番号
- 保護
- ハードリンクの数
- ユーザー識別子
- グループ識別子
- デバイスタイプ
- サイズ
- アクセス時刻
- 更新時刻
- 状態変更時刻
- 最適ブロックサイズ
- 割り当てられているブロック

stat と lstat は、どちらもファイル名を用いて問い合わせをします。fstat は論理装置を用いて問い合わせをします。

stat：ファイル名によるファイルの状態の取得

関数は、次のように呼び出します。

INTEGER*4 stat *ierr* = stat (*name*, *statb*)			
name	character*n	入力	ファイルの名前
statb	INTEGER*4	出力	ファイル状態の情報が格納される 要素数 13 の配列
戻り値	INTEGER*4	出力	*ierr*=0: 正常 *ierr*>0: エラーコード

例 1 : stat():

```
      character name*18 /'MyFile'/
      INTEGER*4 ierr, stat, lunit/1/, statb(13)
      open( unit=lunit, file=name )
      ierr = stat ( name, statb )
      if ( ierr .ne. 0 ) stop 'stat: error'
      write(*,*)'所有者の UID = ',statb(5),',
     1   ブロック数 = ',statb(13)
      end
```

fstat：論理装置によるファイルの状態の取得

関数

INTEGER*4 fstat *ierr* = fstat (*lunit*, *statb*)			
lunit	INTEGER*4	入力	論理装置番号
statb	INTEGER*4	出力	ファイル状態の情報が格納される要素数 13 の配列
戻り値	INTEGER*4	出力	*ierr*=0: 正常 *ierr*>0: エラーコード

関数は、次のように呼び出します。

例 2 : fstat():

```
      character name*18 /'MyFile'/
      INTEGER*4 fstat, lunit/1/, statb(13)
      open( unit=lunit, file=name )
      ierr = fstat ( lunit, statb )
      if ( ierr .ne. 0 ) stop 'fstat: error'
      write(*,*)'所有者の UID = ',statb(5),',
     1      ブロック数 = ',statb(13)
      end
```

lstat：ファイル名によるファイルの状態の取得

関数は、次のように呼び出します。

ierr = lstat (*name*, *statb*)			
name	character*n	入力	ファイル名
statb	INTEGER*4	出力	ファイル状態の情報が格納される要素数 13 の配列
戻り値	INTEGER*4	出力	*ierr*=0: 正常 *ierr*>0: エラーコード

例 3 : lstat():

```
      character name*18 /'MyFile'/
      INTEGER*4 lstat, lunit/1/, statb(13)
      open( unit=lunit, file=name )
      ierr = lstat ( name, statb )
      if ( ierr .ne. 0 ) stop 'lstat: error'
      write(*,*)'所有者の UID = ',statb(5),',
     1   ブロック数 = ',statb(13)
      end
```

ファイル状態を格納する配列の詳細

INTEGE*4 型の配列 *statb* に返される情報の意味は、stat(2) での構造体 stat の説明と同じです。

予備の値は含まれません。順序は以下のとおりです。

statb(1)	i ノードが存在するデバイス
statb(2)	この i ノードの番号
statb(3)	保護
statb(4)	ファイルへのハードリンクの数
statb(5)	所有者のユーザー識別子
statb(6)	所有者のグループ識別子
statb(7)	i ノードに対応するデバイスのデバイスタイプ
statb(8)	ファイルの合計サイズ
statb(9)	ファイルの最終アクセス時刻
statb(10)	ファイルの最終更新時刻
statb(11)	ファイルの最終状態変更時刻
statb(12)	ファイルシステム入出力操作の最適ブロックサイズ
statb(13)	実際に割り当てられているブロックの数

参照: stat(2)、access(3F)、perror(3F)、および time(3F)

注意: パス名は、<sys/param.h> で定義されている MAXPATHLEN より長くすることはできません。

stat64、lstat64、fstat64：ファイルの状態の取得

stat、lstat、fstat の 64 ビット 「ロングファイル」(Solaris 2.6 と 7) バージョンです。これらのルーチンは、要素数 13 の配列 statb を INTEGER*8 で宣言しなければならない点を除いて、非 64 ビットルーチンと同じです。

system：システムコマンドの実行

関数は、次のように呼び出します。

INTEGER*4 system *status* = system(*string*)			
string	character*n	入力	実行するコマンドを含む文字列
戻り値	INTEGER*4	出力	実行されたシェルの終了状態。この値の説明については *wait(2)* を参照

例: system():

```
        character*8 string / 'ls s*' /
        INTEGER*4 status, system
        status = system( string )
        if ( status .ne. 0 ) stop 'system: error'
        end
```

関数 system は、あたかもコマンドとしてキー入力されたかのように、ユーザーのシェルに文字列 *string* を渡します。

注意: *string* の長さは 1024 文字を超えることができません。

system が環境変数 SHELL を見つけた場合、System はコマンドインタプリタ (シェル) として SHELL の値を使用し、それ以外は sh(l) を使用します。

現プロセスはコマンドが終了するまで待機します。

従来、cc はそれぞれ異なった前提で発展してきました。

- cc が system を呼び出す場合、シェルは通常、Bourne シェルです。

system 関数は開いているファイルをすべてフラッシュします。

- 出力ファイルに関しては、バッファは実ファイルにフラッシュされます。
- 入力ファイルに関しては、ポインタの位置が予測不能になります。

参照 : execve(2)、wait(2)、system(3)

sysytem() はマルチスレッド対応ではありません。マルチスレッドプログラム、または並列プログラムからは呼び出さないでください。

time、ctime、ltime、gmtime：システム時間の取得

これらのルーチンは以下の関数を実行します。

time	標準バージョン : システム時間を整数値 (0 GMT 1970/1/1 を起点とした秒数) として読み取る VMS バージョン : システム時間を文字 (*hh*: *mm*: *ss*) として読み取る
ctime	システム時間を ASCII 文字列に変換する
ltime	システム時間を現地時間の月、日などに分解する
gmtime	システム時間を GMT の月、日などに分解する

time：システム時間の取得

*time()*は、次のように呼び出します。

INTEGER*4 time *or* INTEGER*8 *n* = time()			標準バージョン
戻り値	INTEGER*4	出力	0: 0: 0 GMT 1970/1/1 を起点とした秒数
	INTEGER*8	出力	64 ビット環境では、time は INTEGER*8 の値を戻す

関数 time() は、グリニッジ時間で 1970 年 1 月 1 日 00 時 00 分 00 秒からの時間を秒数で示す整数を返します。これはオペレーティングシステム時計の値です。

例: time() ― オペレーティングシステムに付属する標準バージョン

```
demo% cat ttime.f
        INTEGER*4  n, time
        n = time()
        write(*,*)  '1970/1/1  0 時 GMT  からの秒数  =  ',  n
        end
demo% f95 ttime.f
demo% a.out
  1970/1/1  0 時 GMT  からの秒数  =  913240205
demo%
```

ctime：システム時間の文字への変換

関数 ctime は、システム時間 *stime* を変換して、24 文字の ASCII 文字列として返します。

関数は、次のように呼び出します。

CHARACTER ctime*24 *string* = ctime(*stime*)			
stime	INTEGER*4	入力	time() で読み取ったシステム時間 (標準バージョン)
戻り値	character*24	出力	文字列に変換されたシステム時間。 ctime と *string* は CHARACTER*24 として型宣言する

以下に ctime が戻す値の書式を示します。詳細については、ctime(3C) のマニュアルページを参照してください。

例: ctime():

```
demo% cat tctime.f
        character*24 ctime, string
        INTEGER*4  n, time
        n = time()
        string = ctime( n )
        write(*,*) 'ctime:', string
        end
demo% f95 tctime.f
demo% a.out
 ctime:Wed Dec  9 13:50:05 1998
demo%
```

ltime：システム時間の月、日など (現地時間) への分解

このルーチンはシステム時間を現地時間の月、日などに分解します。

サブルーチンは、次のように呼び出されます。

call ltime(*stime*, *tarray*)			
stime	INTEGER*4	入力	time() で読み取ったシステム時間 (標準バージョン)
tarray	INTEGER*4(9)	出力	現地時間の日、月、年、... に分解されたシステム時間

tarray の要素の意味については、次のセクションを参照してください。

例: ltime():

```
demo% cat tltime.f
        integer*4  stime, tarray(9), time
        stime = time()
        call ltime( stime, tarray )
        write(*,*) 'ltime:', tarray
        end
demo% f95 tltime.f
demo% a.out
 ltime: 25 49 10 12 7 91 1 223 1
demo%
```

gmtime：システム時間の月、日など (GMT) への分解

このルーチンはシステム時間を GMT の月、日などに分解します。

サブルーチン

call gmtime(*stime*, *tarray*)			
stime	INTEGER*4	入力	time() で読み取ったシステム時間 (標準バージョン)
tarray	INTEGER*4(9)	出力	GMT の日、月、年、... に分解されたシステム時間

例: gmtime:

```
demo% cat tgmtime.f
        integer*4  stime, tarray(9), time
        stime = time()
        call gmtime( stime, tarray )
        write(*,*) 'gmtime:', tarray
        end
demo% f95t tgmtime.f
demo% a.out
 gmtime:   12  44  19  18  5  94  6  168  0
demo%
```

以下に ltime と gmtime のtarray() の値 (インデックス、単位、範囲) を示します。

1	秒 (0 - 61)	6	年 - 1900
2	分 (0 - 59)	7	曜日 (日曜 = 0)
3	時間 (0 - 23)	8	日 (通年) (0 - 365)
4	日 (1 - 31)	9	夏時間: 夏時間が有効な場合、1
5	月 (0 - 11)		

これらの値は、C ライブラリルーチン ctime(3C) で定義されています。システムが 59 を超える秒を返す理由についてもここで説明されています。idate(3F) と fdate(3F) も参照してください。

ctime64、gmtime64、ltime64：64 ビット環境用のシステム時間ルーチン

これらは、ctime、gmtime、ltime の対応するバージョンで、64 ビット環境での移植性を実現します。変数 *stime* が INTEGER*8 であることを除けば、これらのルーチンは 32 ビット用と同じです。

32 ビット環境で INTEGER*8 の *stime* を指定して ctime64 を使用すると、*stime* の値が INTEGER*4 の範囲を超えている場合、すべてアスタリスク (*) が返されます。gmtime と ltime の場合、*tarray* 配列が -1 で埋められます。

ttynam、isatty：端末ポートの名前の読み取り

ttynam と isatty は、端末ポート名に関する処理を行います。

ttynam：端末ポートの名前の読み取り

関数 ttynam は論理装置 lunit に結合されている端末デバイスのパス名を空白で埋めて返します。

関数は、次のように呼び出します。

CHARACTER ttynam*24 *name* = ttynam(*lunit*)			
lunit	INTEGER*4	入力	論理装置
戻り値	character*n	出力	*name* が空白でない場合: *name* は *lunit* 上のデバイスのパス名。サイズ *n* は、もっとも長いパス名が入るのに十分な大きさにする *name* が空の文字列 (すべて空白): *lunit* はディレクトリ /dev の中の端末デバイスと結合されていない

isatty：装置が端末であるかどうかの確認

関数 isatty は、論理装置 *lunit* が端末装置かどうかによって、*true* または *false* を返します。

関数は、次のように呼び出します。

terminal = isatty(*lunit*)			
lunit	INTEGER*4	入力	論理装置
戻り値	LOGICAL	出力	*terminal* = 真: 端末デバイスである *terminal* = 偽: 端末デバイスではない

例: *lunit* が tty であるかどうかを確認します。

```
        character*12 name, ttynam
        INTEGER*4 lunit/5/
        logical*4 isatty, terminal
        terminal = isatty( lunit )
        name = ttynam( lunit )
        write(*,*) '端末 = ', terminal, ', 名前 = "', name, '"'
        end
```

出力は次のように表示されます。

```
 端末 = T, 名前 = "/dev/ttyp1  "
```

unlink：ファイルの削除

関数は、次のように呼び出します。

INTEGER*4 unlink *n* = unlink (*patnam*)			
patnam	character*n	入力	ファイル名
戻り値	INTEGER*4	出力	*n*=0: 正常 *n*>0: エラー

関数 unlink は、パス名 *patnam* で指定されたファイルを削除します。これがこのファイルに対する最後のリンクである場合、ファイルの内容はすべて失われます。

例: unlink()- tunlink.data ファイルを削除します。

```
demo% cat tunlink.f
       call unlink( 'tunlink.data' )
       end
demo% f95 tunlink.f
demo% ls tunl*
tunlink.f tunlink.data
demo% a.out
demo% ls tunl*
tunlink.f
```

参照 : unlink(2)、link(3F)、perror(3F)

注意: パス名を <sys/param.h> で定義されている MAXPATHLEN より長くすることはできません。

wait：プロセス終了の待機

関数は、次のように呼び出します。

INTEGER*4 wait *n* = wait(*status*)			
status	INTEGER*4	出力	子プロセスの終了状態
戻り値	INTEGER*4	出力	*n*>0: 子プロセスのプロセス識別子 *n*<0: n は (システムエラーコード)wait(2) を参照

wait は、シグナルを受信するか、またはその子プロセスの 1 つが終了するまで呼び出し元のプロセスを保留にします。最後の wait が呼び出された後で、子プロセスのどれかが終了した場合、wait はただちに戻ります。子プロセスがない場合、wait はエラーコードを伴ってただちに戻ります。

例: wait()() を使用したコード部分

```
      INTEGER*4 n, status, wait
      ...
      n = wait( status )
      if ( n .lt. 0 ) stop 'wait: error'
      ...
      end
```

参照 : wait(2)、signal(3F)、kill(3F)、perror(3F)

第2章

Fortran 95 組み込み関数

この章では、f95 コンパイラで認識される組み込み関数名を一覧表示します。

標準の Fortran 95 総称組み込み関数

この節では、Fortran 95 規格で使用される Fortran 95 総称組み込み関数を機能によってグループ分けしています。

ここで示す引数は、cmplx(Y=B, KIND=M, X=A) のようにキーワード形式で使用される場合、引数キーワードとして使用できる名前です。

これらの総称組み込み手続きの仕様についての詳細は、Fortran 95 規格を参照してください。

引数存在問合せ関数

総称組み込み名	内容の説明
PRESENCE	引数の存在

数値関数

総称組み込み名	内容の説明
ABS (A)	絶対値
AIMAG (Z)	複素数の虚部
AINT (A [, KIND])	整数へ切り捨て
ANINT (A [, KIND])	四捨五入
CEILING (A [, KIND])	数以上の最小整数
CMPLX (X [, Y, KIND])	複素数型への変換
CONJG (Z)	複素数の共役
DBLE (A)	倍精度実数型への変換
DIM (X, Y)	超過分
DPROD (X, Y)	倍精度の実数積
FLOOR (A [, KIND])	数以下の最大整数
INT (A [, KIND])	整数型への変換
MAX (A1, A2 [, A3,...])	最大値
MIN (A1, A2 [, A3,...])	最小値
MOD (A, P)	剰余関数
MODULO (A, P)	モジュロ関数
NINT (A [, KIND])	四捨五入の整数化
REAL (A [, KIND])	実数型への変換
SIGN (A, B)	符号の付け替え

数学関数

総称組み込み名	内容の説明
ACOS (X)	逆余弦
ASIN (X)	逆正弦
ATAN (X)	逆正接
ATAN2 (Y, X)	逆正接

総称組み込み名	内容の説明
COS (X)	余弦
COSH (X)	双曲線余弦
EXP (X)	指数関数
LOG (X)	自然対数
LOG10 (X)	共通対数 (底 10)
SIN (X)	正弦
SINH (X)	双曲線正弦
SQRT (X)	平方根
TAN (X)	正接
TANH (X)	双曲線正接

文字関数

総称組み込み名	内容の説明
ACHAR (I)	ASCII 照合手順での指定位置にある文字
ADJUSTL (STRING)	左整合
ADJUSTR (STRING)	右整合
CHAR (I [, KIND])	処理系照合手順での 指定位置にある文字
IACHAR (C)	ASCII 照合手順での文字の位置
ICHAR (C)	処理系照合手順での文字の位置
INDEX (STRING, SUBSTRING [, BACK])	部分文字列の開始位置
LEN_TRIM (STRING)	後続の空白文字なしの長さ
LGE (STRING_A, STRING_B)	字句的に等しいか大きい
LGT (STRING_A, STRING_B)	字句的に大きい
LLE (STRING_A, STRING_B)	字句的に等しいか小さい
LLT (STRING_A, STRING_B)	字句的に小さい
REPEAT (STRING, NCOPIES)	反復連結

総称組み込み名	内容の説明
SCAN (STRING, SET [, BACK])	集合内の文字に対し文字列を走査
TRIM (STRING)	後続の空白文字を削除
VERIFY (STRING, SET [, BACK])	1 つの文字列の文字集合を検証

文字問合せ関数

総称組み込み名	内容の説明
LEN (STRING)	文字要素の長さ

種別関数

総称組み込み名	内容の説明
KIND (X)	*kind* 型パラメータ値
SELECTED_INT_KIND (R)	指定した範囲の整数 *kind* 型パラメータ値
SELECTED_REAL_KIND ([P, R])	指定した精度と範囲の実数 *kind* 型パラメータ値

論理関数

総称組み込み名	内容の説明
LOGICAL (L [, KIND])	異なる *kind* 型パラメータで論理型オブジェクトを変換

数値問合せ関数

総称組み込み名	内容の説明
DIGITS (X)	数体系の有効数字の数
EPSILON (X)	1 に比較した場合おおよそ無視可能な数
HUGE (X)	数体系の最大数
MAXEXPONENT (X)	数体系の最大指数
MINEXPONENT (X)	数体系の最小指数
PRECISION (X)	10 進数精度
RADIX (X)	数体系の底
RANGE (X)	10 進数の指数範囲
TINY (X)	数体系の最小の正の数

ビット問合せ関数

総称組み込み名	内容の説明
BIT_SIZE (I)	数体系のビット数

ビット操作関数

総称組み込み名	内容の説明
BTEST (I, POS)	ビットの検査
IAND (I, J)	論理 AND
IBCLR (I, POS)	ビットを消去
IBITS (I, POS, LEN)	ビット抽出
IBSET (I, POS)	ビットを設定
IEOR (I, J)	排他的論理和
IOR (I, J)	包括的論理和

総称組み込み名	内容の説明
ISHFT (I, SHIFT)	論理シフト
ISHFTC (I, SHIFT [, SIZE])	循環シフト
NOT (I)	論理補数

転換関数

総称組み込み名	内容の説明
TRANSFER (SOURCE, MOLD [, SIZE])	第 1 引数を第 2 引数の型として処理

浮動小数点操作関数

総称組み込み名	内容の説明
EXPONENT (X)	数体系数の指数部
FRACTION (X)	1 つの数の小数部
NEAREST (X, S)	指定された方向にあるもっとも近い異なる処理系の数
RRSPACING (X)	指定された数に近い数体系数の相対空間の逆
SCALE (X, I)	実数に底を乗算して整数累乗へ
SET_EXPONENT (X, I)	1 つの数の指数部を設定
SPACING (X)	指定された数に近い数体系数の絶対空間

ベクトルおよび行列の乗算関数

総称組み込み名	内容の説明
DOT_PRODUCT (VECTOR_A, VECTOR_B)	2 つの一次元配列の内積
MATMUL (MATRIX_A, MATRIX_B)	行列の乗算

配列集計関数

総称組み込み名	内容の説明
ALL (MASK [, DIM])	すべての値が真のとき、真
ANY (MASK [, DIM])	任意の値が真のとき、真
COUNT (MASK [, DIM])	配列の真の要素の数
MAXVAL (ARRAY, DIM [, MASK]) ÅyÇ‹Ç¾ÇÕ MAXVAL (ARRAY [, MASK])Åz	配列の最大値
MINVAL (ARRAY, DIM [, MASK]) ÅyÇ‹Ç¾ÇÕ MINVAL (ARRAY [, MASK])Åz	配列の最小値
PRODUCT (ARRAY, DIM [, MASK]) ÅyÇ‹Ç¾ÇÕ PRODUCT (ARRAY [, MASK])Åz	配列要素の積
SUM (ARRAY, DIM [, MASK]) ÅyÇ‹Ç¾ÇÕ SUM (ARRAY [, MASK])Åz	配列要素の合計

配列問合せ関数

総称組み込み名	内容の説明
ALLOCATED (ARRAY)	配列割り付け状態
LBOUND (ARRAY [, DIM])	配列の次元下限
SHAPE (SOURCE)	配列またはスカラーの形状
SIZE (ARRAY [, DIM])	配列の要素の合計数
UBOUND (ARRAY [, DIM])	配列の次元下限

配列構成関数

総称組み込み名	内容の説明
MERGE (TSOURCE, FSOURCE, MASK)	選別によるマージ
PACK (ARRAY, MASK [, VECTOR])	配列を選別下の一次元配列へパック
SPREAD (SOURCE, DIM, NCOPIES)	次元を追加することにより、配列を複製
UNPACK (VECTOR, MASK, FIELD)	一次元配列を選別下の配列へアンパック

配列再形成関数

総称組み込み名	内容の説明
RESHAPE (SOURCE, SHAPE[, PAD, ORDER])	配列を再形成

配列操作関数

総称組み込み名	内容の説明
CSHIFT (ARRAY, SHIFT [, DIM])	循環シフト
EOSHIFT (ARRAY, SHIFT [, BOUNDARY, DIM]	切り捨て桁送り
TRANSPOSE (MATRIX)	二次元配列の入れ替え

配列内位置関数

総称組み込み名	内容の説明
MAXLOC (ARRAY, DIM [, MASK]) ÅyÇ‹Ç¾ÇÕ MAXLOC (ARRAY [, MASK])Åz	配列の最大値の位置
MINLOC (ARRAY, DIM [, MASK]) ÅyÇ‹Ç¾ÇÕ MINLOC (ARRAY [, MASK])Åz	配列の最小値の位置

ポインタ結合状態問合せ関数

総称組み込み名	内容の説明
ASSOCIATED (POINTER [, TARGET])	関連付け状態問い合わせまたは比較
NULL ([MOLD])	分離したポインタを戻す

組み込みサブルーチン

総称組み込み名	内容の説明
CPU_TIME (TIME)	処理系の時間を取得
DATE_AND_TIME ([DATE, TIME, ZONE, VALUES])	日付と時間を取得
MVBITS (FROM, FROMPOS, LEN, TO, TOPOS)	1 つの整数から別の整数へとビットをコピー
RANDOM_NUMBER (HARVEST)	擬似乱数を戻す
RANDOM_SEED ([SIZE, PUT, GET])	擬似乱数ジェネレータを初期化または再開
SYSTEM_CLOCK ([COUNT, COUNT_RATE, COUNT_MAX])	システム時間からデータを取得

組み込み関数の個別名

表 2-1 Fortran 95 組み込み関数の個別名および総称名

	個別名	総称	引数の型
	ABS (A)	ABS (A)	基本実数
	ACOS (X)	ACOS (X)	基本実数
	AIMAG (Z)	AIMAG (Z)	基本複素数
	AINT (A)	AINT (A)	基本実数
	ALOG (X)	LOG (X)	基本実数
	ALOG10 (X)	LOG10 (X)	基本実数
#	AMAX0 (A1, A2 [, A3,...])	REAL (MAX (A1, A2 [, A3,...]))	基本整数

表 2-1　Fortran 95 組み込み関数の個別名および総称名 (続き)

	個別名	総称	引数の型
#	AMAX1 (A1, A2 [, A3,...])	MAX (A1, A2 [, A3,...])	基本実数
#	AMIN0 (A1, A2 [, A3,...])	REAL (MIN (A1, A2 [, A3,...]))	基本整数
#	AMIN1 (A1, A2 [, A3,...])	MIN (A1, A2 [, A3,...])	基本実数
	AMOD (A, P)	MOD (A, P)	基本実数
	ANINT (A)	ANINT (A)	基本実数
	ASIN (X)	ASIN (X)	基本実数
	ATAN (X)	ATAN (X)	基本実数
	ATAN2 (Y, X)	ATAN2 (Y, X)	基本実数
	CABS (A)	ABS (A)	基本複素数
	CCOS (X)	COS (X)	基本複素数
	CEXP (X)	EXP (X)	基本複素数
#	CHAR (I)	CHAR (I)	基本整数
	CLOG (X)	LOG (X)	基本複素数
	CONJG (Z)	CONJG (Z)	基本複素数
	COS (X)	COS (X)	基本実数
	COSH (X)	COSH (X)	基本実数
	CSIN (X)	SIN (X)	基本複素数
	CSQRT (X)	SQRT (X)	基本複素数
	DABS (A)	ABS (A)	倍精度
	DACOS (X)	ACOS (X)	倍精度
	DASIN (X)	ASIN (X)	倍精度
	DATAN (X)	ATAN (X)	倍精度
	DATAN2 (Y, X)	ATAN2 (Y, X)	倍精度
	DCOS (X)	COS (X)	倍精度
	DCOSH (X)	COSH (X)	倍精度
	DDIM (X, Y)	DIM (X, Y)	倍精度
	DEXP (X)	EXP (X)	倍精度
	DIM (X, Y)	DIM (X, Y)	基本実数
	DINT (A)	AINT (A)	倍精度

表 2-1　Fortran 95 組み込み関数の個別名および総称名 (続き)

	個別名	総称	引数の型
	DLOG (X)	LOG (X)	倍精度
	DLOG10 (X)	LOG10 (X)	倍精度
#	DMAX1 (A1, A2 [, A3,...])	MAX (A1, A2 [, A3,...])	倍精度
#	DMIN1 (A1, A2 [, A3,...])	MIN (A1, A2 [, A3,...])	倍精度
	DMOD (A, P)	MOD (A, P)	倍精度
	DNINT (A)	ANINT (A)	倍精度
	DPROD (X, Y)	DPROD (X, Y)	基本実数
	DSIGN (A, B)	SIGN (A, B)	倍精度
	DSIN (X)	SIN (X)	倍精度
	DSINH (X)	SINH (X)	倍精度
	DSQRT (X)	SQRT (X)	倍精度
	DTAN (X)	TAN (X)	倍精度
	DTANH (X)	TANH (X)	倍精度
	EXP (X)	EXP (X)	基本実数
#	FLOAT (A)	REAL (A)	基本整数
	IABS (A)	ABS (A)	基本整数
#	ICHAR (C)	ICHAR (C)	基本文字
	IDIM (X, Y)	DIM (X, Y)	基本整数
#	IDINT (A)	INT (A)	倍精度
	IDNINT (A)	NINT (A)	倍精度
#	IFIX (A)	INT (A)	基本実数
	INDEX (STRING, SUBSTRING)	INDEX (STRING, SUBSTRING)	基本文字
#	INT (A)	INT (A)	基本実数
	ISIGN (A, B)	SIGN (A, B)	基本整数
	LEN (STRING)	LEN (STRING)	基本文字
#	LGE (STRING_A, STRING_B)	LGE (STRING_A, STRING_B)	基本文字
#	LGT (STRING_A, STRING_B)	LGT (STRING_A, STRING_B)	基本文字
#	LLE (STRING_A, STRING_B)	LLE (STRING_A, STRING_B)	基本文字
#	LLT (STRING_A, STRING_B)	LLT (STRING_A, STRING_B)	基本文字

表 2-1　Fortran 95 組み込み関数の個別名および総称名 (続き)

個別名	総称	引数の型
# MAX0 (A1, A2 [, A3,...])	MAX (A1, A2 [, A3,...])	基本整数
# MAX1 (A1, A2 [, A3,...])	INT (MAX (A1, A2 [, A3,...]))	基本実数
# MIN0 (A1, A2 [, A3,...])	MIN (A1, A2 [, A3,...])	基本整数
# MIN1 (A1, A2 [, A3,...])	INT (MIN (A1, A2 [, A3,...]))	基本実数
MOD (A, P)	MOD (A, P)	基本整数
NINT (A)	NINT (A)	基本実数
# REAL (A)	REAL (A)	基本整数
SIGN (A, B)	SIGN (A, B)	基本実数
SIN (X)	SIN (X)	基本実数
SINH (X)	SINH (X)	基本実数
# SNGL (A)	REAL (A)	倍精度
SQRT (X)	SQRT (X)	基本実数
TAN (X)	TAN (X)	基本実数
TANH (X)	TANH (X)	基本実数

の記号が付いた関数は、実引数として使用することができません。「倍精度」は、倍精度の実数を意味します。

Fortran 2000 モジュールルーチン

Fortran 2000 ドラフト規格には一連の組み込みモジュールが用意されています。このモジュールでは、IEEE 算術演算と C 言語との相互運用性をサポートする関数を定義します。これらのモジュールは新しい関数とサブルーチンを定義するもので、Sun Studio 8 Fortran 95 コンパイラ内で実装されます。

IEEE 算術演算と例外のモジュール

Fortran 2000 ドラフト規格は、対象言語で新しい関数をサポートして、IEEE 算術演算および IEEE 例外を処理できるようにするために、`IEEE_EXCEPTIONS`、`IEEE_ARITHMETIC`、および `IEEE_FEATURES`の 3 つのモジュールを備えています。

ドラフト規格は、組み込み関数、要素別処理関数、種別関数、要素別処理サブルーチン、および非要素別処理サブルーチンのセットを定義します。これらのセットについては、次の表に記載しています。

これらの関数およびサブルーチンにアクセスするには、呼び出し側のルーチンで次のモジュールを指定する必要があります。

```
USE、INTRINSIC :: IEEE_ARITHMETIC、IEEE_EXCEPTIONS
```

詳細は、ドラフト規格の第 14 章 (`http://www.j3-fortran.org`) を参照してください。

問合せ関数

モジュール `IEEE_EXCEPTIONS` には、次の問合せ関数が含まれています。

関数	内容の説明
`IEEE_SUPPORT_FLAG(FLAG[,X])`	プロセッサが例外をサポートしているかどうかの問い合わせ
`IEEE_SUPPORT_HALTING(FLAG)`	プロセッサが、例外停止後の制御をサポートしているかどうかの問い合わせ

モジュール IEEE_ARITHMETIC には、次の問合せ関数が含まれています。

関数	内容の説明
IEEE_SUPPORT_DATATYPE([X])	プロセッサが IEEE 算術演算をサポートしているかどうかの問い合わせ
IEEE_SUPPORT_DENORMAL([X])	プロセッサが、非正規化数をサポートしているかどうかの問い合わせ
IEEE_SUPPORT_DIVIDE([X])	プロセッサが、IEEE 規格で指定されている精度での除算をサポートしているかどうかの問い合わせ
IEEE_SUPPORT_INF([X])	プロセッサが IEEE 無限大をサポートしているかどうかの問い合わせ
IEEE_SUPPORT_IO([X])	プロセッサが、書式付き入出力において IEEE 基本変換の丸めをサポートしているかどうかの問い合わせ
IEEE_SUPPORT_NAN([X])	プロセッサが IEEE 非数をサポートしているかどうかの問い合わせ
IEEE_SUPPORT_ROUNDING(VAL[,X])	プロセッサが 特別な丸めモードをサポートしているかどうかの問い合わせ
IEEE_SUPPORT_SQRT([X])	プロセッサが IEEE 平方根をサポートしているかどうかの問い合わせ
IEEE_SUPPORT_STANDARD([X])	プロセッサが IEEE のすべての機能をサポートしているかどうかの問い合わせ

要素別処理関数

モジュール IEEE_ARITHMETIC には、IEEE_SUPPORT_DATATYPE(X) および IEEE_SUPPORT_DATATYPE(Y) が真であるような実数 X と Y について、次の要素別処理関数が含まれています。

関数	内容の説明
IEEE_CLASS(X)	IEEE クラス
IEEE_COPY_SIGN(X,Y)	IEEE copysign 関数
IEEE_IS_FINITE(X)	値が有限かどうかの判定

IEEE_IS_NAN(X)	値が IEEE 非数であるかどうかの判定
IEEE_IS_NORMAL(X)	値が正規であるかどうかの判定
IEEE_IS_NEGATIVE(X)	値が負かどうかの判定
IEEE_LOGB(X)	IEEE 浮動小数点書式の非バイアス指数
IEEE_NEXT_AFTER(X,Y)	Y 方向に向かって、X の次の表現可能文字を返す
IEEE_REM(X,Y)	IEEE REM 剰余関数 X - Y*N で、N は、実際の X/Y にもっとも近い整数を表す
IEEE_RINT(X)	現在の丸めモードに従って整数値を返す
IEEE_SCALB(X,I)	X*2**I を返す
IEEE_UNORDERED(X,Y)	IEEE unordered 関数 X または Y が NaN の場合は真、それ以外の場合は偽
IEEE_VALUE(X,CLASS)	IEEE 値の生成

種別関数

モジュール IEEE_ARITHMETIC には、次の変形関数が含まれています。

関数	内容の説明
IEEE_SELECTED_REAL_KIND([P,][R])	指定した精度と範囲の IEEE 実数 に対する *kind* 型パラメータ値

要素別処理サブルーチン

モジュール IEEE_EXCEPTIONS には、次の要素別処理サブルーチンが含まれています。

サブルーチン	内容の説明
IEEE_GET_FLAG(FLAG,FLAG_VALUE)	例外フラグの取得
IEEE_GET_HALTING_MODE(FLAG,HALTING)	例外に対する停止モードの取得

非要素別処理サブルーチン

モジュール IEEE_EXCEPTIONS には、次の非要素別処理サブルーチンが含まれています。

サブルーチン	内容の説明
IEEE_GET_STATUS(STATUS_VALUE)	浮動小数点環境の現在の状態の取得
IEEE_SET_FLAG(FLAG,FLAG_VALUE)	例外フラグの設定
IEEE_SET_HALTING_MODE(FLAG,HALTING)	例外時における続行または停止の制御
IEEE_SET_STATUS(STATUS_VALUE)	浮動小数点環境の状態の復元

モジュール IEEE_ARITHMETIC には、次の非要素別処理サブルーチンが含まれています。

サブルーチン	内容の説明
IEEE_GET_ROUNDING_MODE(ROUND_VAL)	現在の IEEE 丸めモードの取得
IEEE_SET_ROUNDING_MODE(ROUND_VAL)	現在の IEEE 丸めモードの設定

C 結合モジュール

Fortran 2000 のドラフト規格には、C 言語手続きを参照する方法が用意されています。ISO_C_BINDING モジュールは、組み込みモジュール関数として 3 つのサポートプロシージャを定義します。これらの関数にアクセスするには、呼び出し側のルーチンで以下の指定が必要です。

```
USE、INTRINSIC :: ISO_C_BINDING、ONLY: C_LOC、C_PTR, C_ASSOCIATED
```

モジュールで定義されるプロシージャは次のとおりです。

関数	内容の説明
C_LOC(X)	引数の C アドレスを返す
C_ASSOCIATED(C_PTR_1 [, C_PTR_2])	C_PTR_1 の結合状態を示すか、または C_PTR_1 と C_PTR_2 が同じエンティティに関連付けられているかどうかを示す
C_F_POINTER(CPTR, FPTR [, SHAPE])	C ポインタのターゲットとポインタを関連付け、その形状を指定する

ISO_C_BINDING 組み込みモジュールの詳細は、Fortran 2000 ドラフト規格の第15 章 (http://www.j3-fortran.org/) を参照してください。

非標準の Fortran 95 組み込み関数

次に挙げる関数は f95 コンパイラにより組み込み関数と見なされますが、Fortran 95 規格の一部ではありません。

基本線形代数関数 (BLAS)

-xknown_lib=blas でコンパイルすると、コンパイラは続くルーチンへの呼び出しを組み込みルーチンと認識し、最適化を行い、Sun Performance Library 実装へリンクします。コンパイラはこれらのルーチンのユーザー供給バージョンを無視します。

表 2-2　BLAS 組み込み関数

関数	内容の説明
CAXPY	スカラーの積およびベクトルプラスベクトル
DAXPY	
SAXPY	
ZAXPY	

表 2-2　BLAS 組み込み関数 (続き)

関数	内容の説明
CCOPY	ベクトルをコピー
DCOPY	
SCOPY	
ZCOPY	
CDOTC	ドット積 (内部積)
CDOTU	
DDOT	
SDOT	
ZDOTC	
ZDOTU	
CSCAL	ベクトルを伸縮
DSCAL	
SSCAL	
ZSCAL	

これらのルーチンの詳細については、『Sun Performance Library User's Guide』を参照してください。

区間演算組み込み関数

次の表は、区間演算 (-xia) のコンパイル時に、コンパイラによって認識される組み込み関数を示します。詳細については、『Fortran 95 区間演算プログラミングリファレンス』を参照してください。

DINTERVAL	DIVIX	INF	INTERVAL
ISEMPTY	MAG	MID	MIG
NDIGITS	QINTERVAL	SINTERVAL	SUP
VDABS	VDACOS	VDASIN	VDATAN
VDATAN2	VDCEILING	VDCOS	VDCOSH
VDEXP	VDFLOOR	VDINF	VDINT
VDISEMPTY	VDLOG	VDLOG10	VDMAG
VDMID	VDMIG	VDMOD	VDNINT
VDSIGN	VDSIN	VDSINH	VDSQRT

VDSUP	VDTAN	VDTANH	VDWID
VQABS	VQCEILING	VQFLOOR	VQINF
VQINT	VQISEMPTY	VQMAG	VQMID
VQMIG	VQNINT	VQSUP	VQWID
VSABS	VSACOS	VSASIN	VSATAN
VSATAN2	VSCEILING	VSCOS	VSCOSH
VSEXP	VSFLOOR	VSINF	VSINT
VSISEMPTY	VSLOG	VSLOG10	VSMAG
VSMID	VSMIG	VSMOD	VSNINT
VSSIGN	VSSIN	VSSINH	VSSQRT
VSSUP	VSTAN	VSTANH	VSWID
WID			

その他のベンダーの組み込み関数

f95 コンパイラは、Cray Research, Inc. などの他のベンダーの Fortran コンパイラで定義された従来のさまざまな組み込み関数を認識します。これらは使用することはできません。

表 2-3 Cray CF90 および他のコンパイラの組み込み関数

関数	引数	内容の説明
CLOC	([C=]c)	文字オブジェクトのアドレスを取得
COMPL	([I=]i)	単語のビットごとの補数。代わりに NOT(i) を使用
COT	([X=]x)	総称余接 (DCOT、QCOT も同様)
CSMG	([I=]i,[J=]j,[K=]k)	条件付スカラーマージ
DSHIFTL	([I=]i,[J=]j,[K=]k)	i と j を k ビット、倍オブジェクト左シフト
DSHIFTR	([I=]i,[J=]j,[K=]k)	i と j を k ビット、倍オブジェクト右シフト
EQV	([I=]i,[J=]j)	論理等価。代わりに IOER(i,j) を使用
FCD	([I=]i,[J=]j)	文字ポインタを構築
GETPOS	([I=]i)	ファイル位置を取得
IBCHNG	([I=]i, [POS=]j)	単語の中で指定のビットへ変更する総称関数

表 2-3　Cray CF90 および他のコンパイラの組み込み関数 (続き)

関数	引数	内容の説明
ISHA	([I=]i, [SHIFT=]j)	総称演算シフト
ISHC	([I=]i, [SHIFT=]j)	総称循環シフト
ISHL	([I=]i, [SHIFT=]j)	総称左シフト
LEADZ	([I=]i)	先頭の 0 ビットの数をカウント
LENGTH	([I=]i)	転送された Cray 単語の数を戻す
LOC	([I=]i)	変数のアドレスを戻す (57 ページの「loc : オブジェクトのアドレスを戻す」 を参照)
NEQV	([I=]i,[J=]j)	論理非等価。代わりに IOER(i,j) を使用
POPCNT	([I=]i)	1 に設定されたビットの数をカウント
POPPAR	([I=]i)	ビット生成パリティを演算
SHIFT	([I=]i,[J=]j)	左循環をシフト。代わりに ISHFT(i,j) または ISHFTC(i,j,k) を使用
SHIFTA	([I=]i,[J=]j)	符号拡張子付きの演算シフト
SHIFTL	([I=]i,[J=]j)	ゼロ充填で左シフト。代わりに ISHFT(i,j) または ISHFTC(i,j,k) を使用
SHIFTR	([I=]i,[J=]j)	ゼロ充填で右シフト。代わりに ISHFT(i,j) または ISHFTC(i,j,k) を使用
TIMEF	()	最初の呼び出し以降の経過時間を戻す
UNIT	([I=]i)	BUFFERIN または BUFFEROUT の状態を戻す
XOR	([I=]i,[J=]j)	排他的論理和。代わりに IOER(i,j) を使用

VMS Fortran 77 組み込み関数のリストについては、第 3 章を参照してください。

その他の拡張子

Fortran 95 コンパイラは、次の組み込み関数を認識します。

MPI_SIZEOF

MPI_SIZEOF(*x*, *size*, *error*)

指定された変数、*x* のマシン表現のサイズをバイトで戻します。*x* が配列の場合、底要素のサイズを戻し、配列全体のサイズは戻しません。

x	入力、変数または任意の型の配列
size	出力、整数、*x* のバイト数でのサイズ
error	出力、整数、エラーが検出されるとエラーコードへ設定、検出されない場合はゼロ

メモリー関数

メモリー割り当て、再割り当て、割り当て解除関数 `malloc()`、`realloc()`、`free()` は、`f95` 組み込み関数として実装されます。詳細については、62 ページの「`malloc`、`malloc64`、`realloc`、`free` : 記憶領域の割り当て/再割り当て/割り当て解除」を参照してください。

第3章

FORTRAN 77 および VMS 組み込み関数

この章では、以前の FORTRAN 77 から Fortran 95 へのプログラムの移行をサポートするために `f95` に組み込まれ、提供される FORTRAN 77 組み込み関数のセットを一覧で示します。

`f95` は、この章で一覧表示している FORTRAN 77 および VMS 関数すべてを組み込み関数として認識します。前の章で一覧表示した Fortran 95 の関数すべても同様です。以前の FORTRAN 77 プログラムを`f95` に移行するサポートとして、`-f77=intrinsics` を指定してコンパイルすると、コンパイラは FORTRAN 77 および VMS 関数のみを組み込み関数として認識し、Fortran 95 組み込み関数は認識しません。

ANSI FORTRAN 77 規格以外の組み込み関数には、✠ 印を付けています。非標準組み込み関数およびライブラリ関数を使用するプログラムは、他のプラットフォームへの移植性がない可能性があります。

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 3-2) や照会関数 (表 3-7) 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 – `-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。135 ページの「注意」、および『Fortran ユーザーズガイド』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。

算術関数

表 3-1　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	INTEGER	INTEGER
				ABS	REAL	REAL
				DABS	DOUBLE	DOUBLE
				CABS	COMPLEX	REAL
				QABS ✠	REAL*16	REAL*16
				ZABS ✠	DOUBLE COMPLEX	DOUBLE
				CDABS ✠	DOUBLE COMPLEX	DOUBLE
				CQABS ✠	COMPLEX*32	REAL*16
切り捨て 注 (1) 参照	int(a)	1	AINT	AINT	REAL	REAL
				DINT	DOUBLE	DOUBLE
				QINT ✠	REAL*16	REAL*16
四捨五入	a ≥ 0 の場合 int(a+.5) a < 0 の場合 int(a-.5)	1	ANINT	ANINT	REAL	REAL
				DNINT	DOUBLE	DOUBLE
				QNINT ✠	REAL*16	REAL*16
四捨五入の整数化	a ≥ 0 の場合 int(a+.5) a < 0 の場合 int(a-.5)	1	NINT	NINT	REAL	INTEGER
				IDNINT	DOUBLE	INTEGER
				IQNINT ✠	REAL*16	INTEGER
剰余 注 (1) 参照	a1-int(a1/a2)*a2	2	MOD	MOD	INTEGER	INTEGER
				AMOD	REAL	REAL
				DMOD	DOUBLE	DOUBLE
				QMOD ✠	REAL*16	REAL*16
符号の付け替え	a2 ≥ 0 の場合 |a1| a2 < 0 の場合 -|a1|	2	SIGN	ISIGN	INTEGER	INTEGER
				SIGN	REAL	REAL
				DSIGN	DOUBLE	DOUBLE
				QSIGN ✠	REAL*16	REAL*16

表 3-1　算術関数 (続き)

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
超過分	a1 > a2 の場合 a1-a2 a1 ≤ a2 の場合 0	2	DIM	IDIM	INTEGER	INTEGER
				DIM	REAL	REAL
				DDIM	DOUBLE	DOUBLE
				QDIM ✠	REAL*16	REAL*16
倍精度化または 4 倍精度化乗算	a1 * a2	2	-	DPROD	REAL	DOUBLE
				QPROD ✠	DOUBLE	REAL*16
最大値の選択	max(a1, a2, …))	&138;2	MAX	MAX0	INTEGER	INTEGER
				AMAX1	REAL	REAL
				DMAX1	DOUBLE	DOUBLE
				QMAX1 ✠	REAL*16	REAL*16
			AMAX0	AMAX0	INTEGER	REAL
			MAX1	MAX1	REAL	INTEGER
最小値の選択	min(a1, a2, …))	&138;2	MIN	MIN0	INTEGER	INTEGER
				AMIN1	REAL	REAL
				DMIN1	DOUBLE	DOUBLE
				QMIN1 ✠	REAL*16	REAL*16
			AMIN0	AMIN0	INTEGER	REAL
			MIN1	MIN1	REAL	INTEGER

型変換関数

表 3-2　Fortran 77 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	引数の型	関数の型
INTEGER 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
REAL 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
			FLOATK	INTEGER*8	REAL*4
DOUBLE 注 (3) 参照	1	DBLE	DBLE	INTEGER	DOUBLE PRECISION
			DFLOAT	INTEGER	DOUBLE PRECISION
			DFLOATK	INTEGER*8	DOUBLE PRECISION
			DREAL ✠	REAL	DOUBLE PRECISION
			-	DOUBLE	DOUBLE PRECISION
			-	COMPLEX	DOUBLE PRECISION
			-	COMPLEX*16	DOUBLE PRECISION
			-	REAL*16	DOUBLE PRECISION
			-	COMPLEX*32	DOUBLE PRECISIONDOUBLE PRECISION
			DBLEQ ✠-	REAL*16COMPLEX*32	DOUBLE PRECISION

表 3-2　Fortran 77 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	引数の型	関数の型
REAL*16 注 (3) 参照	1	QREAL✠	QREAL ✠	INTEGER	REAL*16
		QEXT ✠	QFLOAT ✠	INTEGER	REAL*16
				REAL	REAL*16
			-	INTEGER	REAL*16
			QEXT ✠	DOUBLE	REAL*16
			QEXTD ✠	REAL*16	REAL*16
			-	COMPLEX	REAL*16
			-	COMPLEX*16	REAL*16
			-	COMPLEX*32	REAL*16
			-		
COMPLEX 注 (4) と (8) 参照	1 または 2	CMPLX	-	INTEGER	COMPLEX
			-	REAL	COMPLEX
			-	DOUBLE	COMPLEX
			-	REAL*16	COMPLEX
			-	COMPLEX	COMPLEX
			-	COMPLEX*16	COMPLEX
			-	COMPLEX*32	COMPLEX
DOUBLE COMPLEX 注 (8) 参照	1 または 2	DCMPLX✠	-	INTEGER	DOUBLE COMPLEX
			-	REAL	DOUBLE COMPLEX
			-	DOUBLE	DOUBLE COMPLEX
			-	REAL*16	DOUBLE COMPLEX
			-	COMPLEX	DOUBLE COMPLEX
			-	COMPLEX*16	DOUBLE COMPLEX
			-	COMPLEX*32	DOUBLE COMPLEX

表 3-2　Fortran 77 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	引数の型	関数の型
COMPLEX*32 注 (8) 参照	1 または 2	QCMPLX✠	-	INTEGER	COMPLEX*32
			-	REAL	COMPLEX*32
			-	DOUBLE	COMPLEX*32
			-	REAL*16	COMPLEX*32
			-	COMPLEX	COMPLEX*32
			-	COMPLEX*16	COMPLEX*32
			-	COMPLEX*32	COMPLEX*32
INTEGER 注 (5) 参照	1	-	ICHAR	CHARACTER	INTEGER
		-	IACHAR ✠		
CHARACTER 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 3-3　Fortran 77 三角関数

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
正弦 注 (7) 参照	sin(a)	1	SIN	SIN	REAL	REAL
				DSIN	DOUBLE	DOUBLE
				QSIN ✠	REAL*16	REAL*16
				CSIN	COMPLEX	COMPLEX
				ZSIN ✠	DOUBLE COMPLEX	DOUBLE COMPLEX
				CDSIN ✠	DOUBLE COMPLEX	DOUBLE COMPLEX
				CQSIN ✠	COMPLEX*32	COMPLEX*32
正弦 (度) 注 (7) 参照	sin(a)	1	SIND ✠	SIND ✠	REAL	REAL
				DSIND ✠	DOUBLE	DOUBLE
				QSIND ✠	REAL*16	REAL*16
余弦 注 (7) 参照	cos(a)	1	COS	COS	REAL	REAL
				DCOS	DOUBLE	DOUBLE
				QCOS ✠	REAL*16	REAL*16
				CCOS	COMPLEX	COMPLEX
				ZCOS ✠	DOUBLE COMPLEX	DOUBLE COMPLEX
				CDCOS ✠	DOUBLE COMPLEX	DOUBLE COMPLEX
				CQCOS ✠	COMPLEX*32	COMPLEX*32
余弦 (度) 注 (7) 参照	cos(a)	1	COSD ✠	COSD ✠	REAL	REAL
				DCOSD ✠	DOUBLE	DOUBLE
				QCOSD ✠	REAL*16	REAL*16
正接 注 (7) 参照	tan(a)	1	TAN	TAN	REAL	REAL
				DTAN	DOUBLE	DOUBLE
				QTAN ✠	REAL*16	REAL*16
正接 (度) 注 (7) 参照	tan(a)	1	TAND ✠	TAND ✠	REAL	REAL
				DTAND ✠	DOUBLE	DOUBLE
				QTAND ✠	REAL*16	REAL*16
逆正弦 注 (7) 参照	arcsin(a)	1	ASIN	ASIN	REAL	REAL
				DASIN	DOUBLE	DOUBLE
				QASIN ✠	REAL*16	REAL*16

表 3-3　Fortran 77 三角関数 (続き)

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
逆正弦 (度) 注 (7) 参照	arcsin(a)	1	ASIND ✠	ASIND ✠	REAL	REAL
				DASIND ✠	DOUBLE	DOUBLE
				QASIND ✠	REAL*16	REAL*16
逆余弦 注 (7) 参照	arccos(a)	1	ACOS	ACOS	REAL	REAL
				DACOS	DOUBLE	DOUBLE
				QACOS ✠	REAL*16	REAL*16
逆余弦 (度) 注 (7) 参照	arccos(a)	1	ACOSD ✠	ACOSD ✠	REAL	REAL
				DACOSD ✠	DOUBLE	DOUBLE
				QACOSD ✠	REAL*16	REAL*16
逆正接 注 (7) 参照	arctan(a)	1	ATAN	ATAN	REAL	REAL
				DATAN	DOUBLE	DOUBLE
				QATAN ✠	REAL*16	REAL*16
	arctan (a1/a2)	2	ATAN2	ATAN2	REAL	REAL
				DATAN2	DOUBLE	DOUBLE
				QATAN2 ✠	REAL*16	REAL*16
逆正接 (度) 注 (7) 参照	arctan(a)	1	ATAND ✠	ATAND ✠	REAL	REAL
				DATAND ✠	DOUBLE	DOUBLE
				QATAND ✠	REAL*16	REAL*16
	arctan (a1/a2)	2	ATAN2D ✠	ATAN2D ✠	REAL	REAL
				DATAN2D ✠	DOUBLE	DOUBLE
				QATAN2D ✠	REAL*16	REAL*16
双曲線正弦 注 (7) 参照	sinh(a)	1	SINH	SINH	REAL	REAL
				DSINH	DOUBLE	DOUBLE
				QSINH ✠	REAL*16	REAL*16
双曲線余弦 注 (7) 参照	cosh(a)	1	COSH	COSH	REAL	REAL
				DCOSH	DOUBLE	DOUBLE
				QCOSH ✠	REAL*16	REAL*16
双曲線正接 注 (7) 参照	tanh(a)	1	TANH	TANH	REAL	REAL
				DTANH	DOUBLE	DOUBLE
				QTANH ✠	REAL*16	REAL*16

その他の数学関数

表 3-4 その他の Fortran 77 数学関数

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
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

表 3-4　その他の Fortran 77 数学関数 (続き)

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
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
常用対数	log10(a)	1	LOG10	ALOG10	REAL	REAL
				DLOG10	DOUBLE	DOUBLE
				QLOG10 ✠	REAL*16	REAL*16
誤差関数 (以下参照)	erf(a)	1	ERR	ERF ✠	REAL	REAL
				DERF ✠	DOUBLE	DOUBLE
誤差関数	1.0 - erf(a)	1	ERR	ERFC ✠	REAL	REAL
				DERFC ✠	DOUBLE	DOUBLE

■ 誤差関数 :2/sqrt(pi) に、0 から exp(-t*t) dt の a の値を掛ける

文字関数

表 3-5　Fortran 77 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 整数への変換 参照 : 表 3-2.	1 1	CHAR ACHAR ✠ ICHAR IACHAR ✠	INTEGER CHARACTER	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 3-6 Fortran 77 ビット単位関数

ビット単位操作	引数の数	個別名	引数の型	関数の型
補数	1	NOT	INTEGER	INTEGER
論理積	22	AND IAND	INTEGERINTEGER	INTEGERINTEGER
内包的論理和	22	OR IOR	INTEGERINTEGER	INTEGERINTEGER
排他的論理和	22	XOR IEOR	INTEGERINTEGER	INTEGERINTEGER
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、15 ページの「bit：ビット関数: and、or、bit、setbit」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 3-7 Fortran 77 環境照会関数

定義	引数の数	総称名	引数の型	関数の型
進法の基底	1	EPBASE	INTEGER	INTEGER
			REAL	INTEGER
			DOUBLE	INTEGER
			REAL*16	INTEGER
有効ビット数	1	EPPREC	INTEGER	INTEGER
			REAL	INTEGER
			DOUBLE	INTEGER
			REAL*16	INTEGER
最小指数	1	EPEMIN	REAL	INTEGER
			DOUBLE	INTEGER
			REAL*16	INTEGER
最大指数	1	EPEMAX	REAL	INTEGER
			DOUBLE	INTEGER
			REAL*16	INTEGER
最小非ゼロ数	1	EPTINY	REAL	REAL
			DOUBLE	DOUBLE
			REAL*16	REAL*16
表現可能の最大	1	EPHUGE	INTEGER	INTEGER
			REAL	REAL
			DOUBLE	DOUBLE
			REAL*16	REAL*16
イプシロン 注 (16) 参照	1	EPMRSP	REAL	REAL
			DOUBLE	DOUBLE
			REAL*16	REAL*16

メモリー ✠

以下の関数はすべて FOTRAN 77 の規格外です。

表 3-8 Fortran 77 メモリー関数

組み込み関数	定義	引数の数	個別名	引数の型	関数の型
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

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。-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ることに注意してください。
  - 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

- 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。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、もっとも近い整数関数、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) COMPLEX

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, square_root (0.75))`
- `RT3 = RT1 * (-0.5, square_root (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、`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`

`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、`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

VMS Fortran ではその他のビット単位操作も可能ですが、実装されていません。

(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異なります。

(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、FORTRAN 77 の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、f95 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 3-9　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 3-10　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

表 3-10　vms 度単単位を用いる三角関数 (続き)

総称名	個別名	関数	引数の型	結果の型
ASIND		逆正弦	-	-
	ASIND		REAL*4	REAL*4
	DASIND		REAL*8	REAL*8
	QASIND		REAL*16	REAL*16
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 3-11　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
	KIBITS		INTEGER*8	INTEGER*8
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		INTEGER*2	INTEGER*2
	JISHFT		INTEGER*4	INTEGER*4
	KISHFT		INTEGER*8	INTEGER*8
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

表 3-11　vms ビット操作関数 (続き)

総称名	個別名	関数	引数の型	結果の型
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
	KIOR		INTEGER*8	INTEGER*8
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
	KIEOR		INTEGER*8	INTEGER*8
NOT		ビット単位補数	-	-
	INOT		INTEGER*2	INTEGER*2
	JNOT		INTEGER*4	INTEGER*4
	KNOT		INTEGER*8	INTEGER*8
IBSET		a1 で、ビット a2 を 1 に設定し、新しい a1 を	-	-
	IIBSET	戻す	INTEGER*2	INTEGER*2
	JIBSET		INTEGER*4	INTEGER*4
	KIBSET		INTEGER*8	INTEGER*8
BTEST		a1 のビット a2 が 1 ならば、TURE を戻す	-	-
	BITEST		INTEGER*2	INTEGER*2
	BJTEST		INTEGER*4	INTEGER*4
	BKTEST		INTEGER*8	INTEGER*8
IBCLR		a1 で、ビット a2 を 1 に設定し、新しい a1 を	-	-
	IIBCLR	戻す	INTEGER*2	INTEGER*2
	JIBCLR		INTEGER*4	INTEGER*4
	KIBCLR		INTEGER*8	INTEGER*8

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。コンパイラでは特定の 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 3-12 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		INTEGER*4	INTEGER*4
KIABS		INTEGER*8	INTEGER*8
IMAX0	最大 [1]	INTEGER*2	INTEGER*2
JMAX0		INTEGER*4	INTEGER*4
IMIN0	最小 [1]	INTEGER*2	INTEGER*2
JMIN0		INTEGER*4	INTEGER*4
IIDIM	超過分 [2]	INTEGER*2	INTEGER*2
JIDIM		INTEGER*4	INTEGER*4
KIDIM		INTEGER*8	INTEGER*8
IMOD	a1/a2 の剰余	INTEGER*2	INTEGER*2
JMOD		INTEGER*4	INTEGER*4
IISIGN	符号の付け替え、	INTEGER*2	INTEGER*2
JISIGN	|a1|*sign(a2)	INTEGER*4	INTEGER*4
KISIGN		INTEGER*8	INTEGER*8

1. 引数は 2 個以上でなければならない
2. 超過分 a1-min(a1,a2)

索引

E

F

M

N

O

P

Q

R

S

T

U

V

W

す

せ

そ

た

ち

て

と

な