G95

無料で数値計算

http://www.g95.org

G95 の特長

- 無料の Fortran 95 対応コンパイラ.
- 現在 (2006 年 9 月) g95 のバージョンは 0.91.
- GNU オープンソース, GPL ライセンス.
- コンパイルしたプログラムの動作は, プログラム自体に記述される多数の環境変数により変更可能.
- TR15581– 割り付け可能な仮引数や派生型の要素.
- F2003 スタイルの手続ポインタ, 構造体コンストラクタ, 相互運用性.
- F2003 の組み込み手続とモジュール.
- サブルーチンの仮引数に `VALUE` と指定すると, 値渡し.
- OPEN, READ, 及び WRITE 文で小数点を表す記号としてコンマが選択可能.
- 各括弧 [and] を行列のコンストラクタとして利用可.
- `IMPORT` 文. 引用仕様 (interface) 本体で用いると, 親の有効範囲 (スコープ) 内の言語要素が参照可能に.
- `MIN()` と `MAX()` は, 数値型だけでなく文字にも.
- `OPEN` は “透過的” (ストリーム) 入出力に対応.
- g77 のアプリケーション・バイナリ・インターフェース (ABI) への下位互換性.
- 既定の整数には 32 または 64 ビットが利用可.
- `SYSTEM()` でコマンドを実行.
- タブを使ったソースを許容.
- 変数や関数等の名前に $ の使用を許容するオプション.
- ホラリス (Hollerith) データが利用可.
- 拡張機能として倍精度複素数 DOUBLE COMPLEX.
- 可変長の名前付き `COMMON`.
- 数値型と文字型を `COMMON` や `EQUIVALENCE` に混在.
- `INTEGER` の種別 (kind) は: 1, 2, 4, 8.
- `LOGICAL` の種別は: 1, 2, 4, 8.
- `REAL` の種別は : 4, 8.
- `REAL(KIND=10)` が x86 互換システムで利用可. 19 桁の精度があり, 値の範囲は $10^{\pm 4931}$.
- リスト整形された浮動小数点出力では, その数をただ一つに区別するに必要な最少の桁を印字.
- VAX スタイルのデバッグ (D) 行.
- C スタイルの文字定数 (例 ’hello\nworld’).
- \ と $ 編集記述子.
- VAX スタイルのシステム組込定数 (SECNDS 他.)
- Unix システム拡張ライブラリ (getenv, etime, stat, 他.)
- 不適合あるいは未割り付け配列を実行時に検出 — `http://ftp.aset.psu.edu/pub/ger/fortran/test/results.txt` の表 IV 参照
- メモリリークを検出 — `http://ftp.aset.psu.edu/pub/ger/fortran/test/results.txt` の表 V 参照
- 実行時エラーのトレースバック.
- モジュールのコンパイルの連鎖を防止する賢いコンパイル
- F 互換オプション. `http://www.fortran.com/F` 参照. G95 は, F コンパイラとして構築可.
- プログラムの一時停止/再開機能有 (x86/Linux).
- 廃止された real または double precision の繰り返し変数 (loop index) は削除.
- バグの報告に対する対応はたいてい迅速.
- GCC 4.0.3 と 4.1.1 リリース・バージョンで構築.
- x86, PowerPC, 64 ビット Opteron, 64 ビット Itanium, 64 ビット Alpha 用 Linux で利用可.
- Windows 上の Cygwin, MinGW, 及び Interix で利用可.
- Power Mac G4 及び x86 を使った Mac OS X で利用可.
- x86 を使った FreeBSD, HP-UX 11, SPARC 及び x86 を使った Solaris, OpenBSD, NetBSD, AIX, IRIX, Alpha を使った Tru64 UNIX で利用可.
- Fink パッケージからもインストール可.
- 多くのプラットフォーム用の ‘安定’ かつ 最新版バイナリが `http://ftp.g95.org` から入手可.

g95 に関する電子メールを交換した方から, 「ひとりですごい仕事を成し遂げましたね」と言われることがよくあります. そんなときはいつも笑いながら, 一人でやった訳ではないと説明します. g95 の開発に積極的に関わった人々の数は, 千人に近いのではないでしょうか. コードを書いた人が全ての仕事をしたと思い込んでしまい, 実際には不具合の原因となるコードをいくつも見つけるということが非常に有意義な仕事であるということは, 見逃しがちです. モダンな Fortran コンパイラを書くというような複雑な仕事は, 一人でできることではないことを私自身が一番よく分かっています.

多くのもののように, g95 もやはり欲求不満から生まれました. 私は, 学位論文のコードを g77 を使って Fortran 77 で書きました. Fortran は数値計算には, 大変素晴らしい言語です. プログラムを書くことより, 答えを求めることに主な関心がある人たちには, 手っ取り早い道具です. ところが, 私の学位論文のコードは, 比較的高度なデータ構造を多用していました. 連結リスト, オクトリ, 疎行列を使って, 有限要素格子生成, ポワッソン方程式の解, 多極分解, 共役勾配最小化等多くの計算幾何のコードを書きました. Fortran 77 を使っていたために, コードは不格好になってしまいましたが, 動的メモリ配列やユーザ定義型があれば, どれほど有り難かったかは計り知れません. 学位論文が片付くにつれて, 新しいことに取り組もうという思いが高まってきたのです.

より高度な言語の機能により便利にするという以上に, Bill Kahan の仕事に大いに刺激を受けました. Bill の論文を多数読んで思ったことは, 数値計算はややこしいものだが, 誰も気にしないくらいに誤りを減らす方法は見つかるということです. この意味で, ユーザはしばしばライブラリの著者に全てを委ねることになります.

コンパイラは確かに素晴らしいのですが, ライブラリにはいつもより多くの関心を持ち続けていました. コンパイラの動作は標準によりかなり厳密に定義されており, むしろライブラリにおいて, 革新と実験を自由に行なうことができます. かなり初期の段階から, 他のベンダと比較してライブラリに注力しました. コアファイルの再開機能は, 長年ほしいと思っていたもので, ようやく実装する機会を得ました.

g95 を書くのは大変楽しい経験でした. この先数十年に渡って維持していければいいなぁと思っています.

2006 年 10 月
アリゾナ州 Mesa にて
Andy Vaught

ライセンス

G95 自体は, GNU 一般公衆許諾契約書 (GPL) の下でライセンスされている.

条文の詳細は `http://www.gnu.org/licenses/gpl.ja.html` 参照.

実行時ライブラリも, 概ね GPL に準拠するものとする. ただし, 例外として g95 のライブラリを GPL に準拠していないコードにリンクする権利, 及びリンクされた組み合わせを GPL に準拠することなく, GPL に全く影響を受けずに配布する権利を g95 のユーザに与える.

インストール方法

Unix (Linux/Mac OS X/Solaris/Irix 等.):
コンソールを開き, g95 をインストールしたいディレクトリに移動. g95 をダウンロードしてインストールするには, 次のようなコマンドを実行する (Linux/x86 の場合).

```
wget -O - http://ftp.g95.org/g95-x86-linux.tgz | tar xvfz -
ln -s $PWD/g95-install/bin/i686-pc-linux-gnu-g95 /usr/bin/g95
```

次のようなファイルやディレクトリが得られるはずである.

```
./g95-install/
./g95-install/bin/
./g95-install/bin/i686-pc-linux-gnu-g95
./g95-install/lib/gcc-lib/i686-pc-linux-gnu/4.1.1/
./g95-install/lib/gcc-lib/i686-pc-linux-gnu/4.1.1/f951
./g95-install/lib/gcc-lib/i686-pc-linux-gnu/4.1.1/crtendS.o
./g95-install/lib/gcc-lib/i686-pc-linux-gnu/4.1.1/crtend.o
./g95-install/lib/gcc-lib/i686-pc-linux-gnu/4.1.1/crtbeginT.o
./g95-install/lib/gcc-lib/i686-pc-linux-gnu/4.1.1/crtbeginS.o
./g95-install/lib/gcc-lib/i686-pc-linux-gnu/4.1.1/crtbegin.o
./g95-install/lib/gcc-lib/i686-pc-linux-gnu/4.1.1/cc1
./g95-install/lib/gcc-lib/i686-pc-linux-gnu/4.1.1/libf95.a
./g95-install/lib/gcc-lib/i686-pc-linux-gnu/4.1.1/libgcc.a
./g95-install/INSTALL
./g95-install/G95Manual.pdf
```

ファイル `cc1` は, 同じディレクトリにある `f951` へのシンボリックリンクである.

Cygwin

`-mno-cygwin` オプションを使えば, Cygwin 版の g95 で動作に cygwin1.dll へのアクセスを必要としない実行ファイルを構築できるので, 他のシステムで簡単に走らせることができる. また, 実行ファイルは GNU GPL ライセンスに伴う制限に拘束されない. -mno-cygwin オプションが動作する Cygwin 版をインストールするには, Cygwin のサイト `http://www.cygwin.com` から入手できる, mingw ライブラリがインストールされていなければならない.

`http://ftp.g95.org/g95-x86-cygwin.tgz` からバイナリをダウンロードして Cygwin ディレクトリ (通常 c:\Cygwin). Cygwin セッションを開始し, 次のコマンドを実行する.

```
cd /
tar -xvzf g95-x86-cygwin.tgz
```

この結果, g95 実行ファイルは, `/usr/local/bin` ディレクトリにインストールされる. 注意: Winzip を使って, tar アーカイブを展開しないこと. 必要なリンクが適切に設定されないことがある.

MinGW

MS-Windows 環境用の g95 バイナリは, 自己展開するインストーラにパッケージされている. 現在, 2 種類の版がある. Windows 98 のユーザは gcc 4.0.3 を使って構築された g95 パッケージ `http://ftp.g95.org/g95-MinGW.exe` を利用すること. Windows NT, XP 及び 2000 のユーザは, 上記パッケージまたは gcc 4.1.1 で構築された `http://ftp.g95.org/g95-MinGW-41.exe` のどちらでも利用できる.

フリーの MinGW/Msys システムは, `http://www.mingw.org` から入手できる binutils パッケージの `ld.exe` (リンカ), 及び `as.exe` (GNU アセンブラ) 等 g95 が必要とする GNU GCC のファイルを提供する. インストー

ラのスクリプトは, 2 種類のインストールを扱う. もし MinGW が見つからない場合は, 必要最低限の MinGW binutils プログラムとライブラリを g95 とともに, ユーザが指定したディレクトリにインストールする. インストールするディレクトリを PATH に含めて, 環境変数 LIBRARY_PATH がインストールするディレクトリを指すようにする.

MinGW が既にシステムにインストールされている場合, 潜在的な衝突を回避するため, g95 を MinGW のルート・ディレクトリ (通常 C:\mingw) にインストールすることを推奨する. インストーラが MinGW を検出した場合, MinGW ファイル・システムにインストールを試みる. MinGW\bin を PATH に含め, 次のように環境変数を設定する.

LIBRARY_PATH=*path-to-MinGW*/lib

Window 98 と Window ME では, システムの autoexec.bat を編集し, 再起動することにより, 変更が反映される.

Windows XP ユーザに対する注: MinGW の初期設定では, ヒープに 8 MB しかない. もし, アプリケーションがより多くのメモリを必要とするなら, -Wl,--heap=0x01000000 をつけてコンパイルする. プログラムが動くようになるまで, --heap の 16 進数の値を増大させる.

G95 の実行

G95 は, どのように入力ファイルがコンパイルされるかを拡張子から判断する. Fortran のソースとして許容されるファイル名の拡張子は, .f, .F, .for, .FOR, .f90, .F90, .f95, .F95, .f03 及び .F03 に限定されている. ファイル名拡張子から, Fortran のソースが固定書式あるいは自由書式として取り扱うかが決定される. .f, .F, .for 及び .FOR で終わるファイルは, 古い f77 ファイル互換の固定書式のソースであるとみなされる. .f90, .F90, .f95, .F95, .f03 及び .F03 で終わるファイルは, 自由書式のソースとみなされる. 大文字の拡張子は C プリプロセッサを通し, 小文字の拡張子は通さないのが既定の動作である.

g95 で Fortran のソースをコンパイルするための基本的なオプションは次の通り.

-c　コンパイルのみ. リンカを起動しない.

-v　g95 が実際に起動するプログラムとその引数を表示する. とくにパスを確認するのに便利である.

-o　出力ファイル (オブジェクト・ファイルまたは実行ファイル) の名前を指定する. Windows では, 拡張子 .exe が自動的に付加される. 出力ファイル名が指定されない場合は, unix では a.out, Windows では a.exe が既定の実行ファイル名となる.

簡単な例:

g95 -c hello.f90
hello.f90 をコンパイルして, hello.o という名前のオブジェクト・ファイルを作る.

g95 hello.f90
hello.f90 をコンパイル, リンクし a.out (Unix) あるいは a.exe (Windows) という実行ファイルを生成する.

g95 -c h1.f90 h2.f90 h3.f90
複数のソース・ファイルをコンパイル. 問題がなければ, オブジェクト・ファイル h1.o, h2.o 及び h3.o ができる.

g95 -o hello h1.f90 h2.f90 h3.f90
複数のソース・ファイルをコンパイルし, これらをリンクして hello (Unix) あるいは hello.exe (Windows) という名前の実行ファイルを生成する.

主要なオプション

g95 [-c | -S | -E]	コンパイル及びアセンブル | アセンブリ・コードの生成 | ソースの表示
[-g] [-pg]	デバッグオプション
[-O[n]]	最適化レベル, $n = 0, 1, 2, 3$
[-s]	デバッグ情報の除去
[-W*warn*] [-pedantic]	警告スイッチ
[-I*dir*]	ヘッダを検索するディレクトリに追加
[-L*dir*]	ライブラリを検索するディレクトリに追加
[-D macro[=value]...]	マクロの定義

`[-U macro ]`	マクロの消去
`[-f option ...]`	一般的なコンパイル・オプション
`[-m machine-option ...]`	機種依存オプション. GCC のマニュアル参照
`[-o outfile ]`	出力ファイルの名前を指定
`infile`	

G95 のオプション

Usage: g95 [options] file...

`-pass-exit-codes`	その段階で最も高位のエラー・コードで終了.
`--help`	オプション一覧を表示.
`--target-help`	ターゲットに依存したコマンドライン・オプションを表示. (‘`-v --help`’ を使用するとサブ・プロセスのオプションを表示).
`-dumpspecs`	構築されたもの全てのスペック文字列を表示.
`-dumpversion`	コンパイラのバージョンを表示.
`-dumpmachine`	コンパイラがターゲットにするプロセッサを表示.
`-print-search-dirs`	検索するディレクトリを表示.
`-print-libgcc-file-name`	コンパイラ添付のライブラリの名前を表示.
`-print-file-name=`*lib*	ライブラリ *lib* への絶対パスを表示.
`-print-prog-name=`*prog*	コンパイラのコンポーネント *prog* への絶対パスを表紙.
`-print-multi-directory`	libgcc のさまざまなバージョンのディレクトリを表示.
`-print-multi-lib`	コマンドライン・オプションと複数のライブラリを検索するディレクトリとの対応を表示.
`-print-multi-os-directory`	OS ライブラリの相対パスを表示.
`-Wa,`*options*	コンマで区切られた *options* をアセンブラに渡す.
`-Wp,`*options*	コンマで区切られた *options* プリプロセッサに渡す.
`-Wl,`*options*	コンマで区切られた *options* リンカに渡す.
`-Xassembler` *arg*	*arg* をアセンブラに渡す.
`-Xpreprocessor` *arg*	*arg* をプロセッサにプロセッサに渡す.
`-Xlinker` *arg*	*arg* をリンカに渡す.
`-save-temps`	中間ファイルを消去しない.
`-pipe`	中間ファイルではなく, パイプを使う.
`-time`	サブプロセスの実行時間. (MinGW, Mac OS X) 等にはない.
`-specs=`*file*	*file* の内容で組み込みの仕様を上書きする.
`-std=`*standard*	入力ソースが *standard* に準拠すると見なす.
`-B` *directory*	*directory* をコンパイラの検索パスに追加する.
`-b` *machine*	インストールされていたら, gcc をターゲット *machine* 用に動かす.
`-V` *version*	gcc バージョン *version* がインストールされていれば使用する.
`-v`	コンパイラが起動するプログラムを表示する.
`-M`	Makefile の依存関係を標準出力に表示する.
`-###`	`-v` と同様. ただし, オプションはクォートされ, コマンドは実行されない.
`-E`	プリプロセスのみで, コンパイル, アセンブル, リンクはしない.
`-S`	コンパイルのみで, アセンブルやリンクはしない.
`-c`	コンパイルとアセンブルはするが, リンクはしない.
`-o` *file*	結果を *file* に出力する.
`-x` *language*	後続の入力ファイルの *language* を指定する. 認められるのは言語は, c, c++, assembler, none で ‘none’ はファイルの拡張子から推定される既定の動作に戻ることを意味する.

`-g`, `-f`, `-m`, `-O`, `-W`, もしくは `--param` から始まるオプションは, 自動的に g95 が起動するサブ・プロセスに渡される. それ以外のオプションをサブ・プロセスに渡すには, `-W`*letter* オプションを使わなくてはならない. オプションに関するバグ報告の手引きは, `http://www.g95.org` 参照.

既定では, g95 でコンパイルされたプログラムは最適化されない. `-O`*n* の *n* は, 0 から 3 までの最適化のレベルを指定する. 0 は最適化なしで, 数字が大きくなるほど積極的な最適化が行われる. 最適化を指示すると,

コンパイラは高速化のためにコードを変更することが許可される．計算の結果への影響は，たいていごくわずかである．-O と -O1 とは同義である．

-O2 -march=*arch*

を指定するだけで，かなりの高速化が得られる．*arch* は pentium4, athlon, opteron のようなプロセッサのアーキテクチャである．このほか Fortran でよく用いられるのは，-funroll-loops, -fomit-frame-pointer, -malign-double 及び -msse2 である．g95 でコンパイルする際に使用できる全てのオプションについては，http://gcc.gnu.org/onlinedocs/gcc-4.1.1/gcc 参照．

プリプロセッサのオプション

G95 は，C プリプロセッサの構成要素を含むファイルを取り扱うことができる．

-cpp	入力ファイルが必ず C プリプロセッサで処理されるようにする．
-no-cpp	入力ファイルがプリプロセスされないようにする．
-D name[=value]	プリプロセッサのマクロを定義する．
-U name	プリプロセッサのマクロ定義を削除する．
-E	プリプロセッサを通したソースを表示して終了．
-I *directory*	include するファイルまたはモジュールの検索パスに *directory* を含める．ファイルは，次の順で様々なディレクトリから検索される．メインのソースのあるディレクトリ，カレント・ディレクトリ，-I で指定されたディレクトリ，環境変数 G95_INCLUDE_PATH で指定されたディレクトリ及びシステム・ディレクトリ．

Fortran のオプション

-Wall	ほとんどの警告メッセージを有効にする．
-Werror	警告をエラーとして扱う．
-Werror=*numbers*	コンマで区切って列挙した警告をエラーとして扱う．
-Wextra	-Wall でも有効化されない以下の警告を有効にする． -Wobsolescent, -Wunused-module-vars, -Wunused-module-procs, -Wunused-internal-procs, -Wunused-parameter, -Wunused-types, -Wmissing-intent and -Wimplicit-interface.
-Wglobals	同一のソースファイル内の手続の使用と定義とを照合する．既定では照合するが，-Wno-globals により無効化できる．
-Wimplicit-none	-fimplicit-none と同じ．
-Wimplicit-interface	定義されていない interface の使用を警告する．
-Wline-truncation	ソースの行の打ち切りを警告する．
-Wmissing-intent	引数に intent がないことを警告する．
-Wobsolescent	廃止予定の要素を警告する．
-Wno=*numbers*	コンマで区切られた番号の警告を無視する．
-Wuninitialized	未初期化変数があれば警告する．-O2 が必要．
-Wunused-internal-procs	未使用の内部手続を警告する．
-Wunused-vars	未使用変数があれば警告する．
-Wunused-types	未使用モジュール型があれば警告する．-Wall で有効となる警告に含まれない．
-Wunset-vars	代入されない変数があれば警告する．
-Wunused-module-vars	未使用のモジュール変数があれば警告する．ONLY 節を組み立てるのに便利．
-Wunused-module-procs	未使用のモジュール手続があれば警告する．ONLY 節を組み立てるのに便利．
-Wunused-parameter	未使用の定数があれば警告する．-Wall で有効となる警告に含まれない．
-Wprecision-loss	暗黙の型変換での精度低下があれば警告する．
-fbackslash	文字定数にバックスラッシュかあれば，エスケープコードとして解釈する．既定で有効である．バックスラッシュを文字通りとするには，-fno-backslash を指定する．
-fc-binding	手続の C プロトタイプを標準出力に印字する．
-fd-comment	固定書式で，D 行を実行可能文とする．
-fdollar-ok	構成要素の名前にドル記号を使用可能にする．

`-fendian=`*value* 書式なし入出力のエンディアンを指定する. *value* は `big` または `little`. 環境変数の値よりも優先する.

`-ffixed-form` ソースファイルを固定書式として扱う.

`-ffixed-line-length-132` 固定書式で, 1 行あたり 132 文字とする.

`-ffixed-line-length-80` 固定書式で, 1 行あたり 80 文字とする.

`-ffree-form` ソースファイルを自由書式として扱う.

`-ffree-line-length-huge` 非常に長い (10k) 行を許容する.

`-fimplicit-none` 暗黙の型宣言を認めない. ただし, 明示的に `IMPLICIT` 文で指定されていれば, そちらを優先する.

`-fintrinsic-extensions` `-std=` mode の場合でも, g95 の拡張組み込み関数を利用可能にする.

`-fintrinsic-extensions=` `-std=` mode の場合でも, 指定された g95 の組み込み関数を利用可能にする. 関数はコンマ区切りで指定し, 大文字・小文字は問わない.

`-fmod=`*directory* モジュール・ファイルを *directory* に出力する.

`-fmodule-private` モジュールの構成要素へのアクセス属性を既定で `PRIVATE` とする.

`-fmultiple-save` `SAVE` 属性の多重指定を許容する.

`-fone-error` 一つ目のエラーでコンパイルを強制終了する.

`-ftr15581` TR15581 で定義された動的割り付け配列拡張を `-std=F` や `-std=f95` モードでも利用可能にする.

`-std=F` F の機能に含まれないものがあれば警告する. `http://www.fortran.com/F` 参照.

`-std=f2003` Fortran 2003 への適合性を厳密に検査する.

`-std=f95` Fortran 95 への適合性を厳密に検査する.

`-i4` 指定のない整数は kind=4 (32 ビット) と見なす.

`-i8` 指定のない整数は kind=8 (64 ビット) と見なす.

`-r8` 指定のない実数は倍精度と見なす.

`-d8` `-i8` かつ `-r8` を意味する.

コード生成のオプション

`-fbounds-check` 配列と部分文字列の範囲を実行時に検査する.

`-fcase-upper` 公開シンボルを全て大文字にする.

`-fleading-underscore` 公開された名前の前にアンダスコアを付加する.

`-fonetrip` `DO`-loops を必ず 1 回は実行する. (`FORTRAN` 66 のバグ仕様).

`-fpack-derived` 派生型のメモリ配置をできるだけコンパクトにする. 必要なメモリは少なくなるが, 遅くなることもある.

`-fqkind=`*n* 'q' で記された指数を持つ実数の種別 (kind) を *n* とする.

`-fsecond-underscore` アンダスコアで終わる名前に 2 つ目のアンダスコアを付加する (既定). `-fno-second-underscore` を使えば抑制される.

`-fshort-circuit` 最初の被演算子で式の値が確定した場合, `.AND.` 及び `.OR.` 演算子に続く非演算子の計算を省略する.

`-fsloppy-char` 文字識別子に文字以外のデータを書き込むエラーを抑制し, INTEGER と CHARACTER との比較を許す.

`-fstatic` ローカル変数をできるだけ静的メモリに置く. これは, 静的リンク (`-static`) とは異なる.

`-ftrace=` `-ftrace=frame` は, プログラムが以上終了した場合に, スタックのトレースバックを出力するコードを挿入する. これにより実行速度は遅くなる. `-ftrace=full` は, さらに算術例外が発生した行番号を見つける (さらに遅くなる). 既定は, `-ftrace=none`.

`-funderscoring` 大域の名前の末尾にアンダスコア付加する. このオプションは, 既定で有効である. `-fno-underscoring` を使えば抑制される.

`-max-frame-size=`*n* スタックフレームの大きさがどれくらいか, 配列が動的に割り付けられる前にバイト単位でする.

`-finteger=`*n* 初期化されていないスカラ整数変数の値を *n* とする.

`-flogical=`*value*	初期化されていない論理型のスカラ局所変数を初期化する．有効な *values* は, `none`, `true` 及び `false`. である．
`-freal=`*value*	初期化されていない実数及び複素数変数を初期化する．有効な *values* は, `none`, `zero`, `nan`, `inf`, `+inf` 及び `-inf` である．
`-fpointer=`*value*	スカラ・ポインタを初期化する．有効な *values* は `none`, `null` 及び `invalid` である．
`-fround=`*value*	コンパイル時の丸め方を決める．*value* は, `nearest` (既定), `plus` (正の無限大), `minus` (負の無限大) または `zero` (0).
`-fzero`	数値型の変数を 0 で初期化する．論理型は false に, ポインタは null に初期化する．他の初期化オプションは, この設定よりも優先する．

ディレクトリのオプション

`-I`*directory*	*directory* をインクルード及びモジュールのファイル検索パスに追加する．
`-L`*directory*	*directory* ライブラリのファイル検索パスに追加する．
`-fmod=`*directory*	モジュール・ファイルを *directory* に保存する．

環境変数

g95 の実行時環境は, プログラムの実行時のふるまいを変更するための多数のオプションを提供している．実行時オプションは, 環境変数を通じて制御可能である．g95 でコンパイルされたプログラムを `--g95` オプションをつけて実行すると, 全ての実行時オプションを標準出力に書き出す．各変数の値は常に文字列であるが, 文字列は整数または真偽としても解釈される．真偽値としては, 最初の文字のみが検査される．その値は, '`t`', '`f`', '`y`', '`n`', '`1`' もしくは '`0`' でなければならない (大文字も可)．不正な値でもエラーにはならず, 既定値が使用される．`LIBRARY_PATH` のような g95 で使用される GCC の環境変数については, GCC の文献を参照.

`G95_STDIN_UNIT`	整数	標準入力に接続されるユニット番号．負値の場合は, 接続されない．既定は 5.
`G95_STDOUT_UNIT`	整数	標準出力に接続されるユニット番号．負値の場合は, 接続されない．既定は 6.
`G95_STDERR_UNIT`	整数	標準エラー出力に接続されるユニット番号．負値の場合は, 接続されない．既定は 0.
`G95_USE_STDERR`	真偽	ライブラリ出力を標準出力ではなく, 標準エラー出力に送る．既定は Yes.
`G95_ENDIAN`	文字列	書式なしデータの入出力に使用されるエンディアン形式．値は, `BIG`, `LITTLE` または `NATIVE`. 既定値は `NATIVE`.
`G95_CR`	真偽	書式付き逐次レコードで復帰 (CR 文字) を出力する．Cygwin を使わない Windows 版では TRUE, それ以外では FALSE が既定値.
`G95_INPUT_CR`	真偽	改行文字だけでなく, 復帰改行文字 (CR-LF) をレコードの区切り文字として扱う．既定値は TRUE.
`G95_IGNORE_ENDFILE`	真偽	逐次アクセスモードで, ENDFILE レコードを超える読み込みを試行しても無視する．既定値は FALSE.
`G95_TMPDIR`	文字列	スクラッチ・ファイルのためのディレクトリ．`TMP` 環境変数より優先する．もし `TMP` が指定されていない場合は, `/var/tmp` が使用される．既定値なし.
`G95_UNBUFFERED_ALL`	文字列	TRUE のとき, 全ての出力がバッファされない．この場合, 大容量の書き出しが遅くなるが, データを直ちに表示するようにできる．既定値は FALSE.
`G95_SHOW_LOCUS`	真偽	TRUE のとき, 実行時エラーが発生したファイル名と行番号を印字する．既定値は TRUE.
`G95_STOP_CODE`	真偽	TRUE のとき, stop コードでプログラムの終了コードを呼び出す．既定は TRUE.

G95_OPTIONAL_PLUS	真偽	可能なら正の値の前に + をつける. 既定は FALSE.
G95_DEFAULT_RECL	整数	逐次ファイルにおける既定の最大レコード長. 接続済のユニットでの行の長さを決めるときに利用できる. 既定値は 50000000.
G95_LIST_SEPARATOR	文字列	リスト出力を書き出す際に使用される区切り文字. 複数の空白文字や最大ひとつのコンマを含めることが可能. 既定値は ひとつの空白文字.
G95_LIST_EXP	整数	リスト出力で指数形式を使わない最大値. 10 の指数で与える. 既定値は 6.
G95_COMMA	真偽	コンマを小数点を表す既定の文字として入出力に使用する. 既定値は FALSE.
G95_EXPAND_UNPRINTABLE	真偽	書式付きの出力で, 表示できない文字を \ を使って表示する. 既定値は FALSE.
G95_QUIET	真偽	書式付き出力で, ベル文字 (\a) の印字を抑制する. 既定値は FALSE.
G95_SYSTEM_CLOCK	整数	組み込み関数 SYSTEM_CLOCK() により返される 1 秒あたりの刻み数. 0 とすると, 時計を使用しない. 既定値は 100000.
G95_SEED_RNG	真偽	TRUE のとき, プログラムの実行時に乱数生成に種を与える. 既定値は FALSE.
G95_MINUS_ZERO	真偽	TRUE なら, 書式付き (リストではない) 出力で, 内部の値がマイナスゼロでも符号なしのゼロを印字する. これは, 従来から用いられているが, 非標準のゼロの印字方法. 既定値は FALSE.
G95_ABORT	真偽	TRUE なら, プログラムの異常終了時にコアをダンプする. プログラム中の問題のある場所を特定するときに利用できる. 既定値は FALSE.
G95_MEM_INIT	文字列	メモリ割り付けの初期化方法. 既定値は NONE で初期化なし (高速). NAN とすると, 仮数 0x00f95 の NAN. または, 任意の 16 進値.
G95_MEM_SEGMENTS	整数	プログラム終了時に割り付けたままのメモリの最大値を表示する. 0 たと表示なし, 負の値だと全てを表示. 既定値は 25.
G95_MEM_MAXALLOC	真偽	TRUE なら, プログラム実行中に割り付けられたメモリの最大値をバイト単位で表示する. 既定値は FALSE.
G95_MEM_MXFAST	整数	Fastbin からの処理要求サイズの最大値. Fastbin は, 高速だが断片化しやすい. 既定値は 64 バイト.
G95_MEM_TRIM_THRESHOLD	整数	オペレーティング・システムに戻されるまで保持される最大メモリ量. -1 だとシステムには戻さない. 実行時間の長いプログラムで有用. 既定値は 262144.
G95_MEM_TOP_PAD	整数	OS からメモリを受ける際に余計に割り付ける量. 更なるメモリ要求の際に高速になりうる. 既定値は 0.
G95_SIGHUP	文字列	SIGHUP に対するプログラムのふるまい. IGNORE, ABORT, DUMP または DUMP-QUIT. 既定値は ABORT. Unix のみ.
G95_SIGINT	文字列	SIGINT に対するプログラムのふるまい. IGNORE, ABORT, DUMP または DUMP-QUIT. 既定値は ABORT. Unix のみ.
G95_SIGQUIT	文字列	SIGQUIT に対するプログラムのふるまい. IGNORE, ABORT, DUMP または DUMP-QUIT. 既定値は ABORT. Unix のみ.
G95_CHECKPOINT	整数	x86 Linux で, コアファイルのダンプのチェックポイント間の秒数. 0 ならダンプなし.
G95_CHECKPOINT_MSG	真偽	TRUE なら, プロセスがチェックポイントされたときに, 標準エラー出力にメッセージを印字する. 既定は TRUE.
G95_FPU_ROUND	文字列	浮動小数点の丸めモード. 値は, NEAREST, UP, DOWN または ZERO. 既定値は NEAREST.
G95_FPU_PRECISION	文字列	中間結果の精度. 値は 24, 53 または 64. 既定値は 64. x86 及び 互換プロセッサのみ.
G95_FPU_DENORMAL	真偽	非正規数を検出した場合に浮動小数点例外を発生させる. 既定値は FALSE.

G95_FPU_INVALID	真偽	無効な演算に対して, 浮動小数点例外を発生させる. 既定値は FALSE.
G95_FPU_ZERODIV	真偽	0 での除算に対して, 浮動小数点例外を発生させる. 既定値は FALSE.
G95_FPU_OVERFLOW	真偽	オーバフローに対して, 浮動小数点例外を発生させる. 既定値は FALSE.
G95_FPU_UNDERFLOW	真偽	アンダフローに対して, 浮動小数点例外を発生させる. 既定値は FALSE.
G95_FPU_INEXACT	真偽	桁落ちに対して, 浮動小数点例外を発生させる. 既定値は FALSE.
G95_FPU_EXCEPTIONS	真偽	マスクされた浮動小数点例外をプログラム終了後に表示する. 既定値は FALSE.
G95_UNIT_x	文字列	ユニット x の既定のユニット名を変更する. 既定値は fort.x
G95_UNBUFFERED_x	真偽	TRUE なら, unit x はバッファしない. 既定値は FALSE.

実行時のエラーコード

g95 でコンパイルされたプログラムを --g95 オプションをつけて実行するとエラーコードの一覧を標準出力に出力する.

-2 レコードの末尾
-1 ファイルの末尾
0 正常終了
オペレーティング・システムのエラーコード (1 - 199)
200 矛盾するステートメント・オプション
201 不正なステートメント・オプション
202 ステートメント・オプションの欠落
203 別のユニットで使用中のファイル
204 未接続ユニット
205 FORMAT エラー
206 誤った ACTION の指定
207 ENDFILE レコードを超えた読み込み
208 読み込み中の不正な値
209 読み込み中の数値オーバフロー
210 メモリ不足
211 割り付け済の配列
212 不正なポインタの解放
214 書式なし逐次アクセスファイル中の壊れたレコード
215 レコード長 (RECL) を超えたデータの読み込み
216 レコード長 (RECL) を超えたデータの書き出し

Fortran 2003 への対応

G95 は Fortran 2003 のいくつかの機能を実装している. Fortran 2003 の仕様については http://www.kcl.ac.uk/kis/support/cit/fortran/john_reid_new_2003.pdf を参照.

- 次の組み込み手続が利用可能: COMMAND_ARGUMENT_COUNT(), GET_COMMAND_ARGUMENT(), GET_COMMAND() 及び GET_ENVIRONMENT_VARIABLE()
- 単精度及び倍精度実数の DO loop インデックス変数は g95 では実装されいていない.
- 角括弧 [と] を (/ と /) の代わりに配列のコンストラクタとして利用可.
- TR 15581 — 割り付け可能なユーザ定義型. ALLOCATABLE 属性の使用を仮引数の定義, 関数の結果及び構造体の構成要素として利用可.
- ストリーム入出力 — F2003 のストリームアクセスは, Fortran プログラムがレコードの構造を考慮せずにバイナリファイルの読み書きを可能にする機能. Clive Page が書いてくれたこの機能についての解説が http://www.star.le.ac.uk/~cgp/streamIO.html にある.
- IMPORT 文. インターフェース部に使用され親スコープの要素へのアクセスを可能にする.

- 欧州で使われている実数の書式 — DECIMAL='COMMA' と OPEN, READ 及び WRITE 文で指定すれば, 小数点にコンマを使用する.
- MIN() 及び MAX() は, 数値型だけでなく文字に対しても使用できる.
- 型宣言属性 VALUE を副プログラムの仮引数に使用すると, 実引数は値で渡される.
- F2003 スタイルの構造体コンストラクタをサポート.
- F2003 スタイルの手続ポインタをサポート.
- F2003 では, 構造体の BIND(C) 属性, ISO_C_BINDING モジュールにより C との相互運用性が容易.

G95 プログラムとのインターフェース

g95 は独立した実行可能ファイルを生成するが, 時に他のプログラム, 通常 C とのインターフェースが必要になることがある. 複数の言語を使ったプログラムの第一の困難は, 公開されたシンボルの名前である. G95 は, 公開された名前にアンダスコアをひとつ, あるいはアンダスコアが含まれている場合は二つ付ける f2c の規約に従っている. -fno-second-underscore 及び -fno-underscoring オプションは, C コンパイラ互換の名前を g95 に生成させるときに用いる. nm コマンドを使えば, 二つのコンパイラにより生成された .o ファイル中の名前を確認できる. G95 も公開された名前は小文字にする. -fupper-case オプションが指定されると, 全て大文字にする. モジュールの名前は, *module-name*_MP_*entity-name* と表される.

リンク後は, Fortran が C の関数を呼ぶ場合と, C が Fortran のサブルーチンを呼ぶ場合とがある. 後者の場合, Fortran のサブルーチンは, 同じ方法でヒープが初期化されていることを想定するような Fortran ライブラリのサブルーチンを呼ぶことがよくある. C から手動で初期化するには, g95_runtime_start() を呼び Fortran ライブラリを初期化し, 終了時は g95_runtime_stop() を呼ぶ. g95_runtime_start() のプロトタイプは, 次の通り.

```
void g95_runtime_start(int argc, char *argv[]);
```

ライブラリはコマンドライン・オプションを処理できなければならない. もしこれがやりにくくて, プログラムがコマンドライン引数を必要としなければ, argc=0 と argv=NULL を渡せばよい. Mac OS X では, g95 を使ってリンカを起動し, gcc でコンパイルされたオブジェクト・ファイルをリンクする場合, -lSystemStubs を含める.

F2003 は, C とのインターフェースを容易にする機能を多数提供している. BIND(C) 属性は, C (あるいはほかの言語) から容易に参照できるような Fortran のシンボルを生成する. 例えば,

```
SUBROUTINE foo(a) BIND(C)
```

は, foo という名前の属性をアンダスコアを付けずに生成する. 全ての文字は, 小文字に置き換えられる. また,

```
SUBROUTINE foo(a) BIND(C, name='Foo1')
```

とすると, シンボルの名前を Foo1 とする. Fortran の中では, このサブルーチンは, 通常通り foo, FOO, その大文字・小文字の組み合わせで参照できる.

C プログラムは引数を値で渡すが, Fortran では参照で渡される. F2003 は, VALUE 属性により値で渡される仮引数を指定できる. 例えば,

```
SUBROUTINE foo(a)
    INTEGER, VALUE :: a
    ...
```

のように定義されたサブルーチンは, Fortran から呼び出し可能であるが, 仮引数は実引数ともはや結びついていないので, 仮引数の値を変更しても実引数の値は変更されなくなる.

大域変数も同様にアクセスできる. 次のサブルーチンでは, 変数 VAR の値を印字する以外にアクセスはできない.

```
SUBROUTINE print_it
    INTEGER, BIND(C, name='VAR') :: v
    PRINT *, v
END SUBROUTINE
```

Fortran では型が複数の種別 (kind) を持つのに対し, C では全てが別々の型である. 同一のオブジェクトを指定するために, F2003 で提供される組み込みモジュール ISO_C_BINDING では, Fortran の種別から C の型への対応がつけられる. USE されると, 次の PARAMETER が定義される.

c_int	C の int に対応する整数種別
c_short	C の short に対応する整数種別
c_long	C の long に対応する整数種別
c_long_long	C の long long に対応する整数種別
c_signed_char	C の char に対応する整数種別
c_size_t	C の size_t に対応する整数種別
c_intptr_t	C の ポインタとサイズが同一の整数種別
c_float	C の float に対応する実数種別
c_double	C の double に対応する実数種別

ISO_C_BINDING には, ほかにも多数の機能がある. このモジュールを使用すれば, 次のようなプログラムが書ける.

```
SUBROUTINE foo
    USE, INTRINSIC :: ISO_C_BINDING
    INTEGER(KIND=C_INT) :: int_var
    INTEGER(KIND=C_LONG_LONG) :: big_integer
    REAL(KIND=C_FLOAT) :: float_var
    ...
```

乱数生成器の使用

```
REAL INTENT(OUT)::  harvest CALL random_number(harvest)
```

REAL 型乱数の格納された REAL 型のスカラまたは配列 harvest, $0 \leq$ harvest < 1 を返す.
乱数生成器の初期化.

```
INTEGER, OPTIONAL, INTENT(OUT) ::  sz
INTEGER, OPTIONAL, INTENT(IN) ::  pt(n1)
INTEGER, OPTIONAL, INTENT(OUT) ::  gt(n2)
CALL random_seed(sz,pt,gt)
```

sz は, 初期値を格納するのに必要な既定の整数の数で, g95 は 4 を返す. 引数 pt は, 長さ n1 $\geq$ sz である既定の整数配列で, ユーザが与えた種の値格納されている. 引数 gt は長さ n2 $\geq$ sz の既定の整数配列で, 現在の種の値が格納されている.

引数なしで RANDOM_SEED() を呼ぶと, 種は呼び出し時に決められた値に初期化される. これは, プログラムを毎回起動する度に異なる乱数列を生成する場合に用いることができる. プログラムの起動された時刻をもとにした値で種を初期化するには, 環境変数 G95_SEED_RNG TRUE にする. どちらでもない場合には, RANDOM_NUMBER() は常に同じ数の列を生成する.

基礎となる生成器は, George Marsaglia が開発した xor シフト生成器である.

定義済みプリプロセッサ・マクロ

常に定義されているマクロには, 次のようなものがある.

```
__G95__ 0
__G95_MINOR__ 91
__FORTRAN__ 95
__GNUC__ 4
```

条件付きマクロは次の通り.

```
unix windows hpux linux solaris irix aix netbsd freebsd openbsd cygwin
```

コアファイルの再開機能

x86 Linux システムでは, g95 でコンパイルされたプログラムの実行は中断したり, 再開したりできる. もし, 通常コントロール+バックスラッシュと結びつけられている QUIT シグナルを送って実行を中断すると, プログラムは dump という名前の実行ファイルをカレント・ディレクトリに書き出す. 次の実行例で説明する.

```
andy@fulcrum:~/g95/g95 % cat tst.f90
  b = 0.0
  do i=1, 10
      do j=1, 3000000
          call random_number(a)
          a = 2.0*a - 1.0
          b = b + sin(sin(sin(a)))
      enddo
      print *, i, b
  enddo
  end
andy@fulcrum:~/g95/g95 % g95 tst.f90
andy@fulcrum:~/g95/g95 % a.out
 1 70.01749
 2 830.63153
 3 987.717
 4 316.48703
 5 -426.53815
 6 25.407673      (control-\ hit)
Process dumped
 7 -694.2718
 8 -425.95465
 9 -413.81763
 10 -882.66223
andy@fulcrum:~/g95/g95 % ./dump
Restarting
............Jumping
 7 -694.2718
 8 -425.95465
 9 -413.81763
 10 -882.66223
andy@fulcrum:~/g95/g95 %
```

全ての接続されたファイルは元の場所に存在していなければならない. 他の言語とリンクしている場合には, 動作しないこともある. 主な用途は実行時の状態を再起動の前後での保持を可能にすることであるが, ほかにも長時間かかるジョブを短時間のキューに入れたり, 実行中のプロセスを別のマシンに移したりするのに利用できる. プログラムのチェックポイントを自動的に生成するには, 環境変数 G95_CHECKPOINT にダンプ間の秒数を指定する. 0 はダンプなしを意味する. 新しいチェックポイント・ファイルは古いファイルを上書きする.

賢いコンパイル

モジュール foo のソースコードがファイル foo.f95 に含まれているとする. foo.f95 に対する変更には, 次のように 2 種類ある.

1. foo の使い方を変える変更. 例えば, 手続のインターフェース変更に伴うもの.
2. foo の使い方は変えない変更. 一方, その実装, 例えば, 手続本体のバグの修正.

どちらの変更も, 一般的にオブジェクト・ファイル foo.o の中身を変えるが, 最初の変更だけが foo.mod の内容を変更する. モジュールの再コンパイルをする際, g95 は .mod ファイルの更新が必要かどうかを検出することができる. 後者の変更の場合, .mod はそのまま保持される.

g95 のこの機能により, 大規模なプログラムを構築する際, 不必要なコンパイルの連鎖を防ぐことができる. 実際, 直接 (USE FOO 文により) あるいは, 間接的に (foo を使用してるモジュールを使用により, または foo を使用しているモジュールを使用しているモジュールの使用により等), たくさんの異なるソースファイルが foo.mod に依存しているとする. 前者の類の変更を foo.f95 に加えると, 依存しているソースファイル全ての再コンパイルを引き起こす. 幸いなことに, このようなことはそう多くはない. より頻繁に生ずる後者の類の変更では, foo.f95 だけをコンパイルし直せば, 新たなオブジェクトファイル foo.o は直ちに他のオブジェクトファイルにリンクし, 実行可能プログラムを更新できる.

G95 で拡張された組み込み関数

ACCESS
```
  INTEGER FUNCTION access(filename, mode)
      CHARACTER(LEN=*) :: filename
      CHARACTER(LEN=*) :: mode
  END FUNCTION access
```
filename という名前のファイルが指定された mode でアクセスできるか検査する. ここで, mode は rwxRWX のうち 1 またはそれ以上の文字. 許可に問題がなければ 0, 問題があれば 0 以外の値を返す.

ALGAMA
```
  REAL FUNCTION algama(x)
      REAL, INTENT(IN) :: x
  END FUNCTION algama
```
$\Gamma(x)$ の自然対数を返す. ALGAMA は, どの種別の実数でも引数にとる汎用関数.

BESJ0
```
  REAL FUNCTION besj0(x)
      REAL, INTENT(IN) :: x
  END FUNCTION besj0
```
0 次の第 1 種 Bessel 関数を返す. この関数は汎用関数である.

BESJ1
```
  REAL FUNCTION besj1(x)
      REAL, INTENT(IN) :: x
  END FUNCTION besj1
```
1 次の第 1 種 Bessel 関数を返す. この関数は汎用関数である.

BESJN
```
  REAL FUNCTION besjn(n,x)
      INTEGER, INTENT(IN) :: n
      REAL, INTENT(IN) :: x
  END FUNCTION besjn
```
n 次の第 1 種 Bessel 関数を返す. この関数は汎用関数である.

BESY0
```
  REAL FUNCTION besy0(x)
      REAL, INTENT(IN) :: x
  END FUNCTION besy0
```
0 次の第 2 種 Bessel 関数を返す. この関数は汎用関数である.

BESY1
```
  REAL FUNCTION besy1(x)
      REAL, INTENT(IN) :: x
  END FUNCTION besy1
```
1 次の第 2 種 Bessel 関数を返す. この関数は汎用関数である.

BESYN

```
  REAL FUNCTION besyn(n,x)
      INTEGER, INTENT(IN) :: n
      REAL, INTENT(IN) :: x
  END FUNCTION besyn
```

n 次の第 2 種 Bessel 関数を返す. この関数は汎用関数である.

CHMOD

```
  INTEGER FUNCTION chmod(file,mode)
      CHARACTER(LEN=*), INTENT(IN) :: file
      INTEGER, INTENT(IN) :: mode
  END FUNCTION chmod
```

Unix のファイル許可を変更する. エラーが発生した場合は, 0 でない値を返す.

DBESJ0

```
  DOUBLE PRECISION FUNCTION dbesj0(x)
      DOUBLE PRECISION, INTENT(IN) :: x
  END FUNCTION dbesj0
```

0 次の第 1 種 Bessel 関数を返す.

DBESJ1

```
  DOUBLE PRECISION FUNCTION dbesj1(x)
      DOUBLE PRECISION, INTENT(IN) :: x
  END FUNCTION dbesj1
```

1 次の第 1 種 Bessel 関数を返す.

DBESJN

```
  DOUBLE PRECISION FUNCTION dbesjn(n,x)
      INTEGER, INTENT(IN) :: n
      DOUBLE PRECISION, INTENT(IN) :: x
  END FUNCTION dbesjn
```

n 次の第 1 種 Bessel 関数を返す.

DBESY0

```
  DOUBLE PRECISION FUNCTION dbesy0(x)
      DOUBLE PRECISION, INTENT(IN) :: x
  END FUNCTION debsy0
```

0 次の第 2 種 Bessel 関数を返す.

DBESY1

```
  DOUBLE PRECISION FUNCTION dbesy1(x)
      DOUBLE PRECISION, INTENT(IN) :: x
  END FUNCTION dbesy1
```

1 次の第 2 種 Bessel 関数を返す.

DBESYN

```
  DOUBLE PRECISION FUNCTION dbesyn(n,x)
      INTEGER, INTENT(IN) :: n
      REAL, INTENT(IN) :: x
  END FUNCTION dbesyn
```

n 次の第 2 種 Bessel 関数を返す.

DCMPLX

```
  DOUBLE COMPLEX FUNCTION dcmplx(x,y)
  END FUNCTION dcmplx
```

倍精度の `CMPLX`. `x` 及び `y` はどの数値型あるいは種別でもよい.

DERF

```
  DOUBLE PRECISION FUNCTION derf(x)
      DOUBLE PRECISION, INTENT(IN) :: x
  END FUNCTION derf
```

x に対する倍精度の誤差関数を返す.

DERFC

```
  DOUBLE PRECISION FUNCTION derfc(x)
      DOUBLE PRECISION, INTENT(IN) :: x
  END FUNCTION derfc
```

x に対する倍精度の相補誤差関数を返す.

DFLOAT

```
  DOUBLE PRECISION FUNCTION dfloat(x)
  END FUNCTION dfloat
```

数値 x を倍精度に変換する. 組込関数 DBLE の別名.

DGAMMA

```
  DOUBLE PRECISION FUNCTION dgamma(x)
      DOUBLE PRECISION, INTENT(IN) :: x
  END FUNCTION dgamma
```

$\Gamma(x)$ の近似値を返す.

DLGAMA

```
  DOUBLE PRECISION FUNCTION dlgama(x)
      DOUBLE PRECISION, INTENT(IN) :: x
  END FUNCTION dlgama
```

$\Gamma(x)$ の自然対数を返す.

DREAL

```
  DOUBLE PRECISION FUNCTION dreal(x)
  END FUNCTION dreal
```

数値 x を倍精度にする. 組込み関数 DBLE の別名.

DTIME

```
  REAL FUNCTION dtime(tarray)
      REAL, OPTIONAL, INTENT(OUT) :: tarray(2)
  END FUNCTION dtime
```

現在のプロセスにおいて, 前に DTIME を呼んでから経過した user time の秒数を tarray(1) に, system time の秒数を tarray(2) に格納する. 返り値は, 二つの時刻の和.

ERF

```
  REAL FUNCTION erf(x)
      REAL, INTENT(IN) :: x
  END FUNCTION erf
```

x に対する誤差関数を返す. この関数は, 汎用関数である.

ERFC

```
  REAL FUNCTION erfc(x)
      REAL, INTENT(IN) :: x
  END FUNCTION erfc
```

x に対する相補誤差関数を返す. この関数は, 汎用関数である.

ETIME

```
  REAL FUNCTION etime(tarray)
      REAL, OPTIONAL, INTENT(OUT) :: tarray(2)
  END FUNCTION etime
```

現在のプロセスにおいて, 経過した user time の秒数を tarray(1) に, system time の秒数を tarray(2) に格納する. 返り値は, 二つの時刻の和.

FNUM

```
  INTEGER FUNCTION fnum(unit)
      INTEGER, INTENT(IN) :: unit
  END FUNCTION fnum
```

unit に対応するファイル識別番号を返す. unit が接続されていないときは -1 を返す.

FSTAT

```
  INTEGER FUNCTION fstat(unit, sarray)
      INTEGER, INTENT(IN) :: unit
      INTEGER, INTENT(OUT) :: sarray(13)
  END FUNCTION fstat
```

Fortran 入出力 unit で開かれているファイルについてのデータを取得し, 配列 sarray() に格納する. この配列の値は, 以下のようにシステム関数 fstat(2) が返す構造に準拠している. sarray(1) デバイス番号, sarray(2) i ノード番号, sarray(3) ファイルモード, sarray(4) リンク数, sarray(5) 所有者のユーザ ID, sarray(6) 所有者のグループ ID, sarray(7) デバイスのサイズ, sarray(8) ファイルサイズ, sarray(9) アクセス時刻, sarray(10) 修正時刻, sarray(11) 変更時刻, sarray(12) ブロックサイズ, sarray(13) 割り付け済ブロック.

FDATE

```
  CHARACTER(LEN=*) FUNCTION fdate()
  END FUNCTION fdate
```

現在の日付と時刻を Day Mon dd hh:mm:ss yyyy という形式で返す.

FTELL

```
  INTEGER FUNCTION ftell(unit)
      INTEGER, INTENT(IN) :: unit
  END FUNCTION ftell
```

Fortran ファイル unit の現在のオフセットを返す. unit が開かれていない場合は, -1 を返す.

GAMMA

```
  REAL FUNCTION gamma(x)
      REAL, INTENT(IN) :: x
  END FUNCTION gamma
```

$\Gamma(x)$ の近似値を返す. GAMMA は汎用関数で, 引数はどの実数種別でもよい.

GETCWD

```
  INTEGER FUNCTION getcwd(name)
      CHARACTER(LEN=*), INTENT(OUT) :: name
  END FUNCTION
```

name の中の現在の作業ディレクトリを返す. エラーが発生したら, 0 でない値を返す.

GETGID

```
  INTEGER FUNCTION getgid()
  END FUNCTION getgid
```

現在のプロセスのグループ ID を返す.

GETPID

```
  INTEGER FUNCTION getpid()
  END FUNCTION getpid
```

現在のプロセスのプロセス ID を返す.

GETUID

```
  INTEGER FUNCTION getuid()
  END FUNCTION getuid
```

ユーザ ID を返す.

HOSTNM
```
  INTEGER FUNCTION hostnm(name)
      CHARACTER(LEN=*), INTENT(OUT) :: name
  END FUNCTION hostnm
```
name にシステムのホスト名を格納する. エラーが発生した場合は, 0 以外の値を返す.

IARGC
```
  INTEGER FUNCTION iargc()
  END FUNCTION iargc
```
コマンドライン引数の数を返す. (プログラムの名前自体は含めない.)

ISATTY
```
  LOGICAL FUNCTION isatty(unit)
      INTEGER, INTENT(IN) :: unit
  END FUNCTION isatty
```
unit で指定された Fortran 入出力ユニットがターミナル・デバイスに接続されている場合のみに .true. を返す.

ISNAN
```
  LOGICAL FUNCTION isnan(x)
      REAL, INTENT(IN) :: x
  END FUNCTION isnan
```
x が 数値でない値 (NaN) のときに .true. を返す. この関数は, 汎用関数である.

LINK
```
  INTEGER FUNCTION link(path1, path2)
      CHARACTER(LEN=*), INTENT(IN) :: path1, path2
  END FUNCTION link
```
path1 から path2 に (ハード) リンクを作る.

LNBLNK
```
  INTEGER FUNCTION lnblnk(string)
      CHARACTER(LEN=*), INTENT(IN) :: string
  END FUNCTION lnblnk
```
標準の len_trim 関数の別名. 文字列中の空白でない最後の文字のインデックスを返す.

LSTAT
```
  INTEGER FUNCTION LSTAT(file, sarray)
      CHARACTER(LEN=*), INTENT(IN) :: file
      INTEGER, INTENT(OUT) :: sarray(13)
  END FUNCTION LSTAT
```
file がシンボリックリンクのとき, リンク自体のデータを返す. 詳細は FSTAT() 関数を参照. エラーが発生したときは, 0 でない値を返す.

RAND
```
  REAL FUNCTION rand(x)
      INTEGER, OPTIONAL, INTENT(IN) :: x
  END FUNCTION rand
```
一様疑似乱数 $0 \leq$ rand < 1 を返す. x が 0 のときは, 次の値が返される. x が 1 のときは, 乱数生成器は srand(0) を呼んで初期化される. x がその他の値のときは, srand に新しい種を与えて乱数が生成される.

SECNDS
```
  INTEGER FUNCTION secnds(t)
      REAL, INTENT(IN) :: t
  END FUNCTION secnds
```
t の値を差し引いた午前 0 時からの秒数でローカル時刻を返す. この関数は, 汎用関数である.

SIGNAL
```
  FUNCTION signal(signal, handler)
      INTEGER, INTENT(IN) :: signal
      PROCEDURE, INTENT(IN) :: handler
  END FUNCTION signal
```
Unix の signal コールへのインターフェース. エラーが発生したときは, 0 でない値を返す.

SIZEOF
```
  INTEGER FUNCTION sizeof(object)
  END FUNCTION sizeof
```
引数 object は式または型の名前. object サイズをバイト単位で返す.

STAT
```
  INTEGER FUNCTION stat(file, sarray)
      CHARACTER(LEN=*), INTENT(IN) :: file
      INTEGER, INTENT(OUT) :: sarray(13), status
  END FUNCTION stat
```
与えられた file についてのデータを取得し, 配列 sarray に格納する. 詳細は, fstat() 関数を参照. エラーが発生したときは, 0 でない値を返す.

SYSTEM
```
  INTEGER FUNCTION system(cmd)
      CHARACTER(LEN=*), INTENT(IN) :: cmd
  END FUNCTION system
```
文字列 cmd で指定した外部コマンドを呼ぶ. システムの終了コードを返す.

TIME
```
  INTEGER FUNCTION time()
  END FUNCTION time
```
Unix 関数 time 形式の整数として符号化された現在時刻を返す.

UNLINK
```
  INTEGER FUNCTION unlink(file)
      CHARACTER(LEN=*), INTENT(IN) :: file
  END FUNCTION unlink
```
ファイル file を削除する. エラーが発生したときは, 0 でない値を返す.

%VAL()
実引数列に指定した場合は, 変数は値で渡される. この擬関数は推奨されないが, 互換性のために実装されている. F2003 の VALUE 属性が値渡しの標準的な仕組みである.

%REF()
実引数列に指定した場合は, 変数は参照で渡される.

G95 で拡張された内部サブルーチン

ABORT
```
  SUBROUTINE abort()
  END SUBROUTINE abort
```
SIGABORT をプログラム自身に送ることにより, プログラムはコアをダンプして終了する (Unix).

CHDIR
```
  SUBROUTINE chdir(dir)
      CHARACTER(LEN=*), INTENT(IN) :: dir
  END SUBROUTINE
```
dir にカレントの作業ディレクトリを設定する.

DTIME

```
  SUBROUTINE dtime(tarray, result)
      REAL, OPTIONAL, INTENT(OUT) :: tarray(2), result
  END SUBROUTINE dtime
```

現在のプロセスにおいて, 前に DTIME を呼んから経過した user time の秒数を tarray(1) に, system time の秒数を tarray(2) に, 二つの時刻の和を result に格納する.

ETIME

```
  SUBROUTINE etime(tarray, result)
      REAL, OPTIONAL, INTENT(OUT) :: tarray(2), result
  END SUBROUTINE etime
```

現在のプロセスにおける user time の秒数を tarray(1) に, system time の秒数を tarray(2) に, 二つの時刻の和を result に格納する.

EXIT

```
  SUBROUTINE exit(code)
      INTEGER, OPTIONAL, INTENT(IN) :: code
  END SUBROUTINE exit
```

開かれている Fortran の入出力を閉じてから, 状態 code でプログラムを終了する. このサブルーチンは, 汎用サブルーチンである.

FDATE

```
  SUBROUTINE fdate(date)
      CHARACTER(LEN=*), INTENT(OUT) :: date
  END SUBROUTINE fdate
```

現在の日付と時刻を Day Mon dd hh:mm:ss yyyy という書式で date に格納する.

FLUSH

```
  SUBROUTINE flush(unit)
      INTEGER, INTENT(IN) :: unit
  END SUBROUTINE flush
```

現在出力として開かれている Fortran ファイル unit のバッファを消去する.

FSTAT

```
  SUBROUTINE FSTAT(unit, sarray, status)
      INTEGER, INTENT(IN) :: unit
      INTEGER, INTENT(OUT) :: sarray(13), status
  END SUBROUTINE fstat
```

Obtains data about the file open on Fortran I/O unit and places them in the array sarray(). Sets status to nonzero on error. See the fstat function for information on how sarray is set. Fortran 入出力 unit に開かれているファイルについてのデータを取得し, 配列 sarray() に格納する. エラーが発生したときは, 0 でない値を status に格納する. sarray が格納される情報については, fstat 関数を参照.

GETARG

```
  SUBROUTINE getarg(pos, value)
      INTEGER, INTENT(IN) :: pos
      CHARACTER(LEN=*), INTENT(OUT) :: value
  END SUBROUTINE
```

pos 番目のコマンドライン引数を value に格納する.

GETENV

```
  SUBROUTINE getenv(variable, value)
      CHARACTER(LEN=*), INTENT(IN) :: variable
      CHARACTER(LEN=*), INTENT(OUT) :: value
  END SUBROUTINE getenv
```

環境変数 variable を取得し, その値を value に格納する.

GETLOG

```
  SUBROUTINE getlog(name)
      CHARACTER(LEN=*), INTENT(OUT) :: name
  END SUBROUTINE getlog
```

プロセスのログイン名を name に格納する.

IDATE

```
  SUBROUTINE idate(m, d, y)
      INTEGER :: m, d, y
  END SUBROUTINE idate
```

現在の月を m に, 日を d に, 年を y に格納する. このサブルーチンは実装間であまり移植性が高くないので, 新しいコードでは標準の DATE_AND_TIME を使用すること.

LSTAT

```
  SUBROUTINE lstat(file,sarray,status)
      CHARACTER(LEN=*), INTENT(IN) :: file
      INTEGER, INTENT(OUT) :: sarray(13), status
  END SUBROUTINE lstat
```

file がシンボリックリンクのとき, リンク自体のデータを取得する. 詳細は, fstat() 参照.

RENAME

```
  SUBROUTINE rename(path1, path2, status)
      CHARACTER(LEN=*), INTENT(IN) :: path1, path2
      INTEGER, OPTIONAL, INTENT(OUT) :: status
  END SUBROUTINE rename
```

ファイル path1 を path2 に名前を変更する. status 引数が与えられていれば, エラーが発生したときに, 0 でない値をこれに格納する.

SIGNAL

```
  SUBROUTINE signal(signal, handler, status)
      INTEGER, INTENT(IN) :: signal
      PROCEDURE, INTENT(IN) :: handler
      INTEGER, INTENT(OUT) :: status
  END SUBROUTINE signal
```

Unix の signal システム・コールへのインターフェース. エラーが発生したときは, status に 0 でない値を格納する.

SLEEP

```
  SUBROUTINE sleep(seconds)
      INTEGER, INTENT(IN) :: seconds
  END SUBROUTINE sleep
```

プロセスの実行を seconds 秒間一時停止する.

SRAND

```
  SUBROUTINE srand(seed)
      INTEGER, INTENT(IN) :: seed
  END SUBROUTINE srand
```

乱数生成器を再初期化する. 詳しくは srand() 関数を参照.

STAT

```
  SUBROUTINE stat(file, sarray, status)
      CHARACTER(LEN=*), INTENT(IN) :: file
      INTEGER, INTENT(OUT) :: sarray(13), status
  END SUBROUTINE
```

与えられたファイルのデータを取得し配列 sarray に格納する. 詳細は fstat() 参照. エラーが発生した場合は, status に 0 でない値を格納する.

SYSTEM

```
  SUBROUTINE system(cmd, result)
      CHARACTER(LEN=*), INTENT(IN) :: cmd
      INTEGER, OPTIONAL, INTENT(OUT) :: result
  END SUBROUTINE system
```

コマンド cmd をシェルに渡す. result　が与えられているときは, これに cmd のシステム終了コードを格納する.

UNLINK

```
  SUBROUTINE unlink(file, status)
      CHARACTER(LEN=*), INTENT(IN) :: file
      INTEGER, INTENT(OUT) :: status
  END SUBROUTINE unlink
```

ファィル file を削除する. エラーが発生したときは, status に 0 でない値を格納する.