NEC

SX-Aurora TSUBASA

Fortran コンパイラ　ユーザーズガイド

SX-Aurora TSUBASA

輸出する際の注意事項

本製品（ソフトウェアを含む）は、外国為替および外国貿易法で規定される規制貨物（または役務）に該当することがあります。

その場合、日本国外へ輸出する場合には日本国政府の輸出許可が必要です。

なお、輸出許可申請手続きにあたり資料等が必要な場合には、お買い上げの販売店またはお近くの当社営業拠点にご相談ください。

は　し　が　き

本書は、Vector Engine 向けの NEC Fortran コンパイラの使用法について説明したものです。

備考

(1) 本書は 2022 年 3 月発行の第 26 版です。

(2) NEC Fortran コンパイラは、以下の言語仕様標準に準拠しています。

- ISO/IEC 1539-1:2004 Programming languages - Fortran
- OpenMP Application Program Interface Version 4.5

(3) ISO/IEC 1539-1:2010 Programming languages - Fortran の一部機能にも対応しています。

(4) OpenMP Application Program Interface Version 5.0 の一部機能にも対応しています。

(5) 本書では Vector Engine を略して VE と表記しています。

(6) 本書の読者は、Linux 上での Fortran/C/C++言語でのソフトウェア開発の知識を有していることを前提としています。

(7) 本書内の製品名、ブランド名、社名などは、一般に各社の表示、商標または登録商標です。

目　次

第1章　Fortran コンパイラ

1.1　概要

NEC Fortranコンパイラは、Fortranプログラムをコンパイル・リンクし、VEのCPUで実行するためのバイナリを作成するコンパイラです。本コンパイラは、VEのハードウェア性能を限界まで容易に引き出せるよう、次の最適化機能を実装しています。

- 自動ベクトル化機能
- 自動並列化機能、OpenMP並列化機能
- 自動インライン展開機能
- 性能情報取得機能

各種コンパイラオプションの指定により、これらの機能を選択しながら、その能力を最大限に利用することができます。最適化機能の詳細およびコンパイラオプションについては、2章以降を参照してください。

1.2　コンパイラの起動

(1)　環境変数の設定

NEC Fortranコンパイラ起動時にパスの指定を省略したいときに、環境変数PATHを設定してください。NEC Fortranコンパイラは、標準で/opt/nec/ve配下にインストールされます。環境変数PATHに/opt/nec/ve/binを追加してください。

NEC Fortranコンパイラには、モジュールファイルやライブラリなどのパスを設定する環境変数が用意されていますが、NEC Fortranコンパイラは、既定のパスを自動的にサーチするため、これらの環境変数を設定しなくてもご利用いただけます。環境変数は、コンパイラに付属しないOSSのライブラリのパスを常に追加したい場合など、標準以外のディレクトリをサーチする必要があるときに設定してください。

環境変数については、「第2章 環境変数」を参照してください。

(2)　起動例

コンパイラの起動例を説明します。指定しているコンパイラオプションの詳細については、「第3章 コンパイラオプション」を参照してください。

- Fortranソースファイル(a.f90)のコンパイル、リンク

```
$ nfort a.f90
```

- 複数のソースファイルのコンパイル、リンク

```
$ nfort a.f90 b.f90
```

- 作成する実行ファイルの名前(prog.out)を指定するコンパイル、リンク

```
$ nfort -o prog.out a.f90
```

- 最大限のベクトル化、最適化を行うコンパイル、リンク

```
$ nfort -O4 a.f90
```

- 副作用を伴わないレベルでベクトル化、最適化を行うコンパイル、リンク

```
$ nfort -O1 a.f90
```

- ベクトル化、最適化を行わないコンパイル、リンク

```
$ nfort -O0 a.f90
```

- 自動並列化を行うコンパイル、リンク

```
$ nfort -mparallel a.f90
```

- 自動インライン展開を行うコンパイル、リンク

```
$ nfort -finline-functions a.f90
```

- 特定のバージョンのコンパイラを使用してコンパイル、リンク

```
$ /opt/nec/ve/bin/nfort-X.X.X a.f90 (X.X.Xはコンパイラのバージョン番号)
```

1.3　プログラムの実行

プログラムを実行するときの例を以下に示します。

- プログラムを実行

```
$ ./a.out
```

- 利用するVEを指定してプログラムを実行

```
$ env VE_NODE_NUMBER=1 ./a.out                (ノード番号1のVEで実行)
```

- 入力ファイル、入力パラメタを指定して実行

```
$ ./a.out data1.in 10                    （ファイルdata1.in、値10を入力）
```

- 入力ファイルをリダイレクトして実行

```
$ ./a.out < data2.in
```

- スレッド数を指定して並列プログラムを実行

```
$ nfort -mparallel -O3 a.f90 b.f90
$ export OMP_NUM_THREADS=4
$ ./a.out
```

- 外部ファイル装置に事前にファイルを接続してプログラムを実行

```
$ export VE_FORT9=DATA9                  （装置番号9にファイルDATA9を接続）
$ ./a.out
```

- プロファイラ(ngprof)の使用
  コンパイル、リンク時に**-pg**を指定し、コンパイルしたプログラムを実行すると性能測定結果がファイルgmon.outに出力されます。gmon.outをngprofコマンドで解析、表示できます。

```
$ nfort -pg a.f90
$ ./a.out
$ ls gmon.out
gmon.out
$ ngprof
（性能情報が表示される）
```

1.4　コマンドラインの指定形式

- コンパイラのコマンドラインの指定形式は以下のとおりです。

```
nfort  [ コンパイラオプション | ファイル ] ...
```

1.5　コンパイラオプションの指定

- コンパイラオプションはハイフン(-)に続けて指定します。複数のコンパイラオプションを指定するとき、空白で区切って指定します。

  例

```
$ nfort -v -c a.f90          (正)
$ nfort -vc a.f90            (誤)
```

- コンパイラはファイルとして以下の拡張子を認識します。その他の拡張子は、オブジェクトファイルとして扱われます。

拡張子	ファイル種別
.F　.FOR　.FTN　.FPP　.F90　.F95　.F03 .f　.for　.ftn　.fpp　.f90　.f95　.f03 .i　.i90	Fortranソースファイル
.c	Cソースファイル
.S　.s	アセンブラソース

- コンパイラオプションとファイルは、オプションファイルに記述することができます。オプションファイルは、コンパイラの起動時に常に指定するコンパイラオプションやファイルを指定するファイルです。コンパイラオプション、ファイルはコマンドラインと同じ規則で指定します。オプションファイルは、環境変数**HOME**に設定されたホームディレクトリに”.nfortinit”というファイル名で作成します。

  例

```
$ cat ~/.nfortinit
-O3 -finline-functions
$ nfort -v a.f90
/opt/nec/ve/libexec/fcom … -O3 -finline-functions … a.f90
```

1.6　モジュールファイルのサーチ

ソースファイルがモジュールを含むとき、そのモジュールを他のソースファイルから引用するための情報として、モジュールごとにモジュールファイルが出力されます。組込みモジュールのモジュールファイルはあらかじめ所定の場所に用意されています。

(1)　組込みでないモジュールのモジュールファイルのサーチ

引用しているモジュールがコンパイル中のソースファイルに含まれていないとき、次のディレクトリからモジュールファイルをサーチします。モジュールファイルのサーチ順序は以下のとおりです。

(a)　ソースファイルのあるディレクトリ
(b)　**-module**で指定されたディレクトリ
(c)　カレントディレクトリ
(d)　**-I**で指定されたディレクトリ
(e)　**-B**で指定されたディレクトリ直下のincludeディレクトリ
(f)　環境変数**NFORT_INCLUDE_PATH**で指定されたディレクトリ
(g)　-**isystem**で指定されたディレクトリ
(h)　/opt/nec/ve/nfort/*バージョン番号*/include
(i)　**-isysroot**が指定されているとき、**-isysroot**で指定されたディレクトリ直下のinclude、**-isysroot**が指定されていないとき、/opt/nec/ve/include

(2)　組込みモジュールのモジュールファイルのサーチ

組込みモジュールは、**INTRINSIC**属性付きの**USE**文で引用されます。**INTRINSIC**属性付きで引用されたモジュールのモジュールファイルは次のディレクトリ配下をサーチします。

(a)　**-fintrinsic-modules-path**が指定されているとき、**-fintrinsic-modules-path**で指定されたディレクトリ。
(b)　**-fintrinsic-modules-path**が指定されていないとき、/opt/nec/ve/nfort/*バージョン番号*/include

1.7　INCLUDE行、および、#includeで取り込まれるファイルのサーチ

INCLUDE行、および、**#include** *"ファイル名"*で取り込まれるファイルがサーチされるディレクトリの順番は以下のとおりです。

(1)　ソースファイルのあるディレクトリ

(2)　カレントディレクトリ

(3)　**-I** で指定されたディレクトリ

(4)　**-B** で指定されたディレクトリ直下の include ディレクトリ

(5)　環境変数 **NFORT_INCLUDE_PATH** で指定されたディレクトリ

(6)　**-isystem** で指定されたディレクトリ

(7)　/opt/nec/ve/nfort/*バージョン番号*/include

(8)　**-isysroot** が指定されているとき、**-isysroot** で指定されたディレクトリ直下の include、**-isysroot** が指定されていないとき、/opt/nec/ve/include

1.8　ライブラリのサーチ

ライブラリがサーチされるディレクトリの順番は以下のとおりです。

(1)　**-L** で指定されたディレクトリ

(2)　**-B** で指定されたディレクトリ

(3)　環境変数 **NFORT_LIBRARY_PATH** で指定されたディレクトリ

(4)　/opt/nec/ve/nfort/*バージョン番号*/lib

(5)　環境変数 **VE_LIBRARY_PATH** で指定されたディレクトリ

(6)　/opt/nec/ve/lib/gcc

(7)　/opt/nec/ve/lib

1.9　演算例外

1.9.1　演算例外発生時の演算結果

浮動小数点数や整数の演算中にオーバーフロー、アンダーフロー、ゼロ除算、無効演算、または、精度落ちが発生したときの処置について説明します。

(1)　ゼロ除算

整数型の演算において、除数がゼロの除算が行われたとき、演算結果は不定になります。

整数型でない演算において、除数がゼロの除算が行われたとき、被除数が正なら正の無限大、負なら負の無限大の演算結果になります。

環境変数**VE_FPE_ENABLE**に”DIV”を設定したとき、例外が発生し、エラーメッセージが標準エラー出力に出力されます。環境変数**VE_FPE_ENABLE**に”DIV”を設定しなかったとき、ゼロ除算例外は発生しません。

(2)　浮動小数点オーバーフロー

実数型、複素数型の演算において、オーバーフローが発生したとき、値が正なら正の有限の最大値を、負なら負の無限大が演算結果になります。

環境変数**VE_FPE_ENABLE**に”FOF”を設定したとき、例外が発生し、エラーメッセージが標準エラー出力に出力されます。環境変数**VE_FPE_ENABLE**に”FOF”を設定しなかったとき、浮動小数点オーバーフロー例外は発生しません。

(3)　浮動小数点アンダーフロー

実数型、複素数型の演算において、アンダーフローが発生したとき、ゼロが演算結果になります。

環境変数**VE_FPE_ENABLE**に”FUF”を設定したとき、例外が発生し、エラーメッセージが標準エラー出力に出力されます。環境変数**VE_FPE_ENABLE**に”FUF”を設定しなかったとき、浮動小数点アンダーフロー例外は発生しません。

(4)　無効演算

実数型、複素数型の演算において、無効演算が発生したとき、演算結果は不定、または、NaNになります。

環境変数**VE_FPE_ENABLE**に”INV”を設定したとき、例外が発生し、エラーメッセージが標準エラー出力に出力されます。環境変数**VE_FPE_ENABLE**に”INV”を設定しなかったとき、無効演算例外は発生しません。

(5)　精度落ち

実数型、複素数型の演算において、精度落ちが発生したとき、丸めた値が演算結果となります。

環境変数**VE_FPE_ENABLE**に”INE”を設定したとき、例外が発生し、エラーメッセージが標準エラー出力に出力されます。環境変数**VE_FPE_ENABLE**に”INE”を設定しなかったとき、精度落ち例外は発生しません。

(6)　ベクトル命令実行中の演算例外

ベクトル命令の実行中にオーバーフロー、アンダーフロー、ゼロ除算が発生したときの演算結果は、スカラ命令のときと同じです。ただし、一つのベクトル命令の実行中に複数個の演算例外が発生するときでも、例外は1回しか発生しません。

1.9.2　演算例外マスクの変更

Fortranプログラムでは、演算例外マスクを変更することにより、演算例外発生時に例外を発生させるかどうかを制御できます。

環境変数**VE_FPE_ENABLE**により、演算例外マスクを変更できます。環境変数**VE_FPE_ENABLE**の指定では、どの演算例外を発生させるかを指定します。

例

```
$ export VE_FPE_ENABLE=FOF,DIV
$ ./a.out
```

浮動小数点オーバーフロー時(FOF)とゼロ除算時(DIV)に例外を発生させるように、演算例外マスクを変更します。

1.9.3　トレースバック機能との連携

演算例外マスクを変更し、演算例外を発生させることで、トレースバック機能で例外発生箇所を容易に特定できます。

例

```
$ nfort -traceback=verbose below.f90 out.f90 watch.f90 hey.f90 ovf.f90
...
$ export VE_TRACEBACK=VERBOSE
$ export VE_FPE_ENABLE=DIV
$ ./a.out
Runtime Error: Divide by zero at 0x600008001088
[ 0] 0x600008001088 below_           below.f90:3
[ 1] 0x600018001168 out_             out.f90:3
[ 2] 0x600020001168 watch_           watch.f90:3
[ 3] 0x600010001168 hey_             hey.f90:3
[ 4] 0x60000001cab8 MAIN__           ovf.f90:5
```

例ではbelow.f90の3行目付近でDivide by zeroの例外が発生したことがわかります。

1.9.4　演算例外マスクを変更するときの注意事項

演算例外マスクの変更は手続から呼び出しているシステムライブラリ関数にも適用されます。したがって、システムライブラリ関数で精度落ちなどの演算例外が発生したときも、エラーが発生することがあります。

1.10 実行時の終了コード

プログラムを実行したときの終了コードを以下に示します。

終了コード	意味
0	正常終了
1	実行時エラー
2	文字型の終了コードを指定した**ERROR STOP**文で終了
137	実行時エラー(アボート時)
n	整数型の終了コード*n*を指定した**STOP**文で終了、または、終了コード*n*を指定した組込みサブルーチン**EXIT**で終了

第2章　　環境変数

2.1　コンパイル時に参照される環境変数

HOME

コンパイラは、オプションファイルを取得するために環境変数**HOME**を参照し、ユーザのホームディレクトリを取得します。環境変数**HOME**に値が設定されていないとき、オプションファイルは参照されません。

NFORT_COMPILER_PATH

nfortから起動するFortranコンパイラ(fcom)をサーチするディレクトリを指定します。複数指定するときはコロン(:)で区切って指定します。先に指定したディレクトリが優先されます。Fortranコンパイラがみつからなかったとき、nfortは標準のディレクトリにあるFortranコンパイラを自動的に起動します。この環境変数は、標準以外のディレクトリを常にサーチしたい場合に設定します。

例

```
$ export NFORT_COMPILER_PATH="$HOME/libexec:$HOME/wk/libexec"
```

NFORT_INCLUDE_PATH

#include、または、**INCLUDE**文で取り込まれるファイル、モジュールファイルをサーチするディレクトリを指定します。複数指定するときはコロン(:)で区切って指定します。先に指定したディレクトリが優先されます。この環境変数は、標準以外のディレクトリを常にサーチしたい場合に設定します。

例

```
$ export NFORT_INCLUDE_PATH="$HOME/include:$HOME/wk/include"
```

NFORT_LIBRARY_PATH

ライブラリをサーチするディレクトリを指定します。複数指定するときはコロン(:)で区切って指定します。先に指定したディレクトリが優先されます。この環境変数は、標準以外のディレクトリを常にサーチしたい場合に設定します。

例

```
$ export NFORT_LIBRARY_PATH="$HOME/lib:$HOME/wk/lib"
```

NFORT_PROGRAM_PATH

Foratranコンパイラから起動するVE用のアセンブラおよびリンカをサーチするディレクトリを指定します。複数指定するときはコロン(:)で区切って指定します。先に指定したディレクトリが優先されます。アセンブラおよびリンカがみつからなかったとき、Fortranコンパイラは標準のディレクトリにあるアセンブラおよびリンカを自動的に起動します。この環境変数は、標準以外のディレクトリを常にサーチしたい場合に設定します。

例

```
$ export NFORT_PROGRAM_PATH="$HOME/bin:$HOME/wk/bin"
```

PATH

nfortをサーチするディレクトリを追加します。複数指定するときはコロン(:)で区切って指定します。先に指定したディレクトリが優先されます。Fortranコンパイラをインストールしたディレクトリ配下のbinディレクトリを追加してください。この環境変数を設定すると、nfortを起動するとき、パスの指定を省略できます。標準のディレクトリにインストールしたときは、/opt/nec/ve/binを追加します。環境変数**PATH**はFortranコンパイラ以外のアプリケーションにも影響を与えます。既存の環境変数**PATH**に追加する形で設定してください。

例

```
$ export PATH="/opt/nec/ve/bin:$PATH"
```

TMPDIR

コンパイラとコマンドが一時ファイルを作成するディレクトリを指定します。

(既定値: /tmp)

VE_LIBRARY_PATH

システムライブラリをサーチするディレクトリを指定します。複数指定するときはコロン(:)で区切って指定します。この環境変数は、標準以外のディレクトリを常にサーチしたい場合に設定します。

例

```
$ export VE_LIBRARY_PATH="$HOME/lib:$HOME/wk/lib"
```

2.2　実行時に参照される環境変数

LD_LIBRARY_PATH

書式付き配列一括出力、および、並び配列一括出力のVHへのオフロード機能で利用するライ

ブラリを格納するディレクトリを指定します。書式付き配列一括出力、および、並び配列一括出力のVHへのオフロード機能については、「5.3 その他の高速化機能」を参照してください。

例

```
$ export LD_LIBRARY_PATH=/opt/nec/ve/nfort/lib64
```

OMP_NUM_THREADS / VE_OMP_NUM_THREADS

OpenMPと自動並列化プログラムで使用するスレッド数を設定します。値を設定しないとき、スレッド数はVEのコア数と同じです。

例

```
$ export OMP_NUM_THREADS=4
```

OMP_STACKSIZE / VE_OMP_STACKSIZE

OpenMPと自動並列化プログラムで各スレッドが使用するスタックサイズの上限をキロバイト単位で指定します。単位として"M"を使用するとメガバイト、"G"を使用するとギガバイトとして指定することができます。値を設定しないとき、各スレッドが使用するスタックサイズは4メガバイトとなります。

例

```
$ export OMP_STACKSIZE=1G
```

VE_ADVANCEOFF

Advance-offモードを制御します。"YES"を設定するとAdvance-offモードが有効になります。他の値を設定したとき、Advance-offモードは無効になります。Advance-offモードを有効にしたとき、実行時間が大幅に長くなる場合があります。

例

```
$ export VE_ADVANCEOFF=YES
```

VE_ERRCTL_ALLOCATE

割付け変数、または、ポインタの割付けに関する実行時エラーが発生したときのプログラムの実行を制御します。以下の値を指定できます。

ABORT

エラーメッセージを出力し、実行を異常終了させます。(既定値)

MSG

エラーメッセージを出力し、実行を継続させます。

NOMSG

エラーメッセージを出力せず、実行を継続させます。

例

```
$ export VE_ERRCTL_ALLOCATE=MSG
```

VE_ERRCTL_DEALLOCATE

割付け変数、または、ポインタの解放に関する実行時エラーが発生したときのプログラムの実行を制御します。以下の値を指定できます。

ABORT

エラーメッセージを出力し、実行を異常終了させます。

MSG

エラーメッセージを出力し、実行を継続させます。

NOMSG

エラーメッセージを出力せず、実行を継続させます。(既定値)

例

```
$ export VE_ERRCTL_DEALLOCATE=ABORT
```

VE_FMTIO_OFFLOAD

書式付き配列一括出力、および、並び配列一括出力をVHにオフロードすることを制御します。”YES”、または、”ON”を指定するとVHへのオフロードが有効になります。書式付き配列一括出力、および、並び配列一括出力のVHへのオフロード機能については、「5.3 その他の高速化機能」を参照してください。

例

```
$ export VE_FMTIO_OFFLOAD=YES
```

VE_FMTIO_OFFLOAD_THRESHOLD

書式付き配列一括出力、および、並び配列一括出力をVHにオフロードする配列の要素数の閾値を指定します。指定した値より要素数の小さい配列はVHにオフロードされません。既定値は10です。

例

```
$ export VE_FMTIO_OFFLOAD_THRESHOLD=20
```

VE_FORT*n*

外部ファイル装置*n*に接続するファイル名を設定します。既定のファイル名は”fort.*n*”です。この変数を設定すると、既定のファイル名から設定したファイル名に変更されます。

例

```
$ export VE_FORT9=DATA9
```

VE_FORT_ABORT

致命的エラーを検出したときにコアダンプを出力するかどうかを制御します。”YES”を指定することでコアダンプが出力されます。

備考　　Fortranの「Runtime error」以外の原因によるコアダンプの出力は、環境変数**VE_FORT_ABORT**では制御できません。

例

```
$ export VE_FORT_ABORT=YES
```

VE_FORT_ACCUMULATE_THREAD_CPU_TIME

マルチスレッドのプログラムで、組込み手続**CPU_TIME**が返却する値を制御します。"YES"を指定すると全スレッドのCPU時間の合計を返却します。

例

```
$ export VE_FORT_ACCUMULATE_THREAD_CPU_TIME=YES
```

VE_FORT_DEFAULTFILE

入出力ファイルのデフォルトのディレクトリパスの起点をカレントディレクトリから指定したディレクトリに変更します。**OPEN**文の**FILE**指定子、または、環境変数VE_FORT*n*で指定されたファイル名が絶対パス名で指定されているとき、本環境変数の指定は無視されます。
指定されたデフォルトのディレクトリパス名の最後が “/” でないとき、指定されたデフォルトのディレクトリパス名の最後に ”/” を付与します。既定のデフォルトのディレクトリパス名はカレントディレクトリです。
指定した値の組み合わせで使用されるパス名を以下に示します。

OPEN文のFILE指定子の値	VE_FORT_DEFAULTFILE環境変数値	VE_FORT*n*環境変数値	使用されるパス名
指定なし	指定なし	指定なし	./fort.*n*
指定なし	指定なし	test.dat	./test.dat
指定なし	無効	/usr/tmp/t.dat	/usr/tmp/t.dat
指定なし	/tmp	指定なし	/tmp/fort.*n*
指定なし	/tmp	testdata	/tmp/testdata
指定なし	/usr	lib/testdata	/usr/lib/testdata
file.dat	/usr/group	無効	/usr/group/file.dat
/tmp/file.dat	無効	無効	/tmp/file.dat
file.dat	指定なし	無効	./file.dat

例

```
$ export VE_FORT_DEFAULTFILE=/foo/
```

VE_FORT_EXPRCW

拡張されたフォーマットを使用する書式なし入出力の外部ファイル装置を設定します。装置番号はコンマで区切って複数指定できます。2ギガバイトを超える記録を扱うことができます。

例

```
$ export VE_FORT_EXPRCW=10,11
```

VE_FORT_FILEINF

ファイル入出力解析情報を出力します。”YES”、または、”DETAIL”が設定されているとき、ファイルクローズ時に標準エラー出力にファイル入出力解析情報を出力します。この情報から、入出力に関して意図どおりの動作をしているか、あるいは、性能改善を検討しなければならない外部ファイル装置はないかなどの情報を得ることができます。表示項目(パスなど)にマルチバイト文字が含まれるとき、表示が正しくされない可能性があります。ファイル入出力解析情報の詳細については「付録 D ファイル入出力解析情報」を参照してください。

例

```
$ export VE_FORT_FILEINF=DETAIL
```

VE_FORT_FMT_NO_WRAP_MARGIN

並び出力の折り返しを制御します。"YES"を指定したとき、最大レコード長まで列の折り返しが無効になります。

例

```
$ export VE_FORT_FMT_NO_WRAP_MARGIN=YES
```

VE_FORT_FMTBUF[*n*]

入出力処理で使用するレコードバッファのサイズをバイト単位で指定します。このバッファは、装置ごとに用意します。使用するすべての装置や特定の装置に対して指定することができます。バッファのサイズは、135より小さくできません。135より小さい値を指定したとき、135になります。値が設定されていないとき、**OPEN**文に指定した**RECL**指定子の値となります。

環境変数**VE_FORT_FMTBUF**、**RECL**指定子どちらも設定されているとき、小さいほうの値となります。

標準入出力ファイル、標準エラー出力ファイルに対する本環境変数の指定は無視されます。

環境変数**VE_FORT_FMTBUF**と**VE_FORT_RECORDBUF**が指定されたとき、優先度は以下になります。

高	**VE_FORT_RECORDBUF***u*	・特定の外部ファイル装置指定
｜	**VE_FORT_FMTBUF***u*	・特定の外部ファイル装置指定
｜	**VE_FORT_RECORDBUF**	・すべての外部ファイル装置指定
低	**VE_FORT_FMTBUF**	・すべての外部ファイル装置指定

入出力処理で使用するレコードバッファの既定値は、以下になります。

- 標準入出力ファイル/ストリーム
  65536バイト
- 順編成ファイル
  65536バイト または **RECL**指定値
- 直編成ファイル
  RECL指定値

例(1) すべての外部ファイル装置に対する指定

```
$ export VE_FORT_FMTBUF=32768
```

例(2) 外部ファイル装置 1 に対する指定

```
$ export VE_FORT_FMTBUF1=60000
```

VE_FORT_FOR_PRINT

PRINT文、または、アスタリスク（*）指定した**WRITE**文が出力に使用するファイル名を指定します。環境変数が指定されていないとき、出力先は標準出力になります。

```
$ export VE_FORT_FOR_PRINT=FILENAME
```

備考　この環境変数を使用したときの論理装置番号は、使用していない論理装置番号が自動的に割り当てられます。エラーメッセージなどに含まれる論理装置番号は、マイナスを含む数値で表記されます。

VE_FORT_FOR_READ

論理装置番号の指定を省略、または、アスタリスク（*）指定した**READ**文が入力に使用するファイル名を指定します。環境変数が指定されていないとき、入力先は標準入力になります。

```
$ export VE_FORT_FOR_READ=FILENAME
```

備考　この環境変数を使用したときの論理装置番号は、使用していない論理装置番号が自動的に割り当てられます。エラーメッセージなどに含まれる論理装置番号は、マイナスを含む数値で表記されます。

VE_FORT_FOR_TYPE

TYPE文が出力に使用するファイル名を指定します。環境変数が指定されていないとき、出力先は標準出力になります。

```
$ export VE_FORT_FOR_TYPE=FILENAME
```

備考　この環境変数を使用したときの論理装置番号は、使用していない論理装置番号が自動的に割り当てられます。エラーメッセージなどに含まれる論理装置番号は、マイナスを含む数値で表記されます。

VE_FORT_MEM_BLOCKSIZE

プログラムの開始時にメモリの確保/解放を高速化するために確保されるメモリのブロックサイズをメガバイト単位で指定します。単位として"M"を使用するとメガバイト、"G"を使用するとギガバイトとして指定することができます。指定する値は0または2のべき乗数でなければなりません。値を設定しないとき、ブロックサイズは64メガバイトとなります。プログラムの開始時に各プロセスで初期領域として3個のメモリブロックが確保されます。

例 16 メガバイトを指定

```
$ export VE_FORT_MEM_BLOCKSIZE=16M
```

VE_FORT_NML_DELIM_BLANK

NAMELIST出力において、**DELIM**指定子が省略されたときの文字型配列の出力フォーマットを制御します。”YES”が設定されているとき、値を空白で区切って出力します。既定(”NO”)の動作では、値を連続で出力します。

DELIM指定子を指定したとき、本環境変数の指定は無視されます。

例

```
$ ./a.out
&NML
 C = abcdefg
 /
$ export VE_FORT_NML_DELIM_BLANK=YES
$ ./a.out
&NML C = a b c d e f g/
```

VE_FORT_NML_REPEAT_FORM

NAMELIST出力において、配列で同じ値が二つ以上連続するときの出力フォーマットを制御します。既定(”YES”)の動作では同じ値をまとめたフォーマット(連続する個数*値)で出力します。”NO”が指定されたとき、まとめずに出力します。

備考　バージョン3.0.7以前のコンパイラでは、常にまとめず出力されます。

例

```
$ ./a.out
&NML
 R = 3*1.0000000,    2.0000000,    3.0000000
 /
$ export VE_FORT_NML_REPEAT_FORM=NO
$ ./a.out
&NML R =   1.0000000   1.0000000   1.0000000   2.0000000   3.0000000/
```

VE_FORT_NORCW

制御レコードを一切付与しない形式の書式なし入出力の外部ファイル装置を設定します。装置番号はコンマで区切って複数指定できます。ストリームファイルの書式なし入出力と同じ

扱いとなり、標準のレコード形式よりも入出力が高速化されます。
ただし、以下の制約が付くことに注意してください。

- 入力する記録の長さは、出力した記録の長さと一致する必要があります。違っているときは、結果不正となります。
- **BACKSPACE**文は使用できません。

例

```
$ export VE_FORT_NORCW=10,11
```

VE_FORT_PARTRCW

制御レコードの形式を変更して扱う書式なし入出力の外部ファイル装置を設定します。装置番号はコンマで区切って複数指定できます。標準のレコード形式よりも入出力が高速化されます。
ただし、以下の制約が付くことに注意してください。

- 入力する記録の長さは、出力した記録の長さと一致する必要があります。違っているときは、実行時エラーになります。

例

```
$ export VE_FORT_PARTRCW=10,11
```

VE_FORT_PAUSE

PAUSE文の実行を制御します。"NO"を指定すると**PAUSE**文は無視されます。

例

```
$ export VE_FORT_PAUSE=NO
```

VE_FORT_RECLUNIT

書式なしファイルに対して**OPEN**文の実行時に扱う**RECL**指定子の値の単位を指定します。指定可能な単位は"BYTE"、または、"WORD"です。既定の単位は"BYTE"です。"WORD"は4バイト単位です。

例

```
$ export VE_FORT_RECLUNIT=WORD
```

VE_FORT_RECORDBUF[*n*]

入出力処理で使用するレコードバッファのサイズをバイト単位で指定します。レコードバッファは装置ごとに用意され、使用するすべての装置や特定の装置に対して指定できます。13

5より小さい値は設定できません。135より小さい値を指定したとき、135になります。値が指定されていないとき、**OPEN**文の**RECL**に指定した値となります。

環境変数**VE_FORT_RECORDBUF**、**RECL**指定子のどちらも指定されているとき、小さい方の値となります。

標準入出力ファイル、標準エラー出力ファイルに対する本環境変数の指定は無視されます。

環境変数**VE_FORT_FMTBUF**と**VE_FORT_RECORDBUF**が指定されたとき、優先度は以下になります。

高	**VE_FORT_RECORDBUF***u*	・特定の外部ファイル装置指定
｜	**VE_FORT_FMTBUF***u*	・特定の外部ファイル装置指定
｜	**VE_FORT_RECORDBUF**	・すべての外部ファイル装置指定
低	**VE_FORT_FMTBUF**	・すべての外部ファイル装置指定

入出力処理で使用するレコードバッファの既定値は、以下になります。

- 標準入出力ファイル/ストリーム
  65536バイト
- 順編成ファイル
  65536バイト または **RECL**指定値
- 直編成ファイル
  RECL指定値

例(1) すべての外部ファイル装置に対する指定

```
$ export VE_FORT_RECORDBUF=32768
```

例(2) 外部ファイル装置 1 に対する指定

```
$ export VE_FORT_RECORDBUF1=60000
```

VE_FORT_SETBUF[*n*]

入出力処理が準備するI/Oバッファのサイズをキロバイト単位で指定します。I/Oバッファは装置ごとに用意され、使用するすべての装置や特定の装置に対して指定できます。値0を指定したとき、バッファレスI/Oとなり、入出力処理でI/Oバッファが使用されません。ただし、バッファレスI/Oは、実行時性能を大幅に低下させるため、注意が必要です。

標準入力に対する指定は無視されます。標準出力、および、標準エラー出力には、バッファレスI/Oのみ指定できます。

値が設定されていないとき、I/Oバッファのサイズは、以下になります。

- 順編成ファイル/ストリームファイル

  レコードバッファの環境変数への設定値 ≦ 512キロバイト

  　512キロバイト

  レコードバッファの環境変数への設定値 > 512キロバイト

  　レコードバッファの環境変数への設定値をキロバイト単位に切り上げた値

- 直編成ファイル

  記録の長さ≦4,096バイト

  　4キロバイト

  記録の長さ＞2,048,000,000バイト

  　2,000,000キロバイト

  記録の長さが上記以外

  　記録の長さをキロバイト単位に切り上げた値

備考　　“レコードバッファの環境変数への設定値”は、環境変数**VE_FORT_FMTBUF**、または、**VE_FORT_RECORDBUF**に設定した値(バイト単位)となります。

例(1) すべての外部ファイル装置に対する指定

```
$ export VE_FORT_SETBUF=10
```

例(2) 外部ファイル装置 1 に対する指定

```
$ export VE_FORT_SETBUF1=20
```

VE_FORT_SUBRCW

レコード分割するフォーマットを使用する書式なし入出力の外部ファイル装置を設定します。装置番号はコンマで区切って複数指定できます。2ギガバイトを超える記録を扱うことができます。

環境変数**VE_FORT_EXPRCW**、**VE_FORT_NORCW**、または、**VE_FORT_PARTRCW**のいずれかに同じ装置番号を指定したとき、その装置番号に対する本環境変数の指定は無視されます。

例

```
$ export VE_FORT_SUBRCW=10,11
```

VE_FORT_UFMTADJUST[n]

書式なし入出力時のデータの長さ調整を制御します。使用するすべての装置や特定の装置に対して指定できます。この環境変数が設定されているとき、入出力並びの型とは異なる精度のデータの入出力が可能になります。以下の値が指定できます。値はコンマで区切って複数指定できます。

ALL

INT,LOG,REAL,DBLが指定されたものとみなします。

DBL

入出力並びの型が4倍精度実数型または4倍精度複素数型のとき、ファイル上の型は倍精度実数型または倍精度複素数型とみなして入出力を行います。

INT

入出力並びの型が8バイト整数型のとき、ファイル上の型は4バイト整数型とみなして入出力を行います。

LOG

入出力並びの型が8バイト論理型のとき、ファイル上の型は4バイト論理型とみなして入出力を行います。

NO

精度の調整は行いません。

REAL

入出力並びの型が倍精度実数型または倍精度複素数型のとき、ファイル上の型は単精度実数型または単精度複素数型とみなして入出力を行います。

例(1) 外部ファイル装置10に対してすべての型の調整を指定

```
$ export VE_FORT_UFMTADJUST10=ALL
```

例(2) 外部ファイル装置10以外の装置に対してすべての型の調整を指定

```
$ export VE_FORT_UFMTADJUST=ALL
$ export VE_FORT_UFMTADJUST10=NO
```

例(3) 外部ファイル装置10に対して実数型と複素数型のみ調整を指定

```
$ export VE_FORT_UFMTADJUST10=REAL,DBL
```

VE_FORT_UFMTENDIAN

ビッグエンディアンモードのファイルとして扱う書式なし入出力の外部ファイル装置を設定します。以下の形式を指定することができます。

ALL

すべての装置番号を対象とします。

装置番号 **|** ***装置番号,装置番号*** **|** ***装置番号-装置番号***

対象とする装置番号を指定します。

コンマ区切りで、対象とする複数の装置番号を指定します。

ハイフンで、対象とする連続する複数の装置番号を指定します。

big[:*装置番号*] | little[:*装置番号*]

エンディアン形式を指定します。

コロンの後に、対象とする装置番号を指定します。

;（セミコロン）

セミコロンの前に指定したエンディアン形式から除外するエンディアン形式と装置番号を指定します。

例(1) 外部ファイル装置10に対する指定

```
$ export VE_FORT_UFMTENDIAN=10
```

例(2) 外部ファイル装置10、11に対する指定

```
$ export VE_FORT_UFMTENDIAN=10,11
```

例(3) 外部ファイル装置10、11、12に対する指定

```
$ export VE_FORT_UFMTENDIAN=10-12
```

例(4) 外部ファイル装置10、11、12以外をビッグエンディアンとする指定

```
$ export VE_FORT_UFMTENDIAN=big;little:10-12
```

VE_FORT_UFMTENDIAN_NOVEC

ビッグエンディアンモードのファイルとして扱う書式なし入出力の外部ファイル装置のうち、エンディアン変換処理をスカラで行う外部ファイル装置を設定します。装置番号はコンマで区切って複数指定できます。

例

```
$ export VE_FORT_UFMTENDIAN_NOVEC=10,11
```

VE_FPE_ENABLE

実行時の浮動小数点例外を制御します。この変数が設定されているとき、特定の例外が有効になります。以下の値を指定できます。値はコンマで区切って複数指定できます。

DIV

ゼロ除算例外

FOF

浮動小数点オーバーフロー例外

FUF

浮動小数点アンダーフロー例外

INV

無効演算例外

INE

精度落ち例外

例

```
$ export VE_FPE_ENABLE=DIV
```

VE_INIT_HEAP

実行時にヒープに割り付ける領域を初期化する値を設定します。値を設定しないとき、初期化は行われません。以下の値を指定できます。

ZERO

0(ゼロ)で初期化します。

NAN

倍精度実数型のQuiet NaN(0x7fffffff7fffffff)で初期化します。

NANF

単精度実数型のQuiet NaN(0x7fffffff)で初期化します。

SNAN

倍精度実数型のSignaling NaN(0x7ff4000000000000)で初期化します。

SNANF

単精度実数型のSignaling NaN(0x7fa00000)で初期化します。

0x*XXXX*

最大16桁の16進数で指定された値で初期化します。指定された値が8桁以上の16進数であ

るとき、8バイト単位で初期化します。そうでないときは4バイト単位で初期化します。

例

```
$ export VE_INIT_HEAP=ZERO
```

VE_INIT_STACK

実行時にスタックに割り付ける変数を初期化する値を設定します。値が設定されなかったとき、0(ゼロ)で初期化されます。コンパイル時には**-minit-stack=runtime**を指定してください。

以下の値を指定できます。

ZERO

0(ゼロ)で初期化します。

NAN

倍精度実数型のQuiet NaN(0x7fffffff7fffffff)で初期化します。

NANF

単精度実数型のQuiet NaN(0x7fffffff)で初期化します。

SNAN

倍精度実数型のSignaling NaN(0x7ff4000000000000)で初期化します。

SNANF

倍精度実数型のSignaling NaN(0x7fa00000)で初期化します。

0x*XXXX*

最大16桁の16進数で指定された値で初期化します。指定された値が8桁以上の16進数であるとき、8バイト単位で初期化します。そうでないときは4バイト単位で初期化します。

例

```
$ nfort -minit-stack=runtime a.f90
$ export VE_INIT_STACK=SNAN
$ ./a.out
```

VE_LD_LIBRARY_PATH

実行時に動的リンカが共有ライブラリをサーチするディレクトリを指定します。複数指定するときはコロン(:)で区切って指定し、先に指定したディレクトリが優先されます。動的リンカは標準のディレクトリを自動的にサーチします。この環境変数は、標準以外のディレクトリを常にサーチしたい場合に設定します。例えば、Fortranコンパイラに付属しないOSSの共

有ライブラリのディレクトリを常にサーチしたいときに設定します。

例

```
$ export VE_LD_LIBRARY_PATH="${HOME}/lib:$VE_LD_LIBRARY_PATH"
```

VE_NODE_NUMBER

プログラムが実行される VEノード番号を指定します。

VE_PROGINF

“YES”、または、”DETAIL”が設定されているとき、プログラムの実行終了時に標準エラー出力にプログラム実行解析情報を出力します。プログラム実行解析情報の詳細については「PROGINF/FTRACE ユーザーズガイド」を参照してください。

VE_TRACEBACK

実行時に致命的エラーが発生したときのトレースバック情報の出力を制御します。トレースバック情報を出力するとき、プログラムのコンパイル時とリンク時に**-traceback**を指定する必要があります。この変数に”FULL”、または、”ALL”が設定されたとき、環境変数**VE_TRACEBACK_DEPTH**で指定した値を上限にトレースバック情報が出力されます。その他の値が設定されたとき、エラーが発生した手続のトレースバック情報のみ出力されます。値が設定されなかったとき、トレースバック情報は出力されません。

トレースバック情報に表示されたアドレス情報から、致命的エラーの発生個所がわかります。

例

```
$ nfort -traceback a.f90
...
$ export VE_TRACEBACK=FULL
$ export VE_FPE_ENABLE=DIV
$ ./a.out
Runtime Error: Divide by zero at 0x600000000cc0
[ 1] Called from 0x7f5ca0062f60
[ 2] Called from 0x600000000b70
Floating point exception
```

naddr2lineコマンドを使って表示アドレスのファイル位置を求めます。例ではa.f90の3行目付近で”Divide by zero”の例外が発生したことがわかります。

例

```
$ naddr2line -e ./a.out -a 0x600000000cc0
```

```
0x0000600000000cc0
a.f90:3
```

また、コンパイル時とリンク時に**-traceback=verbose**を指定して作成したプログラムの実行時に、環境変数**VE_TRACEBACK**に"VERBOSE"を指定すると、ファイル名や行番号情報が出力されます。

例

```
$ export VE_TRACEBACK=VERBOSE
$ ./a.out
Runtime Error: Overflow at 0x600008001088
[ 0] 0x600008001088 below_           below.f90:3
[ 1] 0x600018001168 out_             out.f90:3
[ 2] 0x600020001168 watch_           watch.f90:3
[ 3] 0x600010001168 hey_             hey.f90:3
[ 4] 0x60000001cab8 MAIN__           ovf.f90:5
```

VE_TRACEBACK_DEPTH

環境変数**VE_TRACEBACK**にてトレースバック情報を出力する際に、トレースバック情報の上限の値を設定します。値を指定しない場合は、"50"が設定されます。この環境変数に"0"を設定した場合、上限なしとなります。

第3章　　コンパイラオプション

本章では、Fortranコンパイラのコンパイラオプションについて説明します。
コンパイラオプションは、次のカテゴリに分類できます。

- 全体オプション
  コンパイラシステム全体を制御するオプションです。
- 最適化・ベクトル化オプション
  最適化、および、ベクトル化を制御するためのコンパイラオプションです。
- 並列化オプション
  自動並列化機能を制御するためのコンパイラオプションです。
- インライン展開オプション
  インライン展開機能を制御するためのコンパイラオプションです。
- コード生成オプション
  性能測定やスタック領域の初期化を行う付加的なコードを生成するためのコンパイラオプションです。
- デバッグオプション
  デバッグをサポートするオブジェクトファイルを生成するためのコンパイラオプションです。
- 言語仕様制御オプション
  Fortranの言語仕様を制御するためのコンパイラオプションです。
- メッセージオプション
  コンパイルメッセージの出力を制御するためのコンパイラオプションです。
- リスト出力オプション
  コンパイルリストの出力を制御するためのコンパイラオプションです。
- プリプロセッサオプション
  プリプロセスを制御するためのコンパイラオプションです。
- アセンブラオプション
  コンパイラのコマンドラインで指定できるアセンブラを制御するためのオプションです。
- リンカオプション
  コンパイラのコマンドラインで指定できるリンカを制御するためのオプションです。

- ディレクトリオプション
  コンパイラシステムを構成するコマンドや、リンカ、システムヘッダファイル、システムライブラリのあるディレクトリを指定するためのコンパイラオプションです。

3.1　全体オプション

-S

コンパイルのみ行いアセンブラソースを出力する。

-c

コンパイルのみ行いオブジェクトファイルを出力する。

-cf=*conf*

コンパイル、リンクするとき*conf*で指定されたコンフィギュレーションファイルを適用する。

-**clear**

本オプションより前に指定されたコンパイラオプション、ファイルをすべて無視する。

-**fsyntax-only**

文法チェックのみ行う。

-o *ファイル名*

出力するプリプロセス結果、アセンブラソース、オブジェクトファイル、実行ファイルの名前を指定する。複数のソースファイルを指定し、**-S**、**-c**、または、**-E**を指定するとき、指定できない。

-x *language*

ファイルの言語の種類を指定する。このオプションはファイル拡張子に従う自動選択より優先される。仕様は、コマンドライン上のこのオプションに続く(もし次の**-x**があればそれまで)すべてのファイルに適用される。*language*に指定できる言語の種類は以下である。

f77

固定形式のFortranソースファイルとしてコンパイルする。

f77-cpp-input

プリプロセスし、固定形式のFortranソースファイルとしてコンパイルする。

f95

自由形式のFortranソースファイルとしてコンパイルする。

f95-cpp-input

プリプロセスし、自由形式のFortranソースファイルとしてコンパイルする。

assembler

アセンブラソースファイルとしてアセンブルする。

assembler-with-cpp

プリプロセスし、プリプロセスされたファイルをアセンブルする。

@*file-name*

コンパイラオプションを*file-name*で指定されたファイルから読み込む。読み込んだコンパイラオプションは、***@file-name***を指定した位置に挿入される。

3.2　最適化・ベクトル化オプション

-O[*n*]

最適化レベル(*n*)を指定する。*n*に指定できる値は以下である。

4

Fortran言語仕様を逸脱した副作用を伴う最大限の最適化・自動ベクトル化を適用する。

3

副作用を伴う最適化・自動ベクトル化、および、多重ループの最適化を適用する。

2

副作用を伴う最適化・自動ベクトル化を適用する。(既定値)

1

副作用を伴わない最適化・自動ベクトル化を適用する。

0

最適化、自動ベクトル化、並列化、インライン展開を適用しない。

-fargument-alias

引数が互いに、または、非局所オブジェクトへのエイリアスであると仮定して最適化・自動ベクトル化を適用する。

-fargument-noalias

引数が互いに、または、非局所オブジェクトへのエイリアスでないと仮定して最適化・自動ベクトル化を適用する。(既定値)

-f[no-]associative-math

演算順序の変更を伴う最適化・自動ベクトル化を適用する[しない]。

(既定値: -fassociative-math)

-f[no-]aggressive-associative-math

より激しく演算順序の変更を伴う最適化・自動ベクトル化を適用する[しない]。

(既定値: -fno-aggressive-associative-math)

-f[no-]assume-contiguous

形状引継ぎ配列を連続であると仮定し、最適化・自動ベクトル化を適用する[しない]。

(既定値: **-fno-assume-contiguous**)

-f[no-]copyin-intent-out

INTENT(OUT)属性を持つ仮引数に対応する実引数のcopy-inの処理を省略する[しない]。(既定値: **-fcopyin-intent-out)**

-f[no-]cse-after-vectorization

ベクトル化後にも共通式削除の最適化を適用する[しない]。

(既定値: **-fno-cse-after-vectorization**)

-f[no-]fast-formatted-io

書式付き入出力の高速バージョンを使用する[しない]。

(既定値: -ffast-formatted-io)

-f[no-]fast-math

ベクトル化されたループ外でスカラ高速バージョンの数学関数を使用する[しない]。

(既定値: -ffast-math)

-f[no-]ignore-asynchronous

ASYNCHRONOUS属性を無視する[しない]。(既定値: **-fno-ignore-asynchronous**)

-f[no-]ignore-induction-variable-overflow

最適化を行うためにインダクション変数のオーバーフローを無視する[しない]。

(既定値: **-fno-ignore-induction-variable-overflow**)

-f[no-]ignore-volatile

VOLATILE属性を無視する[しない]。(既定値: **-fno-ignore-volatile**)

-fivdep

すべてのループに**ivdep**指示行を指定する。

-fivdep-omp-worksharing-loop

OpenMP機能により並列化されたループに**ivdep**指示行を指定する。ただし、**safelen**、**simdlen**句の指定された**simd**が指定されたループを除く。

-f[no-]loop-collapse

多重ループの一重化を許可する[しない]。**-O***n*（*n*=2,3,4)が有効でなければならない。(既定値: **-fno-loop-collapse**)

-floop-count=*n*

繰返し数に合った最適化を行うため、コンパイル時に繰返し数を算定できないループの繰返し数を*n*回と仮定する。(既定値: **-floop-count**=5000)

-f[no-]loop-fusion

ループ融合を許可する[しない]。**-O***n*（*n*=2,3,4)が有効でなければならない。(既定値: **-fno-loop-fusion**)

-f[no-]loop-interchange

ループ入れ換えを許可する[しない]。**-O***n*（*n*=2,3,4)が有効でなければならない。(既定値: **-fno-loop-interchange**)

-f[no-]loop-normalize

ループの正規化を許可する[しない]。このとき、コンパイラは、ループの繰返し数がループ本体で変更されないと仮定する。(既定値: **-fno-loop-normalize**)

-f[no-]loop-split

外部手続の呼出し前後でのループ分割を許可する[しない]。**-O***n*（*n*=2,3,4)が有効でなければならない。(既定値: **-fno-loop-split**)

-f[no-]loop-strip-mine

ループストリップマイニングを許可する[しない]。**-O***n*（n=2,3,4)が有効でなければならない。(既定値: **-fno-loop-strip-mine**)

-f[no-]loop-unroll

ループアンローリングを許可する[しない]。**-O***n*（*n*=2,3,4)が有効でなければならない。(既定値: **-floop-unroll**)

-floop-unroll-complete=*m*

ループの繰返し数が定数でコンパイル時に算定でき*m*回以下であるとき、そのループのループ展開(完全ループアンローリング)を許可する。**-O***n*（*n*=2,3,4)が有効でなければならない。(既定値: **-floop-unroll-complete**=4)

備考

別名オプションとして**-floop-unroll-completely**=*m*も使用できる。

-floop-unroll-complete-nest=*m*

DOループをループ展開(完全ループアンローリング)する際、最内側から数えてm重目のループまで展開する。ただし、最外側のDOループを除く。

配列式に対応するループをループ展開(完全ループアンローリング)する際、1次元目から*m*次元目まで展開する。(既定値: **-floop-unroll-complete-nest**=3)

備考

別名オプションとして-**floop-unroll-completely-nest**=*m*も使用できる。

-floop-unroll-max-times=*n*

最大アンロール段数を*n*段とする。本オプションが有効でないとき、コンパイラは適切なアンロール段数を自動的に決定する。

-f[no-]matrix-multiply

行列積の高速なベクトルライブラリ呼出しへの変換を許可する[しない]。**-O***n*（*n*=2,3,4)、かつ、**-fassociative-math**が有効でなければならない。

(既定値: **-fno-matrix-multiply**)

-f[no-]move-loop-invariants

条件下の不変式のループ外への移動を許可する[しない]。

(既定値: **-fmove-loop-invariants**)

-f[no-]move-loop-invariants-if

ループ内不変のIF構造のループ外への移動を許可する[しない]。-**O***n*（*n*=2,3,4)が有効でなければならない。(既定値: **-fno-move-loop-invariants-if**)

-f[no-]move-loop-invariants-unsafe

副作用を伴う可能性のあるコードのループ外への移動を許可する[しない]。副作用を伴う可能性のあるコードの例は以下のとおり。

- 除算
- 1バイト、または、2バイトの記憶領域へのメモリ参照

(既定値: **-fno-move-loop-invariants-unsafe)**

-f[no-]move-nested-loop-invariants-outer

多重ループ内の不変式の外側ループ外への移動を許可する[しない]。このオプションが指定されたとき、不変式は直前のループ前に移動される。

(既定値: -**fmove-nested-loop-invariants-outer**)

-fnamed-alias

ポインタの指示先がエイリアスをもつと仮定して最適化・自動ベクトル化を適用する。

-fnamed-noalias

ポインタの指示先がエイリアスをもたないと仮定して最適化・自動ベクトル化を適用する。(既定値)

-fnamed-noalias-aggressive

ポインタの指示先がエイリアスをもたないと仮定してより激しく最適化・自動ベクトル化を適用する。

-f[no-]outerloop-unroll

外側ループのアンローリングを許可する[しない]。**-O***n* (*n*=2,3,4)が有効でなければならない。(既定値: **-fno-outerloop-unroll**)

-fouterloop-unroll-max-size=*n*

外側ループアンローリングの対象とするループの最大サイズを*n*とする。
(既定値: **-fouterloop-unroll-max-size=4**)

-fouterloop-unroll-max-times=*n*

外側ループの最大アンロール段数を*n*段とする。*n*は2のべき乗数でなければならない。本オプションが有効でないとき、コンパイラは適切なアンロール段数を自動的に決定する。

-f[no-]precise-math

べき乗算のベクトル演算において、べき指数(exponent)が整数値であるときの計算精度を高めたべき乗算アルゴリズムを適用する[しない]。計算精度は高まるが、計算速度は低下する。(既定値: **-fno-precise-math**)

-f[no-]reciprocal-math

“x/y”の”x * (1/y)”への変更を許可する[しない]。(既定値: **-freciprocal-math**)

-f[no-]reorder-logical-expression

論理式の各項を任意の順番で評価することを許可する[しない]。左から右へと順に評価する。(既定値: **-freorder-logical-expression**)

-f[no-]replace-loop-equation

ループ繰返し条件の「!=」、「==」、「.NE.」、「.EQ.」を「<=」、または、「>=」に置換する[しない]。(既定値: **-fno-replace-loop-equation**)

-f[no-]replace-matmul-to-matrix-multiply

MATMULの手続呼出しのベクトル行列積ライブラリ呼出しへの置換を許可する[しない]。
(既定値: **-freplace-matmul-to-matrix-multiply**)

-m[no-]array-io

入出力文中の配列式、および、DO形並びの最適化を許可する[しない]。

(既定値: -marray-io)

-m[no-]list-vector

同一の非線形添字をもつ同一配列が左辺と右辺に現れる代入文のベクトル化を許可する[しない]。(既定値: **-mno-list-vector**)

-mretain-*keyword*

優先的にLLC(Last-Level Cache)に保持する配列、または、ポインタのベクトルメモリアクセスの種類を指定する。*keyword*に指定できるメモリアクセスの種類は以下である。

all

すべてのベクトルメモリアクセス。(既定値)

list-vector

非線形添字をもつベクトルメモリアクセス。

none

優先度は指定しない。

-msched-*keyword*

命令の並べ換えのレベルを指定する。*keyword*で指定できる命令の並べ換えのレベルは以下である。

none

命令の並べ換えを行わない。

insns

基本ブロック内での命令の並べ換えを行う。

block

基本ブロック内での命令の並べ換えを行う。**-msched-insns**より命令の並べ換えを適用する範囲を広くし、より強力に命令を並べ換える。(既定値)

interblock

基本ブロックをまたがって命令の並べ換えを行う。

-mstack-arrays

自動配列、作業配列をスタックに割り付ける。(既定値)

-mno-stack-arrays

自動配列、作業配列をヒープ領域に割り付ける。

-muse-mmap

ALLOCATE文/**DEALLOCATE**文でのメモリの確保/解放に**mmap**/**munmap**関数を使用する。

-m[no-]vector

自動ベクトル化を適用する[しない]。(既定値: **-mvector**)

-m[no-]vector-advance-gather

ベクトル収集命令をループ本体の前方に移動する[しない]。前方に移動することにより、後続の演算命令などとオーバーラップさせ、計算時間を短縮できることがある。

(既定値: **-mvector-advance-gather**)

-mvector-advance-gather-limit=*n*

前方に移動するベクトル収集命令の個数の上限を指定する。

(既定値: **-mvector-advance-gather-limit=**56)

-m[no-]vector-dependency-test

依存関係の判定を用いた条件ベクトル化を適用する[しない]。**-O***n*(*n*=2,3,4)が有効でなければならない。(既定値: **-mvector-dependency-test**)

-m[no-]vector-floating-divide-instruction

ベクトル浮動小数点除算において、ベクトル浮動小数点除算命令を使用する[しない]。使用しないとき、逆数近似命令を用いてベクトル浮動小数点除算する。

(既定値: **-mno-vector-floating-divide-instruction**)

-m[no-]vector-fma

ベクトル積和演算命令の使用を許可する[しない]。(既定値: **-mvector-fma**)

-mvector-intrinsic-check

ベクトル化された部分から呼び出された数学関数、組込み演算の引数の値の範囲を実行時に検査する。 本機能の対象となる数学関数、組込み演算は以下のとおり。(ただし、引数は倍精度実数型のみで同型の引数をもつ個別名も対象となる。)

ACOS、**ACOSH**、**ASIN**、**ATAN**、**ATAN2**、**ATANH**、**COS**、**COSD**、**COSH**、**COTAN**、**EXP**、**EXP10**、**EXP2**、**EXPC**、**FACT**、**LOG10**、**LOG2**、**LOG**、**SIN**、**SIND**、**SINH**、**SQRT**、**TAN**、**TANH**、べき乗

-m[no-]vector-iteration

ベクトル漸化式命令の使用を許可する[しない]。(既定値: **-mvector-iteration**)

-m[no-]vector-iteration-unsafe

結果不正を伴うことがあるベクトル漸化式命令の使用を許可する[しない]。

(既定値: **-mvector-iteration-unsafe**)

-m[no-]vector-loop-count-test

ループの繰返し数判定を用いた条件ベクトル化を適用する[しない]。**-O***n*(*n*=2,3,4)が有効でなければならない。(既定値: **-mno-vector-loop-count-test**)

-m[no-]vector-low-precise-divide-function

低精度のベクトル浮動小数点数除算を使用する[しない]。通常精度版と比較して高速に処理されるが、除算結果の仮数部に最大1ビットの誤差が含まれることがある。

(既定値: **-mno-vector-low-precise-divide-function**)

-m[no-]vector-merge-conditional

THEN節、ELSE IF節、ELSE節のベクトルロードとストアをマージすることを許可する[しない]。(既定値: **-mno-vector-merge-conditional**)

-m[no-]vector-packed

パックドベクトル命令の使用を許可する[しない]。(既定値: **-mno-vector-packed**)

-m[no-]vector-power-to-explog

ベクトル化されたループ中のべき乗算R1**R2をEXP(R2*LOG(R1))の呼び出しに置き換えることを許可する[しない]。R1、R2は単精度、または、倍精度実数型である。置き換え前のべき乗算に比べて実行時間は短縮されるが、計算結果が誤差レベルで変わることがある。(既定値: **-mno-vector-power-to-explog**)

-m[no-]vector-power-to-sqrt

ベクトル化されたループ中のべき乗算R1**R2において、R2の値が0.5、または、1.0/3.0などの特別な値のとき、べき乗算をSQRT、CBRTを使用した計算に置き換えることを許可する[しない]。R1、R2は単精度、または、倍精度実数型である。SQRT、CBRTを使用した計算のとき、べき乗算に比べて実行時間は短縮されるが、計算結果が誤差レベルで変わることがある。(既定値: **-mvector-power-to-sqrt**)

-m[no-]vector-reduction

ベクトルリダクション命令の使用を許可する[しない]。(既定値: **-mvector-reduction**)

-m[no-]vector-shortloop-reduction

リダクション演算を含むループにおいてループの繰返し数判定を用いた条件ベクトル化を適用する[しない]。**-O***n*(*n*=2,3,4)が有効でなければならない。

(既定値: -**mno-vector-shortloop-reduction**)

-m[no-]vector-sqrt-instruction

ベクトルSQRTにおいて、ベクトルSQRT命令を使用する[しない]。使用しないとき、逆数

近似命令を用いてベクトルSQRT演算する。

(既定値: -mno-vector-sqrt-instruction)

-mvector-threshold=*n*

ベクトル化の対象とするループの最小繰返し数(*n*)を指定する。

(既定値: **-mvector-threshold**=5)

-mwork-vector-kind=none

部分ベクトル化(ループ分割によるベクトル化)を許可しない。

3.3　並列化オプション

-fopenmp

OpenMP機能を使用する。**-pthread**は暗黙的に有効となる。

-m[no-]create-threads-at-startup

最初に実行されるParallelリージョン、または、並列ループの開始時に、OpenMP・自動並列化のためのスレッドを生成する[しない]。既定値では、、プログラムの実行開始時にスレッドを生成する。(既定値: **-mcreate-threads-at-startup**)

備考

本オプションを指定するとき、**-static-nec**、または、**-static**を指定すること。

-mparallel

自動並列化を適用する。**-pthread**は暗黙的に有効となる。

-mparallel-innerloop

内側ループの並列化を許可する。

-m[no-]parallel-omp-routine

-fopenmpと**-mparallel**が同時に指定されたとき、OpenMPディレクティブを含む手続を自動並列化する[しない]。(既定値: **-mparallel-omp-routine**)

-mparallel-outerloop-strip-mine

外側ループストリップマイニングで得られる多重ループの並列化を許可する。

-mparallel-sections

並列化されたセクションの生成を許可する。

-mparallel-threshold=*n*

自動並列化を適用するループを選択する際のしきい値(n)を指定する。

しきい値以上の作業量を持つループに自動並列化を適用する。

(既定値: **-mparallel-threshold**=2000)

-mschedule-dynamic

-mschedule-runtime

-mschedule-static

-mschedule-chunk-size=*n*

OpenMP並列化、自動並列化において、**schedule**句によりスレッドのスケジューリング種別、サイズの指定が行われなかった場合のスケジューリング種別、サイズを指定する。

-pthread

pthreadライブラリを用いたマルチスレッドのサポートを有効にする。

3.4　インライン展開オプション

-**finline-abort-at-error**

サーチ対象のソースファイル内で定義された手続の生成に失敗した際、コンパイルを終了する。本オプションが指定されなかったとき、エラーのあったファイルをサーチせず、コンパイルを継続する。(既定値: **-fno-inline-abort-at-error**)

-f[no-]inline-copy-arguments

自動インライン展開する手続の実引数のコピーを生成する[しない]。コピーを生成しないとき、手続の仮引数が対応する実引数に置き換えられる。

(既定値: -f**inline-copy-arguments**)

-finline-directory=*ディレクトリ名*

インライン展開する手続をサーチするとき、指定されたディレクトリにあるすべてのソースファイルをサーチする。複数指定するときにコロン(:)で区切って指定する。

-fno-inline-directory=*ディレクトリ名*

インライン展開する手続をサーチするとき、指定されたディレクトリにあるすべてのソースファイルをサーチしない。複数指定するときにコロン(:)で区切って指定する。**-finline-file**、または、**-finline-directory**でサーチ対象となったソースファイルをサーチしないとき指定する。

-finline-file=*文字列*

インライン展開する手続をサーチするとき、指定されたソースファイルをサーチする。複数指定するときにコロン(:)で区切って指定する。**all**が指定されたとき、コマンドラインで指定されたコンパイル対象のすべてのソースファイルもサーチする。

-fno-inline-file=*文字列*

インライン展開する手続をサーチするとき、指定されたソースファイルをサーチしない。複数指定するときにコロン(:)で区切って指定する。**-finline-file**、または、**-finline-directory**でサーチ対象となったソースファイルをサーチしないとき指定する。

-finline-functions

自動インライン展開を適用する。

-finline-max-depth=*n*

自動インライン展開する手続の深さを指定する。 (既定値: **-finline-max-depth**=2)

-finline-max-function-size=*n*

自動インライン展開する手続の大きさ(手続の中間言語の量)を指定する。
(既定値: **-finline-max-function-size**=50)

-finline-max-times=*n*

自動インライン展開後の手続の大きさ(手続の中間言語の量)の上限を「インライン展開前の手続の大きさ×*n*」とする。 (既定値: **-finline-max-times**=6)

-f[no-]inline-suppress-diagnostics

サーチ対象のソースファイル内で定義された手続の生成に失敗した際、エラーメッセージを出力しない[する]。-finline-file、または、-finline-directoryでサーチ対象となったソースファイルの内、正常にサーチされているものを確認したいときに**-fno-inline-suppress-diagnosticsを**指定する。
(既定値: **-finline-suppress-diagnostics**)

-mgenerate-il-file

クロスファイルインライニングのためのILファイルをカレントディレクトリに出力する。ファイル名は、「ソースファイル名.fil」である。**-o** ファイル名で名前を変更できる。

-mread-il-file *ILファイル名*

インライン展開する手続をサーチするとき、指定されたILファイルをサーチする。複数指定するときにコロン(:)で区切って指定する。**-finline-file**、**-finline-directory**、または、**-mgenerate-il-file**のいずれかと同時に指定されたとき無視される。

3.5　コード生成オプション

-finstrument-functions

手続の入口と出口にトレース用の関数呼出しを挿入する。挿入する関数は以下。

```
void __cyg_profile_func_enter(void *this_fn, void *call_site);
void __cyg_profile_func_exit(void *this_fn,void *call_site);
```

-fpic

-fPIC

位置独立コードを生成する。

-ftrace

ftrace機能用のオブジェクトファイル、および、実行ファイルを生成する。

(既定値: **-no-ftrace**)

-**p**

-pg

プロファイラ情報(ngprof)を出力するオブジェクトファイル、実行ファイルを生成する。

-[no-]proginf

PROGINF機能用の実行ファイルを生成する[しない]。(既定値: **-proginf**)

3.6　デバッグオプション

-fbounds-check

-**fcheck=bounds**と同じ。

-fcheck=*keyword*

*keyword*に応じた実行時のチェックを有効にする。複数指定するときにはコロン(:)で区切って指定する。例えば「-fcheck=all:noalias」のように指定するとalias以外の全てのチェックを有効にすることができる。

all

下記すべてのチェックを有効にする。

[no]alias

仮引数のエリアスへの代入のチェックを有効にする[無効にする]。

[no]bits

ビットintrinsicな引数のチェックを有効にする[無効にする]。

[no]bounds

配列の上下限のチェックを有効にする[無効にする]。

[no]dangling

不正なポインタのチェックを有効にする[無効にする]。

[no]do

DOループのステップ値が0かどうかのチェックを有効にする[無効にする]。

[no]iovf

整数オーバーフローのチェックを有効にする[無効にする]。

[no]pointer

ポインタ参照のチェックを有効にする[無効にする]。

[no]present

省略可能な引数の参照のチェックを有効にする[無効にする]。

[no]recursion

不正な再帰呼出しのチェックを有効にする[無効にする]。

-g

DWARFフォーマットでデバッグ情報を生成する。

-minit-stack=*value*

実行時にスタックに割り付ける領域を指定された値で初期化する。*value*に指定できる値は以下である。

zero

0(ゼロ)で初期化する。

nan

倍精度浮動小数点型のQuiet NaN(0x7fffffff7fffffff)で初期化する。

nanf

単精度浮動小数点型のQuiet NaN(0x7fffffff)で初期化する。

snan

倍精度浮動小数点型のSignaling NaN(0x7ff4000000000000)で初期化する。

snanf

単精度浮動小数点型のSignaling NaN(0x7fa00000)で初期化する。

runtime

環境変数**VE_INIT_STACK**に設定された値で初期化する。

0x*XXXX*

最大16桁の16進数で指定された値で初期化する。指定された値が8桁以上の16進数であるとき、8バイト単位で初期化する。そうでないときは4バイト単位で初期化する。

-mmemory-trace

メモリ領域の確保/解放のトレース情報を出力するためのコードを生成する。

-mmemory-trace-full

ソースコードの情報とともにメモリ領域の確保/解放のトレース情報を出力するためのコードを生成する。

-traceback[=verbose]

実行時に環境変数**VE_TRACEBACK**がセットされているとき、トレースバック情報を出力するオブジェクトファイル、実行ファイルを生成する。

verboseを指定した場合、トレースバック情報を出力する際に、ファイル名や行番号情報を出力するための情報を追加したオブジェクトファイル、実行ファイルを生成する。トレースバック情報を出力する際に、これらの情報を出力するためには、実行時に環境変数**VE_TRACEBACK=VERBOSE**を指定する必要がある。

3.7　言語仕様制御オプション

-bss

局所変数を.bssセクションに割り付ける。

-fdefault-integer=*n*

基本整数型、および、基本論理型の大きさをバイトサイズで指定する。*n*は4、または、8でなければならない。 (既定値: **-fdefault-integer**=4)

結果の型や引数の型が基本整数型、または、基本論理型である組込み手続にも影響し、その結果や引数には以下のいずれかの型を指定しなければならない。

- 基本整数型

n=4の場合、基本整数型、または、**INTEGER**(4)

n=8の場合、基本整数型、または、**INTEGER**(8)

- 基本論理型

n=4の場合、基本論理型、または、**LOGICAL**(4)

n=8の場合、基本論理型、または、**LOGICAL**(8)

-fdefault-double=*n*

DOUBLE PRECISION型、および、**DOUBLE COMPLEX**型の実部、虚部の大きさをバ

イトサイズで指定する。*n*は8、または、16でなければならない。 (既定値: **-fdefault-double**=8)

-fdefault-real=*n*

基本実数型、および、基本複素数型の実部、虚部の大きさをバイトサイズで指定する。*n*は4、または、8でなければならない。 (既定値: **-fdefault-real**=4)

結果の型や引数の型が基本実数型、または、基本複素数型である組込み手続にも影響し、その結果や引数には以下のいずれかの型を指定しなければならない。

- 基本実数型

n=4の場合、基本実数型、または、**REAL**(4)

n=8の場合、基本実数型、または、**REAL**(8)

- 基本複素数型

n=4の場合、基本複素数型、または、**COMPLEX**(4)

n=8の場合、基本複素数型、または、**COMPLEX**(8)

-fextend-source

固定形式のソースファイルに記述できる１行の最大文字数を2048文字に拡張する。

-ffree-form

ソースファイルが自由形式であることを指定する。ソースファイルの拡張子が**.f90**、**.f95**、**.f03**、**.F90**、**.F95**、**.F03**のときの既定値。

-ffixed-form

ソースファイルが固定形式であることを指定する。ソースファイルの拡張子が**.f**、**.F**のときの既定値。

-ff90-sign

組込み関数SIGNの第二引数において、正の実数0.0と負の実数-0.0を区別しない。第二引数が負の実数-0.0のとき、結果の値の符号は正となる。

-fmax-continuation-lines=*n*

継続行の行数の上限を指定する。*n*は511以上、かつ、4095以下でなければならない。

(既定値: **-fmax-continuation-lines**=1023)

-fno-realloc-lhs

-fno-realloc-lhs-arrayと-fno-realloc-lhs-scalarを同時に有効にする[しない]。

(既定値: **-frealloc-lhs**)

-fno-realloc-lhs-array

Fortran 2003言語仕様では、代入文の左辺が割り付けられていない、または、右辺と異

なる形状で割り付けられた割付け配列のとき、右辺の形状に再割り当てするよう定められている。

この仕様を無視する。このとき、左辺の割付け配列が、右辺の形状通りに割り付けられていなければプログラムは正しく動作しない。

(既定値: **-frealloc-lhs-array**)

-fno-realloc-lhs-scalar

Fortran 2003 言語仕様では、代入文の左辺が割り付けられていないとき、再割り当てするよう定められている。

この仕様を無視する。このとき、左辺の割付け変数が割り付けられていなければプログラムは正しく動作しない。

(既定値: **-frealloc-lhs-scalar**)

-masync-io

READ文または**WRITE**文に**ASYNCHRONOUS**=”YES”が指定されたとき、非同期でデータを転送することを指定する。書式なし入出力で非同期入出力が有効になる。

-save

各プログラム単位において(RECURSIVEであるものは例外とする)、**SAVE**文がすべての局所変数に指定されているものとする。

-std=*standard*

Fortran言語仕様を指定する。*standard*はf95、f2003、f2008、f2018のいずれかである。これは拡張仕様に関するメッセージ出力にのみ影響します。

(既定値: **-std**=f2008)

-use *module*

*module*で指定したモジュールファイル内のすべての公開要素を引用する。*module*はコンマで区切って複数指定できる。

3.8　メッセージオプション

-Wall

すべての警告レベルの文法診断メッセージを出力する。

-**Werror**

すべての警告レベルの文法診断メッセージを致命的エラーとして扱う。

-Wextension

Fortran拡張言語仕様に対する警告メッセージを出力する。

-Wobsolescent

旧Fortran言語仕様に対する警告メッセージを出力する。

-Woverflow

整数オーバーフローに対する警告メッセージを出力する。

-Woverflow-errors

整数オーバーフローに対するエラーを出力し、コンパイルを中断する。

-fdiag-inline=*n*

自動インライン展開に関する診断メッセージのレベル*n*を指定する。 (0: 出力しない、1: 通常レベル、2: 詳細レベル) (既定値: **-fdiag-inline**=1)

-fdiag-parallel=*n*

自動並列化に関する診断メッセージのレベル*n*を指定する。 (0: 出力しない、1: 通常レベル、2: 詳細レベル) (既定値: **-fdiag-parallel**=1)

-fdiag-vector=*n*

ベクトル化に関する診断メッセージのレベル*n*を指定する。 (0: 出力しない、1: 通常レベル、2: 詳細レベル) (既定値: **-fdiag-vector**=1)

-pedantic-errors

言語仕様から逸脱している場合に致命的エラーを出力する。

-w

すべての警告メッセージを抑止する。

3.9　リスト出力オプション

-report-file=*ファイル名*

既定のファイルの代わりに指定されたファイルにリスト結果を出力する。

-report-append-mode

「上書きモード」の代わりに「追加モード」で出力ファイルを開く。このオプションは**-report-file**の指定が無いと使用できない。

-report-all

コード生成リスト、診断メッセージリスト、編集リスト、インラインリスト、オプションリスト、ベクトルリストを出力する。

-[no-]report-cg

コード生成モジュールの最適化情報リストを出力する[しない]。

(既定値: **-no-report-cg**)

-[no-]report-diagnostics

診断メッセージリストを出力する[しない]。

(既定値: **-no-report-diagnostics**)

-[no-]report-format

編集リストを出力する[しない]。

(既定値: **-no-report-format**)

-[no-]report-inline

インライン展開モジュールの最適化情報リストを出力する[しない]。

(既定値: **-no-report-inline**)

-[no-]report-option

オプションリストを出力する[しない]。

(既定値: **-no-report-option**)

-[no-]report-vector

ベクトル化モジュールの最適化情報リストを出力する[しない]。

(既定値: **-no-report-vector**)

3.10 プリプロセッサオプション

-D*macro*[=*defn*]

#defineと同様に*macro*を値*defn*で定義する。=*defn*が指定されなかったとき*macro*は10進整数1に定義される。

-E

プリプロセスのみ行いプリプロセス結果を標準出力に出力する。

-dM

プリプロセス結果の代わりに、**#define**、**-D**で定義されたマクロ定義、および、定義済みマクロを標準出力に出力する。**-E**と同時に指定されていないとき無視される。

-fpp

ソースファイルをコンパイル前に**fpp**でプリプロセスすることを指定する。ファイルの拡張子が**.F**、**.F90**、**.F95**、**.F03**のときの既定値。

-nofpp

ソースファイルをコンパイル前に**fpp**でプリプロセスしないことを指定する。ファイルの

拡張子が**.f**、**.f90**、**.f95**、.**f03**のときの既定値。

-fpp-name=*name*

デフォルトで使用されるFortranプリプロセッサの代わりのFortranプリプロセッサ名を指定する(パスの有無は任意)。

-I*ディレクトリ名*

#includeで指定されたファイルを*ディレクトリ名*で指定するディレクトリからサーチする。

-isysroot *ディレクトリ名*

#includeで指定されたヘッダファイルを*ディレクトリ名*で指定されたディレクトリ配下の**include**ディレクトリからサーチする。

-isystem *ディレクトリ名*

#includeで指定されたヘッダファイルを**-I**で指定されるディレクトリの後、かつ、標準のシステムディレクトリより前に、*ディレクトリ名*で指定されたディレクトリからサーチする。

-M

プリプロセス結果の代わりにファイルの依存情報を出力する。

-nostdinc

ヘッダファイルについて標準のシステムディレクトリはサーチしない。

-P

行ディレクティブをプリプロセス結果に出力しない。

-U*macro*

*macro*の定義を削除する。

-Wp,*option*

プリプロセッサ(**fpp**)のオプションを指定する。複数のオプションや引数はコンマ(,)で区切って指定する。

3.11 アセンブラオプション

-Wa,*option*

アセンブラ(**nas**)のオプションを指定する。複数のオプションや引数はコンマ(,)で区切って指定する。

-Xassembler *option*

アセンブラ(**nas**)のオプションを指定する。引数を持ったオプションのとき、オプション

と引数をそれぞれ本オプションで指定する。

-assembly-list

アセンブリリストを出力する。出力ファイル名は、入力ファイル名の拡張子を「.O」に変更したものとなる。

3.12 リンカオプション

-cxxlib

C++ライブラリをリンクする。

-Bdynamic

実行時にダイナミックリンクライブラリをリンクする。**-Bstatic**を指定しない場合の既定値である。

-Bstatic

利用者のライブラリを静的リンクする。

-L*ディレクトリ名*

ライブラリを既定のディレクトリより先に*ディレクトリ名*で指定したディレクトリからサーチする。

-l*ライブラリ名*

規定したディレクトリからライブラリ名が"lib*ライブラリ名*"であるライブラリをサーチする。

-nostartfiles

リンク時に標準のシステムのスタートアップファイルを使用しない。

-nostdlib

リンク時に標準のシステムライブラリとスタートアップファイルを使用しない。

-rdynamic

リンク時に未使用のシンボルを含むすべてのシンボルをダイナミックなシンボルテーブルに追加する。

-static

ライブラリを静的リンクする。

-static-nec

NEC SDKのライブラリを静的リンクする。

-shared

共有オブジェクトを生成する。

-Wl,*option*

リンカ(**nld**)のオプションを指定する。複数のオプションや引数はコンマ(,)で区切って指定する。

-Xlinker *option*

リンカ(**nld**)のオプションを指定する。引数を持ったオプションのとき、引数それぞれを本オプションで指定する。

-z *keyword*

リンカ(**nld**)の**-z**と同じ。

3.13 ディレクトリオプション

--sysroot=*ディレクトリ名*

ヘッダファイルとライブラリをサーチするディレクトリを指定する。ヘッダファイルは*ディレクトリ名*で指定されたディレクトリ配下の**include**ディレクトリ配下、ライブラリは*ディレクトリ名*で指定されたディレクトリ配下の**lib**ディレクトリ配下からサーチする。

-B*ディレクトリ名*

コマンド、ヘッダファイル、ライブラリをサーチするディレクトリを指定する。コマンドとライブラリは*ディレクトリ名*で指定されたディレクトリ、ヘッダファイルは*ディレクトリ名*で指定されたディレクトリ配下の**include**ディレクトリ配下からサーチする。

-fintrinsic-modules-path *ディレクトリ名*

組込みモジュールのモジュールファイルをサーチするディレクトリを指定する。

-module *ディレクトリ名*

-J *ディレクトリ名*

モジュールファイルの出力先ディレクトリを指定する。また、指定したディレクトリはモジュールファイルのサーチにも使用される。

3.14 その他オプション

--help

コンパイラの用法を表示する。

-print-file-name=*library*

リンク時に使用されるライブラリファイル(*library*)の絶対パスを出力する。このオプションを指定したとき、コンパイル、および、リンクを行わない。ライブラリファイルが存在しないときは*library*に指定した文字列を出力する。

-print-prog-name=*program*

コンパイル中に呼びだされるコンパイラシステムのコマンド名(*program*)を表示する。このオプションを指定したとき、コンパイル、および、リンクを行わない。指定されたコマンドが存在しないときは*program*で指定した名前を表示する。

-noqueue

コンパイラのライセンス数が使用制限に達しているとき、使用可能になるまで待ち合わせずに終了する。

-v

コンパイルの各ステージで起動されたコマンドを表示する。

--version

コンパイラのバージョンと著作権を表示する。

3.15 オプション指示行で指定できないコンパイラオプション

次のコンパイラオプションは、オプション指示行で指定できません。

- 全体オプション

  -S、**-c**、**-cf**=*conf*、**-fsyntax-only**、**-o** *ファイル名*、**-x** *language*、@*file-name*

- 最適化・ベクトル化オプション

  -muse-mmap

- 並列化オプション

  -mno-create-threads-at-startup、**-pthread**

- インライン展開オプション

  -finline-abort-at-error、**-mgenerate-il-file** *ILファイル名*

- コード生成オプション

  -no-proginf

- デバッグオプション

  -mmemory-trace、**-mmemory-trace-full**、**-traceback**

- 言語仕様制御オプション

  -masync-io、**-use module**

- メッセージオプション

  -Werror

- プリプロセッサオプション

  -D*macro*[=*defn*]、**-E**、**-fpp**、**-nofpp**、**-fpp-name**=*name*、**-M**、**-P**、**-U***macro*、**-Wp,**option

- アセンブラオプション

  -Wa,*option*、**-Xassembler** *option*、**-assembly-list**

- リンカオプション

  -Bdynamic、**-Bstatic**、**-L***ディレクトリ名*、**-l***ライブラリ名*、**-nostartfiles**、**-nostdlib**、**-rdynamic**、**-static**、**-static-nec**、**-shared**、**-Wl**,*option*、**-Xlinker** *option*、**-z** keyword

- ディレクトリオプション

  --sysroot=ディレクトリ名、**-B**ディレクトリ名

- その他オプション

  --help、**-print-file-name**=*library*、**-print-prog-name**=*program*、**-noqueue**、**-v**、**--version**

3.16 最適化レベルとオプションの既定値

-O*n*と最適化を個別に制御するオプションの対応は以下のとおりです。
ただし、-O*n*は最適化全体のレベルを制御するもので、最適化を個別に制御するオプションの有効、無効を同等にしても同じ命令コードが作成されるとは限りませんので注意してください。ある最適化を効果的に適用するには、別の補助的な最適化も適用するなど最適化同士が相互に関連しており、それらが連携して動作するよう-O*n*で制御しています。例えば-O0 に対して、-O1 指定時に既定値として設定される最適化オプションを指定しても -O1 と同じにはなりません。

オプション名	-O4	-O3	-O2	-O1	-O0
-fassociative-math	✓	✓	✓	-	-
-ffast-math	✓	✓	✓	✓	-
-fignore-induction-variable-overflow	✓	-	-	-	-
-fignore-volatile	✓	-	-	-	-
-finline-copy-arguments	-	✓	✓	✓	✓
-floop-collapse	✓	✓	-	-	-
-floop-fusion	✓	✓	-	-	-
-floop-interchange	✓	✓	-	-	-
-floop-normalize	✓	✓	-	-	-
-floop-strip-mine	✓	✓	-	-	-
-floop-unroll	✓	✓	✓	-	-
-floop-unroll-complete=4	✓	✓	✓	-	-
-floop-unroll-complete-nest=3	✓	✓	✓	-	-
-fmatrix-multiply	✓	✓	-	-	-
-fmove-loop-invariants	✓	✓	✓	✓	-
-fmove-loop-invariants-if	✓	✓	-	-	-
-fmove-loop-invariants-unsafe	✓	-	-	-	-
-fmove-nested-loop-invariants-outer	✓	✓	✓	✓	-
-fnamed-alias	-	-	-	✓	✓
-fnamed-noalias	✓	✓	✓	-	-
-fouterloop-unroll	✓	✓	-	-	-
-freciprocal-math	✓	✓	✓	-	-
-freplace-loop-equation	✓	-	-	-	-
-freplace-matmul-to-matrix-multiply	✓	✓	✓	✓	-
-marray-io	✓	✓	✓	✓	-
-msched-none	-	-	-	-	✓

オプション名	-O4	-O3	-O2	-O1	-O0
-msched-block	✓	✓	✓	✓	-
-mvector	✓	✓	✓	✓	-
-mvector-dependency-test	✓	✓	✓	-	-
-mvector-fma	✓	✓	✓	-	-
-mvector-merge-conditional	✓	✓	-	-	-

第4章　　コンパイラ指示行

本章では、Fortranコンパイラのコンパイラ指示行ついて説明します。

コンパイラ指示行は以下の形式で記述します。

```
!NEC$ コンパイラ指示オプション（自由形式）
*NEC$ コンパイラ指示オプション（固定形式）
cNEC$ コンパイラ指示オプション（固定形式）
```

備考　　以下の形式でも記述できます。ただし、この形式は将来廃止予定のため、上記形式での記述を推奨します。

```
!$NEC コンパイラ指示オプション（自由形式）
*$NEC コンパイラ指示オプション（固定形式）
c$NEC コンパイラ指示オプション（固定形式）
```

利用可能なコンパイラ指示オプションについて以降で説明します。

[no]advance_gather

直後のベクトルループ中のベクトル収集命令をループ本体の前方に移動する[しない]。前方に移動することにより、後続の演算命令などとオーバーラップさせ、計算時間を短縮できることがある。

[no]always_inline

本指示行の指定された手続を常にインライン展開の対象とする。呼び出される手続に指定する。ただし、本手続の手続呼出しに**noinline**が有効であるとき、インライン展開されない。**-O***n*[*n*=2,3,4]、**-finline-functions**、**-fopenmp**、または、**-mparallel**が指定されたときのみ有効である。

[no]assoc

演算順序の変更を伴うループ変形を許可する[しない]。

[no]assume

直後の配列式の形状、または、**DO**ループのループ長を仮定するとき、配列宣言の利用を許可する[しない]。

atomic

直後の文が、総和、累積などのマクロ演算の式であることを示す。

cncall

利用者定義の手続呼出しがあっても並列化を許可する。

collapse

多重ループの一重化を許可する。

[no]concurrent

ループの並列化を許可する[しない]。**-mparallel**が有効であるときのみ効果がある。

concurrentに続けて、ループの分割方法を指定するOpenMPと同じ**schedule**句を指定できる。指定できるschedule種別は以下である。

- schedule(static [,*chunk-size*])
- schedule(dynamic [,*chunk-size*])
- schedule(runtime)

dependency_test

データ依存関係の判定をベクトルコード、スカラコードの選択の条件とする条件ベクトル化を許可する。

forced_collapse

直後の多重ループが一重化可能かどうか不明な場合でも強制的に一重化する。一重化しても予期しない結果、結果不正とならないことはプログラマが保証しなければならない。

gather_reorder

直後のDOループ中に現れる非線形添字をもつベクトルロード、ベクトルストアは互いに重なることが無いものと仮定して命令の並べ換えを行うことを許可する。

[no]inline

本指示行の直後の文、**BLOCK**構文、**DO**構文、**IF**構文に含まれる手続呼出しをインライン展開の対象とする[しない]。**-O***n*[*n*=2,3,4]、**-finline-functions**、**-fopenmp**、または、**-mparallel**が指定されたときのみ有効である。

inline_complete

inlineと同様であるが、インライン展開される手続がさらに手続を呼び出しているときその手続もインライン展開の対象とし、手続呼出しがなくなるまでインライン展開を試みる。**-O***n*[*n*=2,3,4]、**-finline-functions**、**-fopenmp**、または、**-mparallel**が指定されたときのみ有効である。

[no]inner

次元数が1である配列式、または、最内側ループの自動並列化を許可する[しない]。最内側ループに指定したときのみ効果がある。

[no]interchange

ループ入れ換えを許可する[しない]。

ivdep

配列の依存関係が不明なとき、ベクトル化不可の依存関係ではないと仮定してベクトル化することを許可する。ベクトル化不可のループをベクトル化し、結果が不正となることがある。

[no]list_vector

同一の非線形添字をもつ同一配列が左辺と右辺に現れる代入文のベクトル化を許可する[しない]。加算、減算のみ含まれる代入文を対象とする。

loop_count(*n*)

ループの繰返し数が不明なとき、繰返し数を整定数*n*回と仮定する。

loop_count_test

ループ長による条件ベクトル化を許可する。

[no]lstval

ループ内で定義され、ループの後で値が参照されるかどうかが不明な変数の終値を保証する[しない]。

move / move_unsafe / nomove

move

不変式のループ外への移動を許可する。

move_unsafe

副作用を伴う場合でも、不変式のループ外への移動を許可する。

nomove

不変式のループ外への移動を許可しない。

nofma

ベクトル積和命令を使用したベクトル化を許可しない。

nofuse

直前のループとのループ融合を許可しない。

nosync

配列要素間、ポインタ式の指示先間の依存関係が不明であるときでも、依存関係がないものと仮定して並列化する。

options “compiler-option [compiler-option]...”

オプション指示行を指定することによりコマンドラインと同じようにコンパイラオプションを指定できる。

指定方法

- オプション指示行は、ソースファイルの先頭に記述する。
- オプション指示行は、複数続けて指定できる。
- 空行、コメント行、**#line**に限り、オプション指示行の直前、間に記述できる。
- オプション指示行は、ソースファイルの先頭の**#include**により取り込まれるファイルの先頭に記述できる。

注意事項

- オプション指示行は継続できない。
- オプション指示行で指定された**-I**で指定されたディレクトリは、オプション指示行の記載されたヘッダファイルのサーチの対象外となる。
- オプション指示行を読み取る際、**#include**により取り込まれるファイルのネスト数の上限値は1000となる。
- オプション指示行は**INCLUDE**行で取り込まれるファイルには記述できない。
- 「3.15 オプション指示行で指定できないコンパイラオプション」で示すリンクやコンパイラの動作環境を制御するオプションはオプション指示行で指定できない。
- オプション指示行で**-fopenmp**、**-mparallel**、または、**-ftrace**を指定したとき、リンク時にもそれらのオプションを指定しなければならない。

outerloop_unroll(*n*) / noouterloop_unroll

outerloop_unroll(*n*)

外側ループのアンローリングを許可する。アンロール段数は*n*を超えない最大の2のべき乗数となる。

noouterloop_unroll

外側ループのアンローリングを許可しない。

[no]packed_vector

直後の実行文中に現れる**DO**ループ中でパックドベクトル命令の使用を許可する[しない]。

parallel do

DOループを強制並列化する。ループが並列実行できることはプログラマが保証しなければならない。**-mparallel**が有効であるときのみ効果がある。

parallel doに続けて、ループの分割方法を指定するOpenMPと同じ**schedule**句を指定できる。指定できるschedule種別は以下である。

- schedule(static [,*chunk-size*])
- schedule(dynamic [,*chunk-size*])
- schedule(runtime)

また、**PRIVATE**句も指定できます。*name*では、文字型、派生型でないスカラ変数名、形状明示配列名を指定できる。

pvreg(*array-name*)

配列*array-name*にメモリの代わりにベクトルレジスタを割り当てる。

コンパイラはこの指定行を含む手続内のすべての**pvreg**指定された配列の参照を、ロード／ストアではなく、ベクトルレジスタの参照として翻訳する。

PVREG指定される配列は以下の条件を満たさなければならない。

- ローカル配列
- 配列のタイプは**INTEGER(KIND=4)**、**REAL(KIND=4)**または、それらの別名でなければならない
- 1次元配列
- 配列要素の数は最大のパックドベクトル長(=512)以下
- パックドベクトル化されるループ中でのみ定義／参照されること
- すべてのループにおいて添字が同じであること
- **VREG**指定される配列を指定してはならない

retain(*array-name*)

*array-name*の配列、または、ポインタの指示先を可能なかぎりLLC(Last-Level Cache)に保

持することを指定する。

備考　この指示行を有効にするとき、**-mretain-list-vector**、または、**-mretain-none** を指定してください。

select_concurrent

多重ループのループのうち、直後のループに対して他のループより優先して自動並列化を適用する。

select_vector

多重ループのループのうち、直後のループに対して他のループより優先して自動ベクトル化を適用する。

shortloop

ループの繰返し数が、システムの最大ベクトルレジスタ長(=256)を超えないものとしてベクトル化することを許可する。

[no]shortloop_reduction

リダクション演算を含むループの繰返し数による条件ベクトル化を許可する[しない]。

-fassociative-mathが有効であるときのみ効果がある。

[no]sparse

sparse

ループ中の条件下の数学関数の実行回数が、ループの繰返し数より非常に小さいものとして、ベクトル化する。

nosparse

ループ中の条件文下の数学関数の実行回数が、ループの繰返し数に近いものとして、ベクトル化 する。

unroll(*n*) / nounroll

unroll(*n*)

ループを*n*段にアンロールすることを許可する。

nounroll

ループのアンローリングを許可しない。

unroll_complete

直後のループの繰返し数が定数でコンパイル時に算定できるとき、そのループをループ展開(完全ループアンローリング)することを許可する。

備考　　別名としてunroll_completelyも使用できる。

[no]vector

自動ベクトル化を許可する[しない]。

vector_threshold(*n*)

直後の**DO**ループまたは配列式のベクトル化を行う最小の繰返し数を*n*回とする。

[no]vob

直後の実行文中に現れる配列式、または、**DO**ループの終了後に実行されるスカラロード、スカラストア、ベクトルロードが、その配列式、または、**DO**ループ中のベクトルストアを追い越すことを許可しない[する]。

多重ループの外側ループに指定したとき、その内側ループには効果が無い。

[no]vovertake

直後の実行文中に現れる配列式、または、**DO**ループ中のベクトルストアを、後続のスカラロード、スカラストア、ベクトルロードが追い越して実行することを許可する[しない]。

- 配列式、または、**DO**ループ中のベクトルストアと、ループ中またはループ後のスカラロード、スカラストア、ベクトルロードの領域に重なりがあるとき、実行結果が不正になることがある。
- 多重ループの外側ループに指定したとき、その内側ループには効果が無い。

vreg(*array-name*)

配列*array-name*にメモリの代わりにベクトルレジスタを割り当てる。
コンパイラはこの指定行を含む手続内のすべての**VREG**指定された配列の参照を、ロード・ストアではなく、ベクトルレジスタの参照として翻訳する。
VREG指定される配列は以下の条件を満たさなければならない。

- ローカル配列
- 配列のタイプはINTEGER(KIND=4)、INTEGER(KIND=8)、REAL(KIND=4)、REAL(KIND=8)、または、それらの別名でなければならない
- 1次元配列
- 配列要素の数は最大のベクトル長(=256)以下
- ベクトル化されるループ中でのみ定義／参照されること
- すべてのループにおいて添字式の値が同じであること

- **PVREG**指定される配列を指定してはならない

[no]vwork

部分ベクトル化(ループ分割によるベクトル化)を許可する[しない]。**novwork**を指定した場合、外側ループ、およびベクトル化不可の部分を含むループは、ループ全体がベクトル化されなくなる。

第5章　　最適化・ベクトル化

本章では、コンパイラのもつ最適化機能、自動ベクトル化によるプログラムの高速化にかかわる機能を説明します。

5.1　コードの最適化

コードの最適化は、プログラム中の制御の流れやデータの流れを解析することによって、不要な演算を可能なかぎり削除し、また、ループ中の演算を必要最小限のものとします。さらに可能ならば、演算をそれと等価なより高速な演算と置き換えることにより、プログラム実行時間の短縮を図ります。

5.1.1　コンパイラの適用する最適化

Fortranコンパイラは次の最適化を適用します。()内でそれらを有効にするオプションを示します。

- 共通式の削除（**-O**[*n*](*n*=1,2,3,4)）
- 条件下の不変式のループ外への移動（-O[*n*](*n*=1,2,3,4), **-fmove-loop-invariants**, **-fmove-loop-invariants-unsafe**）
- 単純代入の削除（**-O**[*n*](*n*=1,2,3,4)）
- 不要コードの削除（**-O**[*n*](*n*=1,2,3,4)）
- べき乗の最適化（**-O**[*n*](*n*=1,2,3,4)）
- 除算の乗算化（**-O**[*n*](*n*=2,3,4), **-freciprocal-math**）
- ループ融合（**-O**[*n*](*n*=3,4)）
- 算術IF文の最適化（**-O**[*n*](*n*=1,2,3,4)）
- 定数の演算・型変換のコンパイル時計算（**-O**[*n*](*n*=1,2,3,4)）
- 複素数演算の最適化（**-O**[*n*](*n*=1,2,3,4)）
- 単項演算子のマイナスの除去（**-O**[*n*](*n*=1,2,3,4)）
- 分岐に関する最適化（**-O**[*n*](*n*=1,2,3,4)）
- 演算の強度縮小とテスト変数の置換（**-O**[*n*](*n*=1,2,3,4)）
- 不要終値保証の除去（**-O**[*n*](*n*=1,2,3,4)）
- 組込み手続のインライン展開（**-O**[*n*](*n*=1,2,3,4)）

- 入出力文のDO形並びの最適化（**-O**[*n*](*n*=1,2,3,4), **-marray-io**）
- 命令の並べ換えによる最適化（**-msched**-*keyword*）

5.1.2　最適化による副作用

- 式、コードの削除により、計算を行う場所、回数が変わり、エラーの発生位置、回数が最適化を適用しなかった場合と比べて変わることがあります。
- 条件下の不変式のループ外への移動により、実行されないはずの式が実行され、本来発生しないはずのエラーや演算例外が発生することがあります。
- べき乗の最適化を適用したとき、アンダーフローが発生しても例外を検出しません。
- 除算の乗算化により、演算結果にわずかの差異が生じます。浮動小数点数の演算では、この差異はほとんど無視してかまいませんが、無視したくないときコンパイラオプションで最適化を抑止してください。
- 命令の並べ換えによる最適化により、ある条件が成立したときのみ実行される計算が、基本ブロックをまたがって移動され、常に実行されるようになると、発生しないはずの実行時エラーが検出されることがあります。また、コンパイル時間やコンパイラが使用するメモリ量を著しく増加させることがあります。

5.2　ベクトル化機能

5.2.1　ベクトル化

変数や配列の各要素のことをスカラデータと呼びます。これに対し、行列の行要素、列要素、対角要素など、規則的に並んだデータ列をベクトルデータと呼びます。ベクトルデータを処理するスカラ命令列を、等価な処理を行うベクトル命令で置き換えることをベクトル化といいます。コンパイラがプログラムを解析してベクトル命令で実行可能な部分を自動的に検出し、その部分に対してベクトル命令を生成することを自動ベクトル化といいます。**-O**[*n*]（*n*=1,2,3,4)が有効であるとき、自動ベクトル化が有効になります。

この最適化を制御するコンパイラオプションは、**-mvector**です。

この最適化を制御するコンパイラ指示オプションは、**[no]vector**です。

5.2.2　部分ベクトル化

ループ、または、配列式にベクトル化可能な部分とベクトル化不可の部分が混在しているとき、そのループ、または、配列式をベクトル化可能な部分(ベクトルコード部分)とベクトル化不可の部分(スカラコード部分)に分割し、ベクトル化可能な部分のみをベクトル化します。このベクトル化方法は、部分ベクトル化と呼ばれます。**-O**[*n*](*n*=1,2,3,4)が有効である

とき、この最適化が有効になります。

この最適化を制御するコンパイラオプションは、**-mwork-vector-kind**=noneです。

この最適化を制御するコンパイラ指示オプションは、**[no]vwork**です。

5.2.3　マスク演算の最適化

IF文を含む**DO**ループに対してマスク付きの演算によりベクトル化を可能としているが、**IF**文が入れ子となり、複合条件となっているようなときには、マスク間の同一な演算が生じ、実行効率が低下することがあります。これを避けるために、マスク演算に対して次のような最適化を行います。**-O**[*n*] (*n*=1,2,3,4)が有効であるとき、この最適化が有効になります。

(1)　同一の演算は、共通式として処理します。

この例ではA(I).LE.0.0が共通式として処理されます。

例

```
DO I = 1, N
    IF (A(I).LE.0.0)THEN
        X(I) = A(I) * B(I)
    END IF
    Y(I) = A(I) + B(I)
    IF (A(I).LE.0.0.AND.B(I).EQ.0.0) THEN
        Z(I) = A(I)
    END IF
END DO
```

↓　ベクトル化

```
M1i       ←      0: if Ai > 0.0
                 1: if Ai <= 0.0
Xi        ←      Ai * Bi (if M1i = 1)
Yi        ←      Ai + Bi
M2i       ←      0: if Bi ≠ 0.0
                 1: if Bi = 0.0
M3i       ←      M1i  AND  M2i
Zi        ←      Ai  (if M3i = 1)
```

(2)　**IF** 文が入れ子となり、複合条件となっている場合にも、共通式として処理されます。

この例ではY(I).GT.0.0が共通式として処理されます。

例

```
DO I = 1, N
```

```
    IF (X(I).GT.0.0) THEN
        IF (Y(I).GT.0.0) THEN
            Z(I) = Y(I) / X(I)
        ELSE
            Z(I) = 0.0
        END IF
    ELSE
        IF (Y(I).GT.0.0) THEN
            Z(I) = X(I) / Y(I)
        END IF
    END IF
END DO
```

↓　ベクトル化

```
M1i       ←      0: if Xi <= 0.0
                 1: if Xi >  0.0
M2i       ←      0: if Yi <= 0.0
                 1: if Yi >  0.0
M3i       ←      M1i  AND  M2i
Zi        ←      Yi / Xi (if M3i = 1)
M4i       ←      M1i  AND  M2i
Zi        ←      0.0    (if M4i = 1)
M5i       ←      M1i  AND  M2i
Zi        ←      Yi / Xi (if M5i = 1)
```

5.2.4　マクロ演算

以下のようなパターンの式はベクトル化の定義・参照の条件を満たしていませんが、コンパイラは特別なパターンと認識し、専用のベクトル命令を用いてベクトル化します。-O[*n*] (*n*=1,2,3,4)が有効であるとき、この最適化が有効になります。

(1)　総和/内積

```
S = S ± exp              (exp:式)
```

次のような、複数の文から構成されている総和/内積もベクトル化されます。

```
t1 = S ± exp1            (expi:式)
t2 = t1 ± exp2
      ...
S = tn ± expn
```

このベクトル化は、-mvector-reductionで制御します。

(2)　累積

```
S =   S   *   exp                    (exp:式)
```

次のような、複数の文から構成されている累積もベクトル化されます。

```
t1    =   S   *   exp1          (expi:式)
t2    =   t1  *   exp2
      ...
S     =   tn  *   expn
```

このベクトル化は、-mvector-reductionで制御します。

(3)　漸化式

```
A(I)      =   exp     ±   A(I-1)                    (exp:式)
A(I)      =   exp     *   A(I-1)
A(I)      =   exp1±   A(I-1)   *   exp2
A(I)      =   (exp1   ±   A(I-1)) *   exp2
```

次のような複数の文から構成される漸化式もベクトル化されます。

```
T =   exp1±   A(I-1)                        (expi:式)
A(I) = t * exp2
```

このベクトル化は、-mvector-iteration、および、-mvector-iteration-unsafeで制御します。

(4)　最大値/最小値

(a) 関数型

例

```
DO I = 1, N
    XMAX = MAX(XMAX, X(I))
END DO
```

(b)　最大値/最小値のみを求める

例

```
DO I = 1, N
```

```
    IF (XMAX .LT. X(I)) THEN
        XMAX = X(I)
    END IF
END DO
```

(c)　最大値/最小値とインデックスを求める

例

```
DO I = 1, N
    IF (XMIN .GT. X(I)) THEN
        XMIN = X(I)
        IX = I
    END IF
ENDDO
```

(d)　最大値/最小値とインデックス、および、その他の値を求める

例

```
DO J = 1, N
    DO I = 1, N
        IF (XMIN .GT. X(I, J)) THEN
            XMIN = X(I, J)
            IX = I
            IY = J
        END IF
    END DO
END DO
```

(e)　絶対値比較

例

```
DO I = 1, N
    IF (ABS(XMIN) .GT. ABS(X(I))) THEN
        XMIN = X(I)
    ENDIF
ENDDO
```

(5)　サーチ

ある条件を満たす要素をサーチするループをベクトル化します。

例

```
DO I = 1, N
    IF (X(I) .EQ. 0.0) THEN
        EXIT
    ENDIF
ENDDO
```

このとき、ループは以下の条件を満たさなければなりません。

- 最内側のループである。
- ループの外への分岐はただ一つである。
- ループの外への分岐条件はループの繰返しに依存する。
- ループの外への分岐の前に配列要素、ポインタの指示先への代入がない。
- ループの外への分岐以外は、ベクトル化の条件をすべて満たしている。

(6) 圧縮

ある条件を満たす要素を圧縮するループをベクトル化します。

例

```
J = 0
DO I = 1, N
    IF (X(I) .GT. 0.0) THEN
        J = J + 1
        Y(J) = Z(I)
    END IF
END DO
```

(7) 伸長

ある条件を満たす要素に値を伸長するループをベクトル化します。

例

```
J = 0
DO I = 1, N
    IF (X(I) .GT. 0.0) THEN
        J = J + 1
        Z(I) = Y(J)
    END IF
END DO
```

5.2.5　条件ベクトル化

条件ベクトル化とは、一つのループに対してベクトル化したコードとスカラのコード、特定のパターンのみ高速に実行できるコードなど、数種類のコードを用意し、実行時に条件を調べて、適切なコードを選択して実行するようなループの変形のことをいいます。実行時に調べる条件は以下があります。

- 依存関係
- ループ長
- リダクション演算を含むループのループ長

  -O[*n*](*n*=2,3,4)が有効であるとき、この最適化が有効になります。この最適化を制御するコンパイラオプションは、実行時に調べる条件ごとに以下です。

- 依存関係による条件ベクトル化の制御は、**-mvector-dependency-test**です。
- ループ長による条件ベクトル化の制御は、**-mvector-loop-count-test**です。
- リダクション演算を含むループのループ長による条件ベクトル化の制御は、**-mvector-shortloop-reduction**です。

  この最適化を制御するコンパイラ指示オプションは、実行時に調べる条件ごとに以下です。

  - 依存関係による条件ベクトル化の制御は、**dependency_test**です。
  - ループ長による条件ベクトル化の制御は、**loop_count_test**です。
  - リダクション演算を含むループのループ長による条件ベクトル化の制御は、**[no]shortloop_reduction**です。

5.2.6　外側ループストリップマイニング

繰返し数が最大ベクトルレジスタ長(=256)より長いループをベクトル化するとき、コンパイラは内部的に繰返し数を最大ベクトルレジスタ長に収まるように分割しています。これをストリップマイニングと呼びます。密な多重ループの内側ループに、外側ループのインダクション変数を添字式に含まない配列要素があるとき、コンパイラはストリップループを外側に移動します。**-O**[*n*](*n*=3,4)が有効であるとき、この最適化が有効になります。

この最適化を制御するコンパイラオプションは、**-floop-strip-mine**です。

備考　密な多重ループとは、すべての直接の入れ子関係をなすループにおいて、外側ループの**DO**文と内側ループの**DO**文の間、および、内側ループの**ENDDO**文と外側ループの**ENDDO**文の間に実行文が現れないもの(および、それと等価な構造の多重ループ)です。

例　　密な多重ループ

```
DO I = 1, 10
    DO J = 1, 1000
        A(J) = A(J) + B(J, I) * C(J, I)
    ENDDO
ENDDO
```

例　　密でない多重ループ(DO 文の間に実行分が現れる)

```
DO K = 1, 10
    D(K) = 0.0
    DO J = 1, 20
        DO I = 1, 30
            A(I, J, K) = B(I, J, K) * C(I, J, K)
        END DO
        X(K, J) = Y(K, J) + Z(K, J)
    END DO
END DO
```

例　　密でない多重ループ(ENDDO 間に別のループが現れる)

```
DO K = 1, 10
    DO J = 1, 20
        DO I = 1, 10
            S(I, J, K) = T(I, J, K) * U(I, J, K)
        END DO
        DO I = 1, 30
            A(I, J, K) = B(I, J, K) * C(I, J, K)
        END DO
    END DO
END DO
```

5.2.7　ショートループ

ベクトル化されたループのうち、繰返し数が最大ベクトルレジスタ長(=256)以下のループに対して、ループの繰返しの終了判定が省略された命令コードが生成されます。このループはショートループと呼ばれます。**-O**[*n*](*n*=1,2,3,4)が有効であるとき、この最適化が有効になります。

この最適化を制御するコンパイラ指示オプションは、**shortloop**です。

5.2.8　パックドベクトル命令

ベクトルレジスタの各要素を32ビットで二つに分割して、ベクトルレジスタの各要素に2つのデータを格納することをパックといいます。パックしたデータに対して演算を行う命令をパックドベクトル命令といいます。パックドベクトル命令はベクトル命令の2倍のデータを1命令で処理することができます。

パックドベクトル命令の使用を制御するコンパイラオプションは、**-mvector-packed**です。パックドベクトル命令の使用を制御するコンパイラ指示オプションは、**[no]packed_vector**です。

5.2.9　その他のベクトル化コードに適用する最適化

共通式の削除、単純代入の削除、不要コードの削除、除算の乗算化、不要終値保証の除去は、ベクトル化されたコードに対しても行います。その他に、Fortranコンパイラはベクトル化したコードに対して、次の最適化を適用します。()内でそれらを有効にするオプションを示します。

- スカラ演算のくくりだし（**-O**[*n*](*n*=1,2,3,4)）
- 文の入れ換えによるベクトル化（**-O**[*n*](*n*=1,2,3,4)）
- ループの一重化（**-O**[*n*](*n*=3,4)，**-floop-collapse**）
- 外側ループのアンローリング（**-O**[*n*](*n*=3,4)，**-fouterloop-unroll**）
- ループのリローリング（**-O**[*n*](*n*=3,4)）
- 行列積ループの認識（-O[*n*](*n*=3,4)，-fassociative-math，-fmatrix-multiply）
- ループ展開（-O[*n*](*n*=2,3,4)，-floop-unroll-complete=*m*）

5.2.10 ベクトル化機能使用時の注意事項

- 総和演算は、演算順序がベクトル化したときとしないときとで違うため、演算結果が異なることがあります。累積、漸化式、内積演算も同様です。
- 8バイト整数型の漸化式は実数型のベクトル漸化式命令を用いてベクトル化されます。このため結果の精度が52bitを超えるときや、実数型のオーバーフローが発生するとき、実行結果はベクトル化しないときと異なります。
- ベクトル版の数学関数で使用している計算方法は、高速化のため、スカラ版と必ずしも同じ結果にはなりません。
- ベクトル化したときとしないときとで、除算の乗算化などの最適化の適用のされ方が異なります。

- ベクトル化したときとしないときとで、演算順序の変更などの最適化の適用のされ方が異なります。
- ベクトル化すると、数学関数で検出されるエラーや演算例外の検出のされ方が、ベクトル化しないときと異なります。
- コンパイラは、配列の定義・参照関係がベクトル化により正しく保たれるか否かを検査する際、個々の添字式の値が配列宣言の対応する次元の上限、下限の間に収まっていることを前提とします。したがって、この条件に反するループをベクトル化したとき、その実行結果は保証されません。
- IF文を含むループがベクトル化されると、条件付きで実行される部分に関し、演算は必要な部分しか行われませんが、配列要素は、ループ構造で決まる繰返し数分だけ参照されます。すなわち、本来参照されない配列要素が参照されます。したがって、配列については繰返し数分だけの領域を用意しておかないと、メモリアクセス例外を起こすことがあります。
- 飛び出しを含むループがベクトル化されると、飛び出しが起こる繰返し回より後の繰返しも実行されます。したがって、本来実行されない演算が実行されたり、本来参照されないデータが参照されることにより、エラーや例外が発生することがあります。
- ベクトル処理できるデータのアラインメントは、そのデータの型のサイズと同じ（4バイト、または、8バイト）でなければなりません。アラインメントの条件を満たさない配列要素の参照、定義を含むループがベクトル化されたとき、実行時に例外となることがあります。そのとき、コンパイル時に
  -mno-vectorを指定してプログラム全体のベクトル化をやめるか、**!NEC$ NOVECTOR**を指定し、問題のループのベクトル化を抑止してください。ベクトル処理できるアラインメントを満たさなくなる可能性のあるデータは仮引数です。コンパイラは、仮引数をアラインメントの条件を満たすものと仮定してベクトル化します。

5.3　その他の高速化機能

5.3.1　配列一括出力の VH へのオフロード

書式付き配列一括出力処理、および、並び配列一括出力処理をVH上にオフロードすることでプログラムの実行を高速化します。この機能を利用するにはプログラム実行時に環境変数**VE_FMTIO_OFFLOAD**に**YES**、または、**ON**を指定し、環境変数**LD_LIBRARY_PATH**に/opt/nec/ve/nfort/lib64を指定してください。

例　　書式付き配列一括出力

```
SUBROUTINE FUN
INTEGER I(100)
I=100
WRITE(*,'(I5)') I
END
```

例　　並び配列一括出力

```
SUBROUTINE FUN
INTEGER I(100)
I=100
WRITE(*,*) I
END
```

5.3.2　バッファの効率的な利用

順編成ファイルの書式なし入出力では、レコードバッファや入出力バッファのサイズを変更することで、入出力を高速に行える場合があります。

5.3.2.1 レコードバッファ

順編成ファイルの書式なし入出力は、レコードバッファを利用して入出力項目並びとデータのやり取りを行います。このため、順編成ファイルの書式なし記録の最大記録長より、レコードバッファの値を大きくすると入出力が高速化されます。レコードバッファのサイズを変更するには環境変数**VE_FORT_RECORDBUF**を利用します。

5.3.2.2 入出力バッファ

ファイルとの入出力は、ファイルと入出力バッファの間で行われます。ファイルが存在するファイルシステムには最適な転送サイズが存在します。入出力バッファのサイズを最適な転送サイズに合わせることで入出力が高速化されます。また、メモリサイズに問題がなければ、ファイルサイズより大きな値を入出力バッファのサイズにすることでも入出力が高速化されます。入出力バッファのサイズを変更するには環境変数**VE_FORT_SETBUF**を利用します。

第6章　　インライン展開機能

6.1　自動インライン展開

コンパイラがソースファイルを解析、サーチして、インライン展開すべき手続呼出しを選択し、インライン展開します。
この最適化を制御するコンパイラオプションは、**-finline-functions**です。

6.2　明示的インライン展開

6.2.1　説明

明示的インライン展開では、利用者がインライン展開を指示する指示行(インライン展開指示行)をソースファイルに指定し、インライン展開します。このとき**-finline-functions**の指定は必要ありません。ただし、インライン展開指示行は**-O***n*[*n*=2,3,4]、**-finline-functions**、**-fopenmp**、または、**-mparallel**が指定されたときのみ有効です。
インライン展開を指示する指示行には次のものがあります。

- **always_inline**
  本指示行の指定された手続を常にインライン展開の対象とする。呼び出される手続に指定する。ただし、本手続の手続呼出しに**noinline**が有効であるとき、インライン展開されない。
- **inline**
  本指示行の直後の文、**BLOCK**構文、**DO**構文、**IF**構文に含まれる手続呼出しをインライン展開の対象とする。
- **inline_complete**
  inlineと同様であるが、インライン展開される手続がさらに手続を呼び出しているときその手続もインライン展開の対象とし、手続呼出しがなくなるまでインライン展開を試みる。
- **noinline**
  本指示行の直後の文、**BLOCK**構文、**DO**構文、**IF**構文に含まれる手続呼出しをインライン展開しない。**always_inline**の指定された手続もインライン展開しない。

6.2.2　インライン展開指示行の指定

(1)　呼び出される手続

always_inlineは呼び出される手続に指定します。

例

```
      SUBROUTINE SUB
!NEC$ ALWAYS_INLINE
      ...
      END SUBROUTINE
```

(2)　直後の文

inline / **inline_complete** / **noinline**は、直後の文中のすべての手続呼出しに作用します。

例

```
!NEC$ INLINE
    X = FUNC1(A) + FUNC2(A)
    Y = FUNC3(A)
```

FUNC1()、FUNC2()の呼び出しをインライン展開対象とする。FUNC3()は対象ではない。

(3)　**BLOCK** 構文 / **DO** 構文 / **IF** 構文

inline / inline_complete / noinlineは、直後のBLOCK構文、DO構文、IF構文のすべての手続呼出しに作用します。

例

```
!NEC$ INLINE
    DO I=1,N
        CALL SUB1
        CALL SUB2
    END DO
```

SUB1、SUB2の呼び出しをインライン展開対象とする。

6.2.3　注意事項

- always_inline / inline / inline_complete / noinlineは、-O*n*[*n*=2,3,4]、-finline-functions、-fopenmp、または、-mparallelが指定されたときのみ有効です。
- **always_inline**を指定した手続の定義は削除されません。
- **noinline**指定された手続呼出しから**always_inline**を持つ手続を呼び出しているとき、その手続はインライン展開しない。

- **BLOCK**構文、**DO**構文、**IF**構文がネストしており、それぞれに相反する指示行が指定されているとき、内側の構文についてはその構文に指定された指示行が優先適用されます。

例

```
!NEC$ INLINE
    BLOCK
        CALL SUB1          ! インライン展開の対象
!NEC$ NOINLINE
        BLOCK
            CALL SUB2      ! インライン展開しない
        END BLOCK
    END BLOCK
```

6.3　クロスファイルインライニング

コンパイル対象のソースファイルとは別のソースファイルに含まれる手続をインライン展開することをクロスファイルインライニングと呼びます。
NEC Fortranコンパイラでは、自動インライン展開のオプションに加えて、インライン展開する手続をサーチする別のソースファイルを指定することでクロスファイルインライニングできます。
以下にサーチするソースファイルの指定例を示します。

(1)　ソースファイルを指定

```
$ nfort -c -finline-functions -finline-file=sub.f90 call.f90
```

(2)　ソースファイルとコマンドラインで指定されたすべてのソースファイルを指定

```
$ nfort -c -finline-functions -finline-file=sub2.f90:all call.f90 sub.f90
```

(3)　ディレクトリにあるすべてのソースファイルを指定

```
$ ls dir
sub.f90 sub2.f90 sub3.f90
$ nfort -c -finline-functions -finline-directory=dir sub.f90
```

(4)　ディレクトリにあるすべてのソースファイルを指定したが、あるソースファイルを除外

```
$ ls dir
sub.f90 sub2.f90 sub3.f90
$ nfort -c -finline-functions -finline-directory=dir -fno-inline-file=sub2.f90
```

```
call.f90
```

サーチ対象にソースファイルを指定する他にILファイルを指定する方法もあります。コンパイルするソースファイルが多いとき、この方法の方がコンパイル時間を短縮できる場合があります。

(5)　IL ファイルを生成後、IL ファイルを読み込んで手続をサーチする。

```
$ nfort -mgenerate-il-file sub.f90
$ nfort -c -finline-functions -mread-il-file sub.fil main.f90
```

6.4　インライン展開の阻害要因

インライン展開の阻害要因として次の項目があります。

- インライン展開される手続が見つからない。
- 呼出し元の手続の引数がインライン展開される手続の引数に一致しない。
- 呼出し元の手続とインライン展開される手続に、同じ名前でサイズが異なる名前付き共通ブロックが含まれる。
- インライン展開される手続に変数群入出力文が含まれる。
- インライン展開される手続に**SAVE**属性を持つ変数が含まれる。
- インライン展開される手続で引用される関数名が、呼出し元の手続で使用される非関数名と衝突する。
- インライン展開される手続にOpenMPディレクティブが指定されている。
- インライン展開される手続に再帰的な手続呼出しが含まれる。

6.5　インライン展開機能使用時の注意事項

- インライン展開機能を利用するとき、インライン展開後のオブジェクトファイルの大きさに注意してください。多数の手続をインライン展開してしまうと、プログラムのコードサイズが肥大化し、命令キャッシュからコードがあふれてしまい、プログラム全体の実行性能が低下することがあります。
- クロスファイルインライニングで、大きなプログラムや多数のプログラムをサーチする場合、それ以降の処理量が増えて、コンパイル時間が延びたり、コンパイル時に使用するメモリ量が増加したりすることがあります。

- **-finline-file**や**-finiline-directory**でサーチ対象となったソースファイルから参照するモジュールが見つからないとき、そのソースファイルをサーチせずにコンパイルを継続するため、コンパイル順序によって手続がインライン展開されるかどうか変わる場合があります。コンパイルを継続せずに終了したいとき、**-finline-abort-at-error**を指定して下さい。

6.6　インライン展開機能に対する制限事項

- クロスファイルインライニングは、スレッドプライベートな共通ブロックが指定された**EQUIVALENCE**文を含むソースファイルをサーチできません。
- クロスファイルインライニングは、**PRIVATE**属性をもつ変数を参照するモジュール手続をインライン展開できません。

第7章　　自動並列化・OpenMP 並列化機能

本章では、自動並列化、OpenMP並列化機能、それらの利用にあたって留意すべき項目について説明します。

7.1　自動並列化機能

7.1.1　自動並列化

自動並列化機能は、プログラム内の並列実行できるループや文の集まりを抽出し、並列処理できるようにプログラムを変形、および、並列処理制御のための処理の挿入を自動的に行います。

この最適化を制御するコンパイラオプションは、**-mparallel**です。

7.1.2　作業量による条件並列化

一般に、並列処理はオーバーヘッドを伴うため、分割された各ループの繰返しが、それぞれに十分な作業量をもっていなければ、プログラムの実行時間を増大させることになります。繰返し数がコンパイル時に計算できないとき、その多重ループについて、並列化したコードとそうでないコードの両方を生成しておき、実行時に繰返し数の大小によって適切な方が実行されるようにします。これを「作業量による条件並列化」と呼びます。作業量による条件並列化では、ループ中の演算の数などから、その多重ループが並列化に適している繰返し数の下限値(しきい値)が計算されます。実行時には、このしきい値と、多重ループの繰返し数が比較され、繰返し数がしきい値以上ならば並列化されたコードが実行され、小さければ並列化されていないコードが実行されるようにIF文などが生成されます。

この最適化を制御するコンパイラオプションは、**-mparallel-threshold**=*n*です。

7.1.3　依存関係による条件並列化

ループ中に、コンパイル時に不明なデータ依存関係が存在して並列化を妨げているとき、自動並列化機能は作業量による条件ベクトル化と同様に、依存関係を実行時にテストするためのコードを生成し、依存関係により条件並列化します。1重、または、2重ループで、しきい値のテストが行われるとき、依存関係のテストも同時に行われます。

7.1.4　最内側ループの並列化

外側ループによる適切な並列化ができないとき、最内側のループも並列化の対象とします。本最適化を有効にしない限り、最内側ループは、繰返し数がしきい値を超えることが明らかなときのみ並列化されます。

この最適化を制御するコンパイラオプションは、**-mparallel-innerloop**です。

7.1.5　ループの強制並列化

プログラマはあるDOループが並列化可能なことを知っているが、コンパイラには並列化可能であることが認識できないため自動並列化が行われないようなとき、強制並列化指示行の**!NEC$ PARALLEL DO**によりループを並列化できます。このとき、ループが並列実行できることは、プログラマが保証しなければなりません。

PARALLEL DOに続けて、ループの分割方法を指定するOpenMPと同じ**SCHEDULE**句を指定できます。指定できるスケジューリング種別は以下です。

- **SCHEDULE**(STATIC [,*chunk-size*])
- **SCHEDULE**(DYNAMIC [,*chunk-size*])
- **SCHEDULE**(RUNTIME)

また、**PRIVATE**句も指定できます。*name*では、文字型、派生型でないスカラ変数名、形状明示配列名を指定できます。

- **PRIVATE**(*name*[,...])

ループが、総和、累積などのマクロ演算の文を含むとき、**!NEC$ ATOMIC**をその文の直前に指定します。

強制並列化指示行の指定例を以下に示します。

例

```
      SUBROUTINE SUB(SUM, A, N)
      INTEGER::N
      REAL(KIND=8)::A(N,N), SUM
      ...
!NEC$ PARALLEL DO
      DO J = 1, N
         DO I = 1, N
!NEC$ ATOMIC
            SUM = SUM + A(I, J)
         ENDDO
      ENDDO
      ...
      END
```

7.2　OpenMP並列化

7.2.1　OpenMP 並列化の利用

OpenMPを利用するには、コンパイル、リンク時に**-fopenmp**を指定します。OpenMPディレクティブや注意事項の詳細については、OpenMP仕様を参照してください。

例　　OpenMP ディレクティブの挿入

```
FUNCTION FUN(N, A)
INTEGER N, I, J
REAL A(N), B(N)
REAL FUN
FUN = 1.0
...
!$OMP PARALLEL DO REDUCTION(+:FUN)
DO J = 1, N
    DO I = 1, N
        FUN = A(J) + B(I) + FUN
    END DO
END DO
RETURN
END FUNCTION
```

7.2.2　OpenMP Version 5.0

OpenMP Version 5.0で追加された構文のうち、以下の構文が利用できます。

- **LOOP**構文
- **PARALLEL LOOP**構文
- **PARALLEL MASTER**構文

7.2.3　OpenMP 並列化に対する拡張機能

OpenMP Version 4.5でサポートされいている環境変数について、環境変数名に接頭子"VE_"をつけた環境変数も利用できます。接頭子"VE_"のあり・なし両方の環境変数が指定されている場合には、接頭子"VE_"ありの環境変数で指定した値が有効となります。

例　　環境変数の指定(**VE_OMP_NUM_THREADS** が有効)

```
$ export OMP_NUM_THREADS=4
$ export VE_OMP_NUM_THREADS=8
```

7.2.4　OpenMP 並列化に対する制限事項

以下の機能は利用できません。

- “Device Constructs”で定義されている機能
  コンパイラはデバイスに対するコードを一切生成せず、TARGETリージョンはホスト上で実行されます。
- **REDUCTION**句の中以外で現れる、“Array Sections”で定義されている構文
- “Cancellation Constructs”で定義されている機能
- “Controlling OpenMP Thread Affinity”で定義されている機能

DISTRIBUTE、**TARGET**、**TEAMS**

複合構文のための指示行中の**DISTRIBUTE**、**TARGET**、**TEAMS**、および、それらに関わる指示句は無視されます。

例：”**TARGET PARALLEL FOR**”は“**PARALLEL FOR**”としてコード生成します。

- **TASKLOOP**構文
- **PARALLEL DO SIMD**構文、および**DO SIMD**構文
  それぞれ**PARALLEL DO**構文、**DO**構文としてコード生成します。
- **SIMD**構文
  SAFELEN句、または、**SIMDLEN**句が指定されていない場合、**ivdep**指示行が指定されているものとみなします。
- **DECLARE REDUCTION**構文
- **ALLOCATE**句
- **BIND**句
- **IF**句でのdirective-name-modifier指定
- **IN_REDUCTION**/**TASK_REDUCTION**句
- **ORDERED**句での*parameter*指定
- **SCHEDULE**句での*modifier*指定
- 配列が指定された**DEPEND**句
- **DEPEND**句での*dependence-type*指定の**SOURCE**、**SINK**
- **CRITICAL**構文での**HINT**指定
- **ATOMIC**構文での**SEQ_CST**指定

- **LINEAR**句での*modifier*指定
- ネスト並列性

7.3　スレッド制御

7.3.1　スレッド数の指定・取得

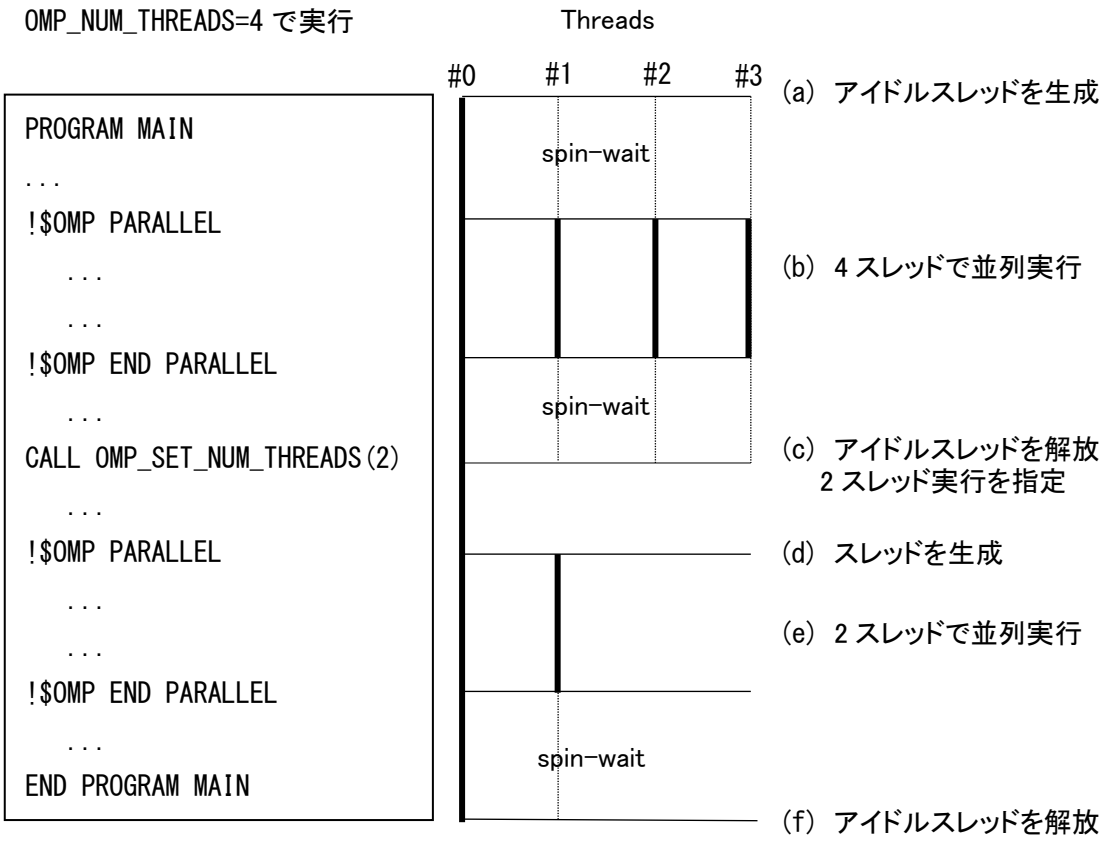
自動並列化されたプログラムでは、OpenMP並列機能をベースに並列処理を実現しています。このため、自動並列化されたプログラム、OpenMP並列化されたプログラムの実行では、環境変数**OMP_NUM_THREADS**、または、**VE_OMP_NUM_THREADS**により実行に使用するスレッド数を指定できます。
また、自動並列化されたプログラムにおいても、OpenMPの実行時ルーチンで、スレッド数を指定、取得できます。

```
SUBROUTINE OMP_SET_NUM_THREADS(num_threads)      // スレッド数の設定
INTEGER num_threads
INTEGER FUNCTION OMP_GET_NUM_THREADS ()          // スレッド数の取得
INTEGER FUNCTION OMP_GET_MAX_THREADS()           // 利用可能スレッド数の取得
INTEGER FUNCTION OMP_GET_THREAD_NUM()            // スレッド番号の取得
```

プログラムの実行開始前に環境変数**OMP_NUM_THREADS**、または、**VE_OMP_NUM_THREADS**によりスレッド数が指定されなかったとき、プログラムで利用可能なVEコア数と同じスレッド数でプログラムの実行が開始されます。

7.3.2　スレッドの生成・解放

自動並列化されたプログラム、OpenMP並列化されたプログラムの実行では、主プログラムの実行開始前にスレッドが自動生成され、プログラムの終了時に解放されます。
次の図を参照してスレッドの生成、解放について説明します。環境変数**OMP_NUM_THREADS**に4が設定されているものとします。



(a) 主プログラムの開始前に、マスタースレッド(#0)により三つのアイドルスレッドが生成されます。アイドルスレッドは、スピンウェイトしながら仕事が割り当てられるのを待ちます。

(b) 並列リージョンに到達すると仕事が割り当てられ、計算が並列実行されます。並列リージョンが終了すると、スレッドはスピンウェイトを開始し、再度仕事が割り当てられるのを待ちます。

(c) 実行時ルーチンOMP_SET_NUM_THREADS(2)で、以降の並列リージョンを2スレッドで実行するよう指定しています(ICV値を2に変更)。ここですべてのアイドルスレッドが解放されます。

(d) 並列リージョンの開始直前で一つのスレッドが生成され、二つのスレッドで並列リージョンの実行が開始されます。

(e) 並列リージョンを2スレッドで並列実行します。

(f) プログラム終了直前に、すべてのアイドルスレッドが解放され、プログラムを終了します。

備考　自動並列化されたプログラムの実行時の解析情報をPROGINF、FTRACEを使って出力するとき、以下の点に注意してください。

- 主プログラムの実行開始前に自動生成されるスレッドのスピンウェイト分の演算数等はPROGINFでは加算されますが、ftraceでは加算されません。
- PROGINFはプロセス全体のカウンタ値から算出した値を表示するため、主プログラムの実行開始前に自動生成されるスレッドのスピンウェイトする部分の演算数等が加算され、総演算数が増加することによりベクトル演算率が低下することがあります。

PROGINF、FTRACEの詳細については「PROGINF/FTRACE ユーザーズガイド」を参照してください。

7.3.3　スレッド生成の遅延

既定の動作では主プログラムの開始前にスレッドが生成されますが、リンク時に次のコンパイラオプションを指定することで、最初に実行される並列リージョンの直前まで、スレッドの生成を遅延させることができます。

```
$ nfort -fopenmp -mno-create-threads-at-startup -static-nec a.o
$ nfort -mparallel -mno-create-threads-at-startup -static-nec b.o
```

7.4　注意事項

- 並列処理時の総CPU時間は、並列処理のオーバーヘッドにより増加します。
- 手続呼出しを含む手続を並列化するとき、共有データの定義、参照が不正にならないかどうか、呼び出される手続の中まで調べなければなりません。
- **-fopenmp**と**-mparallel**が同時に指定されたとき、ループがOpenMPの並列区間の外側にある場合は外側のループが自動並列化の対象となります。OpenMPディレクティブを含むルーチンを自動並列化したくないとき、**-mno-parallel-omp-routine**を指定してください。

第8章　　コンパイルリスト

本章ではFortranコンパイラの出力するコンパイルリストについて説明します。
コンパイルリストは、「*ソースファイル名*.L」という名前でカレントディレクトリに出力されます。

8.1　オプションリスト

オプションリストは、**-report-option**、または、**-report-all**が指定されたとき出力されます。
形式:

```
NEC Fortran Compiler (3.0.7) for Vector Engine   Thu Jun 18 13:25:29 2020   (a)

FILE NAME: fft.f90     (b)

  COMPILER OPTIONS : -report-option    (c)

OPTIONS DIRECTIVE: -O4     (d)

PARAMETER :

Optimization Options :
  (e)                                    (f)
  -On                                   : 4
-fargument-alias                      : disable
-fargument-noalias                        : enable
-fassociative-math                        : enable
```

(a)　オプションリストを作成したコンパイラ、および、リストの作成時刻
(b)　対応するソースファイルの名前
(c)　コマンドラインで指定されたコンパイラオプション
(d)　オプション指示行で指定されたコンパイラオプション
(e)　コンパイラオプション
(f)　コンパイラオプションの値

8.2　診断メッセージリスト

診断メッセージリストは、**-report-diagnostics**、または、**-report-all**が指定されたとき出力されます。

8.2.1　形式

診断メッセージリストの形式は以下のとおりです。

形式:

```
NEC Fortran Compiler (1.0.0) for Vector Engine   Wed Jan 17 14:58:49 2018   (a)

FILE NAME: fft.f90        (b)

PROCEDURE NAME: FFT_3D    (c)
DIAGNOSTIC LIST

 LINE              DIAGNOSTIC MESSAGE
   (d)       (e)        (f)
     7: inl(1222): Inlined
     9: par(1801): Parallel routine generated.: MAIN_$1
     9: vec( 101): Vectorized loop.
```

(a) 診断メッセージを作成したコンパイラ、および、リストの作成時刻

(b)　対応するソースファイルの名前

(c)　診断メッセージに対応するループ、文が含まれる手続の名前

(d)　行番号

(e)　診断メッセージの種別とメッセージ番号

メッセージの種別は以下のとおり。

vec：ベクトル化診断メッセージ

opt：最適化診断メッセージ

inl　：インライン展開診断メッセージ

par：並列化診断メッセージ

(f)　診断メッセージ

8.2.2　注意事項

- あるルーチンにインライン展開されたルーチンの文、ループ対する診断メッセージは、それを呼び出したルーチンの診断メッセージリストには出力されません。インライン展開されたルーチンの診断メッセージについては、そのルーチン自身の診断メッセージリストを

参照してください。

8.3　編集リスト

編集リストは、**-report-format**、または、**-report-all**が指定されたとき出力されます。リストにはソース行とともに、次の情報が手続ごとに出力されます。

- ループのベクトル化情報
- ループの並列化情報
- 手続呼出しのインライン展開情報

8.3.1　形式

編集リストの形式は以下のとおりです。

```
NEC Fortran Compiler (1.0.0) for Vector Engine   Wed Jan 17 15:00:01 2018   (a)
FILE NAME: a.f90     (b)

PROCEDURE NAME: SUB    (c)
FORMAT LIST

 LINE   LOOP      STATEMENT
  (d)    (e)      (f)
     1:           SUBROUTINE SUB(A, B, N, M)
     2:           INTEGER::N, M
     3:           REAL(KIND=8)::A(M, N), B(M, N)
     4: +------>  DO J=1,M
     5: |V----->     DO I=1, N
     6: ||              A(I,J) = A(I,J) + B(I,J)
     7: |V-----      ENDDO
     8: +------   ENDDO
     9:           END SUBROUTINE
```

(a) 編集リストを作成したコンパイラ、および、リストの作成時刻

(b)　対応するソースファイルの名前

(c)　以降のリスト中のソースコードの含まれる手続の名前

(d)　行番号

(e)　ループのベクトル化、並列化、インライン展開に関する情報

(f)　対応するコード

8.3.2　ループのベクトル化、並列化、インライン展開に関する情報

ループの構造、および、ベクトル化、並列化、インライン展開に関する情報を記号で出力します。出力例を以下に示します。

- ループ全体がベクトル化されたとき

```
V------> DO I=1,N
|
V------  END DO
```

- ループが部分ベクトル化されたとき

```
S------> DO I=1,N
|
S------  END DO
```

- ループが条件ベクトル化されたとき

```
C------> DO I=1,N
|
C------  END DO
```

- ループが並列化されたとき

```
P------> DO I=1,N
|
P------  END DO
```

- ループが並列化、かつ、ベクトル化されたとき

```
Y------> DO I=1,N
|
Y------  END DO
```

- ループがベクトル化されなかったとき

```
+------> DO I=1,N
|
+------  END DO
```

- 配列式など、一行にループ全体が含まれるとき

```
V======> A = A + B
```

- ループが一重化されたとき

```
W------> DO I=1,N
|*----->     DO J=1,M
||
|*-----      END DO
W------  END DO
```

- ループが入れ換えられ、ベクトル化されたとき

```
X------> DO I=1,N
|*----->     DO J=1
||
|*-----      END DO
X------  END DO
```

- 外側ループが外側ループアンロールされ、内側ループがベクトル化されたとき

```
U------> DO I=1,N
|V----->     DO J=1
||
|V-----      END DO
U------  END DO
```

- ループが融合されたとき

```
V------> DO I=1,N
|
|         END DO
|         DO I=1,N
|
V------  END DO
```

- ループが展開されたとき

```
*------> DO I=1,4
|
*------  END DO
```

- 17カラム目に表示される文字は行がどう最適化されたかを表す
  - “I” 関数呼び出しがインライン展開された
  - “M” この行を含む多重ループがベクトル行列積ライブラリ呼び出しに置き換えられた
  - “F” 式に対してベクトル積和命令が生成された
  - “R” 配列に**retain**指示行が適用された
  - “G” ベクトル収集命令が生成された
  - “C” ベクトル拡散命令が生成された
  - “V” 配列に**vreg**指示行、または、**pvreg**指示行が適用された

8.3.3　注意事項

- 内部副プログラムはそれを含むプログラム単位に含めて出力されます。
- ループの一部が**INCLUDE**行、**#include**で取り込まれるファイルに含まれているとき、ループの構造、情報が正しく出力されないことがあります。
- 一つの行にループが複数存在するとき、ループの構造、情報が正しく出力されないことがあります。

8.4　モジュール別最適化情報リスト

インライン展開モジュール、ベクトル化モジュール、コード生成モジュールの最適化情報を含んだリストを出力できます。

8.4.1　インライン展開モジュール

インライン展開モジュールの最適化情報リストは、**-report-inline**、または、**-report-all**が指定されたとき出力されます。

形式:

```
NEC Fortran Compiler (3.1.0) for Vector Engine     Thu Sep 17 07:33:16 2020  (a)

FILE NAME: fft.f90      (b)

FUNCTION NAME: func3    (c)

INLINE LIST

  INLINE REPORT: func3 (fft.f90:17)
```

```
 (d)
-> INLINE: func2 (fft.f90:19)              (e)
   -> NOINLINE: func0 (fft.f90:12)         (e)
      *** Source for routine not found.    (f)
   -> INLINE: func1 (fft.f90:13)           (e)
```

(a) インライン展開リストを作成したコンパイラ、および、リストの作成時刻

(b)　対応するソースファイルの名前

(c)　インライン展開状況を表示する手続の名前

(d)　インライン展開する手続の深さ

(e)　インライン展開の結果

(f)　診断メッセージ

8.4.2　ベクトル化モジュール

ベクトル化モジュールの最適化情報リストは、**-report-vector**、または、**-report-all**が指定されたとき出力されます。

形式:

```
NEC Fortran Compiler (3.1.0) for Vector Engine     Thu Sep 17 08:10:39 2020  (a)

FILE NAME: vec.f90      (b)

FUNCTION NAME: func     (c)

VECTORIZATION LIST

  LOOP BEGIN: (vec.f90:3)
    <Unvectorized loop.>              (d)

    LOOP BEGIN: (vec.f90:4)
      <Vectorized loop.>              (d)
      *** The number of VGT,   VSC.    :  0,  0. (vec.c:4)          (e)
      *** The number of VLOAD, VSTORE. :  1,  1. (vec.c:4)          (e)
    LOOP END
  LOOP END
```

(a) ベクトルリストを作成したコンパイラ、および、リストの作成時刻

(b)　対応するソースファイルの名前

(c)　ベクトル化状況を表示する手続の名前

(d)　ベクトル化の結果

(e)　診断メッセージ

8.4.3　コード生成モジュール

コード生成モジュールの最適化情報リストは、**-report-cg**、または、**-report-all**が指定されたとき出力されます。

形式:

```
NEC Fortran Compiler (3.1.0) for Vector Engine     Thu Sep 17 08:10:39 2020  (a)

FILE NAME: vec.f90      (b)

FUNCTION NAME: func     (c)

CODE GENERATION LIST

  Hardware registers    (d)
    Reserved                  : 10 [sl fp lr sp s12 s13 tp got plt s17]
    Callee-saved              : 16 [s18-s33]
    Assigned
      Scalar registers        : 32 [s0-s12 s15-s16 s18-s21 s23-s32 s61-s63]
      Vector registers        : 35 [v0 v30-v63]
      Vector mask registers   :  0
      VREG directive          :  2 [v18-v19]

  Routine stack         (e)
    Total size                : 256 bytes
      Register spill area     :  16 bytes
      Parameter area          :  40 bytes
      Register save area      : 176 bytes
      User data area          :  16 bytes
      Others                  :   8 bytes

    Note: Total size of Routine stack does not include
          the size extended by alloca() and so on.


  LOOP BEGIN: (vec.f90:3)
    LOOP BEGIN: (vec.f90:4)
```

```
      *** The number of VECTOR REGISTER SPILL  (f)
            Total                                    : 14
              Across calls                           : 11
              Not enough registers                   : 1
              Over basic blocks                      : 1
              Others                                 : 1
      *** The number of VECTOR REGISTER RESTORE
            Total                                    : 14
              Across calls                           : 11
              Not enough registers                   : 1
              Over basic blocks                      : 1
              Others                                 : 1
      *** The number of VECTOR REGISTER TRANSFER      : 12

      *** The number of SCALAR REGISTER RESTORE
            Total                                    : 14
              Across calls                           : 11
              Not enough registers                   : 1
              Over basic blocks                      : 1
              Others                                 : 1
      *** The number of SCALAR REGISTER RESTORE
            Total                                    : 14
              Across calls                           : 11
              Not enough registers                   : 1
              Over basic blocks                      : 1
              Others                                 : 1
      *** The number of SCALAR REGISTER TRANSFER      : 21
    LOOP END
  LOOP END
```

(a) コード生成リストを作成したコンパイラ、および、リストの作成時刻

(b)　対応するソースファイルの名前

(c)　コード生成状況を表示する手続の名前

(d)　レジスタの種類ごとのレジスタ使用数

Reserved　　：システムで予約済のレジスタ

Callee-saved　　：手続呼出しを跨いでセーブするレジスタ

Assigned　　：計算、ユーザデータに割り当てられたレジスタ

(e)　スタック情報

Register spill area　：レジスタ退避領域
Parameter area　：引数領域
Register save area　：レジスタセーブ領域
User data area　：ユーザデータ
Others　：その他

(f)　ループ毎のレジスタスピル・リストア・転送の原因

Across calls　：手続呼出しを跨いでいるため
Not enough registers　：レジスタが枯渇しているため
Over basic blocks　：ベーシックブロックを跨いで使用されているため
Others　：その他

第9章　　言語仕様に関する補足

9.1　非標準機能拡張

9.1.1　文

9.1.1.1 COMMON 文

一つの共通ブロックの中に文字型の要素と他の型の要素を混在できます。ただし、実行効率が低下することがあるため、できるだけ混在を避けるのが望ましいです。

9.1.1.2 COMPLEX DOUBLE 文 / COMPLEX DOUBLE PRECISION 文

COMPLEX DOUBLE文、**COMPLEX DOUBLE PRECISION**文で、倍精度複素数型のデータを宣言できます。

種別パラメタ値は、"KIND(0.0D0)"とします。

形式

COMPLEX DOUBLE データ要素宣言並び

COMPLEX DOUBLE PRECISION データ要素宣言並び

ここで、データ要素宣言 ：

実体名[(明示上下限並び)] [/初期値/]

|　実体名[(大きさ引継ぎ配列形状指定)] [/初期値/]

|　関数名

9.1.1.3 COMPLEX QUADRUPLE 文 / COMPLEX QUADRUPLE PRECISION 文

COMPLEX QUADRUPLE文、および、**COMPLEX QUADRUPLE PRECISION**文は、互換用の型宣言文で、その文で宣言された名前のデータ要素が、すべて組込みの4倍精度複素数型であることを指定します。

種別パラメタ値は、"KIND(0.0Q0)"とします。

形式

COMPLEX QUADRUPLE データ要素宣言並び

COMPLEX QUADRUPLE PRECISION データ要素宣言並び

ここで、データ要素宣言 ：

実体名[(明示上下限並び)] [/初期値/]

|　実体名[(大きさ引継ぎ配列形状指定)] [/初期値/]

|　関数名

9.1.1.4 DATA 文

初期値定数に4文字より長いホラリス定数を指定できます。

9.1.1.5 DIMENSION 文

DATA文や型宣言文と同様に、**DIMENSION**文でも初期値を設定できます。

形式

DIMENSION 配列名(配列形状指定)[/初期値表現並び/]

[,配列名(配列形状指定)[/初期値表現並び/]]...

ここでの初期値表現並びは直前の配列名の初期値を表します。

初期値を設定するときの規則は**DATA**文と同じです。

9.1.1.6 DOUBLE 文

DOUBLE文は、互換用の型宣言文で、その文で宣言された名前のデータ要素が、すべて組込みの倍精度実数型であることを指定します。

種別パラメタ値は、"KIND(0.0D0)"とします。

形式

DOUBLE データ要素宣言並び

ここで、データ要素宣言 ：

実体名[(明示上下限並び)] [/初期値/]

| 実体名[(大きさ引継ぎ配列形状指定)] [/初期値/]

| 関数名

9.1.1.7 DOUBLE COMPLEX 文

DOUBLE COMPLEX文は、互換用の型宣言文で、その文で宣言された名前のデータ要素が、すべて組込みの倍精度複素数型であることを指定します。

種別パラメタ値は、"KIND(0.0D0)"とします。

形式

DOUBLE COMPLEX データ要素宣言並び

ここで、データ要素宣言 ：

実体名[(明示上下限並び)] [/初期値/]

| 実体名[(大きさ引継ぎ配列形状指定)] [/初期値/]

| 関数名

9.1.1.8 DOUBLE PRECISION 文

DOUBLE PRECISION文でデータ要素の初期値を指定できます。

形式

DOUBLE PRECISION [[, 属性指定子] ... ::] データ要素宣言並び

ここで、属性指定子 ：

ALLOCATABLE

| DIMENSION(配列形状指定)
| EXTERNAL
| INTENT(授受特性指定)
| INTRINSIC
| OPTIONAL
| PARAMETER
| POINTER
| PRIVATE
| PUBLIC
| SAVE
| TARGET

データ要素宣言 ：

実体名[(明示上下限並び)] [/初期値/]
| 実体名[(大きさ引継ぎ配列形状指定)] [/初期値/]
| 関数名

9.1.1.9 EQUIVALENCE 文

文字型の要素と派生型を除く他の型の要素を結合できます。ただし、他のFortranとの互換上は、なるべくこれを避ける方が望ましいです。

9.1.1.10 FORMAT 文

FORMAT文の書式項目並び中の書式項目を区切るコンマは、文字列編集記述子の前後で省略できます。ただし、X形編集記述子と文字列編集記述子との間のコンマは省略できません。また、*n*X形編集記述子の*n*と*k*P形編集記述子の*k*を省略できます。省略したとき、暗黙値は1とします。さらにデータ編集記述子(B/D/E/EN/ES/F/G/I/L/O/Z)で編集記述子のみを指定することができます。

例

```
      PRINT 10, 3.14, 2.71
      PRINT 20, 3.14, 2.7110
      FORMAT('PI='F4.2' and',X,'E='F4.2)
20    FORMAT('PI='F' and',X,'E='F)
```

この出力は以下となります。

```
PI=3.14 and E=2.71
PI=       3.1400001 and E=       2.7100000
```

9.1.1.11 FUNCTION 文

FUNCTION文では、関数名に続く"([*仮引数名並び*])"を括弧も含めて省略できます。

次の形式で**FUNCTION**文を利用できます。

形式

[型指定子] FUNCTION 関数名 [([仮引数名並び])]

ここで、型指定子 ：

INTEGER[*バイト数]
| REAL[*バイト数]
| DOUBLE PRECISON
| DOUBLE
| QUADRUPLE PRECISON
| QUADRUPLE
| COMPLEX[*バイト数]
| COMPLEX DOUBLE PRECISON
| COMPLEX DOUBLE
| DOUBLE COMPLEX
| COMPLEX QUADRUPLE PRECISON
| COMPLEX QUADRUPLE
| LOGICAL[*バイト数]
| CHARACTER[*文字長]

9.1.1.12 計算型 GO TO 文

以下の計算型GO TO文を利用できます。

形式

GO TO(文番号並び)[,]スカラ整数式

構文規則

文番号並び中の文番号は、その計算形**GO TO**文と同じ有効域内にある、飛び先文の文番号でなければなりません。

一般規則

(1) 一つの文番号並び中に、同じ文番号を2回以上書けます。

(2) 計算形**GO TO**文を実行すると、そのスカラ整数式が評価されます。この値をiとし、文番号並び中の文番号の個数をnとします。1≦i≦nであるときには制御の移行が起こり、文番号並び中のi番目にある文番号をもつ文が次に実行されます。iが1より小さいとき、および、nより大きいとき、プログラムは**CONTINUE**文が実行されたかの

ように継続されます。

例

```
GO TO(100,200,300,400,500), I
```

9.1.1.13算術 IF 文

次の形式のの算術**IF**文を利用できます。

形式

IF(スカラ数値式) 文番号, 文番号, 文番号

構文規則

(1) 各*文番号*は、その算術**IF**文と同じ有効域内にある、飛び先文の文番号でなければなりません。

(2) *スカラ数値*式は複素数型であってはなりません。

(3) *文番号*を最大二つまで省略できます。ただし、コンマは省略できません。*スカラ数値式*に対応する*文番号*が省略されているとき、プログラムは**CONTINUE**文が実行されたかのように継続されます。

また、*文番号*が一つでも省略された算術**IF**文は、**DO**ループの端末文として使用できます。

一般規則

(1) 一つの算術**IF**文中に、同じ文番号を2回以上指定できます。

(2) 算術**IF**文を実行するとスカラ数値式が評価され、続いて制御の移行が起こります。スカラ数値式の値が負であるとき、ゼロであるとき、および、正であるときに、それぞれ1番目、2番目、および、3番目の文番号によって識別された飛び先文が次に実行されます。

例

```
IF( I + J ) 100,200,300
```

9.1.1.14IMPLICIT 文

一つの有効域内のすべての**IMPLICIT**文を通じて、同じ英字を、単独の英字として書くか英字範囲に含めたりして、2回以上指定できます。2回以上指定したとき、後ろで指定したものが有効となります。

"$" を1文字目にもつ名前のデータ要素の暗黙的な型、および、型パラメタを指定できます。

9.1.1.15 PARAMETER 文

PARAMETER文で並びの括弧を省略できます。省略したとき、変数のデータ型は、名前の暗黙的な型付けではなく、定数の形式によって決定されます。

例

```
PARAMETER PI=3.1415927, DPI=3.14159265358979323D0
PARAMETER PIOV2=PI/2, DPIOV2=DPI/2
PARAMETER FLAG=.TRUE., LONGNAME='A STRING OF 25 CHARACTERS'
PRINT *,'PI=',PI
PRINT *,'DPI=',DPI
PRINT *,'PIOV2=',PIOV2
PRINT *,'DPIOV2=',DPIOV2
PRINT *,'FLAG=',FLAG
PRINT *,'LONGNAME=',LONGNAME
END
```

この出力は以下となります。

```
PI=   3.1415927
 DPI=   3.1415926535897931
 PIOV2=   1.5707964
 DPIOV2=   1.5707963267948966
 FLAG= T
 LONGNAME=A STRING OF 25 CHARACTERS
```

9.1.1.16 FORTRAN77 POINTER 文

以下の互換用の**POINTER**文をサポートしています。

概要

POINTER文は、組込み型変数にポインタ変数を対応づける機能をもちます。型宣言文が現れることのできる場所に記述できます。

形式

POINTER (ポインタ変数, データ変数宣言) [,(ポインタ変数, データ変数宣言)]...

ここで、ポインタ変数 : スカラ8バイト型変数

データ変数宣言 :　　スカラ変数名

| 配列名

| 配列名(明示上下限並び)

| 配列名(大きさ引継ぎ配列形状指定並び)

一般規則

(1)　**POINTER** 文は、モジュールの宣言部と初期値設定プログラムに記述できません。

(2)　ポインタ変数は、スカラ変数でなければなりません。

(3)　ポインタ変数は、**ALLOCATABLE** 属性をもってはなりません。

(4)　ポインタ変数に、８バイト整数型以外の型を宣言してはなりません。

(5)　ポインタ変数は、**POINTER** 属性、または、**TARGET** 属性をもってはなりません。

(6)　ポインタ変数は、派生型の要素であってはなりません。

(7)　ポインタ変数は、**PARAMETER** 文、または、**PARAMETER** 属性を含む型宣言文に現れてはなりません。

(8)　ポインタ変数は、**DATA** 文に現れてはなりません。

(9)　データ変数宣言に、形状引継ぎ配列は指定できません。

(10) データ変数宣言は、**ALLOCATABLE**、**INTENT**、**OPTIONAL**、**DUMMY**、**TARGET**、**INTRINSIC**、または、**POINTER** 属性をもってはなりません。

(11) データ変数宣言は、複数の **POINTER** 文に現れてはなりません。

(12) データ変数宣言は、ポインタ変数であってはなりません。

(13) データ変数宣言の配列形状指定は、明示上下限並びか、または、大きさ引継ぎ配列でなければなりません。

(14) データ変数宣言は、**SAVE**、**DATA**、**EQUIVALENCE**、**COMMON**、または、**PARAMETER** 文に現れてはなりません。

(15) データ変数宣言に、構造体、または、構造体の要素が現れてはなりません。

(16) データ変数宣言は、組込み型でなければなりません。

(17) データ変数宣言は、共通ブロックの実体名、仮引数名、関数結果名、または、自動割付け実体名であってはなりません。

(18) データ変数宣言には、初期値を与えてはなりません。

備考

(a) ポインタ変数の扱いは、通常の８バイト整数型変数と同じです。

(b)　ポインタ変数に型が宣言されないとき、暗黙の型には従わず、８バイト整数型に設定されます。

(c)　ポインタ変数は、複数のデータ変数宣言に対して宣言できます。

(d)　データ変数宣言の配列形状指定は、上下限が定数以外のとき、配列の大きさはその副プログラムが呼び出された時点で決定されます。

(e)　データ変数宣言には、記憶域は割り付けられません。実際のアドレスは、対応するポインタ変数の値をバイトアドレスとみなして動的に決定されます。

(f)　データ変数宣言が配列のとき、型宣言文、**DIMENSION** 文、または、**POINTER** 文のいずれかによって、配列形状を指定できます。

(g)　データ変数宣言は、親子結合をしません。

9.1.1.17 QUADRUPLE 文 / QUADRUPLE PRECISION 文

QUADRUPLE文、および、**QUADRUPLE PRECISION**文は、互換用の型宣言文で、その文で宣言された名前のデータ要素が、すべて組込みの4倍精度実数型であることを指定します。

種別パラメタ値は、"KIND(0.0Q0)"とします。

形式

QUADRUPLE データ要素宣言並び

QUADRUPLE PRECISION データ要素宣言並び

ここで、データ要素宣言 ：

実体名[(明示上下限並び)] [/初期値/]

|　実体名[(大きさ引継ぎ配列形状指定)] [/初期値/]

|　関数名

9.1.1.18 RETURN 文

RETURN文のスカラ整数式に、実数型の式を指定できます。

指定された実数型の式は、制御の移行の前に整数式に変換されます。

9.1.1.19 STOP 文

STOP文の終了符号として、文字型、または、基本整数型のスカラ変数名、および、定数名を指定できます。

9.1.2　プログラム

9.1.2.1　継続行

プログラム形式によらず、継続行は開始行と合わせて最大511行まで継続できます。

9.1.2.2　文字$

名前の英字の代わりに通貨記号($)を使用できます。

また、書式付き入出力の編集記述子に通貨記号($)を使用できます。$編集記述子は、出力において、書式制御の最後の記録の行送り制御を抑止することを指定します。$編集記述子を入

力に指定すると無視されます。

9.1.2.3 引数の結合

引用仕様が明示的でない手続は、引数が以下の場合でも引数結合のエラーとせずコンパイルできます。

- 実引数の数が仮引数の数より少ない。
- 引数が文字型で、実引数より仮引数の方が文字長が大きい。

9.1.3 プログラム形式

9.1.3.1 固定形式

(1) 文の継続

第1けたに "&" を書いたとき、その行の第2けた以降は、先行する注釈でない行の継続行とします。

(2) 拡張固定形式

拡張固定形式では、1行の最大長は2048文字です。1行が72文字固定でなく、最大2048文字まで可変であること以外、固定形式と同じです。

拡張固定形式では、開始行と合わせて13200文字まで継続できます。

標準のFortran 2008言語仕様ではソース形式によらず、255行まで継続できます。

-fextend-sourceを指定したとき、拡張固定形式が有効となります。

(3) タブコード行

最初のタブコードが、第1けたから第6けたに現れたとき、最初のタブコードの次の文字が数字ならその文字は第6けたに現れたとみなされます。

また、最初のタブコードの次の文字が数字以外なら、その文字は第7けたに現れたとみなされます。このとき、行の最後の文字までが文の一部となります。

また、最初のタブコードが、第7けた以降に現れたとき、文字定数、ホラリス定数、および、文字列編集記述子を除いて空白とみなされます。

9.1.3.2 自由形式

自由形式では、1行の最大長に制限はありません。

9.1.4 式

9.1.4.1 関係演算子

互換のために、以下の関係演算子を使用できます。

=>

| =<

| ><

| <>

9.1.4.2 論理演算子

互換のために、下記の論理演算子を使用できます。

.XOR.

.XOR.の演算の解釈は、.NEQV.と同じで、優先順位は.OR.と同じです。

9.1.4.3 配列の最大次元数

配列の次元数は最大31まで宣言できます。Fortran 2008標準では最大15、それ以前のFortran標準では最大7です。

9.1.4.4 非 10 進定数表現

引用符、または、アポストロフィがある形式の非10進定数表現は、以下の場所にも記述できます。

- **PARAMETER**文の初期値
- 型宣言文の初期値
- 暗黙的引用仕様をもつ手続の実引数

このとき非10進定数は、型変換されず、その使われ方に応じた型として扱われます。

非10進定数のサイズが、扱われる型のサイズより小さいとき、左に0が補われます。

非10進定数のサイズが、扱われる型のサイズより大きいとき、左端が切り捨てられます。

16進定数表現は、Zの代わりにXを用いた以下の表現でも書けます。

X"16進数字[16進数字]..."

| X'16進数字[16進数字]...'

9.1.4.5 ホラリス型

ホラリス定数を、ホラリス関係式、または、ホラリス代入文で指定できます。

(1) ホラリス関係式

関係式において、一方のオペランドがホラリス定数、または、文字定数のとき、他方のオペランドに整数型、または、実数型のスカラ変数を指定できます。これによりホラリスデータの比較ができます。その変数は、その関係式評価時には、ホラリスデータで確定していなければなりません。ホラリス関係式の解釈は、同一の文字値をもつ文字関係式の解釈と同じです。

例

```
INTEGER DATA
READ(*, 10) DATA
```

```
10     FORMAT(A4)
IF( DATA .EQ. 3HEND ) STOP
```

(2)　ホラリス代入文

代入文において、右辺がホラリス定数、または、文字定数のとき、左辺に文字型を除く任意の型のスカラ変数を指定できます。この代入文の実行により、左辺の変数は右辺のホラリスデータで確定されます。

ホラリス定数、または、文字定数における文字数をn、左辺の変数が含むことができる文字数をgとすると、nがg未満のとき、その要素は定数の右端を(g-n)個の空白により拡張したg個の文字が割り当てられます。gがn未満のとき、その要素は定数の左端のg個の文字が割り当てられます。

例

```
INTEGER TITLE
TITLE = 4HDATA
WRITE(*, 10) TITLE
10 FORMAT(A4)
```

9.1.4.6　添字式と文字位置式

配列要素の添字式、および、文字位置式に実数型の式を指定できます。

指定された実数型の式は、添字の値を計算する前に基本整数型に変換されます。

9.1.5　廃止事項

Fortran 95言語仕様での廃止事項(**PAUSE**文、**ASSIGN**文、および、割当て形**GO TO**文、H形編集記述子)をサポートしています。-**Wobsolescent**を指定したとき、これらの廃止事項がみつかると、その都度'Deleted feature:'の付された警告メッセージが出力されます。

9.2　処理系定義

9.2.1　型

9.2.1.1　種別パラメタと対応するデータ宣言との関係

Fortranコンパイラで指定可能な種別パラメタと対応するデータ宣言との関係は、以下のとおりです。

型	種別パラメタ	対応するデータ宣言
整数型	1	1バイト整数型

整数型	2	2バイト整数型
整数型	4	4バイト整数型(基本整数型)
整数型	8	8バイト整数型
実数型	4	実数型(基本実数型)
実数型	8	倍精度実数型
実数型	16	4倍精度実数型
複素数型	4	複素数型(基本複素数型)
複素数型	8	倍精度複素数型
複素数型	16	4倍精度複素数型
論理型	1	1バイト論理型
論理型	4	4バイト論理型(基本論理型)
論理型	8	8バイト論理型
文字型	1	文字型

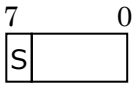
9.2.2　データの内部表現

9.2.2.1　整数型データ

整数型データは、記憶域内で連続した1バイト、2バイト、4バイト、または、8バイトを占め、それぞれ最下位のビットを1の位とした2進形式で表現されます。負の数は2の補数によって表現されます。最上位のビットは符号ビットで、0ならば正、または、0であり、1ならば負です。

(1)　1 バイト整数型

形式



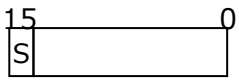
S:符号ビット

表現可能な値

-128 ～ 127 (-2^7 ～ 2^{7-1})

(2)　２バイト整数型

形式



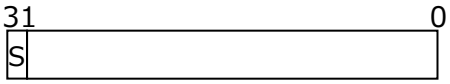
S:符号ビット(0:正 1:負)

表現可能な値

-32768 ～ 32767 (-2^{15} ～ 2^{15-1})

(3)　４バイト整数型

形式



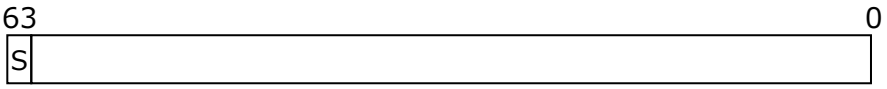
S:符号ビット(0:正 1:負)

表現可能な値

-2147483648 ～ 2147483647 (-2^{31} ～ 2^{31-1})

(4)　８バイト整数型

形式



S:符号ビット(0:正 1:負)

表現可能な値

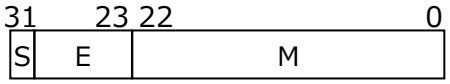
-9223372036854775808 ～ 9223372036854775807 (-2^{63} ～ 2^{63-1})

9.2.2.2　実数型データ

(1)　単精度実数型

単精度実数型データは、記憶域内で連続した4バイトを占めます。最上位のビットは仮数部の符号ビットです。下位の23ビットは仮数部を表し、その上位端のビット(ビット22)を2^{-1}の位とする2進形式で格納され、仮数部の符号ビットが0ならば正の値を表し、1ならば絶対値を表します。下位9ビットのうち、上位側の8ビットは指数部を表し、その下位端のビット(ビット23)を1の位とする2進形式で格納されます。値ゼロは指数部の値が0であることによって表現されます。

形式



S:仮数部の符号ビット(0:正 1:負)

E:指数部 (0≦E≦255)

M:仮数部 (0≦M＜1)

表現可能な値

$(-1)^S * 2^{E-127} * 1.M$

有効けた数は10進約7けたで、絶対値が0、または、約10^{-38}～10^{37}の範囲の値を表せます。

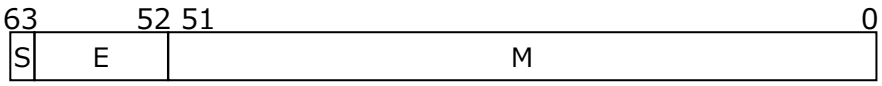
特別な値

NaN	E==255、かつ、M!=0 (Mの先頭ビットが0:signaling NaN、Mの先頭ビットが1:quiet NaN)
無限大	E==255、かつ、M==0
符号付きゼロ	E==0

(2) 倍精度実数型

倍精度実数型データは、記憶域内で連続した8バイトを占めます。最上位のビットは仮数部の符号ビットです。下位の52ビットは仮数部を表し、その上位端のビット(ビット51)を2^{-1}の位とする2進形式で格納され、仮数部の符号ビットが0ならば正の値を表し、1ならば絶対値を表します。上位12ビットのうち、下位側の11ビットは指数部を表し、その下位端のビット(ビット52)を1の位とする2進形式で格納されます。値ゼロは指数部の値が0であることによって表現されます。

形式



S:仮数部の符号ビット(0:正 1:負)

E:指数部 (0≦E≦2047)

M:仮数部 (0≦M＜1)

表現可能な値

$(-1)^S * 2^{E-1023} * 1.M$

有効けた数は10進約16けたで、絶対値が0、または、約10^{-308}～10^{308}の範囲の値を表せます。

特別な値

- NaN　　E==2047、かつ、M!=0
  (Mの先頭ビットが0:signaling NaN、Mの先頭ビットが1:quiet NaN)
- 無限大　　E==2047、かつ、M==0
- 符号付きゼロ　E==0

(3) 4倍精度実数型

4倍精度実数型データは、記憶域内で連続した16バイトを占めます。最上位のビットは仮数部の符号ビットです。下位の112ビットは仮数部を表し、その上位端のビット(ビット111)を2^{-1}の位とする2進形式で格納され、仮数部の符号ビットが0ならば正の値を表し、1ならば絶対値を表します。上位16ビットのうち、下位側の15ビットは指数部を表し、その下位端のビット(ビット112)を1の位とする2進形式で格納されます。値ゼロは指数部の値が0であることによって表現されます。

形式

S:仮数部の符号ビット(0:正 1:負)

E:指数部 (0≦E≦32767)

M:仮数部 (0≦M＜1)

表現可能な値

$(-1)^S * 2^{E-16383} * 1.M$

有効けた数は10進約34けたで、絶対値が0、または、約10^{-4932}～10^{4932}の範囲の値を表せます。

特別な値

- NaN　　E==32767、かつ、M!=0
  (Mの先頭ビットが1:signaling NaN、Mの先頭ビットが0:quiet NaN)
- 無限大　　E==32767、かつ、M==0
- 符号付きゼロ　E1==0

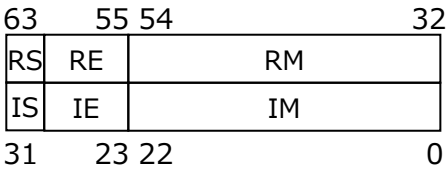
9.2.2.3 複素数型データ

(1) 単精度複素数型

複素数型データは、記憶域内で連続した8バイトを占めます。上位番地を占める4バイトに

は実部が格納され、下位番地を占める4バイトには虚部が格納されます。実部と虚部はそれぞれ実数型データと同じ形式です。

形式



実部RS:仮数部の符号ビット(0:正 1:負)、RE:指数部(0≦RE≦255)、RM:仮数部
虚部IS:指数部の符号ビット(0:正 1:負)、IE:指数部(0≦IE≦255)、IM:仮数部

表現可能な値

$(-1)^{RS} * 2^{RE-127} * 1.RM$、および、$(-1)^{IS} * 2^{IE-127} * 1.IM$
有効けた数は10進約7けたで、絶対値が0、または、約10^{-38}～10^{37}の範囲の値を表せます。

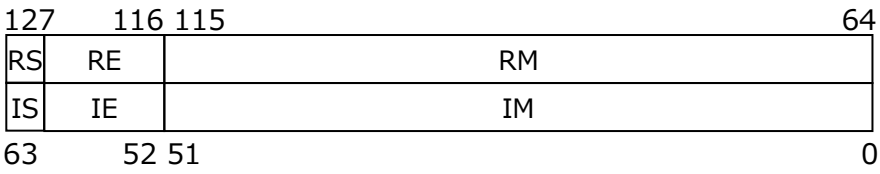
特別な値

- NaN　　RE==255、かつ、RM!=0、または、IE==255、かつ、IM!=0
- 無限大　　RE==255、かつ、RM==0、または、IE==255、かつ、IM==0
- 符号付きゼロ　RE==0、かつ、IE==0

(2)　倍精度複素数型

倍精度複素数型データは、記憶域内で連続した16バイトを占めます。上位番地を占める8バイトには実部が格納され、下位番地を占める8バイトには虚部が格納されます。実部と虚部はそれぞれ倍精度実数型データと同じ形式です。

- **形式**



実部RS:仮数部の符号ビット(0:正 1:負)、RE:指数部(0≦RE≦2047)、RM:仮数部
虚部IS:仮数部の符号ビット(0:正 1:負)、IE:指数部(0≦IE≦2047)、IM:仮数部

表現可能な値

$(-1)^{RS} * 2^{RE-1023} * 1.RM$、および、$(-1)^{IS} * 2^{IE-1023} * 1.IM$
有効けた数は10進約16けたで、絶対値が0、または、約10^{-308}～10^{308}の範囲の値を表せます。

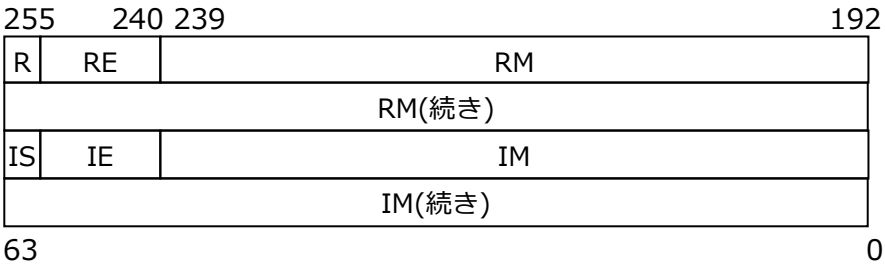
特別な値

- NaN　　RE==2047、かつ、RM!=0、または、IE==2047、かつ、IM!=0
- 無限大　　RE==2047、かつ、RM==0、または、IE==2047、かつ、IM==0
- 符号付きゼロ　RE==0、かつ、IE==0

(3)　4 倍精度複素数型

4倍精度複素数型データは、記憶域内で連続した32バイトを占めます。上位番地を占める16バイトには実部が格納され、下位番地を占める16バイトには虚部が格納されます。実部と虚部はそれぞれ4倍精度実数型データと同じ形式です。

形式



実部RS:仮数部の符号ビット(0:正　1:負)、RE:指数部(0≦RE≦32767)、RM:仮数部
虚部IS:仮数部の符号ビット(0:正　1:負)、IE:指数部(0≦IE≦32767)、IM:仮数部

表現可能な値

$(-1)^{RS} * 2^{RE-16383} * 1.RM$、および、$(-1)^{IS} * 2^{IE-16383} * 1.IM$
有効けた数は10進約34けたで、絶対値が0、または、約10^{-4932}～10^{4932}の範囲の値を表せます。

特別な値

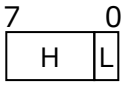
- NaN　　RE==32767、かつ、RM!=0、または、IE==32767、かつ、IM!=0
- 無限大　　RE==32767、かつ、RM==0、または、IE==32767、かつ、IM==0
- 符号付きゼロ　RE==0、かつ、IE==0

9.2.2.4 論理型データ

論理型データは、記憶域内で1バイト、連続した4バイト、または、連続した8バイトを占め、すべてのビットが0のとき「偽」、それ以外は「真」とみなされます。「真」を論理型変数に代入したときの内部表現は、最上位のビットのみを1とします。

(1)　１バイト論理型

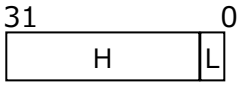
形式



H:上位ビット(H=0) L:最下位ビット(0:偽 1:真)

(2)　４バイト論理型

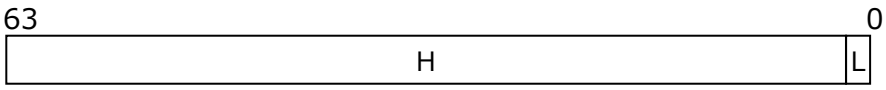
形式



H:上位ビット(H=0) L:最下位ビット(0:偽 1:真)

(3)　８バイト論理型

形式

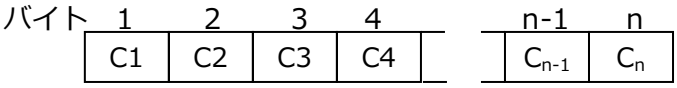


H:上位ビット(H=0) L:最下位ビット(0:偽 1:真)

9.2.2.5 文字型データ

文字型データは、記憶域内で宣言文、または、**IMPLICIT**文で指定された長さだけの連続するバイト、または、文字定数のときにはその文字数だけの連続するバイトを占めます。各バイトにはそれぞれ1文字が格納されます。

形式



Ci　文字データ中の上位からｉ番目の文字

n　型宣言文、または、**IMPLICIT** 文で指定された文字型のスカラ変数、
および、配列要素の長さ(最大 2147483647 文字)。
または、文字定数の長さ(最大 2147483647 文字)。

9.2.2.6 ホラリス型データ

ホラリス型データは、記憶域内で連続した1、2、4、8、16、32バイトを占め、左詰めで格納されます。文字型以外の変数、または、配列要素に格納されますが、いずれのときも必要な数だけ空白が後ろに補われます。

ホラリス定数は、ゼロでない符号なし整定数ｎ、その後ろに文字Hとそれに続くちょうどｎ個の連続した文字の列を書いたものです。ｎ個の文字の列がホラリスデータです。

実引数として使用するときを除き、文字定数をホラリス定数のかわりに指定できます。

例　　5HABCD が倍精度実数型変数に格納されるとき

バイト

1	2	3	4	5	6	7	8
A	B	C	D	E	△	△	△

ここで、△は空白を表します。

ホラリス定数は、ホラリス関係式(一方のオペランドがホラリス定数である関係式)、ホラリス代入文(右辺がホラリス定数である代入文)、FORTRAN77互換形式型宣言文、**DATA**文、**DIMENSION**文、または、明白な引用仕様をもたない手続引用の実引数並びにだけ指定できます。

9.2.2.7　16 進型データ

16進型データは、**DATA**文、型宣言文による初期値設定、または、Z形編集記述子を使用した**READ**文によって記憶域内に右詰めで格納され、対応するデータの型に応じたバイトを占めます。

16進型データは、4ビットで16進数1けたを表現します。

9.2.2.8　8 進型データ

8進型データは、**DATA**文、型宣言文による初期値設定、または、O形編集記述子を使用した**READ**文によって記憶域内に右詰めで格納され、対応するデータの型に応じたバイトを占めます。

8進型データは、3ビットで8進数1けたを表現します。

9.2.2.9　2 進型データ

2進型データは、**DATA**文、型宣言文による初期値設定、または、B形編集記述子を使用した**READ**文によって記憶域内に右詰めで格納され、対応するデータの型に応じたバイトを占めます。

2進型データは、1ビットで2進数1けたを表現します。

9.2.2.10特別な値

以下の特別な値を表現できます。

(1)　非数(NaN)

0.0/0.0のような無効演算を行うと結果は非数となります。

非数は以下の2つのタイプに分類されます。

- Signaling NaN

  この種の非数が演算に対して使用されるとき、無効演算例外が検知されます。

- Quiet NaN

この種の非数は無効演算の結果として返されます。しかし、無効演算例外は検知されません。

(2)　無限大(inf)

無限大は、正の無限大、および、負の無限大に分類されます。

正の無限大(+inf)は、同じ型で表せる他のいずれの数値以上の値です。

負の無限大(-inf)は、同じ型で表せる他のいずれの数値未満の値です。

(3)　符号付きのゼロ(+0.0、および、-0.0)

内部表現では、+0.0、および、-0.0は仮数部の符号ビットによって認識されます。しかし、これらの値は同じ値として扱われます。

```
0.0 .EQ. (-0.0) => true
```

以下に示すように符号付きのゼロは、正の無限大、および、負の無限大を求めるのに有効です。

```
B1 = +0.0
B2 = -0.0
A1 = 1.0 / B1
A2 = 1.0 / B2
WRITE(*, *) "A1 = ", A1, " A2 = ", A2
```

9.2.3　諸元の定義

Fortranコンパイラにおける各種上限値は以下のとおりです。

項目	上限値
INCLUDE行で取り込むファイルのネスト数	63
配列の次元数	31
継続行の行数	1023行
名前の長さ	199文字

9.2.4　定義済みマクロ

すべての定義済みマクロは、**fpp**でプリプロセスが行われ、かつ、以下のいずれかの条件を満たすとき、有効になります。

- -**E**または-**M**が指定されている。
- ファイルの拡張子が**.F**、**.F90**、**.F95**、**.F03**のいずれかである。

定義済みマクロは以下のとおりです。

unix、__unix、__unix__

常に1に定義される。

linux、__linux、__linux__

常に1に定義される。

__gnu_linux__

常に1に定義される。

__ve、__ve__

常に1に定義される。

__ELF__

常に1に定義される。

__NEC__

常に1に定義される。

__FAST_MATH__

-**ffast-math**が有効であるとき1に定義され、そうでないとき定義されない。

_FTRACE

-ftraceが有効であるとき1に定義され、そうでないとき定義されない。

__NEC_VERSION__

コンパイラのバージョンが*X.Y.Z*であるとき、以下の計算式で得られる値が定義される。

*X**10000 + *Y**100 + *Z*

__OPTIMIZE__

コンパイル時に有効となった**-O***n*の最適化レベル(*n*)の値に定義される。

__VECTOR__

自動ベクトル化が有効であるとき1に定義され、そうでないとき定義されない。

__VERSION__

常に使用しているコンパイラのバージョンを示す文字列定数に定義される。

9.2.5　組込み手続に関する注意事項

CPU_TIME

プログラム実行でのCPU時間を返却します。共有並列処理時、Version 3.0.7以降ではCP

U_TIMEを呼び出したスレッドのCPU時間を返却します。以前は、呼び出し時点での全スレッドのCPU時間の合計値を返却していました。Version 3.0.7以降で全スレッドのCPU時間の合計を取得したいときは、環境変数**VE_FORT_ACCUMULATE_THREAD_CPU_TIME**に"YES"を指定してください。

9.3　メモリの確保・解放

Fortranコンパイラは、プログラム実行時の**ALLOCATE**文、**DEALLOCATE**文、各文の処理のために必要なメモリの確保、解放を高速に行えるようメモリブロック機能を実装しています。

メモリブロック機能では、プログラムの実行開始時にあらかじめメモリブロックと呼ばれる領域を確保しておき、実行時に領域が確保されるスカラ変数(基本型、派生型)、大きさの小さい配列をメモリブロック内に割り付けます。このため、メモリの確保、解放ごとのシステムコールが不要となり、スカラ変数、小さい配列の確保、解放を高速化できます。

9.3.1　メモリブロック

メモリブロックは割り付けられるデータの種類により3種類あり、それぞれのプログラム実行開始時のサイズは64メガバイトです。また、それぞれのしきい値は16メガバイトで、しきい値未満のサイズのデータの領域がメモリブロック内に割り付けられます。しきい値以上のサイズのデータは、コンパイラの実行時ルーチンを介して呼び出されるmalloc(3C)、free(3C)によって、確保、解放されます。

メモリブロック名	割り付けられるデータ	サイズ	しきい値
Allocate	ALLOCATABLE 属性を持つスカラ変数、配列	64	16
Pointer	POINTER 属性を持つスカラ変数、配列	64	16
Miscellaneous	各文の処理のための作業領域、作業配列、自動配列	64	16

(単位:メガバイト)

データを割り付ける際メモリブロック内に十分な領域が確保できないとき、「サイズ」の大きさのメモリブロックが追加され、そのブロック内に割り付けられます。

9.3.2　メモリブロックのサイズ・しきい値の変更

メモリブロックのサイズは環境変数**VE_FORT_MEM_BLOCKSIZE**により変更できます。サイズは、M(メガバイト)、G(ギガバイト)単位で指定でき、2のべき乗数のみ指定できます。しきい値は、メモリブロックサイズ/4の値に設定されます。

例

```
$ export VE_FORT_MEM_BLOCKSIZE=32M
```

メモリブロックサイズを32メガバイトに設定します。しきい値は8メガバイトです。

9.4　入出力

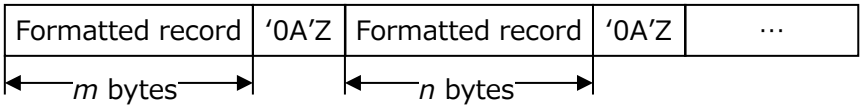
9.4.1　書式付き記録

書式付き記録は、書式付き入出力文、並び入出力文、および、変数群入出力文によって入出力される記録です。

- 書式付き入出力文によって入出力される記録は、書式仕様にしたがって入出力されます。一般にこの記録の長さは可変ですが、Fortranコンパイラの準備するレコードバッファの長さ以上にできません。
- 並び入出力文によって入出力される記録は、その入出力文の入出力項目並びにしたがって入出力されます。並び入出力文では、一度の実行によって一つ以上の記録が入出力されます。
- 変数群入出力文によって入出力される記録は、その入出力文で指定した変数群名の変数群並びに従って入出力されます。変数群入出力文では、一度の実行によって一つ以上の記録が入出力されます。

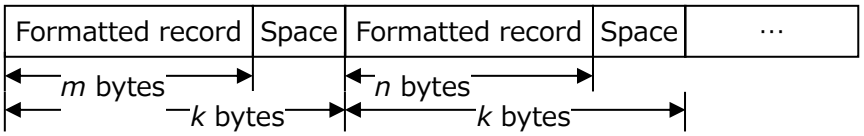
9.4.1.1　順編成ファイルの書式付記録

順編成ファイル中の書式付き記録は、改行コード('0A'Z)によって区切られています。また、各記録の長さは一定ではありません。形式は以下のとおりです。



9.4.1.2　直編成ファイルの書式付記録

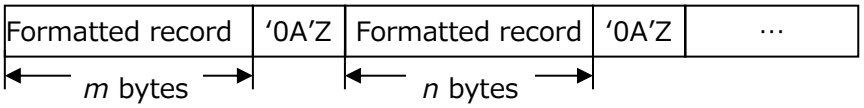
直編成ファイル中の書式付き記録の長さは、**OPEN**文の**RECL**指定子で指定された値です。入出力項目並びの編集による記録の長さが、ファイル中の記録の長さに満たないときには、後ろに空白が詰められます。



(*k*: OPEN 文で指定された記録の長さ)

9.4.1.3 ストリームファイルの書式付記録

ストリームファイル中の書式付き記録は、順編成ファイルと同じように改行コード('0A'Z)によって区切られています。ただし、記録に対して最大の長さは適用されません。形式は以下のとおりです。

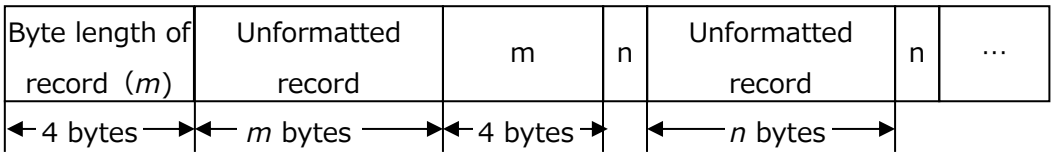


9.4.2　書式なし記録

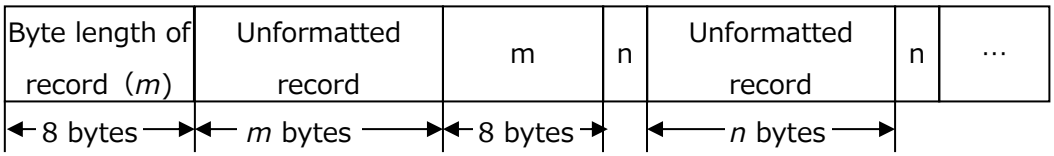
書式なし記録は、書式なし入出力文によってのみ入出力される記録です。書式なし記録の長さは、入出力項目のデータサイズを合計した値と同じです。各データのサイズは「9.2 処理系定義」を参照してください。

9.4.2.1 順編成ファイルの書式なし記録

順編成ファイル中の書式なし記録は、記録の前後にその書式なし記録のバイト長を示す4バイトのデータが付加されて構成されます。

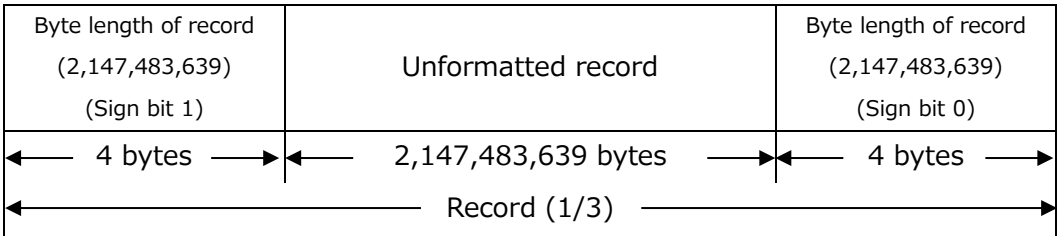


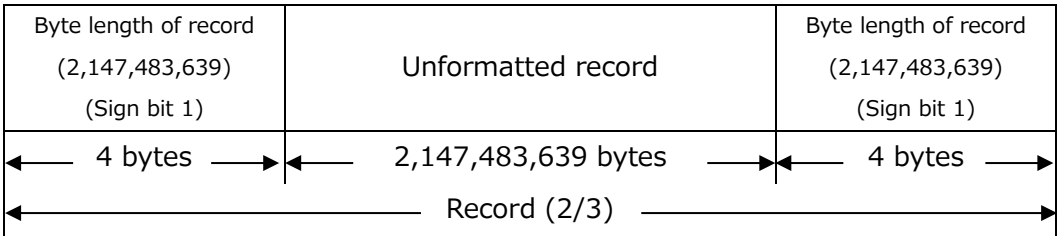
環境変数**VE_FORT_EXPRCW**が設定されているとき、順編成ファイル中の書式なし記録は、記録の前後にその書式なし記録のバイト長を示す8バイトのデータが付加されて構成されます。

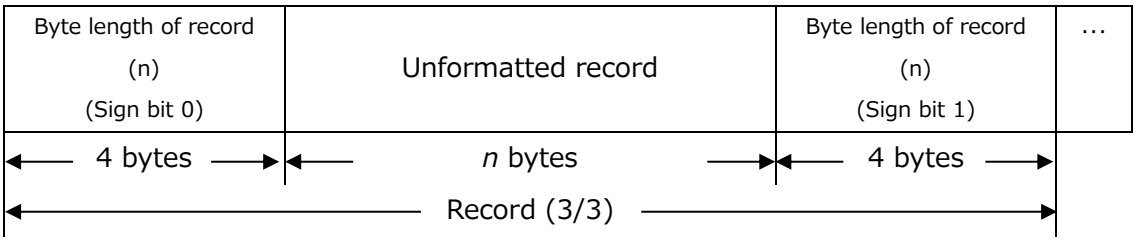


環境変数**VE_FORT_SUBRCW**が設定されているとき、順編成ファイル中の書式なし記録は、記録を一定の長さ(2,147,483,639bytes)以下に分割し、その前後にその書式なし記録の

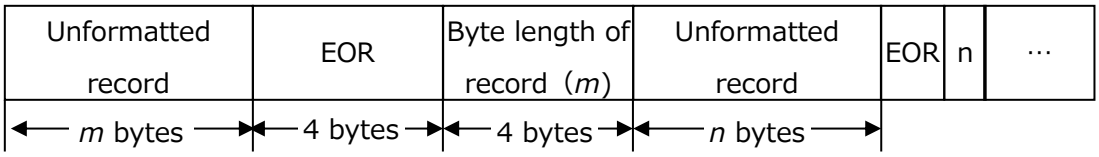
バイト長を示す4バイトのデータが付加されて構成されます。また、記録のバイト長の符号ビットでそのレコードが前後のレコードの記録の一部かを示します。



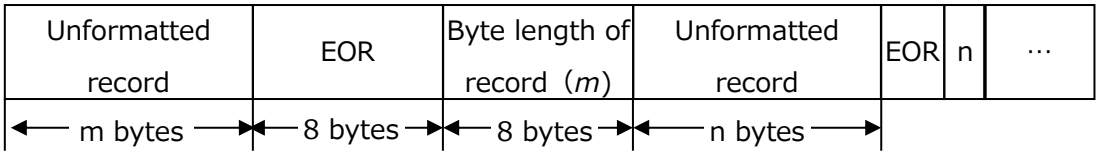




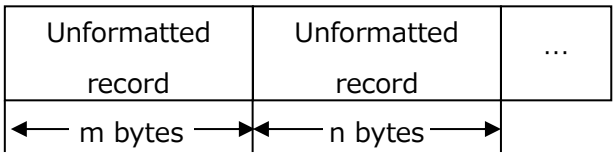
環境変数**VE_FORT_PARTRCW**が指定されたとき、順編成ファイル中の書式なし記録は、記録の後ろに記録の終了を示す4バイトのEORのデータとその書式なし記録のバイト長を示す4バイトのデータが付与されて構成されます。



環境変数**VE_FORT_EXPRCW**と環境変数**VE_FORT_PARTRCW**が同時に指定されたとき、順編成ファイル中の書式なし記録は、記録の後ろに記録の終了を示す8バイトのEORのデータとその書式なし記録のバイト長を示す8バイトのデータが付与されて構成されます。



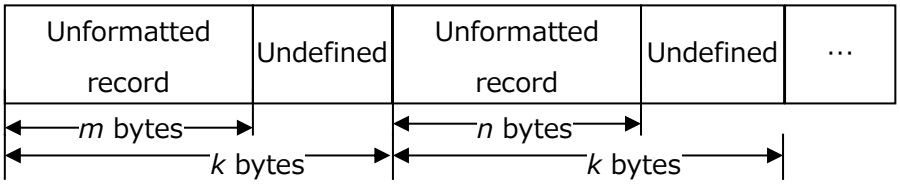
環境変数**VE_FORT_NORCW**が指定されたとき、順編成ファイル中の書式なし記録は、制御レコードを一切付与されず構成されます。これは、ストリームファイルの書式なし記録と同じ構成です。



9.4.2.2　直編成ファイルの書式なし記録

直編成ファイル中の書式なし記録の長さは、**OPEN**文の**RECL**指定子で指定した値です。入出力項目並びによる記録の長さが、ファイル中の記録の長さに満たないとき、残りの値は不定です。

WRITE文でファイルに記録を書き込むときは、不定のデータを無視し、記録の長さは出力項目のデータサイズを合計した値と同じになります。



(*k*: OPEN文で指定された記録の長さ)

9.4.2.3　ストリームファイルの書式なし記録

書式なしストリームファイルは記録構造をもたないバイト列です。

9.4.3　事前接続

外部ファイル装置は、プログラムの実行が開始された時点でそれぞれ固有のファイルを識別できます。これを事前の接続と呼びます。

9.4.3.1　システム標準ファイルの事前の接続

システム標準ファイルは、次に示す外部ファイル装置とそれぞれ事前に接続されています。

外部ファイル装置	システム標準ファイル
0	標準エラー出力

5	標準入力ファイル
6	標準出力ファイル

これらの外部ファイル装置に対する事前の接続は、その外部ファイル装置に対する**OPEN**文を実行しないときにかぎり有効です。一度**OPEN**文を実行すると、その外部ファイル装置に対するシステム標準ファイルと外部ファイル装置の接続は切り離され、再び接続することはできません。すなわち、上記の外部ファイル装置を指定した**OPEN**文を実行した後**CLOSE**文を実行しても、次の外部ファイル装置0、5、6に対する入出力文では、装置が接続されていないのでエラーになります。

次の例では、(a)の**WRITE**文は標準出力ファイルに出力され、(b)の**WRITE**文はファイルDATA6に出力され、(c)の**WRITE**文はエラーが出力されます。

例

```
WRITE(6, *) A, B, C  ------(a) Standard output file
    :
    :
OPEN(6, FILE = "DATA6")
WRITE(6, *) I, J, K  ------(b) DATA6
    :
CLOSE(6)
    :
WRITE(6, *) X, Y, Z  ------(c) Unit 6 は接続されていない
```

9.4.3.2　一般ファイルの事前の接続

0、5、6以外の外部ファイル装置*n*に対しては、ファイルfort.*n*が事前に接続されています。**OPEN**文に**FILE**指定子を指定したときでも、その外部ファイル装置nに対する**CLOSE**文を実行し、再度fort.*n*に**OPEN**文で接続すれば、次の外部ファイル装置nに対する入出力文では、ファイルfort.*n*に再度接続されます。

次の例では、(a)の**WRITE**文はfort.8に出力され、(b)の**WRITE**文はDATA8に出力され、(c)の**WRITE**文では再びfort.8に出力されます。ただし、このとき(a)で出力された記録は(c)の**WRITE**文で上書きされます。

事前に接続するファイルを変更するときは、「2.2 実行時に参照される環境変数」のVE_FORT*n*を参照してください。

例

```
WRITE(8, *) A, B, C  ------(a) fort.8
    :
```

```
      :
OPEN(8, FILE = "DATA8")
WRITE(8, *) I, J, K  ------(b) DATA8
      :
CLOSE(8)
      :
OPEN(8, FILE = "fort.8")
WRITE(8, *) X, Y, Z  ------(c) fort.8
```

9.4.4　名前なしファイル

STATUS="SCRATCH"を指定した**OPEN**文の実行により名前なしファイルが出力されます。名前なしファイルは、<stdio.h>ヘッダファイルでP_tmpdirとして定義されたディレクトリ配下に出力されますが、このディレクトリがアクセス不能なときは、/tmpが最後の手段として使用されます。

ただし、環境変数**TMPDIR**を指定することにより、指定したディレクトリ配下に名前なしファイルを出力できます。

9.4.5　丸めモード

丸めモードは、**OPEN**文、データ転送入出力文において、**ROUND**指定子、または、編集記述子で指定できます。指定がないとき、丸めモードは**PROCESSOR_DEFINED**になります。

丸めモードによる変換結果の値は以下となります。

ROUND指定子	編集記述子	変換結果
UP	RU	元の値以上で最小の表現可能な値
DOWN	RD	元の値以下で最大の表現可能な値
ZERO	RZ	元の値に最も近く、絶対値で元の値未満の値
NEAREST	RN	最も近い表現可能な値 最も近い値が二つあるとき、偶数を選択
COMPATIBLE	RC	最も近い表現可能な値 最も近い値が二つあるとき、ゼロから離れている方を選択
PROCESSOR_DEFINED	RP	**NEAREST**と同等

9.4.6　NAMELIST の入力形式

NAMELISTの入力形式は、NAMELIST名の先頭に追加する文字として"$"、"&"をサポートし

ます。また、終了を示す記号として”$end”、“&end”、“/”をサポートします。

9.4.7　NAMELIST の出力形式

- 数値配列の出力

  NAMELIST出力において、数値配列で同じ値が二つ以上連続するとき同じ値をまとめたフォーマット(連続する個数*値)で出力します。フォーマットは、環境変数**VE_FORT_NML_REPEAT_FORM**で変更できます。詳細は「2.2 実行時に参照される環境変数」を参照してください。

- **DELIM**指定子と文字型配列

  DELIM指定子で”NONE”が指定されたとき、値区切り子を含まない連続した文字で出力します。”QUOTE”、または、”APOSTROPHE”が指定されたとき、連続した同じ文字、または、値をまとめたフォーマット(連続する個数*値)で出力します。

  DELIM指定子が省略されたとき、文字型配列の値は連続で出力します。このフォーマットは、環境変数**VE_FORT_NML_DELIM_BLANK**で変更できます。詳細は「2.2 実行時に参照される環境変数」を参照してください。

  備考　**DELIM**指定子に"NONE"を指定、または、指定していないときの出力レコードは、NAMELISTの入力レコードとして受け付けられません。NAMELISTの入力レコードとして使用するには、**DELIM**指定子で”NONE”以外を指定するか、**DELIM**指定子を指定せずに環境変数**VE_FORT_NML_DELIM_BLANK**に”YES”を指定して出力レコードを作成してください。

- バージョン3.0.7以前のバージョンとの互換

  バージョン3.0.7以前のフォーマットで出力したいとき、環境変数**VE_FORT_NML_REPEAT_FORM**に”NO”を指定してください。

9.5　Fortran 2008 機能

本章では、NEC Fortranコンパイラがサポートする Fortran 2008言語仕様の新機能について説明します。

9.5.1　Coarray を用いた SPMD プログラミング

- SPMD（Single Program Multiple Data）プログラミングモデルが利用できます。
- 単一のイメージに制限されます。並列実行はありません。
- イメージの制御として、以下の文および構文が利用できます。

  ALLOCATEと**DEALLOCATE**、**CRITICAL**と**END CRITICAL**、**END**、**LOCK**と**UNLOCK**、**SYNC ALL**、**SYNC IMAGES**、**SYNC MEMORY**

- 組込み手続**MOVE_ALLOC**でCoarrayが利用できます。
- 以下の新しい組込み関数は利用できます。

  ATOMIC_DEFINE、**ATOMIC_REF**、**IMAGE_INDEX**、**LCOBOUND**、**NUM_IMAGES**、**THIS_IMAGE**、**UCOBOUND**

9.5.2　データの宣言

- 配列の次元数の最大値が7から15に拡大されました。

例

```
REAL ARRAY(2,2,2,2,2,2,2,2,2,2,2,2,2,2,2)
```

- 64ビット**integer**型が利用できます。
- 定義式から形状を推測できる名前付き定数（**PARAMETER**）の配列が利用できます。

例

```
REAL,PARAMETER :: IDMAT3(*,*) = RESHAPE( [ 1,0,0,0,1,0,0,0,1 ], [ 3,3 ] )
REAL,PARAMETER :: YEARDATA(2000:*) = [ 1,2,3,4,5,6,7,8,9 ]
```

- 組込み型の実体を定義するために**TYPE**キーワードを利用できます。

例

```
TYPE(REAL) X
TYPE(COMPLEX(KIND(0d0))) Y
TYPE(CHARACTER(LEN=80)) X
```

- 複数の型束縛手続を一つの型束縛手続定義文で宣言できます。

例

```
PROCEDURE,NOPASS :: A
PROCEDURE,NOPASS :: B=>X
PROCEDURE,NOPASS :: C
```

上の三つの文を以下のように一つの文にまとめることができます。

```
PROCEDURE,NOPASS :: A, B=>X, C
```

- DOUBLEPRECISIONという名前の派生型を定義することはできません。
- 初期化**DO**形反復で組み込み関数が利用できます。

例

```
DATA (X(I),I=1,SIZE(X))/1,2,3,4,5,6,7,8,9,10/
```

- 利用者定義演算子を宣言式中で利用できます。以下の例では、利用者定義演算子.USER.を配列Aの宣言で利用しています。

例

```
MODULE MOD
  INTERFACE OPERATOR(.USER.)
    MODULE PROCEDURE USER
  END INTERFACE
CONTAINS
  INTEGER PURE FUNCTION USER(W)
    REAL,INTENT(IN) :: W
    USER = CEILING(SQRT(ABS(W)))
  END FUNCTION
END MODULE
PROGRAM TEST
  USE MOD
  CALL SUB(17.0)
CONTAINS
  SUBROUTINE SUB(X)
    REAL,INTENT(IN) :: X
    LOGICAL A(.USER.(X))
  END SUBROUTINE
END PROGRAM
```

9.5.3　データの用法と計算

- 構造体構成子において、割付け可能な成分の値は省略できます。このときNULL()を指定したときと同じ振る舞いをします。
- **ALLOCATE**文で配列を割付けるとき、**SOURCE**=、または、**MOLD**=が指定され、それが配列表現であるならば、その配列は式から直接形状を取得できます。

例

```
SUBROUTINE A(X,MASK)
     REAL X(:,:,:)
     LOGICAL MASK(:,:,:)
     REAL,ALLOCATABLE :: Y(:,:,:)
```

```
      ALLOCATE(Y,MOLD=X)
      WHERE (MASK)
           Y = 1/X
           ELSEWHERE
           Y = HUGE(X)
      END WHERE
! ...
END SUBROUTINE
```

- **SOURCE**指定子を伴う**ALLOCATE**文は複数回の割付けができます。

例

```
PROGRAM MULTI_ALLOC
      INTEGER,ALLOCATABLE :: X(:),Y(:,:)
      ALLOCATE(X(3),Y(2,4),SOURCE=42)
      PRINT *,X,Y
END PROGRAM
```

上のプログラムは、“42”を11回出力します。

- 複素数オブジェクトの実部と虚部は複素数部分指定し”%RE”と”%IM”でアクセスできます。

例

```
COMPLEX,PARAMETER :: C = (1,2), CA(2) = [ (3,4),(5,6) ]
```

C%REとC%IMはそれぞれ値1と2を持ちます。そして、CA%REとCA%IMはそれぞれ値[3,5]と[4,6]を持つ配列です。

例

```
COMPLEX :: V, VA(10)
```

実部と虚部は直接代入することができます。VA%IM=0はVAの虚部のそれぞれの要素に0を代入します。

- **ALLOCATE**文において、**MOLD**指定子によって変数に式の型パラメタ、および、実行時の型を指定できます。

例

```
CLASS(*),POINTER :: A,B,C
ALLOCATE(A,B,C,MOLD=125)
```

A、B、Cの型が（基本）整数型になるように割付けています。

- 多相的割付け変数へ代入できます。

例

```
CLASS(*),ALLOCATABLE :: X
…
X = 43
```

Xは割付け済み、解放済みにみかかわらず、(基本)整数型を持ち、値は43となります。

- ターゲットが1より大きい次元を持つとき、次元数を再マッピングしてポインタ代入できます。

例

```
REAL,TARGET :: X(100,100)
REAL,POINTER :: X1(:)
X1(1:SIZE(X)) => X
```

9.5.4　実行制御

- **BLOCK**構文は、実行コード中の実体の宣言を許可します。

例

```
DO I=1,N
     BLOCK
          REAL TMP
          TMP = A(I)**3
          IF (TMP>B(I)) B(I) = TMP
     END BLOCK
END DO
```

ここで変数tmpの有効範囲は**BLOCK**構文内に限られ、その外側に影響を及ぼしません。

- **EXIT**文は**DO**構文だけではなく**ASSOCIATE**、**BLOCK**、**IF**、**SELECT CASE**、**SELECT TYPE**の各名前付き構文でも利用できます。
- **STOP**文で、stop-codeが整数型もしくは基本文字型のどのようなスカラ定数式も指定できます。
- **ERROR STOP**文が利用できます。

例

```
IF (X<=0) ERROR STOP 'X MUST BE POSSIBLE'
```

- **FORALL**構文の最初の文で省略可能な型指定子が指定できます。

例

```
COMPLEX I(100)
REAL X(200)
...
FORALL (INTEGER :: I=1:SIZE(X)) X(I) = I
```

9.5.5　組込み手続と組込みモジュール

- 以下の新しい組込み関数を利用できます。

  ACOSH、**ASINH**、**ATANH**、**BESSEL_J0**、**BESSEL_Y0**、**BESSEL_J1**、**BESSEL_Y1**、**BESSEL_JN**、**BESSEL_YN**、**ERF**、**ERFC**、**ERFC_SCALED**、**GAMMA**、**LOG_GAMMA**、**HYPOT**、**NORM2**、**BGE**、**BGT**、**BLE**、**BLT**、**DSHIFTL**、**DSHIFTR**、**IALL**, **IANY**, **IPARITY**、**LEADZ**、**TRAILZ**、**MASKL**、**MASKR**、**MERGE_BITS**、**PARITY**、**POPCNT**、**EXECUTE_COMMAND_LINE**、**CMDSTAT**、**STORAGE_SIZE**、**IS_CONTIGUOUS**、**FINDLOC**

- 組込み関数**ACOS**、**ASIN**、**ATAN**、**COSH**、**SINH**、**TAN**、**TANH**に複素数型を指定できます。
- 組込み関数**ATAN**は新たに**ATAN2**(*Y*,*X*)と同じ意味で**ATAN**(*Y*,*X*)の形式が利用できます。
- 組込み関数**SELECTED_REAL_KIND**に第三番目の引数*RADIX*が利用できます。
- 標準組込みモジュール**ISO_C_BINDING**で以下の追加手続を利用できます。

例

```
INTERFACE C_SIZEOF
     PURE INTEGER(C_SIZE_T) FUNCTION C_SIZEOF...(X)
          TYPE(*) :: X(..)
     END FUNCTION
END INTERFACE
   use iso_c_binding
   integer(c_int) :: i
   real(c_float) :: r, s(5)
   write(*, *) "size of i = ", c_sizeof(i)
   write(*, *) "size of r = ", c_sizeof(r)
```

```
  write(*, *) "size of s(5) = ", c_sizeof(s)
  end
```

- 組込みモジュール**ISO_FORTRAN_ENV**で以下の名前付き定数を利用できます。
  - スカラ整数定数INT8，INT16，INT32，INT64，REAL32，REAL64，REAL128
  - CHARACTER_KINDS，INTEGER_KINDS，LOGICAL_KINDS，REAL_KINDS
- 組込み関数**MAXVAL**、**MINVAL**で、*KIND*引数に続く省略可能な引数*BACK*が利用できます。

例

```
MAXVAL( [ 5,1,5 ], BACK=.TRUE.)
```

- 組込みモジュールISO_FORTRAN_ENVの**COMPILER_VERSION**関数を利用できます。

例

```
MODULE VERSION_INFO
 USE ISO_FORTRAN_ENV
 CHARACTER(LEN(COMPILER_VERSION()))::COMPILER = COMPILER_VERSION()
END MODULE
PROGRAM SHOW_VERSION_INFO
 USE VERSION_INFO
 PRINT *,COMPILER
END PROGRAM
```

9.5.6　入出力

- **OPEN**文の**NEWUNIT**指定子が利用できます。

例

```
INTEGER UNIT
OPEN(FILE='OUTPUT.LOG',FORM='FORMATTED',NEWUNIT=UNIT)
WRITE(UNIT,*) 'LOGFILE OPENED.'
```

- 別の装置に対する再帰入出力が利用できます。

例

```
WRITE (OUTPUT_UNIT,*) F(100)
```

関数FはOUTPUT_UNIT以外の装置に対して入出力できます。例えば値100が範囲外である場

合に以下のようにエラーメッセージを出力できます。

例

```
WRITE (ERROR_UNIT,*) 'ERROR IN F:',N,'IS OUT OF RANGE'
```

- 書式反復数としてアスタリスク（*）を指定して書式を無限に繰り返すことができます。

例

```
SUBROUTINE S(X)
     LOGICAL X(:)
     PRINT 1,X
     1   FORMAT('X =',*(:,' ',L1))
END SUBROUTINE
```

配列Xの全体をその成分数がいくつであっても一行に表示します。無限の繰り返し回数は書式指定の最上位レベルでのみ許されていて、かつ最後の書式項目でなければなりません。

- 編集記述子G0およびG0.dは先頭と末尾のすべての空白を省略して一般化された編集を行います。

例

```
PRINT 1,1.25,.TRUE.,"HI !",123456789
1 FORMAT(*(G0,','))
```

上の**PRINT**文は以下のように出力されます。

```
1.250000,T,HI !,123456789,
```

9.5.7　プログラムと手続

- **CONTAINS**文の後に空の内部副プログラム部分、モジュール副プログラム部分、および型束縛手続部分が記述できます。型束縛手続部分の場合では（特に効果のない）**PRIVATE**文が（指定する必要のない）**CONTAINS**文の後に許されるようになりました。
- 内部手続が実引数として渡せます。また手続ポインタへ代入もできます。内部手続が仮引数経由もしくは手続ポインタ経由で実行された場合に、親手続の局所変数へアクセスできます。

例

```
SUBROUTINE MYSUB(COEFFS)
     REAL,INTENT(IN) :: COEFFS(0:) ! Coefficients of polynomial.
```

```
      REAL INTEGRAL
      INTEGRAL = INTEGRATE(MYFUNC,0.0,1.0) ! Integrate from 0.0 to 1.0.
      PRINT *,'INTEGRAL =',INTEGRAL
      CONTAINS
      REAL FUNCTION MYFUNC(X) RESULT(Y)
          REAL,INTENT(IN) :: X
          INTEGER I
          Y = COEFFS(UBOUND(COEFFS,1))
          DO I=UBOUND(COEFFS,1)-1,0,-1
              Y = Y*X + COEFFS(I)
          END DO
      END FUNCTION
END SUBROUTINE
```

- 空状態のポインタ、および、割り付けられていない割付け変数は省略可能な非割付け、非ポインタ仮引数に対しての実引数として渡すことができます。
- 非純粋要素別処理手続は**IMPURE**キーワードで定義できます。

例

```
IMPURE ELEMENTAL INTEGER FUNCTION CHECKED_ADDITION(A,B) RESULT(C)
      INTEGER,INTENT(IN) :: A,B
      IF (A>0 .AND. B>0) THEN
          IF (B>HUGE(C)-A) STOP 'POSITIVE INTEGER OVERFLOW'
      ELSE IF (A<0 .AND. B<0) THEN
          IF ((A+HUGE(C))+B<0) STOP 'NEGATVE INTEGER OVERFLOW '
      END IF
      C = A + B
END FUNCTION
```

- 純粋手続の引数が**VALUE**属性を持つ場合、どの**INTENT**属性も必要ありません。

例

```
PURE SUBROUTINE S(A,B)
      REAL,INTENT(OUT) :: A
      REAL,VALUE :: B
      A = B
END SUBROUTINE
```

- 内部手続およびモジュール手続において、**END**文の**FUNCTION**、および、**SUBROUTIN E**キーワードを省略できます。
- **ENTRY**文は削除予定機能として扱われます。
- 行をセミコロンで開始できます。
- 結合名を持つ外部手続の名前が局所識別子として扱われます。

例

```
SUBROUTINE SUB() BIND(C,NAME='one')
PRINT *,'ONE'
END SUBROUTINE
SUBROUTINE SUB() BIND(C,NAME='two')
PRINT *,'TWO'
END SUBROUTINE
PROGRAM TEST
INTERFACE
SUBROUTINE ONE() BIND(C)
END SUBROUTINE
SUBROUTINE TWO() BIND(C)
END SUBROUTINE
END INTERFACE
CALL ONE
CALL TWO
END PROGRAM
```

- **NAME**=指示子を持たないとき、内部手続で**BIND(C)**属性が利用できます。
- **VALUE**属性を持つ仮引数は、配列や長さが1ではない文字型であっても利用できます。ただし、**ALLOCATABLE**属性、**POINTER**属性、coarrayは利用できません。

例

```
PROGRAM VALUE_EXAMPLE_2008
     INTEGER :: A(3) = [ 1,2,3 ]
     CALL S('HELLO?',A)
     PRINT '(7X,3I6)',A
CONTAINS
     SUBROUTINE S(STRING,J)
          CHARACTER(*),VALUE :: STRING
          INTEGER,VALUE :: J(:)
          STRING(LEN(STRING):) = '!'
```

```
        J = J + 1
        PRINT '(7X,A,3I6)',STRING,J
    END SUBROUTINE
END PROGRAM
```

上のプログラムは以下のように出力されます。

```
    Hello!     2     3     4
          1     2     3
```

- 内部手続が総称引用仕様内の個別手続となることができます。

```
PROGRAM F9
  INTERFACE G
    PROCEDURE S
  END INTERFACE
  CALL G
CONTAINS
  SUBROUTINE S
    PRINT *,'OK'
  END SUBROUTINE
END PROGRAM
```

9.5.8　多言語プログラミング

組込みモジュールISO_C_BINDINGの**C_LOC**、**C_FUNLOC**関数を宣言式中で利用できます。

```
INTEGER WORKSPACE(MERGE(10,20,C_ASSOCIATED(X,C_LOC(Y))))
```

9.5.9　サブモジュール

サブモジュールはモジュールを引用仕様宣言部と実装部に分離する仕組みです。インタフェースと実装が同じファイルにあると、インタフェースの変更がなく実装部のみを変更したときでも、そのモジュールを参照するモジュール、さらにそのモジュールを参照するモジュールと再コンパイルする必要があります。サブモジュールを用いて実装部を別のファイルに分離することで、実装部のみの変更の場合、それを参照するモジュールファイルの再コンパイルが不要となります。

サブモジュールのモジュールファイルは「*モジュール名*.*サブモジュール名*.smod」という名前でカレントディレクトリに出力されます。出力先は**-module**オプションで変更することができます。

例

```
MODULE SM
  INTERFACE
    MODULE SUBROUTINE S(N)
    END SUBROUTINE
  END INTERFACE
END MODULE
SUBMODULE (SM) SMOD
CONTAINS
  MODULE SUBROUTINE S(N)
    PRINT 1,N
    IF (N<0) RETURN
    PRINT 1,CEILING(SQRT(REAL(N)))
  1 FORMAT(I4)
  END SUBROUTINE
END SUBMODULE
```

9.6　Fortran 2018 機能

本章では、NEC Fortranコンパイラがサポートする Fortran 2018言語仕様の新機能について説明します。

9.6.1　実行制御

- **ERROR STOP**文と**STOP**文で基本整数型と基本文字型を利用できます。
- **ERROR STOP**文と**STOP**文の**QUIET**指定子が利用できます。

例

```
STOP 13, QUIET = .True.
```

上のプログラムは、ステータス13で正常終了します。

9.6.2　組込み手続と組込みモジュール

- 組込み手続**MOVE_ALLOC**で、省略可能な引数STATとERRMSGが利用できます。

例

```
INTEGER,ALLOCATABLE :: X(:),Y(:)
INTEGER ISTAT
CHARACTER(80) EMSG
...
```

```
CALL MOVE_ALLOC(X,Y,ISTAT,EMSG)
IF (ISTAT/=0) THEN
     PRINT *,' UNEXPECTED ERROR IN MOVE_ALLOC: ',TRIM(EMSG)
```

9.6.3　入出力

- **INQUIRE**文の**RECL**指定子に設定される値が変更になりました。接続されていない装置またはファイルのときは-1が設定され、ACCESS="STREAM"で接続されている装置またはファイルのときは-2が設定されます。以前のFortran標準では、両方とも未定義になりました。

9.6.4　プログラムと手続

- 利用者定義代入文として引用する手続の二番目の引数が**VALUE**属性を持つとき、**INTENT(IN)**属性は必須ではなくなりました。

例

```
MODULE MOD
  INTERFACE OPERATOR(+)
    MODULE PROCEDURE PLUS
  END INTERFACE
CONTAINS
  PURE INTEGER FUNCTION PLUS(A,B)
    INTEGER,VALUE :: A
    LOGICAL,VALUE :: B
    PLUS = MERGE(A+1,A,B)
  END FUNCTION
END MODULE
```

- 利用者定義演算子として引用する関数の引数が**VALUE**属性を持つとき、**INTENT(IN)**属性は必須ではなくなりました。

例

```
MODULE MOD
  INTERFACE ASSIGNMENT(=)
    MODULE PROCEDURE ASGN
  END INTERFACE
CONTAINS
  PURE SUBROUTINE ASGN(A,B)
    INTEGER,INTENT(OUT) :: A
    LOGICAL,VALUE :: B
```

```
    A = MERGE(1,0,B)
  END SUBROUTINE
END MODULE
```

9.6.5　多言語プログラミング

- **C_FUNLOC**の引数に**BIND(C)**属性のない手続を指定できます。

9.6.6　廃止予定事項

- **EQUIVALENCE**文、**COMMON**文、**BLOCK DATA**文はFortran 2018言語仕様では廃止予定であり、**-std**=f2018を指定したとき、警告メッセージが出力されます。

第10章　多言語プログラミング

異なるプログラミング言語のオブジェクトファイルをリンクし、一つの実行ファイルを作成することを多言語プログラミングと呼びます。この章ではCプログラム、C++プログラム、Fortranプログラムを使用した多言語プログラミングの手法について説明します。

10.1 多言語プログラミングのポイント

次の例は、Cプログラム、Fortranプログラムをリンクし、一つの実行ファイルを作成する多言語プログラミングの例です。

C プログラム（ファイル名: a.c）

```
#include <stdlib.h>
#define N 1024
#define SIZE sizeof(double)
main()
{
  double *x = (double *)malloc(SIZE*N);
  double *y = (double *)malloc(SIZE*N);
  double *z = (double *)malloc(SIZE*N);
  int n;
  n = read_data(x, y);
  compute_(x, y, z, &n);
  write_data(z, n);
}
```

C プログラム（ファイル名: b.c）

```
#include <stdio.h>
int read_data(double *x, double *y)
{...}
```

Fortran プログラム（ファイル名: c.f90）

```
SUBROUTINE COMPUTE (X, Y, Z, N)
REAL*8 X(N),Y(N),Z(N)
!!! 計算
I = CHECK_VALUE(Z(N))
IF (I.EQ.0) RETURN
END SUBROUTINE
```

C プログラム（ファイル名: d.c）

```
int check_value_(double *x)
{ ... }
```

この例では、CプログラムからFortranプログラム、FortranプログラムからCプログラムを呼び出しています。呼び出す際には関数名、手続名を実際にプログラミングしている名前から外部シンボル名に変更し、引数、返却値でデータをやりとりし、言語間でデータを共有しています。

多言語プログラミングのポイントは以下です。

- C関数、C++関数、Fortran手続の名前の対応
- C/C++のデータ、Fortranのデータの型の対応

- C/C++ - Fortran間の返却値の受け渡し
- C/C++ - Fortran間の引数による値の受け渡し
- プログラムのコンパイル、リンクによる実行ファイルの作成

以降で、これらのポイントについて説明します。

10.2 C/C++関数名・Fortran手続名の対応

ソースファイルに記述されたC++関数名、Fortran手続名は、コンパイラによって名前が変更され、オブジェクトファイルに外部シンボル名として登録されます。このため、それらの関数、手続を参照するとき、変更後の外部シンボル名で呼び出すようにします。

10.2.1 Fortran 手続の外部シンボル名

(1)　手続の結合ラベルを使用するとき

Fortran 2003言語仕様の「Cとの相互利用」で手続の結合ラベルを使用するとき、Fortranプログラムの手続名は、結合ラベルと同じ文字列の外部シンボル名になります。すなわち、**NAME**指定があるとき指定された名前、省略されたときすべて小文字に変更された外部シンボル名になります。

例

```
SUBROUTINE SUB1(X) BIND(C, NAME="Fortran_Sub1")
...
END SUBROUTINE

SUBROUTINE SUB2(Y) BIND(C)
...
END SUBROUTINE
```

このFortranプログラムでは手続名が以下の外部シンボル名に変更されます。

（手続名）		（外部シンボル名）
SUB1	→	Fortran_Sub1
SUB2	→	sub2

(2)　手続の結合ラベルを使用しないとき

Fortranプログラムの手続名は、次の規則で外部シンボル名に変更されます。

- 手続名はすべて小文字に変更
- 名前の末尾に下線(_)が付加

例

```
SUBROUTINE COMPUTE (X, Y, Z, N)
REAL*8 X(N),Y(N),Z(N)
! 計算
I = CHECK_VALUE(Z(N))
IF (I.EQ.0) RETURN
END SUBROUTINE
```

このFortranプログラムでは手続名が以下の外部シンボル名に変更されます。

（手続名）		（外部シンボル名）
COMPUTE	→	compute_
CHECK_VALUE	→	check_value_

10.2.2 C++関数名の外部シンボル名

C++コンパイラは、ソースファイルに記述された関数名に返却値、引数の型を表す文字列が付加します。この操作を関数名のマングル(mangle)と呼び、C++コンパイラは、この仕組みを利用して引数の型が違う同じ名前の関数の定義を可能としています。

例

（ソースファイルでの表記）		（マングル名）
void func(double *x)		_Z4funcPd
void func(float *x)	→	_Z4funcPf

備考　　マングル名をソースファイルでの表記に戻すことを「デマングル」(demangle)と呼びます。

C関数やFortran手続から呼び出すC++関数は、C結合で宣言し、マングルされないようにします。そして、ソースファイルに記述された関数名で呼び出すようにします。同じように、C++関数から呼び出すC関数やFortran手続のプロトタイプ宣言もC結合で宣言し、マングルされないようにします。

例

```
extern "C" {
    void func(double *x);
    void func1(float *x);
};
```

結合指定はC++言語でのみ利用できます。プロトタイプ宣言などをC言語でも利用するとき

は、結合指定を条件コーディングします。

例

```
#ifdef __cplusplus          // __cplusplusはC++コンパイラによって
                            // 自動的に定義される
extern "C" {
#endif
    void func(double *x);
    void func1(float *x);
#ifdef __cplusplus
};
#endif
```

10.2.3 C/C++関数名、Fortran 手続の対応ルール

- C関数からFortran手続を呼び出すとき、Fortran手続の外部シンボル名でC関数から呼び出します。
- Fortran手続から呼び出されるC関数は、Fortran手続の外部シンボル名で定義します。
- C関数やFortran手続から呼び出されるC++関数は、C結合で宣言します。
- C++関数から呼び出すC関数やFortran手続のプロトタイプ宣言は、C結合で宣言します。

10.2.4 呼出し例

例　　C 関数からの Fortran 手続の呼出し(**BIND** 属性の指定あり)

呼出し側 (C 関数)

```
void cfunc() {
    ...
    sub1();
    ...
}
```

呼び出される側 (Fortran 手続)

```
SUBROUTINE SUB1() BIND(C)
...
END SUBROUTINE SUB1
```

すべて小文字の名前でプロトタイプ宣言し、呼び出します。

例　　C 関数からの Fortran 手続の呼出し(**BIND** 属性の指定なし)

呼出し側 (C 関数)

```
extern int sub_();
void cfunc() {
    ...
    sub_();
    ...
}
```

呼び出される側 (Fortran 手続)

```
SUBROUTINE SUB
...
END SUBROUTINE SUB
```

Fortran手続を下線付きのすべて小文字の名前でプロトタイプ宣言し、呼び出します。

例　　Fortran 手続からの C 関数の呼出し(**BIND** 属性の指定あり)

呼出し側 (Fortran 手続)

```
SUBROUTINE SUB
    USE, INTRINSIC :: ISO_C_BINDING
    INTERFACE
        SUBROUTINE CFUNC() BIND(C)
        END SUBROUTINE CFUNC
    END INTERFACE
    ...
    CALL CFUNC
    ...
END SUBROUTINE SUB
```

呼び出される側 (C 関数)

```
void cfunc() {
...
}
```

Cプログラムにおいて、すべて小文字の名前でC関数を宣言、定義し、Fortranプログラムにおいてすべて大文字の名前で引用仕様の定義、参照します。

例　　Fortran 手続からの C 関数の呼出し(**BIND** 属性の指定なし)

呼出し側 (Fortran 手続)

```
SUBROUTINE SUB
...
CALL CFUNC
...
END SUBROUTINE SUB
```

呼び出される側 (C 関数)

```
int cfunc_() {
...
}
```

下線付きのすべて小文字の名前で、C関数を宣言、定義します。

例　　C++関数からの Fortran 手続の呼出し

呼出し側 (C++関数)

```
extern "C" {
    int sub_(void);
};
void cfunc() {
    ...
    sub_();
    ...
}
```

呼び出される側 (Fortran 手続)

```
SUBROUTINE SUB
...
END SUBROUTINE SUB
```

Fortran手続を下線付きのすべて小文字の名前で、C結合でプロトタイプ宣言し、呼び出します。

例　　Fortran 手続からの C++関数の呼出し

呼出し側 (Fortran 手続)

```
SUBROUTINE SUB
...
CALL CFUNC
...
END SUBROUTINE SUB
```

呼び出される側 (C++関数)

```
extern "C" {
    int cfunc_(void);
};
int cfunc_(void) {
...
}
```

下線付きのすべて小文字の名前、C結合で、C++関数を宣言、定義します。

10.3 データ型

FortranプログラムとC/C++プログラムのデータ型の対応は以下のとおりです。

10.3.1 Fortran の整数型/論理型

型	Fortranの型	C/C++の型
整数型	**INTEGER**	**int** (*1)
	INTEGER(KIND=1) **INTEGER*1**	**signed char**
	INTEGER(KIND=2) **INTEGER*2**	**short**
	INTEGER(KIND=4) **INTEGER*4**	**int**
	INTEGER(KIND=8) **INTEGER*8**	**long**、**long int**、**long long**、 または、**long long int**
論理型	**LOGICAL**	**int** (*1)
	LOGICAL(KIND=1)	**signed char**

LOGICAL(KIND=2)	**short**
LOGICAL(KIND=4)	**int**
LOGICAL(KIND=8)	**long**、**long int**、**long long**、または、**long long int**

(*1) -**fdefault-integer**=**8** 指定時は、**long**、**long int**、**long long**、または、**long long int**

10.3.2 Fortran の実数型/複素数型

型	Fortranの型	C/C++の型
実数型	**REAL**	**float** (*1)
	REAL(KIND=4) **REAL*4**	**float**
倍精度実数型	**DOUBLE PRECISION**	**double** (*2)
	REAL(KIND=8) **REAL*8**	**double**
4倍精度実数型	**QUADRUPLE PRECISION** **REAL(KIND=16)** **REAL*16**	**long double**
複素数型	**COMPLEX**	**float __complex__** (*3)
	COMPLEX(KIND=4) **COMPLEX*8**	**float __complex__**
倍精度複素数型	**COMPLEX(KIND=8)** **COMPLEX*16**	**double __complex__**
4倍精度複素数型	**COMPLEX(KIND=16)** **COMPLEX*32**	**long double __complex__**

(*1) -**fdefault-real**=**8**指定時は、**double**

(*2) -**fdefault-double**=**16**指定時は、**long double**

(*3) -**fdefault-real**=**8**指定時は、**double __complex__**

10.3.3 Fortran の文字型

文字列chを宣言する例で示します。

型	Fortranの型	C/C++の型
文字型	**CHARACTER(LEN=n) ch**	**char ch**[n];

10.3.4 Fortran の派生型

(1) 説明

Fortranプログラムの派生型をCプログラムの構造体と結合するとき、派生型定義と宣言部に**BIND**属性(**BIND**(C))を指定します。派生型の成分と構造体のメンバの数は同じでなければならず、対応する成分・メンバの型が同じでなければなりません。

例

Fortranプログラム

```
USE, INTRINSIC :: ISO_C_BINDING
TYPE, BIND(C) :: STR_TYPE            ! 結合する派生型定義にBIND属性を指定
   REAL(C_DOUBLE) :: S1, S2
END TYPE STR_TYPE

INTERFACE
   SUBROUTINE FUNC(X) BIND(C)
      USE, INTRINSIC :: ISO_C_BINDING
      TYPE(C_PTR) :: X
   END SUBROUTINE FUNC
END INTERFACE

TYPE(C_PTR) :: P
TYPE(STR_TYPE),TARGET :: F_STR

P=C_LOC(F_STR)                    ! 派生型変数F_STRのCアドレス取得
CALL FUNC(P)                          ! C関数の関数呼出しでF_STRのCアドレスPを
渡す
...
```

Cプログラム

```
struct str_type {                          // Fortranと結合する構造体定義
    double s1, s2;
} *c_str;
```

```
void func(struct str_type **x) {
    c_str = *x;                        // c_strの指示先はF_STRになる
    ...
}
```

(2)　注意事項

- 派生型の成分と構造体のメンバの名前は同じである必要はありません。
- ビットフィールド、または、可変長配列を含んでいるCプログラムの構造体は結合できません。
- 4倍精度実数型、複素数型を含む派生型と構造体は結合できません。

10.3.5 C のポインタ

CポインタをFortranのデータと結合するとき、派生型C_PTRを使用します。

(1)　Fortran プログラムと C プログラムのポインタの結合

FortranプログラムでCプログラムのポインタを使用するとき、派生型**C_PTR**を使用します。

例

Fortranプログラム

```
USE, INTRINSIC :: ISO_C_BINDING
   INTERFACE
   SUBROUTINE FUNC(X) BIND(C)
      USE, INTRINSIC :: ISO_C_BINDING
      TYPE(C_PTR) :: X
   END SUBROUTINE FUNC
END INTERFACE

TYPE(C_PTR) :: P
...
CALL FUNC(P)                                ! C関数の関数呼出し
...
```

Cプログラム

```
int *a;

void func(int **p) {
```

```
    *p = a;                                    // FortranのPの指示先は変数aになる
}
```

(2)　Fortran プログラムでの変数の C アドレスの取得

Fortranプログラムの割り付けられた変数のCアドレスを取得するには、組込み手続**C_LOC**を使用します。

例

```
USE, INTRINSIC :: ISO_C_BINDING
INTEGER(C_INT),TARGET :: N
TYPE(C_PTR) :: N_ADDR
N_ADDR = C_LOC(N)                              ! 変数NのCアドレスの取得
```

(3)　C ポインタの比較

Fortranの組込み手続**C_ASSOCIATED**で二つのCポインタの指示先が同じであるかどうかを比較できます。組込み手続**C_ASSOCIATED**は、第1引数と第2引数が指す実体が同じとき真、そうでないとき偽を返却します。第2引数が省略されたとき、第1引数がNULLなら偽、それ以外なら真を返却します。

例

Fortranプログラム

```
USE, INTRINSIC :: ISO_C_BINDING
...
INTEGER(C_INT), BIND(C) :: X, Y
TYPE(C_PTR) :: P1, P2
CALL FUNC(P1, P2)                          ! C関数の関数呼出し
  IF ( C_ASSOCIATED(P1, P2) ) THEN           ! 組込み手続C_ASSOCIATEDは、
   ...                                     ! P1とP2が同じ実体を指すとき.true.、
END IF                                     ! そうでないとき.false.を返却する
```

Cプログラム

```
int x, y;
void func(int **px, int **py) {
    *px = &x;           // 関数func()をFortranプログラムから呼び出したとき、
    *py = &y;           // FortranのP1、P2の指示先はそれぞれ変数x、yになる
}
```

(4)　C ポインタと Fortran のデータポインタの結合

Fortranの組込み手続**C_F_POINTER**でCポインタをFortranのデータポインタに結合できます。組込み手続**C_F_POINTER**は、第1引数の*C_PTR*を第2引数のデータポインタに結合します。

例

Fortranプログラム

```
USE, INTRINSIC :: ISO_C_BINDING
...
INTEGER(C_INT), BIND(C) :: X
TYPE(C_PTR), BIND(C) :: CP
INTEGER(C_INT), POINTER :: FP
...
CALL FUNC(CP)                    ! C関数の関数呼出し
CALL C_F_POINTER(CP, FP)         ! CポインタCPをデータポインタFPに結合する
...
```

Cプログラム

```
int x;
void func(int **px) {
    *px = &x;           // 関数func()をFortranプログラムから呼び出したとき、
}                       // FortranのCPの指示先は変数xになる
```

10.3.6 Fortran の共通ブロック

(1) 説明

Fortranプログラムの共通ブロックをCプログラムと結合するとき、共通ブロックに**BIND**属性(BIND(C))を指定します。**BIND**属性が指定されたFortranプログラムの共通ブロックの成分が1個であるとき、Cプログラムの外部結合をもつ変数と結合できます。また、共通ブロックの成分が複数個あるとき、Cプログラムの外部結合をもつ構造体と結合できます。ただし、共通ブロックの成分と構造体のメンバの数は同じでなければならず、対応する成分・メンバの型が同じでなければなりません。

例

Fortranプログラム

```
USE, INTRINSIC :: ISO_C_BINDING
COMMON /COM1/ F1, F2
COMMON /COM2/ F3
REAL(C_FLOAT) :: F1, F2, F3
```

```
BIND(C) :: /COM1/, /COM2/        ! 結合する共通ブロックにBIND(C)を指定する
...
```

Cプログラム

```
struct { float f1, f2; } com1;
                        // 共通ブロックに複数個の変数が定義されているとき、
                        // Cプログラムの構造体と結合できる
...
float com2;             // 共通ブロックに一つだけ変数が定義されているとき、
                        // Cプログラムの変数と結合できる
...
```

(2)　注意事項

- 共通ブロックの成分と構造体のメンバの名前は同じである必要はありません。
- ビットフィールド、または、可変長配列を含んでいるCプログラムの構造体は結合できません。
- 4倍精度実数型、複素数型を含む共通ブロックと構造体は結合できません。

10.3.7 注意事項

Fortranの複素数型、倍精度複素数型、4倍精度複素数型を、Cの_Complexキーワードを使用して宣言された複素数型、倍精度複素数型、4倍精度複素数型に対応させることはできません。

10.4 関数・手続の型と返却値

ここではC関数、Fortran手続間の返却値の受渡しについて説明します。C++関数は、C関数、Fortran手続から呼び出される、または、それらを呼び出すとき、C結合で定義、宣言されるため、C関数と同じとみなせます。

(1)　整数、論理型、実数型、倍精度実数型、および、4 倍精度実数型の Fortran 手続

「10.3 データ型」で示した対応と同じです。

例　　倍精度実数型のFortran手続の呼出し

呼出し側 (C関数)

```
extern double func_();
...
double a;
a = func_();                // Fortran手続の呼出し
```

```
...
```

呼び出される側 (Fortran手続)

```
REAL(KIND=8) FUNCTION FUNC()
...
FUNC=10.0
...
END FUNCTION FUNC
```

例　　**double**型のC++関数の呼出し

呼出し側 (Fortran手続)

```
REAL(KIND=8) A
...
A = CFUNC()                    ! C++関数の呼出し
...
```

呼び出される側 (C++関数)

```
extern "C" {
    double cfunc_();
};
double cfunc_()
{
    double a;
    ...
    return a;
}
```

(2)　複素数型、倍精度複素数型、4 倍精度複素数型

C/C++関数とFortran手続間では、複素数型、倍精度複素数型、4倍精度複素数型に相当する関数を参照できません。

(3)　文字型の関数

Fortranの文字型の関数では、返却値を返却するための二つの引数が追加されます。二つの引数は、返却値である文字列の設定される領域のアドレスとその長さ(バイト数)です。

例　　文字型のFortran手続の呼出し

呼出し側 (C++関数)

```
extern "C" {
```

```
    int chfunc_(char *res_p, long res_l);
};
char a[17];                                    // 16バイト+終端1バイト分確保
...
chfunc_(a, 16L);                               // Fortran手続の呼出し
a[16] = '¥0';
...
```

呼び出される側 (Fortran手続)

```
CHARACTER*16 FUNCTION CHFUNC
CHFUNC="THIS IS FORTRAN."
RETURN
END FUNCTION CHFUNC
```

文字列を格納するための領域は、C/C++関数側で確保します。また、Fortranの文字列長は終端のヌル文字を含まないので、C/C++関数側で領域を確保するときは1バイト余計に確保します。

例　　文字型の関数としてC関数を呼び出す

呼出し側 (Fortran手続)

```
SUBROUTINE SUB
CHARACTER*20 CFUNC, CH
INTEGER M
...
CH = CFUNC(M)                              ! C関数の呼出し
...
END SUBROUTINE SUB
```

呼び出される側 (C関数)

```
extern int cfunc_(char *a, long b, int *p);

int cfunc_(char *a, long b, int *p)
{
    strcpy(a, "THIS IS C++.");
}
```

Fortran手続の第1引数は、C/C++関数の第3引数になります。

(4)　Fortran 手続のサブルーチン

Fortran手続のサブルーチンは、**int**型のC関数に相当します。

10.5 関数・手続間の引数の受け渡し

10.5.1 Fortran 手続の引数

Fortran手続の引数は、すべてアドレスで渡されます。このため、C/C++関数で引数を受け取るとき、引数の領域を指すポインタ値で受け取ります。逆に、Fortran手続に引数を渡すとき、引数のアドレスをFortran手続に渡します。

(1)　Fortran 手続の **VALUE** 属性をもたない引数を渡す

引数として渡す変数のアドレスを渡します。定数は、1度変数に代入してから渡します。

例

渡す側 (C++関数)

```
extern "C" {
    int func_(int *i, int *j);
}
void cfunc()
{
    int a, b, ret;
    ...
    b = 100;                              // 定数は変数に代入して渡す
    ret = func_(&a, &b);                  // Fortran手続の呼出し
    ...
}
```

受け取る側 (Fortran手続)

```
INTEGER FUNCTION FUNC(I, J)
INTEGER I, J
...
END FUNCTION FUNC
```

(2)　Fortran 手続の **VALUE** 属性をもつ引数を渡す

変数の値を渡します。定数は、引数にして渡します。

例

渡す側 (C++関数)

```
extern "C" {
```

```
    int func_(int i, int j);
}
void cfunc()
{
    int a, ret;
    ...
    ret = func_(a, 100);                    // Fortran手続の呼出し
    ...
}
```

受け取る側 (Fortran手続)

```
INTEGER FUNCTION FUNC(I, J) BIND(C)  ! BIND属性を指定
INTEGER, VALUE :: I, J                    ! VALUE属性を指定
...
END FUNCTION FUNC
```

(3)　Fortran 手続の **VALUE** 属性をもたない引数を受け取る

引数をポインタで受け取ります。

例

渡す側 (Fortran手続)

```
SUBROUTINE SUB
INTEGER K, I, J
...
CALL C_FUNC(I, J)
...
END SUBROUTINE SUB
```

受け取る側 (C++関数)

```
extern "C" {
    int c_func_(int *a, int *b);
}
int c_func_(int *a, int *b)          // 引数をポインタで受け取る
{
    ...
}
```

(4)　Fortran 手続の **VALUE** 属性をもつ引数を受け取る

変数の値を渡します。定数は、引数に指定して渡します。

例

渡す側 (Fortran手続)

```
SUBROUTINE SUB
  INTERFACE
      INTEGER(C_INT) FUNCTION C_FUNC(A, B) BIND(C)
                                              ! 引数仕様にBIND属性を指定
        USE, INTRINSIC :: ISO_C_BINDING
        INTEGER(C_INT), VALUE :: A, B         ! VALUE属性を指定
      END FUNCTION C_FUNC
  END INTERFACE
INTEGER K, I, J
...
K = C_FUNC(I, J)
...
END SUBROUTINE SUB
```

受け取る側 (C++関数)

```
extern "C" {
    int c_func_(int a, int b);
}
int c_func_(int a, int b)                  // 引数を値で受け取る
{
    ...
}
```

10.5.2 留意事項

10.5.2.1暗黙に追加される引数

Fortran手続ではソースファイルに記述された引数のほかに、暗黙的に引数が追加されることがあります。次の引数が追加されます。

- 呼び出すFortran手続が文字型の関数のとき、関数の返却値を格納する領域のアドレスとその返却値の長さ(単位:バイト)
- 引数が文字型の要素であるとき、その引数の長さ(単位:バイト)
- 引数が手続名のとき、手続の返却値のサイズ(単位:バイト)。ただし、文字型の関数以外は0

上記の引数が渡される順番は以下のとおりです。

(1)　返却値を格納する領域のアドレス(呼び出す手続が文字型のときのみ)

(2)　返却値のサイズ(呼び出す手続が文字型のときのみ)

(3)　ソースファイルに記述された引数

引数が文字型のときその長さ、手続名のとき返却値のサイズが、引数の最後に追加されます。

10.6 プログラムのリンク

10.6.1 C プログラムと Fortran プログラムのリンク

CプログラムとFortranプログラムをリンクするとき、Fortranコンパイラ(nfort)でリンクします。

例

```
$ nfort -c a.f90                (Fortranプログラムのコンパイル)
$ ncc -c b.c                    (Cプログラムのコンパイル)
$ nfort a.o b.o                 (Fortranコンパイラによるリンク)
```

10.6.2 C++プログラムと Fortran プログラムのリンク

C++プログラムとFortranプログラムをリンクするとき、Fortranコンパイラ(nfort)でリンクします。リンクするとき、C++コンパイラの実行時ライブラリ(**-cxxlib**)を指定します。

例

```
$ nfort-c a.f90                 (Fortranプログラムのコンパイル)
$ nc++ -c b.cpp                 (C++プログラムのコンパイル)
$ nfort a.o b.o -cxxlib         (Fortranコンパイラによるリンク)
```

10.7 実行時の注意事項

C/C++プログラムとFortranプログラムをリンクしたとき、C/C++関数でstdin、stdout、stderrをクローズしないでください。クローズしたとき、Fortran手続の動作は保証されません。

第11章　ライブラリ・リファレンス

本章では、NEC Fortranコンパイラ独自の組込み手続について説明します。

11.1 組込み手続

11.1.1 ABS(*A*) 個別名

- **機能**

  絶対値

- **分類**

  要素別処理関数

- **引数**

  A: 整数型、実数型、または、複素数型

- **結果の型、および、型パラメタ**

  *A*が複素数型のとき、結果の型はAと同じ種別パラメタの実数型とします。それ以外のとき、結果の型は*A*と同じとします。

- **結果の値**

  *A*が整数型または実数型のとき、結果の値は*A*の絶対値｜*A*｜とします。*A*が複素数(x,y)のとき、結果の値は$(x^2+y^2)^{1/2}$の値とします。

- **個別名**

個別名	引数の型	結果の型	備考
BABS	INTEGER(1)	INTEGER(1)	
IIABS, HABS	INTEGER(2)	INTEGER(2)	
IABS	基本整数型	基本整数型	規格で定義
JIABS	INTEGER(4)	INTEGER(4)	
KIABS	INTEGER(8)	INTEGER(8)	
ABS	基本実数型	基本実数型	規格で定義
DABS	DOUBLE PRECISION型	DOUBLE PRECISION型	規格で定義
QABS	REAL(16)	REAL(16)	

CABS	基本複素数型	基本実数型	規格で定義
CDABS	DOUBLE COMPLEX 型	DOUBLE PRECISION型	
ZABS	COMPLEX(8)	REAL(8)	
CQABS	COMPLEX(16)	REAL(16)	

11.1.2 ACOS(*X*) 個別名

- **機能**

  逆余弦関数

- **分類**

  要素別処理関数

- **引数**

  X:　実数型。|*X*|≦1を満たす値でなければなりません。

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  ラジアンで表した逆余弦**arccos**(*X*)の値

- **個別名**

個別名	引数の型	結果の型	備考
ACOS	基本実数型	基本実数型	規格で定義
DACOS	DOUBLE PRECISION型	DOUBLE PRECISION型	規格で定義
QACOS, QARCOS	REAL(16)	REAL(16)	

11.1.3 ACOSH(*X*) 個別名

- **機能**

  逆双曲線余弦関数

- **分類**

  要素別処理関数

- **引数**

  X:　実数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  逆双曲線余弦**arccosh**(*X*)の値

- **個別名**

個別名	引数の型	結果の型	備考
ACOSH	default real	default real	
DACOSH	DOUBLE PRECISION型	DOUBLE PRECISION型	
QACOSH	REAL(16)	REAL(16)	

11.1.4 AIMAG(*Z*) 個別名

- **機能**

  複素数の虚部を返します。

- **分類**

  要素別処理関数

- **引数**

  Z:　複素数型

- **結果の型、および、型パラメタ**

  *Z*と同じ種別パラメタの実数型

- **結果の値**

  *Z*の値が(*x*,*y*)のとき、結果の値は*y*です。

- **個別名**

個別名	引数の型	結果の型	備考
AIMAG	基本複素数型	基本実数型	
DIMAG	DOUBLE COMPLEX 型	DOUBLE PRECISION型	
QIMAG	COMPLEX(16)	REAL(16)	

11.1.5 AINT(*A*) 個別名

- **機能**

  整数値への切捨て

- **分類**

  要素別処理関数

- **引数**

  A:　実数型

- **結果の型、および、型パラメタ**

  *A*と同じ型

- **結果の値**

  |*A*|<1 のときの値は、0

  |*A*|≧1 のときの値は、*A*の絶対値以下で最大の整数値に*A*の符号を付けた値

- **個別名**

個別名	引数の型	結果の型	備考
AINT	基本実数型	基本実数型	規格で定義
DINT	DOUBLE PRECISION型	DOUBLE PRECISION型	規格で定義
QINT	REAL(16)	REAL(16)	

11.1.6 AMT(*X*)

- **機能**

  仮数部の取出し

- **分類**

  要素別処理関数

- **引数**

  X:　実数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  *X*の仮数部の値

- **個別名**

個別名	引数の型	結果の型	備考
AMT	基本実数型	基本実数型	
DMT	DOUBLE PRECISION型	DOUBLE PRECISION型	
QMT	REAL(16)	REAL(16)	

11.1.7 AND(*I,J*)

IANDの別名の関数。詳細は、「11.1.44 IAND(*I,J*) 個別名11.1.48」を参照してください。

11.1.8 ANINT(*A*) 個別名

- **機能**

  最も近い整数値を返します(四捨五入)。

- **分類**

  要素別処理関数

- **引数**

  A: 実数型

- **結果の型、および、型パラメタ**

  *A*と同じ型

- **結果の値**

  A>0 のときの値は、AINT(*A*+0.5)

  A≦0 のときの値は、AINT(*A*-0.5)

- **個別名**

個別名	引数の型	結果の型	備考
ANINT	基本実数型	基本実数型	規格で定義
DNINT	DOUBLE PRECISION型	DOUBLE PRECISION型	規格で定義
QNINT	REAL(16)	REAL(16)	

11.1.9 ASIN(*X*) 個別名

- **機能**

  逆正弦関数

- **分類**

  要素別処理関数

- **引数**

  X:　実数型。|*X*|≦1を満たす値でなければなりません。

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  ラジアンで表した逆正弦**arcsin**(*X*)の値

- **個別名**

個別名	引数の型	結果の型	備考
ASIN	基本実数型	基本実数型	規格で定義
DASIN	DOUBLE PRECISION型	DOUBLE PRECISION型	規格で定義
QASIN	REAL(16)	REAL(16)	

11.1.10 ASINH(*X*) 個別名

- **機能**

  逆双曲線正弦関数

- **分類**

  要素別処理関数

- **引数**

  X:　実数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  逆双曲線正弦**arcsinh**(*X*)の値

- **個別名**

個別名	引数の型	結果の型	備考
ASINH	基本実数型	基本時数型	
DASINH	DOUBLE PRECISION型	DOUBLE PRECISION型	
QASINH	REAL(16)	REAL(16)	

11.1.11 ATAN(*X*) 個別名

- **機能**

  逆正接関数

- **分類**

  要素別処理関数

- **引数**

  X:　実数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  ラジアンで表した逆正接**arctan**(*X*)の値

- **個別名**

個別名	引数の型	結果の型	備考
ATAN	基本実数型	基本実数型	規格で定義
DATAN	DOUBLE PRECISION型	DOUBLE PRECISION型	規格で定義
QATAN	REAL(16)	REAL(16)	

11.1.12 ATAN2(*Y*,*X*) 個別名

- **機能**

  逆正接関数

- **分類**

  要素別処理関数

- **引数**

  Y:　実数型

  X:　*Y*と同じ種別パラメタの実数型。*Y*が0.0のとき、*X*は0.0であってはなりません。

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  ラジアンで表した複素数(*Y*,*X*)の偏角の主値に対する値

- **個別名**

個別名	引数の型	結果の型	備考
ATAN2	基本実数型	基本実数型	規格で定義
DATAN2	DOUBLE PRECISION型	DOUBLE PRECISION型	規格で定義
QATAN2	REAL(16)	REAL(16)	

11.1.13 ATANH(*X*)　個別名

- **機能**

  逆双曲線正接関数

- **分類**

  要素別処理関数

- **引数**

  X:　実数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  逆双曲線正接**arctanh**(*X*)の値

- **個別名**

個別名	引数の型	結果の型	備考
ATANH	基本実数型	基本実数型	
DATANH	DOUBLE PRECISION型	DOUBLE PRECISION型	

QATANH	REAL(16)	REAL(16)

11.1.14 BTEST(*I*,*POS*) 個別名

- **機能**

  整数値のビットを調べます。

- **分類**

  要素別処理関数

- **引数**

  I:　整数型

  POS:　　整数型。値は、**BIT_SIZE**(*I*)未満で0以上でなければなりません。

- **結果の型、および、型パラメタ**

  基本論理型

- **結果の値**

  *I*の第*POS*ビットが1であるとき、結果の値は真とします。*I*の第*POS*ビットが0であるとき、結果の値は偽とします。

- **個別名**

個別名	引数の型	結果の型	備考
BBTEST	INTEGER(1)	INTEGER(1)	
BITEST, HTEST	INTEGER(2)	INTEGER(2)	
BTEST, BJTEST	INTEGER(4)	INTEGER(4)	
BKTEST	INTEGER(8)	INTEGER(8)	

11.1.15 CANG(*X*)

- **機能**

  偏角

- **分類**

  要素別処理関数

- **引数**

  X:　複素数型

- **結果の型、および、型パラメタ**

  *X*と同じ種別パラメタの実数型

- **結果の値**

  *X*の偏角の値

- **個別名**

個別名	引数の型	結果の型	備考
CANG	default complex	default real	
CDANG, ZANG	DOUBLE COMPLEX 型	DOUBLE PRECISION型	

11.1.16 CBRT(*X*)

- **機能**

  立方根

- **分類**

  要素別処理関数

- **引数**

  X:　実数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  *X*の立方根の値

- **個別名**

個別名	引数の型	結果の型	備考
CBRT	基本実数型	基本実数型	
DCBRT	DOUBLE PRECISION型	DOUBLE PRECISION型	
QCBRT	REAL(16)	REAL(16)	

11.1.17 CLOCK(*D*)

- **機能**

  CPU時間を求めます。

- **分類**

  サブルーチン

- **引数**

  D: **INTENT(OUT)**属性を持つ倍精度実数型、または、4倍精度実数型のスカラ変数。プログラムの実行が開始されてから、このサブルーチンが引用されるまでのCPU時間の累積実行時間(単位：秒、精度はマイクロ秒まで)が設定されます。

11.1.18 CONJG(*Z*) 個別名

- **機能**

  共役複素数

- **分類**

  要素別処理関数

- **引数**

  Z: 複素数型

- **結果の型、および、型パラメタ**

  *Z*と同じ型

- **結果の値**

  *Z*の値が(*x*,*y*)のとき、結果の値は(*x*,-*y*)とします。

- **個別名**

個別名	引数の型	結果の型	備考
CONJG	基本複素数型	基本複素数型	規格で定義
DCONJG	DOUBLE COMPLEX 型	DOUBLE COMPLEX 型	
QCONJG	COMPLEX(16)	COMPLEX(16)	

11.1.19 COS(*X*) 個別名

- **機能**

  余弦関数

- **分類**

  要素別処理関数

- **引数**

  X: 実数型、または、複素数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  余弦**cos**(*X*)の値。*X*が実数型のとき、ラジアンでの値とみなされます。ただし、引数が単精度で値の絶対値が2^{21}×**π**以上であるとき、結果の値はNaNです。*X*が複素数型のとき、その実部がラジアンでの値とみなされます。ただし、引数が単精度で実部の値の絶対値が2^{21}×**π**以上であるとき、結果の値はNaNです。

  他の精度の注意事項は、「11.5 注意事項」を参照してください。

- **個別名**

個別名	引数の型	結果の型	備考
COS	基本実数型	基本実数型	規格で定義
DCOS	DOUBLE PRECISION型	DOUBLE PRECISION型	規格で定義
QCOS	REAL(16)	REAL(16)	
CCOS	基本複素数型	基本複素数型	規格で定義
CDCOS	DOULE COMPLEX型	DOUBLE COMPLEX型	
ZCOS	COMPLEX(8)	COMPLEX(8)	
CQCOS	COMPLEX(16)	COMPLEX(16)	

11.1.20 COSD(*X*)

- **機能**

  余弦関数

- **分類**

  要素別処理関数

- **引数**

  X:　実数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  *X*を度の値としたときの余弦**cos**(*X*)の値。ただし、引数の値の絶対値が2^{21}×180以上であるとき、結果の値はNaNです。

- **個別名**

個別名	引数の型	結果の型	備考
COSD	基本実数型	基本実数型	
DCOSD	DOUBLE PRECISION型	DOUBLE PRECISION型	
QCOSD	REAL(16)	REAL(16)	

11.1.21 COSH(*X*) 個別名

- **機能**

  双曲線余弦関数

- **分類**

  要素別処理関数

- **引数**

  X:　実数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  双曲線余弦**cosh**(*X*)の値

- **個別名**

個別名	引数の型	結果の型	備考
COSH	基本実数型	基本実数型	規格で定義
DCOSH	DOUBLE PRECISION型	DOUBLE PRECISION型	規格で定義
QCOSH	REAL(16)	REAL(16)	

11.1.22 COTAN(*X*)

- **機能**

  余接

- **分類**

  要素別処理関数

- **引数**

  X:　実数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  余接**cotan**(*X*)の値。ただし、引数の値の絶対値が2^{50}×180以上であるとき、結果の値はNaNです。

- **個別名**

個別名	引数の型	結果の型	備考
COTAN	基本実数型	基本実数型	
DCOTAN	DOUBLE PRECISION型	DOUBLE PRECISION型	
QCOTAN	REAL(16)	REAL(16)	

11.1.23 DATE(*A*)

- **機能**

  日付を求めます。

- **分類**

  サブルーチン

- **引数**

  A:　**INTENT(OUT)**属性を持つ長さ8文字の基本文字型のスカラ変数。"*yy-mm-dd*"の形式の日付の値が設定されます。

11.1.24 DATIM(*A*,*B*,*C*)

- **機能**

  日付と時刻を求めます。

- **分類**

  サブルーチン

- **引数**

  A:　**INTENT(OUT)**属性を持つ長さ8文字の基本文字型のスカラ変数。引数Cで指定した形式の日付が設定されます。

  B:　**INTENT(OUT)**属性を持つ基本実数型または長さ8文字の基本文字型のスカラ変数。基本実数型の場合、時間を単位とするそのときの時刻が設定されます。基本文字型の場合、そのときの時刻が"*hh*:*mm*:*ss*"の形式で設定されます。

C(省略可能): **INTENT(IN)**属性を持つ基本整数型のスカラ値。引数*A*へ返す日付の形式を指定します。

1　　*yy-mm-dd*(既定値)

3　　*mm*/*dd*/*yy*

4　　*dd*/*mm*/*yy*

11.1.25 DBLE(*A*) 個別名

- **機能**

  倍精度実数型への変換

- **分類**

  要素別処理関数

- **引数**

  A:　実数型

- **結果の型、および、型パラメタ**

  倍精度実数型

- **結果の値**

  REAL(*A*,KIND(0.0D0))の値

- **個別名**

個別名	引数の型	結果の型	備考
DBLE	基本実数型	DOUBLE PRECISION型	規格で定義
DBLEQ	REAL(16)	DOUBLE PRECISION型	

11.1.26 DCMPLX(*X*,*Y*)

- **機能**

  倍精度複素数型への変換

- **分類**

  要素別処理関数

- **引数**

  X:　整数型、実数型または複素数型

  Y(省略可能): 整数型または実数型。*X*が複素数型のとき、指定できません。

- **結果の型、および、型パラメタ**

  倍精度複素数型

- **結果の値**

  CMPLX(*X*,*Y*,KIND=KIND(0.0D0))の値

11.1.27 DFACT(*I*)

- **機能**

  階乗

- **分類**

  要素別処理関数

- **引数**

  I:　基本整数型

- **結果の型、および、型パラメタ**

  倍精度実数型

- **結果の値**

  *I*の階乗の値を倍精度実数型に変換した値

11.1.28 DFLOAT(*A*)

- **機能**

  倍精度実数型への変換

- **分類**

  要素別処理関数

- **引数**

  A:　整数型

- **結果の型、および、型パラメタ**

  倍精度実数型

- **結果の値**

  REAL(*A*,KIND=KIND(0.0D0))の値

- **個別名**

個別名	引数の型	結果の型	備考
DFLOTI	INTEGER(2)	DOUBLE PRECISION型	

DFLOTJ	基本整数型	DOUBLE PRECISION型
DFLOTK	INTEGER(8)	DOUBLE PRECISION型

11.1.29 DIM(*X*,*Y*) 個別名

- **機能**

  超過分X-Y が正のときはX-Y、正でないときはゼロとします。

- **分類**

  要素別処理関数

- **引数**

  X: 整数型、または、実数型

  Y: *A*と同じ種別パラメタの整数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  *X*より*Y*が小さいとき、*X-Y*。*Y*が*X*以上のとき、ゼロとします。

- **個別名**

個別名	引数の型	結果の型	備考
BDIM	INTEGER(1)	INTEGER(1)	
IIDIM, HDIM	INTEGER(2)	INTEGER(2)	
IDIM	基本整数型	基本整数型	規格で定義
JIDIM	INTEGER(4)	INTEGER(4)	
KIDIM	INTEGER(8)	INTEGER(8)	
DIM	基本実数型	基本時数型	規格で定義
DDIM	DOUBLE PRECISION型	DOUBLE PRECISION型	規格で定義
QDIM	REAL(16)	REAL(16)	

11.1.30 DREAL(*A*)

- **機能**

  倍精度実数型への変換

- **分類**

  要素別処理関数

- **引数**

  A:　複素数型

- **結果の型、および、型パラメタ**

  倍精度実数型

- **結果の値**

  *A*の値が(*x*,*y*)のとき、結果の値は*x*です。

11.1.31 ERF(*X*)　個別名

- **機能**

  誤差関数

- **分類**

  要素別処理関数

- **引数**

  X:　実数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  *X*の誤差関数の値

- **個別名**

個別名	引数の型	結果の型	備考
ERF	基本実数型	基本実数型	
DERF	DOUBLE PRECISION型	DOUBLE PRECISION型	
QERF	REAL(16)	REAL(16)	

11.1.32 ERFC(*X*) 個別名

- **機能**

  補誤差関数

- **分類**

  要素別処理関数

- **引数**

  X: 実数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  1.0から*X*の誤差関数の値を引いた値

- **個別名**

個別名	引数の型	結果の型	備考
ERFC	基本実数型	基本実数型	
DERFC	DOUBLE PRECISION型	DOUBLE PRECISION型	
QERFC	REAL(16)	REAL(16)	

11.1.33 ETIME(*D*)

- **機能**

  システム起動時からの経過時間を求めます。

- **分類**

  サブルーチン

- **引数**

  D: **INTENT(OUT)**属性を持つ倍精度実数型。システム起動時からの経過時間（単位: 秒）が設定されます。

- **備考**

  関数として使用するときの詳細は、「11.4.8 ETIME(*TARRAY*)」を参照してください。

11.1.34 EXIT(*X*)

- **機能**

プログラムの実行を終了します。

- **分類**

  サブルーチン

- **引数**

  X:　**INTENT(IN)**属性を持つ整数型。実行ステータスコードの値を指定します。

11.1.35 EXP(*X*) 個別名

- **機能**

  指数関数

- **分類**

  要素別処理関数

- **引数**

  X:　実数型、または、複素数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  e^{x}の値。*X*が複素数型のとき、虚部の値はラジアンを単位とします。ただし、引数が単精度で虚部の値の絶対値が$2^{21}\times\pi$以上であるとき、結果の値はNaNです。

  他の精度の注意事項は、「11.5 注意事項」を参照してください。

- **個別名**

個別名	引数の型	結果の型	備考
EXP	基本実数型	基本実数型	規格で定義
DEXP	DOUBLE PRECISION型	DOUBLE PRECISION型	規格で定義
QEXP	REAL(16)	REAL(16)	
CEXP	基本複素数型	基本複素数型	規格で定義
CDEXP	DOUBLE COMPLEX型	DOUBLE COMPLEX型	
ZEXP	COMPLEX(8)	COMPLEX(8)	
CQEXP	COMPLEX(16)	COMPLEX(16)	

11.1.36 EXP10(*X*)

- **機能**

  指数

- **分類**

  要素別処理関数

- **引数**

  X:　実数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  10.0^{X}の値

- **個別名**

個別名	引数の型	結果の型	備考
EXP10	基本実数型	基本実数型	
DEXP10	DOUBLE PRECISION型	DOUBLE PRECISION型	
QEXP10	REAL(16)	REAL(16)	

11.1.37 EXP2(*X*)

- **機能**

  指数

- **分類**

  要素別処理関数

- **引数**

  X:　実数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  2.0^{X}の値

- **個別名**

個別名	引数の型	結果の型	備考
EXP2	基本実数型	基本実数型	
DEXP2	DOUBLE PRECISION型	DOUBLE PRECISION型	
QEXP2	REAL(16)	REAL(16)	

11.1.38 EXPC(*X*)

- **機能**

  指数

- **分類**

  要素別処理関数

- **引数**

  X:　実数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  $e^{X-1.0}$の値

- **個別名**

個別名	引数の型	結果の型	備考
EXPC	基本実数型	基本実数型	
DEXPC	DOUBLE PRECISION型	DOUBLE PRECISION型	

11.1.39 EXPC10(*X*)

- **機能**

  指数

- **分類**

  要素別処理関数

- **引数**

  X:　実数型

- **結果の型、および、型パラメタ**

*X*と同じ型

- **結果の値**

  $10.0^{X-1.0}$の値

- **個別名**

個別名	引数の型	結果の型	備考
EXPC10	基本実数型	基本実数型	
DXPC10	DOUBLE PRECISION型	DOUBLE PRECISION型	
QXPC10	REAL(16)	REAL(16)	

11.1.40 EXPC2(*X*)

- **機能**

  指数

- **分類**

  要素別処理関数

- **引数**

  X:　実数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  $2.0^{X-1.0}$の値

- **個別名**

個別名	引数の型	結果の型	備考
EXPC2	基本実数型	基本実数型	
DEXPC2	DOUBLE PRECISION型	DOUBLE PRECISION型	
QEXPC2	REAL(16)	REAL(16)	

11.1.41 FACT(*I*)

- **機能**

  階乗

- **分類**

  要素別処理関数

- **引数**

  I:　基本整数型

- **結果の型、および、型パラメタ**

  基本実数型

- **結果の値**

  *I*の階乗の値を基本実数型に変換した値

11.1.42 FLUSH(*UNIT*)

- **機能**

  バッファの内容をファイルに出力します。

- **分類**

  サブルーチン

- **引数**

  UNIT:　**INTENT(IN)**属性を持つ整数型。バッファを出力する外部ファイル装置を指定します。

11.1.43 GAMMA(X) 個別名

- **機能**

  ガンマ関数

- **分類**

  要素別処理関数

- **引数**

  X:　実数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  *X*のガンマ関数の値

- **個別名**

個別名	引数の型	結果の型	備考
GAMMA	基本実数型	基本実数型	規格で定義
DGAMMA	DOUBLE PRECISION型	DOUBLE PRECISION型	

11.1.44 IAND(*I*,*J*) 個別名

- **機能**

  ビットごとの論理積

- **分類**

  要素別処理関数

- **引数**

  I:　整数型

  J:　*I*と同じ種別パラメタの整数型

- **結果の型、および、型パラメタ**

  *I*と同じ型

- **結果の値**

  結果の値は、*I*と*J*をビットごとに組み合わせ、次の真理値表にしたがって得られる値とします。

I	J	IAND(I,J)
1	1	1
1	0	0
0	1	0
0	0	0

- **備考**

  引数の数は3以上でもかまいません。そのとき、引数は、*I*と同じ種別パラメタの整数型でなければなりません。また、3以上の引数を指定するとき、キーワード指定はできません。

- **個別名**

個別名	引数の型	結果の型	備考
BIAND	INTEGER(1)	INTEGER(1)	
IIAND, HIAND	INTEGER(2)	INTEGER(2)	
JIAND	INTEGER(4)	INTEGER(4)	
KIAND	INTEGER(8)	INTEGER(8)	

11.1.45 IBCLR(*I*,*POS*) 個別名

- 機能

  一つのビットを0に設定します。

- 分類

  要素別処理関数

- **引数**

  I:　整数型

  POS:　　整数型。値は、**BIT_SIZE**(*I*)未満で0以上でなければなりません。

- **結果の型、および、型パラメタ**

  *I*と同じ型

- **結果の値**

  結果の値は、*I*の第*POS*ビットを0に設定した値とします。

- **個別名**

個別名	引数の型	結果の型	備考
BBCLR	INTEGER(1)	INTEGER(1)	
IIBCLR, HBCLR	INTEGER(2)	INTEGER(2)	
JIBCLR	INTEGER(4)	INTEGER(4)	
KIBCLR	INTEGER(8)	INTEGER(8)	

11.1.46 IBITS(*I*,*POS*,*LEN*) 個別名

- 機能

  ビットの並びを取り出します。

- 分類

  要素別処理関数

- **引数**

  I:　整数型

  POS:　　整数型。値は、*POS*+*LEN*が**BIT_SIZE**(*I*)以下で0以上でなければなりません。

  LEN:　　整数型。値は、0以上でなければなりません。

- **結果の型、および、型パラメタ**

  *I*と同じ型

- **結果の値**

  結果の値は、*I*の第*POS*ビットから始まる*LEN*個のビットを右詰めし、残りのビットを0にした値とします。

- **個別名**

個別名	引数の型	結果の型	備考
BBITS	INTEGER(1)	INTEGER(1)	
IIBITS, HBITS	INTEGER(2)	INTEGER(2)	
JIBITS	INTEGER(4)	INTEGER(4)	
KIBITS	INTEGER(8)	INTEGER(8)	

11.1.47 IBSET(*I*,*POS*) 個別名

- 機能

  一つのビットを1に設定します。

- 分類

  要素別処理関数

- **引数**

  I:　整数型

  POS:　　整数型。値は、**BIT_SIZE**(*I*)未満で0以上でなければなりません。

- **結果の型、および、型パラメタ**

  *I*と同じ型

- **結果の値**

  結果の値は、*I*の第*POS*ビットを1に設定した値とします。

- **個別名**

個別名	引数の型	結果の型	備考
BBSET	INTEGER(1)	INTEGER(1)	
IIBSET, HBSET	INTEGER(2)	INTEGER(2)	
JIBSET	INTEGER(4)	INTEGER(4)	
KIBSET	INTEGER(8)	INTEGER(8)	

11.1.48 IEOR(*I*,*J*) 個別名

- 機能

  ビットごとの排他的論理和

- 分類

  要素別処理関数

- **引数**

  I:　整数型

  J:　*I*と同じ種別パラメタの整数型

- **結果の型、および、型パラメタ**

  *I*と同じ型

- **結果の値**

  結果の値は、*I*と*J*をビットごとに組み合わせ、次の真理値表にしたがって得られる値とします。

I	***J***	**IEOR(*I*,*J*)**
1	1	0
1	0	1
0	1	1
0	0	0

- **備考**

  引数の数は3以上でもかまいません。そのとき、引数は、*I*と同じ種別パラメタの整数型でなければなりません。また、3以上の引数を指定するとき、キーワード指定はできません。

- **個別名**

個別名	引数の型	結果の型	備考
BIEOR, BIXOR	INTEGER(1)	INTEGER(1)	
IIEOR, HIEOR, HIXOR, IIXOR	INTEGER(2)	INTEGER(2)	
JIEOR, JIXOR	INTEGER(4)	INTEGER(4)	
KIEOR	INTEGER(8)	INTEGER(8)	

11.1.49 IMAG(*A*)

AIMAGの別名の関数。詳細は、「11.1.4 AIMAG(*Z*) 個別名」を参照してください。

11.1.50 INT(*A*[,*KIND*]) 個別名

- **機能**

  最も近い整数を返します(切り捨て)

- **分類**

  要素別処理関数

- **引数**

  A: 整数型、実数型、または、複素数型

  KIND(省略可能): 整数型。パラメタ種別の値。

- **結果の型、および、型パラメタ**

  整数型。*KIND*を指定したとき、種別パラメタは*KIND*の指定に従います。*KIND*を省略したとき、種別パラメタは基本整数型に対する種別パラメタとします。

- **結果の値**

  *A*が整数型のとき、INT(*A*)の値はAとします。*A*が実数型のとき、|*A*|<1のとき、INT(*A*)は0とします。|*A*|≧1のとき、INT(*A*)の値は、*A*の絶対値以下で最大の整数に*A*の符号を付けたものとします。*A*が複素数型のとき、INT(*A*)の値は*A*の実部に実数型のときと同じ規則を適用して得られる値とします。

- **個別名**

個別名	引数の型	結果の型	備考
INT1	INTEGER(*), REAL(*), COMPLEX(*)	INTEGER(1)	

IJINT	INTEGER(4)	INTEGER(2)	
INT2	INTEGER(*), REAL(*), COMPLEX(*)	INTEGER(2)	
IIFIX, IINT, IINT, HFIX	REAL(4)	INTEGER(2)	
IIDINT	REAL(8)	INTEGER(2)	
IIQINT	REAL(16)	INTEGER(2)	
INT4, JFIX	INTEGER(*), REAL(*), COMPLEX(*)	基本整数型	
JIFIX	REAL(*)	基本整数型	
INT, JINT	基本実数型	基本整数型	INTは、規格で定義
IDINT, JIDINT	DOUBLE PRECISION型	基本整数型	IDINTは、規格で定義
IQINT, JIQINT	REAL(16)	基本整数型	
INT8	INTEGER(*), REAL(*), COMPLEX(*)	INTEGER(8)	
KIFIX, KINT	REAL(4)	INTEGER(8)	
KIDINT	REAL(8)	INTEGER(8)	
KIQINT	REAL(16)	INTEGER(8)	

11.1.51 IOR(*I*,*J*) 個別名

- **機能**

  ビットごとの論理和

- **分類**

  要素別処理関数

- **引数**

  I:　整数型

  J:　*I*と同じ種別パラメタの整数型

- **結果の型、および、型パラメタ**

  *I*と同じ型

- **結果の値**

  結果の値は、*I*と*J*をビットごとに組み合わせ、次の真理値表にしたがって得られる値とします。

I	***J***	**IOR(*I*,*J*)**
1	1	1
1	0	1
0	1	1
0	0	0

- **備考**

  引数の数は3以上でもかまいません。そのとき、引数は、*I*と同じ種別パラメタの整数型でなければなりません。また、3以上の引数を指定するとき、キーワード指定はできません。

- **個別名**

個別名	引数の型	結果の型	備考
BIOR	INTEGER(1)	INTEGER(1)	
IIOR, HIOR	INTEGER(2)	INTEGER(2)	
JIOR	INTEGER(4)	INTEGER(4)	
KIOR	INTEGER(8)	INTEGER(8)	

11.1.52 IRE(*X*)

- **機能**

  指数部の取出し

- **分類**

  要素別処理関数

- **引数**

  X:　実数型

- **結果の型、および、型パラメタ**

  基本整数型

- **結果の値**

  *X*の指数部の値

- **個別名**

個別名	引数の型	結果の型	備考
IRE	基本実数型	基本整数型	
IDE	DOUBLE PRECISION型	基本整数型	
IQE	REAL(16)	基本整数型	

11.1.53 ISHFT(*I*,*SHIFT*) 個別名

- 機能

  論理けた移動

- 分類

  要素別処理関数

- **引数**

  I:　整数型

  SHIFT: 整数型。絶対値が**BIT_SIZE**(*I*)以下でなければなりません。

- **結果の型、および、型パラメタ**

  *I*と同じ型

- **結果の値**

  結果の値は、ビットを*SHIFT*個だけけた(桁)移動して得られる値とします。

- **個別名**

個別名	引数の型	結果の型	備考
BSHFT	INTEGER(1)	INTEGER(1)	
IISHFT, HSHFT	INTEGER(2)	INTEGER(2)	
JISHFT	INTEGER(4)	INTEGER(4)	
KISHFT	INTEGER(8)	INTEGER(8)	

11.1.54 ISHFTC(*I*,*SHIFT*[,*SIZE*]) 個別名

- 機能

  右端のビット並びの循環けた移動を行います。

- 分類

  要素別処理関数

- **引数**

  I:　整数型

  SHIFT: 整数型。絶対値が**BIT_SIZE**(*I*)以下でなければなりません。

  SIZE(省略可能): 整数型。値は、**BIT_SIZE**(*I*)以下で0以上でなければなりません。*SIZE*を省略したとき、**BIT_SIZE**(*I*)の値を指定したものとします。

- **結果の型、および、型パラメタ**

  *I*と同じ型

- **結果の値**

  結果の値は、*I*の右端の*SIZE*個のビットを*SHIFT*個だけ循環けた(桁)移動して得られる値とします。

- **個別名**

個別名	引数の型	結果の型	備考
BSHFTC	INTEGER(1)	INTEGER(1)	
IISHFTC, HSHFTC	INTEGER(2)	INTEGER(2)	
JISHFTC	INTEGER(4)	INTEGER(4)	
KISHFTC	INTEGER(8)	INTEGER(8)	

11.1.55 ISNAN(*X*)

- **機能**

  NaNかどうか検査します。

- **分類**

  要素別処理関数

- **引数**

  X:　実数型

- **結果の型、および、型パラメタ**

  基本論理型

- **結果の値**

  *X*がNaNであるとき、結果の値は真とします。*X*がNaN以外のとき、結果の値は偽としま

す。

11.1.56 IXOR(*I,J*)

IEORの別名の関数。詳細は、「11.1.48 11.1.48IEOR(*I,J*) 個別名」を参照してください。

11.1.57 LGAMMA(*X*)

- **機能**

  対数ガンマ関数

- **分類**

  要素別処理関数

- **引数**

  X: 実数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  *X*の対数ガンマ関数の値

- **個別名**

個別名	引数の型	結果の型	備考
ALGAMA	基本実数型	基本実数型	
DLGAMA	DOUBLE PRECISION型	DOUBLE PRECISION型	

11.1.58 LOC(*X*)

- **機能**

  アドレス取出し。

- **分類**

  変形関数

- **引数**

  X: 任意の型の変数

- **結果の型、および、型パラメタ**

  8バイト整数型

- **結果の値**

*X*のアドレスの値

11.1.59 LOG(*X*) 個別名

- **機能**

  自然対数

- **分類**

  要素別処理関数

- **引数**

  X:　実数型、または、複素数型。*X*が実数型のとき、値は正でなければなりません。*X*が複素数型のとき、値は(0.0,0.0)であってはなりません。

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  対数$\log_e(X)$の値。複素数型のとき、$-\pi<\omega\leqq\pi$の範囲にある虚部ωをもつ主値とします。結果の虚部は、引数の実部が負であって虚部が0.0のときに限り、πとします。

- **個別名**

個別名	引数の型	結果の型	備考
ALOG	基本実数型	基本実数型	規格で定義
DLOG	DOUBLE PRECISION型	DOUBLE PRECISION型	規格で定義
QLOG	REAL(16)	REAL(16)	
CLOG	基本複素数型	基本複素数型	規格で定義
CDLOG	DOUBLE COMPLEX型	DOUBLE COMPLEX型	
ZLOG	COMPLEX(8)	COMPLEX(8)	
CQLOG	COMPLEX(16)	COMPLEX(16)	

11.1.60 LOG10(*X*) 個別名

- **機能**

  常用対数

- **分類**

要素別処理関数

- **引数**

  X:　実数型

- **結果の型および型パラメタ**

  *X*と同じ型

- **結果の値**

  対数$\log_{10}(X)$の値

- **個別名**

個別名	引数の型	結果の型	備考
ALOG10	基本実数型	基本実数型	規格で定義
DLOG10	DOUBLE PRECISION型	DOUBLE PRECISION型	規格で定義
QLOG10	REAL(16)	REAL(16)	

11.1.61 LOG2(*X*)

- **機能**

  対数

- **分類**

  要素別処理関数

- **引数**

  X:　実数型

- **結果の型および型パラメタ**

  *X*と同じ型

- **結果の値**

  対数$\log_{2}(X)$の値

- **個別名**

個別名	引数の型	結果の型	備考
ALOG2	基本実数型	基本実数型	
DLOG2	DOUBLE PRECISION型	DOUBLE PRECISION型	

11.1.62 MAX(*A1*,*A2*[,*A3*,...]) 個別名

- **機能**

  最大値

- **分類**

  要素別処理関数

- **引数**

  An: 全て同じ整数型、または、実数型で同じ種別パラメタ

- **結果の型、および、型パラメタ**

  *An*と同じ型

- **結果の値**

  *An*の最大の値

- **個別名**

個別名	引数の型	結果の型	備考
IMAX0	INTEGER(2)	INTEGER(2)	
AIMAX0	INTEGER(2)	基本実数型	
MAX0, JMAX0	基本整数型	基本整数型	MAX0は、規格で定義
AMAX0, AJMAX0	基本整数型	基本実数型	AMAX0は、規格で定義
DMAX0	基本整数型	DOUBLE PRECISION型	
KMAX0	INTEGER(8)	INTEGER(8)	
AKMAX0	INTEGER(8)	基本実数型	
IMAX1	基本実数型	INTEGER(2)	
MAX1, JMAX1	基本実数型	基本整数型	MAX1は、規格で定義
KMAX1	基本実数型	INTEGER(8)	
AMAX1	基本実数型	基本実数型	規格で定義
DMAX1	DOUBLE PRECISION型	DOUBLE PRECISION型	規格で定義

11.1.63 MAXVL()

- **機能**

  最大ベクトルレジスタ長を求めます。

- **分類**

  問い合わせ関数

- **結果の型、および、型パラメタ**

  基本整数型

- **結果の値**

  システムの最大ベクトルレジスタ長

11.1.64 MIN(*A1*,*A2*[,*A3*,...]) 個別名

- **機能**

  最小値

- **分類**

  要素別処理関数

- **引数**

  An: 全て同じ整数型、または、実数型で同じ種別パラメタ

- **結果の型、および、型パラメタ**

  *An*と同じ型

- **結果の値**

  *An*の最小の値

- **個別名**

個別名	引数の型	結果の型	備考
IMIN0	INTEGER(2)	INTEGER(2)	
AIMIN0	INTEGER(2)	基本実数型	
MIN0, JMIN0	基本整数型	基本整数型	MIN0は、規格で定義
AMIN0, AJMIN0	基本整数型	基本実数型	AMIN0は、規格で定義
DMIN0	基本整数型	DOUBLE PRECISION型	

KMIN0	INTEGER(8)	INTEGER(8)	
AKMIN0	INTEGER(8)	基本実数型	
IMIN1	基本実数型	INTEGER(2)	
MIN1, JMIN1	基本実数型	基本整数型	MIN1は、規格で定義
KMIN1	基本実数型	INTEGER(8)	
AMIN1	基本実数型	基本実数型	規格で定義
DMIN1	DOUBLE PRECISION型	DOUBLE PRECISION型	規格で定義

11.1.65 MOD(*A*,*P*) 個別名

- **機能**

  余り関数

- **分類**

  要素別処理関数

- **引数**

  A: 整数型、または、実数型

  P: *A*と同じ種別パラメタの整数型

- **結果の型、および、型パラメタ**

  *A*と同じ型

- **結果の値**

  *P*が0以外のとき、結果の値は*A*-INT(*A*/*P*)***P*とします。*P*が0のとき、結果は不定となります。

- **個別名**

個別名	引数の型	結果の型	備考
BMOD	INTEGER(1)	INTEGER(1)	
IMOD, HMOD	INTEGER(2)	INTEGER(2)	
MOD	基本整数型	基本整数型	規格で定義
JMOD	INTEGER(4)	INTEGER(4)	
KMOD	INTEGER(8)	INTEGER(8)	
AMOD	基本実数型	基本実数型	規格で定義

DMOD	DOBULE PRECISION型	DOBULE PRECISION型	規格で定義
QMOD	REAL(16)	REAL(16)	

11.1.66 MVBITS(*FROM*,*FROMPOS*,*LEN*,*TO*,*TOPOS*) 個別名

- **機能**

  システム起動時からの経過時間を求めます。

- **分類**

  サブルーチン

- **引数**

  FROM:　**INTENT(IN)**属性を持つ整数型。

  FORMPOS:　**INTENT(IN)**属性を持つ整数型。値は、*FROMPOS*+*LEN*が**BIT_SIZE**(*FROM*)以下で0以上でなければなりません。

  LEN:　**INTENT(IN)**属性を持つ整数型。値は、0以上でなければなりません。

  TO:　**INTENT(OUT)**属性を持つ*TO*と同じ型の整数型。*FROM*の位置*FROMPOS*から始まる長さ*LEN*のビット列を*TO*の位置*TOPOS*に複写します。*TO*の他のどのビットも変更されません。戻るとき、*TOPOS*から始まる*TO*の*LEN*ビットは、呼出し時に*FROMPOS*から始まる*FROM*の*LEN*ビットが持っていた値に等しくなります。

  TOPOS: **INTENT(IN)**属性を持つ整数型。値は、*TOPOS*+*LEN*が**BIT_SIZE**(*TO*)以下で0以上でなければなりません。

- **個別名**

個別名	引数の型	結果の型	備考
BMVBITS	INTEGER(1)	-	
IMVBITS, HMVBITS	INTEGER(2)	-	
JMVBITS	INTEGER(4)	-	
KMVBITS	INTEGER(8)	-	

11.1.67 NINT(*A*[,*KIND*]) 個別名

- **機能**

  最も近い整数を返します(四捨五入)

- **分類**

要素別処理関数

- **引数**

  A: 実数型

  KIND(省略可能): 整数型。パラメタ種別の値。

- **結果の型、および、型パラメタ**

  整数型。*KIND*を指定したとき、種別パラメタは*KIND*の指定に従います。*KIND*を省略したとき、種別パラメタは基本整数型に対する種別パラメタとします。

- **結果の値**

  A>0のとき、NINT(*A*)の値はINT(*A*+0.5)とします。*A*≦0のとき、NINT(*A*)の値はINT(*A*-0.5)とします。

- **個別名**

個別名	引数の型	結果の型	備考
ININT	REAL(4)	INTEGER(2)	
NINT	基本実数型	基本整数型	規格で定義
JNINT	REAL(4)	INTEGER(4)	
KNINT	REAL(4)	INTEGER(8)	
IIDNNT	REAL(8)	INTEGER(2)	
IDNINT	DOLUBLE PRECISION型	基本整数型	規格で定義
JIDNNT	REAL(8)	INTEGER(4)	
KIDNNT	REAL(8)	INTEGER(8)	
IIQNNT	REAL(16)	INTEGER(2)	
IQNINT	REAL(16)	基本整数型	
JIQNNT	REAL(16)	INTEGER(4)	
KIQNNT	REAL(16)	INTEGER(8)	

11.1.68 NOT(*I*) 個別名

- **機能**

  ビットの論理否定

- **分類**

要素別処理関数

- **引数**

  I:　整数型

- **結果の型、および、型パラメタ**

  *I*と同じ型

- **結果の値**

  結果の値は、次の真理値表にしたがって得られる値とします。

I	**NOT(*I*)**
0	1
1	0

- **個別名**

個別名	引数の型	結果の型	備考
BNOT	INTEGER(1)	INTEGER(1)	
INOT, HNOT	INTEGER(2)	INTEGER(2)	
JNOT	INTEGER(4)	INTEGER(4)	
KNOT	INTEGER(8)	INTEGER(8)	

11.1.69 OR(*I,J*)

IORの別名の関数。詳細は、「11.1.51 IOR(*I,J*) 個別名」を参照してください。

11.1.70 QCMPLX(*X,Y*)

- **機能**

  4倍精度複素数型に変換

- **分類**

  要素別処理関数

- **引数**

  X:　整数型、実数型、または複素数型

  Y(省略可能): 整数型または実数型。Xが複素数型のとき、指定できません。

- **結果の型、および、型パラメタ**

  4倍精度実数型

- **結果の値**

CMPLX(*X*,*Y*,KIND=KIND(0.0Q0))の値

11.1.71 QEXT(*X*)

- **機能**

  4倍精度実数型に変換

- **分類**

  要素別処理関数

- **引数**

  X:　整数型、実数型、または複素数型

- **結果の型、および、型パラメタ**

  4倍精度実数型

- **結果の値**

  REAL(*X*,KIND=KIND(0.0Q0))の値

- **個別名**

個別名	引数の型	結果の型	備考
QEXT	基本実数型	REAL(16)	
QEXTD	REAL(8)	REAL(16)	

11.1.72 QFACT(*I*)

- **機能**

  階乗

- **分類**

  要素別処理関数

- **引数**

  I:　基本整数型

- **結果の型、および、型パラメタ**

  4倍精度実数型

- **結果の値**

  *I*の階乗の値を4倍精度実数型に変換した値

11.1.73 QFLOAT(*A*)

- **機能**

4倍精度実数型への変換

- **分類**

  要素別処理関数

- **引数**

  A:　整数型

- **結果の型、および、型パラメタ**

  4倍精度実数型

- **結果の値**

  REAL(*A*,KIND=KIND(0.0Q0))の値

11.1.74　QREAL(*A*)

- **機能**

  4倍精度実数型への変換

- **分類**

  要素別処理関数

- **引数**

  A:　4倍精度複素数型

- **結果の型、および、型パラメタ**

  *A*と同じ種別パラメタの実数型

- **結果の値**

  *A*の値が(*x*,*y*)のとき、結果の値は*x*です。

11.1.75　REAL(*A*[,*KIND*])　個別名

- **機能**

  実数型への変換

- **分類**

  要素別処理関数

- **引数**

  A:　整数型、実数型、または、複素数型

  KIND(省略可能):　整数型。パラメタ種別の値。

- **結果の型、および、型パラメタ**

  実数型。*A*が整数型または実数型のとき、*KIND*を指定したとき、種別パラメタは*KIND*の

指定に従います。*KIND*を省略したとき、種別パラメタは基本実数型に対する種別パラメタとします。*A*が複素数型のとき、*KIND*を指定したとき、種別パラメタは*KIND*の指定に従います。*KIND*を省略したとき、種別パラメタは*A*の種別パラメタとします。

- **結果の値**

  *A*が整数型または実数型のとき、結果の値は*A*の値です。*A*が複素数型のとき、結果の値は*A*の実部の値です。

- **個別名**

個別名	引数の型	結果の型	備考
FLOATI	INTEGER(2)	基本実数型	
REAL, FLOAT	基本整数型	基本実数型	規格で定義
FLOATJ	INTEGER(4)	REAL(4)	
FLOATK	INTEGER(8)	基本実数型	
SNGL	DOUBLE PRECISION型	基本実数型	規格で定義
SNGLQ	REAL(16)	基本実数型	

11.1.76 RSQRT(*X*)

- **機能**

  逆数平方根

- **分類**

  要素別処理関数

- **引数**

  X:　実数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  *X*の平方根の逆数の値の近似値

- **個別名**

個別名	引数の型	結果の型	備考
RSQRT	基本整数型	基本整数型	

DRSQRT	DOBULE PRECISION型	DOBULE PRECISION型
QRSQRT	REAL(16)	REAL(16)

11.1.77 SIGN(*A*,*B*) 個別名

- **機能**

  Aの絶対値とBの符号の積

- **分類**

  要素別処理関数

- **引数**

  A: 整数型、または、実数型

  B: *A*と同じ種別パラメタ、および、同じ型

- **結果の型、および、型パラメタ**

  *A*と同じ型

- **結果の値**

  結果の値は、B≧0のときABS(A)とし、B<0のとき-ABS(A)とします。

- **個別名**

個別名	引数の型	結果の型	備考
BSIGN	INTEGER(1)	INTEGER(1)	
IISIGN, HSIGN	INTEGER(2)	INTEGER(2)	
SIGN, ISIGN	基本整数型	基本整数型	規格で定義
JISIGN	INTEGER(4)	INTEGER(4)	
KISIGN	INTEGER(8)	INTEGER(8)	
SIGN	基本実数型	基本実数型	規格で定義
DSIGN	DOUBLE PRECISION型	DOUBLE PRECISION型	規格で定義
QSIGN	REAL(16)	REAL(16)	

11.1.78 SIN(*X*) 個別名

- **機能**

  正弦関数

- **分類**

  要素別処理関数

- **引数**

  X:　実数型、または、複素数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  正弦**sin**(*X*)の値。*X*が実数型のとき、ラジアンでの値とみなされます。ただし、引数が単精度で値の絶対値が2^{21}×**π**以上であるとき、結果の値はNaNです。*X*が複素数型のとき、その実部がラジアンでの値とみなされます。ただし、引数が単精度で実部の値の絶対値が2^{21}×**π**以上であるとき、結果の値はNaNです。

  他の精度の注意事項は、「11.5 注意事項」を参照してください。

- **個別名**

個別名	引数の型	結果の型	備考
SIN	基本実数型	基本実数型	規格で定義
DSIN	DOUBLE PRECISION型	DOUBLE PRECISION型	規格で定義
QSIN	REAL(16)	REAL(16)	
CSIN	基本複素数型	基本複素数型	規格で定義
CDSIN	DOUBLE COMPLEX型	DOUBLE COMPLEX型	
ZSIN	COMPLEX(8)	COMPLEX(8)	
CQSIN	COMPLEX(16)	COMPLEX(16)	

11.1.79 SIND(*X*)

- **機能**

  正弦関数

- **分類**

  要素別処理関数

- **引数**

  X:　実数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  *X*を度の値としたときの**sin**(*X*)の値です。ただし、引数の値の絶対値が2^{50}×180以上であるとき、結果の値はNaNです。

- **個別名**

個別名	引数の型	結果の型	備考
SIND	基本実数型	基本実数型	
DSIND	REAL(8)	REAL(8)	
QSIND	REAL(16)	REAL(16)	

11.1.80 SINH(*X*) 個別名

- **機能**

  双曲線正弦関数

- **分類**

  要素別処理関数

- **引数**

  X:　実数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  双曲線正弦**sinh**(*X*)の値

- **個別名**

個別名	引数の型	結果の型	備考
SINH	基本実数型	基本実数型	規格で定義
DSINH	DOUBLE PRECISION型	DOUBLE PRECISION型	規格で定義
QSINH	REAL(16)	REAL(16)	

11.1.81 SQRT(*X*) 個別名

- **機能**

平方根

- **分類**

  要素別処理関数

- **引数**

  X:　実数型、または、複素数型。*X*が実数型のとき、値は0.0以上でなければなりません。

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  *X*の平方根の値。複素数型の結果は、実部が0.0以上である主値とします。結果の実部が0.0のとき、虚部は0.0以上とします。

- **個別名**

個別名	引数の型	結果の型	備考
SQRT	基本実数型	基本実数型	規格で定義
DSQRT	DOUBLE PRECISION型	DOUBLE PRECISION型	規格で定義
QSQRT	REAL(16)	REAL(16)	
CSQRT	基本複素数型	基本複素数型	規格で定義
CDSQRT	COMPLEX DOUBLE 型	COMPLEX DOUBLE 型	
ZSQRT	COMPLEX(8)	COMPLEX(8)	
CQSQRT	COMPLEX(16)	COMPLEX(16)	

11.1.82 TAN(*X*) 個別名

- **機能**

  正接関数

- **分類**

  要素別処理関数

- **引数**

  X:　実数型、または、複素数型

- **結果の型、および、型パラメタ**

*X*と同じ型

- **結果の値**

  ラジアンで表した正接**tan**(*X*)の値。ただし、引数が単精度で値の絶対値が2^{21}×**n**以上であるとき、結果の値はNaNです。

  他の精度の注意事項は、「11.5 注意事項」を参照してください。

- **個別名**

個別名	引数の型	結果の型	備考
TAN	基本実数型	基本実数型	規格で定義
DTAN	DOUBLE PRECISION型	DOUBLE PRECISION型	規格で定義
QTAN	REAL(16)	REAL(16)	
CTAN	COMPLEX(4)	COMPLEX(4)	
CDTAN, ZTAN	COMPLEX(8)	COMPLEX(8)	
CQTAN	COMPLEX(16)	COMPLEX(16)	

11.1.83 TANH(*X*) 個別名

- **機能**

  双曲線正接関数

- **分類**

  要素別処理関数

- **引数**

  X:　実数型

- **結果の型、および、型パラメタ**

  *X*と同じ型

- **結果の値**

  双曲線正接**tanh**(*X*)の値

- **個別名**

個別名	引数の型	結果の型	備考
TANH	基本実数型	基本実数型	規格で定義
DTANH	DOUBLE	DOUBLE	規格で定義

	PRECISION型	PRECISION型
QTANH	REAL(16)	REAL(16)

11.1.84 TIME(*A*)

- **機能**

  時刻を求めます。

- **分類**

  サブルーチン

- **引数**

  A:　**INTENT(OUT)**属性を持つ長さ8文字の基本文字型のスカラ変数。"*hh*:*mm*:*ss*"の形式の時刻の値が設定されます。

11.1.85 XOR(*I,J*)

IEORの別名の関数。詳細は、「11.1.48 11.1.48IEOR(*I,J*) 個別名」を参照してください。

11.2 行列積ライブラリ

行列積ライブラリにより、行列と行列または行列とベクトルの積を計算に利用できます。

11.2.1 行列とベクトルの積(*A*, *NAR*, *B*, *NBR*, *C*)

- **機能**

  行列とベクトルの積を求めます。

- **分類**

  サブルーチン

- **引数**

  A:　**INTENT(IN)**属性を持つ整数型または実数型の2次元配列

  NAR:　**INTENT(IN)**属性を持つ整数型

  B:　**INTENT(IN)**属性を持つ*A*と同じ種別パラメタの整数型または実数型の配列

  NBR:　**INTENT(IN)**属性を持つ整数型

  C:　**INTENT(INOUT)**属性を持つ*A*と同じ種別パラメタの整数型または実数型の配列。配列*A*,*B*の積の結果との和または差が格納されます。一部手続は、0で初期化してから演算します。

- **詳細**

手続名と引数種別は以下の組み合わせとなります。

手続名 結果(和)	手続名 結果(差)	配列の引数種別 (*A*,*B*,*C*)	引数種別 (*NAR*,*NBR*)	配列Cの 初期化
VAMXV	VASXV	REAL(KIND= 4)	INTEGER(KIND=4)	有
VDMXV	VDSXV	REAL(KIND= 8)	INTEGER(KIND=4)	有
VQMXV	VQSXV	REAL(KIND=16)	INTEGER(KIND=4)	有
VIMXV	VISXV	INTEGER(KIND=4)	INTEGER(KIND=4)	有
VDMXVL	VDSXVL	REAL(KIND= 8)	INTEGER(KIND=8)	有
VQMXVL	VQSXVL	REAL(KIND=16)	INTEGER(KIND=8)	有
VLMXVL	VLSXVL	INTEGER(KIND=8)	INTEGER(KIND=8)	有
VAMXP	VASXP	REAL(KIND= 4)	INTEGER(KIND=4)	無
VDMXP	VDSXP	REAL(KIND= 8)	INTEGER(KIND=4)	無
VQMXP	VQSXP	REAL(KIND=16)	INTEGER(KIND=4)	無
VIMXP	VISXP	INTEGER(KIND=4)	INTEGER(KIND=4)	無
VDMXPL	VDSXPL	REAL(KIND= 8)	INTEGER(KIND=8)	無
VQMXPL	VQSXPL	REAL(KIND=16)	INTEGER(KIND=8)	無
VLMXPL	VLSXPL	INTEGER(KIND=8)	INTEGER(KIND=8)	無

配列 *C* の初期化を含む手続は、以下動作後に結果(和)/結果(差)を実施します。

```
DO I=1,NAR
  C(I)=0
ENDDO
```

結果(和)の手続の動作は以下です。

```
DO J=1,NBR
  DO I=1,NAR
    C(I) = C(I) + B(J) * A(I, J)
  ENDDO
ENDDO
```

結果(差)の手続の動作は以下です。

```
DO J=1,NBR
  DO I=1,NAR
    C(I) = C(I) - B(J) * A(I, J)
  ENDDO
ENDDO
```

11.2.2 行列とベクトルの積(*A*, *NA*, *IAD*, *B*, *NB*, *C*, *NC*, *NAR*, *NBR*)

- **機能**

  行列とベクトルの積を求めます。

- **分類**

  サブルーチン

- **引数**

  A:　　**INTENT(IN)**属性を持つ整数型または実数型の2次元配列

  NA:　　**INTENT(IN)**属性を持つ整数型。1次元目のストライド。

  IAD:　　**INTENT(IN)**属性を持つ整数型。2次元目のストライド。

  B:　　**INTENT(IN)**属性を持つ**A**と同じ種別パラメタの整数型または実数型の配列

  NB:　　**INTENT(IN)**属性を持つ整数型。1次元目のストライド。

  C:　　**INTENT(INOUT)**属性を持つ**A**と同じ種別パラメタの整数型または実数型の配列。配列A,Bの積の結果との和または差が格納されます。一部手続は、0で初期化してから演算します。

  NC:　　**INTENT(IN)**属性を持つ整数型。1次元目のストライド。

  NAR:　　**INTENT(IN)**属性を持つ整数型

  NBR:　　**INTENT(IN)**属性を持つ整数型

- **詳細**

  手続名と引数種別は以下の組み合わせとなります。

手続名 結果(和)	手続名 結果(差)	配列の引数種別 (A,B,C)	引数種別 (NA,NB,NC,NAR,NBR,IAD)	配列Cの 初期化
VAMXVA	VASXVA	REAL(KIND= 4)	INTEGER(KIND=4)	有
VDMXVA	VDSXVA	REAL(KIND= 8)	INTEGER(KIND=4)	有
VQMXVA	VQSXVA	REAL(KIND=16)	INTEGER(KIND=4)	有

手続名 結果(和)	手続名 結果(差)	配列の引数種別 (A,B,C)	引数種別 (NA,NB,NC,NAR,NBR,IAD)	配列Cの 初期化
VIMXVA	VISXVA	INTEGER(KIND=4)	INTEGER(KIND=4)	有
VDMVAL	VDSVAL	REAL(KIND= 8)	INTEGER(KIND=8)	有
VQMVAL	VQSVAL	REAL(KIND=16)	INTEGER(KIND=8)	有
VLMVAL	VLSVAL	INTEGER(KIND=8)	INTEGER(KIND=8)	有
VAMXPA	VASXPA	REAL(KIND= 4)	INTEGER(KIND=4)	無
VDMXPA	VDSXPA	REAL(KIND= 8)	INTEGER(KIND=4)	無
VQMXPA	VQSXPA	REAL(KIND=16)	INTEGER(KIND=4)	無
VIMXPA	VISXPA	INTEGER(KIND=4)	INTEGER(KIND=4)	無
VDMPAL	VDSPAL	REAL(KIND= 8)	INTEGER(KIND=8)	無
VQMPAL	VQSPAL	REAL(KIND=16)	INTEGER(KIND=8)	無
VLMPAL	VLSPAL	INTEGER(KIND=8)	INTEGER(KIND=8)	無

配列 **C** の初期化を含む手続は、以下動作後に結果(和)/結果(差)を実施します。

```
DO I=1,NAR
  C(NC*I)=0
ENDDO
```

結果(和)の手続の動作は以下です。

```
DO J=1,NBR
  DO I=1,NAR
    C(NC*I) = C(NC*I) + B(NB*J) * A(NA*I, J)
  ENDDO
ENDDO
```

結果(差)の手続の動作は以下です。

```
DO J=1,NBR
  DO I=1,NAR
    C(NC*I) = C(NC*I) - B(NB*J) * A(NA*I, J)
  ENDDO
```

```
ENDDO
```

11.2.3 行列と行列の積(*A*, *NA*, *IAD*, *B*, *NB*, *IBD*, *C*, *NC*, *ICD*, *NAR*, *NAC*, *NBC*)

- **機能**

  行列と行列の積を求めます。

- **分類**

  サブルーチン

- **引数**

  A:　　**INTENT(IN)**属性を持つ整数型または実数型の2次元配列

  NA:　　**INTENT(IN)**属性を持つ整数型。1次元目のストライド。

  IAD:　　**INTENT(IN)**属性を持つ整数型。2次元目のストライド。

  B:　　**INTENT(IN)**属性を持つ*A*と同じ種別パラメタの整数型または実数型の配列

  NB:　　**INTENT(IN)**属性を持つ整数型。1次元目のストライド。

  IBD:　　**INTENT(IN)**属性を持つ整数型。2次元目のストライド。

  C:　　**INTENT(INOUT)**属性を持つ*A*と同じ種別パラメタの整数型または実数型の配列。配列A,Bの積の結果との和または差が格納されます。一部手続は、0で初期化してから演算します。

  NC:　　**INTENT(IN)**属性を持つ整数型。1次元目のストライド。

  ICD:　　**INTENT(IN)**属性を持つ整数型。2次元目のストライド。

  NAR:　　**INTENT(IN)**属性を持つ整数型

  NAC:　　**INTENT(IN)**属性を持つ整数型

  NBC:　　**INTENT(IN)**属性を持つ整数型

- **詳細**

  手続名と引数種別は以下の組み合わせとなります。

手続名 結果(和)	手続名 結果(差)	配列の引数種別 (A,B,C)	引数種別(NA, NB, NC,IAD,IBD,ICD, NAR,NAC,NBC)	配列Cの初期化
VAMXMA	VASXMA	REAL(KIND= 4)	INTEGER(KIND=4)	有
VDMXMA	VDSXMA	REAL(KIND= 8)	INTEGER(KIND=4)	有
VQMXMA	VQSXMA	REAL(KIND=16)	INTEGER(KIND=4)	有
VIMXMA	VISXMA	INTEGER(KIND=4)	INTEGER(KIND=4)	有
VDMMAL	VDSMAL	REAL(KIND= 8)	INTEGER(KIND=8)	有

手続名 結果(和)	手続名 結果(差)	配列の引数種別 (A,B,C)	引数種別(NA, NB, NC,IAD,IBD,ICD, NAR,NAC,NBC)	配列Cの初期化
VQMMAL	VQSMAL	REAL(KIND=16)	INTEGER(KIND=8)	有
VLMMAL	VLSMAL	INTEGER(KIND=8)	INTEGER(KIND=8)	有
VAMXQA	VASXQA	REAL(KIND= 4)	INTEGER(KIND=4)	無
VDMXQA	VDSXQA	REAL(KIND= 8)	INTEGER(KIND=4)	無
VQMXQA	VQSXQA	REAL(KIND=16)	INTEGER(KIND=4)	無
VIMXQA	VISXQA	INTEGER(KIND=4)	INTEGER(KIND=4)	無
VDMQAL	VDSQAL	REAL(KIND= 8)	INTEGER(KIND=8)	無
VQMQAL	VQSQAL	REAL(KIND=16)	INTEGER(KIND=8)	無
VLMQAL	VLSQAL	INTEGER(KIND=8)	INTEGER(KIND=8)	無

配列 **C** の初期化を含む手続は、以下動作後に結果(和)/結果(差)を実施します。

```
DO J=1,NBC
  DO I=1,NAR
    C(NC*I,J)=0
  ENDDO
ENDDO
```

結果(和)の手続の動作は以下です。

```
DO K=1,NAC
  DO J=1,NBC
    DO I=1,NAR
      C(NC*I, J) = C(NC*I, J) + B(NB*K, J) * A(NA*I, K)
    ENDDO
  ENDDO
ENDDO
```

結果(差)の手続の動作は以下です。

```
DO K=1,NAC
  DO J=1,NBC
    DO I=1,NAR
```

```
      C(NC*I, J) = C(NC*I, J) - B(NB*K, J) * A(NA*I, K)
    ENDDO
  ENDDO
ENDDO
```

11.3 UNIXシステム関数インタフェース

UNIXシステム関数インタフェースにより、UNIX固有のシステム関数をFortranプログラムから直接利用できます。UNIXシステム関数インタフェースを利用するには、**USE**文、または、**-use**で以降で説明するモジュールを指定します。

例　　**USE** 文の使用例

```
PROGRAM MAIN
USE F90_UNIX
...
END PROGRAM MAIN
```

例　　コンパイラオプションの使用例

```
$ nfort -use F90_UNIX,F90_UNIX_DIR a.f90
```

以降の手続の記述において、KINDが'*'と表記されているときは、その種別の任意の値が使用できます。

各モジュールを**USE**文、または、**-use**で使用したとき、一部変数名が使用できなくなります。使用できなくなる変数名は以下です。

モジュール名	変数名
F90_UNIX	CLOCK_TICK_KIND, TMS
F90_UNIX_DIR	MODE_KIND
F90_UNIX_ENV	CLOCK_TICK_KIND, ID_KIND, LONG_KIND, SC_ARG_MAX, SC_CHILD_MAX, SC_CLK_TCK, SC_JOB_CONTROL, SC_NGROUPS_MAX, SC_OPEN_MAX, SC_SAVED_IDS, SC_STDERR_UNIT, SC_STDIN_UNIT, SC_STDOUT_UNIT, SC_STREAM_MAX, SC_TZNAME_MAX, SC_VERSION, TIME_KIND, TMS, UTSNAME
F90_UNIX_FILE	F_OK, ID_KIND, MODE_KIND, R_OK, STAT_T, S_IRGRP,

	S_IROTH, S_IRUSR, S_IRWXG, S_IRWXO, S_IRWXU, S_ISGID, S_ISUID, S_IWGRP, S_IWOTH, S_IWUSR, S_IXGRP, S_IXOTH, S_IXUSR, UTIMBUF, W_OK, X_OK
F90_UNIX_PROC	ATOMIC_INT, ATOMIC_LOG, PID_KIND, TIME_KIND, WNOHANG, WUNTRACED

各モジュールを**USE**文、または、**-use**で使用したとき、他のUNIXシステム関数インタフェースのモジュール全体、または、必要な手続をUSEします。各モジュールが、USEするモジュールと手続名は以下のとおりです。

モジュール名	USEするモジュール名と手続名
F90_UNIX	F90_UNIX_PROC : ABORT() F90_UNIX_ENV: GETPID(), GETUID(), GETGID(), IARGC(), HIDDEN_GETARG()=>GETARG(), CLOCK_TICK_KIND(), TIMES(), HIDDEN_GETENV()=>GETENV(), CLOCK_TICKS_PER_SECOND()=>CLK_TCK()
F90_UNIX_ENV	F90_UNIX_ERRNO (全ての手続)
F90_UNIX_FILE	F90_UNIX_ENV (全ての手続) F90_UNIX_ERRNO (全ての手続)
F90_UNIX_PROC	F90_UNIX_ERRNO (全ての手続)

※”=>”はモジュール手続を別名としてUSEすることを示します。

11.3.1 F90_UNIX

F90_UNIXモジュールで提供される手続は、以下のとおりです。

SUBROUTINE ABORT(*MESSAGE*)

CHARACTER(*),OPTIONAL,INTENT(IN) :: *MESSAGE*

I/Oバッファをクリーンアップし、実行を異常終了させます。*MESSAGE*は、‘abort:’が先頭に付加された形で論理装置0に出力されます。

SUBROUTINE EXIT(*STATUS*)

INTEGER(*),OPTIONAL,INTENT(IN) :: *STATUS*

メインプログラムの**END**文、または、非修飾の**STOP**文を実行したように実行を終了します。*STATUS*には、実行ステータスコードが設定されます。引数*STATUS*に指定できる種別パラメタは**INTEGER(KIND=4)**、**INTEGER(KIND=8)**のみです。

SUBROUTINE FLUSH(*LUNIT*)

INTEGER(4),INTENT(IN) :: *LUNIT*

論理装置(*LUNIT*)の出力バッファの内容をファイルに出力します。装置かファイルに接続されていないときは、エラーが発生します。

SUBROUTINE FREE(*IPTR*)

INTEGER(8),INTENT(IN) :: *IPTR*

*IPTR*で指定した領域を解放します。引数*IPTR*は、**MALLOC**により割り当てられた領域のアドレスです。

SUBROUTINE GETARG(*K*,*ARG*)

INTEGER(4),INTENT(IN)::*K*

CHARACTER(*),INTENT(OUT)::*ARG*

詳細は、「11.3.3 F90_UNIX_ENV」の**GETARG**を参照してください。本モジュールを引用してGETARGを利用するときは、オプション引数の*LENARG*、*ERRNO*は指定できません。

SUBROUTINE GETENV(*NAME*,*VALUE*)

CHARACTER(*),INTENT(IN)::*NAME*

CHARACTER(*),INTENT(OUT)::*VALUE*

詳細は、「11.3.3 F90_UNIX_ENV」の**GETENV**を参照してください。本モジュールを引用してGETENVを利用するときは、オプション引数の*LENVALUE*、*ERRNO*は指定できません。

PURE INTEGER(4) FUNCTION GETGID()

呼出し元プロセスのグループ番号を返します。

PURE INTEGER(4) FUNCTION GETPID()

呼出し元プロセスのプロセス番号を返します。

PURE INTEGER(4) FUNCTION GETUID()

呼出し元プロセスのユーザ番号を返します。

PURE INTEGER(4) FUNCTION IARGC()

コマンドの引数の数を返します。これは組込み関数**COMMAND_ARGUMENT_COUNT**と同じ値ですが、プログラム名が得られなかったときには、-1を返します。

INTEGER(8) FUNCTION MALLOC(*ISIZE*)

INTEGER(*),INTENT(IN),VALUE :: *ISIZE*

必要な領域の大きさ(*ISIZE*)を確保し、その先頭アドレスを返します(*ISIZE*の単位はバイトです)。引数*ISIZE*に指定できる種別パラメタは**INTEGER**(KIND=4)、**INTEGER**(KIND=8)のみです。

11.3.2 F90_UNIX_DIR

F90_UNIX_DIRモジュールで提供される手続は、以下のとおりです。

SUBROUTINE CHDIR(*PATH*,*ERRNO*)

CHARACTER(*),INTENT(IN) :: *PATH*

INTEGER(4),OPTIONAL,INTENT(OUT) :: *ERRNO*

作業ディレクトリを*PATH*に変更します。*PATH*の末尾の空白は意味を持つことがあるので注意してください。*ERRNO*を指定したとき、正常に終了すると0が、失敗するとエラーコードが返却されます。*ERRNO*を指定しなかったとき、失敗するとエラーメッセージが出力され、プログラムの実行が終了されます。

SUBROUTINE GETCWD(PATH,LENPATH,ERRNO)

CHARACTER(*),OPTIONAL,INTENT(OUT) :: PATH

INTEGER(4),OPTIONAL,INTENT(OUT) :: LENPATH

INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

作業ディレクトリ情報にアクセスします。*PATH*には、作業ディレクトリの名称が設定されます。その際、作業ディレクトリ名称の長さが*PATH*の長さと異なるとき、空白の追加や文字列の切取りが行われます。*LENPATH*には、作業ディレクトリ名称の長さが設定されます。*ERRNO*を指定したとき、正常に終了すると0が、失敗するとエラーコードが返却

されます。*ERRNO*を指定しなかったとき、失敗するとエラーメッセージが出力され、プログラムの実行が終了されます。

SUBROUTINE LINK(EXIS*TING*,*NEW*,*ERRNO*)
CHARACTER(*),INTENT(IN) :: *EXISTING*,*NEW*
INTEGER(4),OPTIONAL,INTENT(OUT) :: *ERRNO*

既存ファイル(*EXISTING*)に対して、新たなリンク(*NEW*で指定)を作成します。*ERRNO*を指定したとき、正常に終了すると0が、失敗するとエラーコードが返却されます。*ERRNO*を指定しなかったとき、失敗するとエラーメッセージが出力され、プログラムの実行が終了されます。

SUBROUTINE RENAME(*OLD*,*NEW*,*ERRNO*)
CHARACTER(*),INTENT(IN) :: *OLD*
CHARACTER(*),INTENT(IN) :: *NEW*
INTEGER(4),OPTIONAL,INTENT(OUT) :: *ERRNO*

ファイル名称を*OLD*から*NEW*へ変更します。ファイルNEWが存在するとき、*NEW*を削除した後に*OLD*を*NEW*へ変更します。*OLD*、または、*NEW*中の末尾の空白は意味を持つことがあるので注意してください。*ERRNO*を指定したとき、正常に終了すると0が、失敗するとエラーコードが返却されます。*ERRNO*を指定しなかったとき、失敗するとエラーメッセージが出力され、プログラムの実行が終了されます。

SUBROUTINE UNLINK(*PATH*,*ERRNO*)
CHARACTER(*),INTENT(IN) :: *PATH*
INTEGER(4),OPTIONAL,INTENT(OUT) :: *ERRNO*

*PATH*に指定したファイルを削除します。*PATH*中の末尾の空白は意味を持つことがあるので注意してください。*ERRNO*を指定したとき、正常に終了すると0が、失敗するとエラーコードが返却されます。*ERRNO*を指定しなかったとき、失敗するとエラーメッセージが出力され、プログラムの実行が終了されます。

11.3.3 F90_UNIX_ENV

F90_UNIX_ENVモジュールで提供される手続は、以下のとおりです。

SUBROUTINE GETARG(*K*,*ARG*,*LENARG*,*ERRNO*)

INTEGER(*),INTENT(IN) :: *K*
CHARACTER(*),OPTIONAL,INTENT(OUT) :: *ARG*
INTEGER(4),OPTIONAL,INTENT(OUT) :: *LENARG*
INTEGER(4),OPTIONAL,INTENT(OUT) :: *ERRNO*

コマンドの引数番号(*K*)にアクセスします。引数0は、プログラム名です。*ARG*には、引数の文字列が設定されます。*ARG*の長さが引数に指定した文字列の長さと異なるとき、空白の追加や文字列の切取りが行われます。*LENARG*には、引数の文字列の長さが設定されます。*ERRNO*を指定したとき、正常に終了すると0が、失敗するとエラーコードが返却されます。*ERRNO*を指定しなかったとき、失敗するとエラーメッセージが出力され、プログラムの実行が終了されます。

SUBROUTINE GETENV(*NAME*,*VALUE*,*LENVALUE*,*ERRNO*)
CHARACTER(*),INTENT(IN) :: *NAME*
CHARACTER(*),OPTIONAL,INTENT(OUT) :: *VALUE*
INTEGER(4),OPTIONAL,INTENT(OUT) :: *LENVALUE*
INTEGER(4),OPTIONAL,INTENT(OUT) :: *ERRNO*

*NAME*で指定された環境変数にアクセスします。*VALUE*には、変数のテキスト値が設定されます。*VALUE*の長さが引数に指定した文字列の長さと異なるとき、空白の追加や文字列の切取りが行われます。*LENVALUE*には、変数のテキスト値の長さが設定されます。*ERRNO*を指定したとき、正常に終了すると0が、失敗するとエラーコードが返却されます。*ERRNO*を指定しなかったとき、失敗するとエラーメッセージが出力され、プログラムの実行が終了されます。

PURE SUBROUTINE GETHOSTNAME(*NAME*,*LENNAME*)
CHARACTER(*),OPTIONAL,INTENT(OUT) :: *NAME*
INTEGER(4),OPTIONAL,INTENT(OUT) :: *LENNAME*

*NAME*には、現在のホスト名を文字列で設定されます。*NAME*の長さが引数に指定した文字列の長さと異なるとき、空白の追加や切取りが行われます。*LENNAME*には、ホスト名の長さが設定されます。ホスト名が得られなかったときには、*LENNAME*は0になります。

PURE SUBROUTINE GETLOGIN(*S*,*LENS*)
CHARACTER(*),OPTIONAL,INTENT(OUT) :: *S*

INTEGER(4),OPTIONAL,INTENT(OUT) :: *LENS*

呼出し元プロセスのユーザ名（ログイン名）にアクセスします。*S*を指定したとき、名称が文字列で設定されます。ログイン名の長さが*S*の長さと異なるとき、空白の追加や文字列の切取りが行われます。*LENS*を指定したとき、ログイン名の長さが設定されます。

SUBROUTINE ISATTY(*LUNIT*,*ANSWER*,*ERRNO*)

INTEGER(*),INTENT(IN) :: *LUNIT*

LOGICAL(*),INTENT(OUT) :: *ANSWER*

INTEGER(4),OPTIONAL,INTENT(OUT) :: *ERRNO*

論理装置*LUNIT*が端末と接続されているとき、*ANSWER*に.TRUE.が設定されます。*LUNIT*が正しい装置番号ではない、または、どのファイルにも接続されていないとき、エラーが返却されます。

*ERRNO*を指定したとき、正常に終了すると0が、失敗するとエラーコードが返却されます。*ERRNO*を指定しなかったとき、失敗するとエラーメッセージが出力され、プログラムの実行が終了されます。

SUBROUTINE TIME(*ITIME*,*ERRNO*)

INTEGER(4),INTENT(OUT) :: *ITIME*

INTEGER(4),OPTIONAL,INTENT(OUT) :: *ERRNO*

*ITIME*にEpochを起点とする日付/時刻（秒単位）が設定されます。

*ERRNO*を指定したとき、正常に終了すると0が、失敗するとエラーコードが返却されます。*ERRNO*を指定しなかったとき、失敗するとエラーメッセージが出力され、プログラムの実行が終了されます。

SUBROUTINE TTYNAME(*LUNIT*,*S*,*LENS*,*ERRNO*)

INTEGER(*),INTENT(IN) :: *LUNIT*

CHARACTER(*),OPTIONAL,INTENT(OUT) :: *S*

INTEGER(4),OPTIONAL,INTENT(OUT) :: *LENS*

INTEGER(4),OPTIONAL,INTENT(OUT) :: *ERRNO*

論理装置*LUNIT*に接続された端末の名称にアクセスします。*S*を指定したとき、端末名の文字列が設定されます。端末名の長さがSの長さと異なるとき、空白の追加や文字列の切取りが行われます。*LENS*を指定したとき、端末名の長さが設定されます。*LUNIT*が正しい論理装置番号でない、または接続されていないときエラーが返却されます。

*ERRNO*を指定したとき、正常に終了すると0が、失敗するとエラーコードが返却されます。*ERRNO*を指定しなかったとき、失敗するとエラーメッセージが出力され、プログラムの実行が終了されます。

11.3.4 F90_UNIX_ERRNO

F90_UNIX_ERRNOモジュールで提供されるパラメタは、以下のとおりです。

INTEGER(4),OPTIONAL,INTENT(OUT) :: *ERRNO*

UNIXシステム関数インタフェースで提供する多くの手続が規定される省略可能な*ERRNO*引数を用意しています。この引数が与えられたとき、手続からエラーステータスを受け取ります。0は正常終了、0以外はエラーコードを意味します。*ERRNO*引数を省略、かつエラー条件が発生したとき、エラーメッセージとともにプログラムの実行が終了します。

手続が*ERRNO*引数を用意していないとき、手続は常に正常終了することを示します。

11.3.5 F90_UNIX_FILE

F90_UNIX_FILEモジュールで提供されるパラメタは、以下のとおりです。

INTEGER(4),PARAMETER :: F_OK

ファイルが存在するかどうかの確認に使用するフラグです。

INTEGER(4),PARAMETER :: R_OK

ファイルが読み込み可能かどうかの確認に使用するフラグです。

INTEGER(4),PARAMETER :: S_IRGRP

グループに対して、読み取り許可モードを示すビットです。

INTEGER(4),PARAMETER :: S_IROTH

その他に対して、読み取り許可モードを示すビットです。

INTEGER(4),PARAMETER :: S_IRUSR

所有者に対して、読み取り許可モードを示すビットです。

INTEGER(4),PARAMETER :: S_IRWXG

ファイルのアクセス権からグループアクセス権限を選択するためのマスクです。

INTEGER(4),PARAMETER :: S_IRWXO

ファイルのアクセス権からその他のアクセス権限を選択するためのマスクです。

INTEGER(4),PARAMETER :: S_IRWXU

ファイルのアクセス権から所有者のアクセス権限を選択するためのマスクです。

INTEGER(4),PARAMETER :: S_ISGID

ファイルがSET-GROUP-ID(SGID)モードを示すビットです。

INTEGER(4),PARAMETER :: S_ISUID

ファイルがSET-USER-ID(SUID)モードを示すビットです。

INTEGER(4),PARAMETER :: S_IWGRP

グループに対して、書き込み許可モードを示すビットです。

INTEGER(4),PARAMETER :: S_IWOTH

その他に対して、書き込み許可モードを示すビットです。

INTEGER(4),PARAMETER :: S_IWUSR

所有者に対して、書き込み許可モードを示すビットです。

INTEGER(4),PARAMETER :: S_IXGRP

グループに対して、実行許可モードを示すビットです。

INTEGER(4),PARAMETER :: S_IXOTH

その他に対して、実行許可モードを示すビットです。

INTEGER(4),PARAMETER :: S_IXUSR

所有者に対して、実行許可モードを示すビットです。

INTEGER(4),PARAMETER :: W_OK

ファイルが書き込み可能かどうかの確認に使用するフラグです。

INTEGER(4),PARAMETER :: X_OK

ファイルが実行可能かどうかの確認に使用するフラグです。

F90_UNIX_FILEモジュールで提供される型は、以下のとおりです。

STAT_T

```
TYPE STAT_T
    INTEGER(4) ST_MODE
    INTEGER(4) ST_INO
    INTEGER(4) ST_DEV
    INTEGER(4) ST_NLINK
    INTEGER(4) ST_UID
```

```
    INTEGER(4) ST_GID
    INTEGER(4) ST_SIZE
    INTEGER(4) ST_ATIME, ST_MTIME, ST_CTIME
END TYPE
```

ファイルの特性を保持している構造体です。

ST_MODE

ファイルのモード(ユーザ/グループ/その他に対する読み取り/書き込み/実行許可、および SET-GROUP-ID, SET-USER-IDビット)が設定されます。

ST_INO

ファイルのシリアル番号が設定されます。

ST_DEV

ファイルが存在している装置に対するIDが設定されます。

ST_NLINK

ファイルに対するリンクの数が設定されます。

ST_UID

ファイル所有者のユーザ番号が設定されます。

ST_GID

ファイルのグループ番号が設定されます。

ST_SIZE

バイト単位のファイルサイズが設定されます(通常ファイルのみ)。

ST_ATIME

最後にアクセスした時刻が設定されます。

ST_MTIME

最後に変更を加えた時刻が設定されます。

ST_CTIME

最後にファイル状態が変更された時刻が設定されます。

F90_UNIX_FILEモジュールで提供される手続は、以下のとおりです。

PURE SUBROUTINE ACCESS(*PATH*,*AMODE*,*ERRNO*)

CHARACTER(*),INTENT(IN) :: *PATH*

INTEGER(*),INTENT(IN) :: *AMODE*

INTEGER(4),INTENT(OUT) :: *ERRNO*

*AMODE*の値に従ってファイルのアクセス権限ビットをチェックします。*AMODE*の値は、F_OK、または、R_OK、W_OK、X_OKの組合せの必要があります。R_OK、W_OK、X_OKの値は加算、または、組込み関数IORによって結合します。

アクセス権限ビットのチェック結果は、*ERRNO*に設定されます。アクセス可では0、アクセス不可では、その理由を示すエラーコードが設定されます。

SUBROUTINE CHMOD(*PATH*,*MODE*,*ERRNO*)

CHARACTER(*),INTENT(IN) :: *PATH*

INTEGER(*),INTENT(IN) :: *MODE*

INTEGER(4),OPTIONAL,INTENT(OUT) :: *ERRNO*

ファイルモード(ST_MODE)を*MODE*に変更します。*ERRNO*を指定したとき、正常に終了すると0が、失敗するとエラーコードが返却されます。*ERRNO*を指定しなかったとき、失敗するとエラーメッセージが出力され、プログラムの実行が終了されます。

SUBROUTINE FSTAT(*LUNIT*,*BUF*,*ERRNO*)

INTEGER(*),INTENT(IN) :: *LUNIT*

TYPE(STAT_T),INTENT(OUT) :: *BUF*

INTEGER(4),OPTIONAL,INTENT(OUT) :: *ERRNO*

論理装置*LUNIT*に接続されたファイルの情報が*BUF*に設定されます。*LUNIT*が正しい論理装置番号ではない、またはファイルに接続されていないとき、エラーが返却されます。

*ERRNO*を指定したとき、正常に終了すると0が、失敗するとエラーコードが返却されます。*ERRNO*を指定しなかったとき、失敗するとエラーメッセージが出力され、プログラムの実行が終了されます。

SUBROUTINE LSTAT(*PATH*,*BUF*,*ERRNO*)

CHARACTER(*),INTENT(IN) :: *PATH*

TYPE(STAT_T),INTENT(OUT) :: *BUF*

INTEGER(4),OPTIONAL,INTENT(OUT) :: *ERRNO*

ファイルPATHの情報が*BUF*に設定されます。*PATH*がリンクファイルのとき、リンクファイルの情報が設定されます。*ERRNO*を指定したとき、正常に終了すると0が、失敗するとエラーコードが返却されます。*ERRNO*を指定しなかったとき、失敗するとエラーメッセージが出力され、プログラムの実行が終了されます。

SUBROUTINE STAT(*PATH*,*BUF*,*ERRNO*)

CHARACTER(*),INTENT(IN) :: PATH

TYPE(STAT_T),INTENT(OUT) :: BUF

INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

ファイル*PATH*の情報が*BUF*に設定されます。*PATH*がリンクファイルのとき、リンク先のファイルの情報が設定されます。*ERRNO*を指定したとき、正常に終了すると0が、失敗するとエラーコードが返却されます。*ERRNO*を指定しなかったとき、失敗するとエラーメッセージが出力され、プログラムの実行が終了されます。

11.3.6 F90_UNIX_PROC

F90_UNIX_PROCモジュールで提供される手続は、以下のとおりです。

SUBROUTINE ALARM(*SECONDS*,*SUBROUTINE*,*SECLEFT,ERRNO*)

INTEGER(*),INTENT(IN) :: *SECONDS*

INTERFACE

** SUBROUTINE SUBROUTINE()**

** END**

END INTERFACE

OPTIONAL *SUBROUTINE*

INTEGER(4),OPTIONAL,INTENT(OUT) :: *SECLEFT*

INTEGER(4),OPTIONAL,INTENT(OUT) :: *ERRNO*

SECONDS指定の秒数後に手続*SUBROUTINE*に対し“alarm”コールがかかるように設定します。*SECONDS*が0のときには、既存のアラームをキャンセルします。*SUBROUTINE*が存在しなかったとき、それ以前に設定されたサブルーチンとアラームシグナルの対応付けは変更しません。

*SECLEFT*には、先行するアラーム上で残存する秒数が設定されます。既存のアラームが存在しないときには0が設定されます。*ERRNO*を指定したとき、正常に終了すると0が、失敗するとエラーコードが返却されます。*ERRNO*を指定しなかったとき、失敗するとエラーメッセージが出力され、プログラムの実行が終了されます。

SUBROUTINE EXECL(*PATH*,*ARG0...*,*ERRNO*)

CHARACTER(*),INTENT(IN) :: *PATH*

CHARACTER(*),INTENT(IN) :: *ARG0...*

INTEGER(4),OPTIONAL,INTENT(OUT) :: *ERRNO*

現行プロセスに代わり指定したプログラム(*PATH*)を実行します。

新たなプログラムへの引数は、*ARG0*, *ARG1*など、最大で*ARG20*までの名称で指定されます。これらは省略可能な引数ではない点に注意してください。それ自身が省略可能な仮引数の実引数もすべて指定されなくてはなりません。

引数が、配列ではなく個々に提供されるという点以外は、EXECVと同様です。また、引数は個々に提供されるため、長さの指定は必要ありません(長さはそれぞれの引数から求められます)。

*ERRNO*を指定したとき、正常に終了すると0が、失敗するとエラーコードが返却されます。*ERRNO*を指定しなかったとき、失敗するとエラーメッセージが出力され、プログラムの実行が終了されます。

SUBROUTINE EXECLP(*FILE*,*ARG0*…,*ERRNO*)

CHARACTER(*),INTENT(IN) :: *FILE*

CHARACTER(*),INTENT(IN) :: *ARG0*…

INTEGER(4),OPTIONAL,INTENT(OUT) :: *ERRNO*

現行プロセスに代わり、指定したプログラム(FILE)を実行します。

新たなプログラムへの引数は、*ARG0*, *ARG1*など、最大でARG20までの名称で指定されます。これらは省略可能な引数ではない点に注意してください。それ自身が省略可能な仮引数の実引数もすべて指定されなくてはなりません。

実行するプログラムの決定がEXECVPと同じルールに従うという点以外は、**EXECL**と同様です。

*ERRNO*を指定したとき、正常に終了すると0が、失敗するとエラーコードが返却されます。*ERRNO*を指定しなかったとき、失敗するとエラーメッセージが出力され、プログラムの実行が終了されます。

SUBROUTINE EXECV(*PATH*,*ARGV*,*LENARGV*,*ERRNO*)

CHARACTER(*),INTENT(IN) :: *PATH*

CHARACTER(*),INTENT(IN) :: *ARGV(:)*

INTEGER(*),INTENT(IN) :: *LENARGV(:)*

INTEGER(4),OPTIONAL,INTENT(OUT) :: *ERRNO*

現行プロセスに代わり、指定したプログラム(*PATH*)を実行します。

*ARGV*は引数文字列の配列であり、*LENARGV*には各引数の長さが設定されます。*ARGV*のサイズが0でないとき、*ARGV*(1)(:*LENARGV*(1))は引数0(プログラム名)として引き渡されます。

*ERRNO*を指定したとき、正常に終了すると0が、失敗するとエラーコードが返却されます。*ERRNO*を指定しなかったとき、失敗するとエラーメッセージが出力され、プログラムの実行が終了されます。

SUBROUTINE EXECVE(*PATH*,*ARGV*,*LENARGV*,*ENV*,*LENENV*,*ERRNO*)
CHARACTER(*),INTENT(IN) :: *PATH*
CHARACTER(*),INTENT(IN) :: *ARGV(:)*
INTEGER(*),INTENT(IN) :: *LENARGV(:)*
CHARACTER(*),INTENT(IN) :: *ENV(:)*
INTEGER(*),INTENT(IN) :: *LENENV(:)*
INTEGER(4),OPTIONAL,INTENT(OUT) :: ERRNO

動作は環境変数を新たなプログラムに引き渡される点以外は、**EXECV**と同様です。

*ENV*に環境変数の文字列の配列と*LENENV*には各環境変数の長さが設定されます。

*ERRNO*を指定したとき、正常に終了すると0が、失敗するとエラーコードが返却されます。*ERRNO*を指定しなかったとき、失敗するとエラーメッセージが出力され、プログラムの実行が終了されます。

SUBROUTINE EXECVP(*FILE*,*ARGV*,*LENARGV*,*ERRNO*)
CHARACTER(*),INTENT(IN) :: *FILE*
CHARACTER(*),INTENT(IN) :: *ARGV(:)*
INTEGER(*),INTENT(IN) :: *LENARGV(:)*
INTEGER(4),OPTIONAL,INTENT(OUT) :: *ERRNO*

実行対象のプログラム(*FILE*)が環境変数(*PATH*)を使ってサーチされる点以外は、**EXECV**と同様です(*FILE*にスラッシュ文字が含まれている場合には、**EXECVP**は**EXECV**と等価になります)。

*ERRNO*を指定したとき、正常に終了すると0が、失敗するとエラーコードが返却されます。*ERRNO*を指定しなかったとき、失敗するとエラーメッセージが出力され、プログラムの実行が終了されます。

SUBROUTINE FORK(*PID*,*ERRNO*)
INTEGER(4),INTENT(OUT) :: *PID*
INTEGER(4),OPTIONAL,INTENT(OUT) :: *ERRNO*

呼出し元プロセスと全く同一のコピーを新たなプロセスとして作成します。

新たなプロセスでは、*PID*に0が設定されます。呼び出し元プロセスでは、*PID*に新たに生成された(子)プロセスのプロセスIDが設定されます。

*ERRNO*を指定したとき、正常に終了すると0が、失敗するとエラーコードが返却されます。*ERRNO*を指定しなかったとき、失敗するとエラーメッセージが出力され、プログラムの実行が終了されます。

PURE SUBROUTINE SLEEP(*SECONDS*,*SECLEFT*)
INTEGER(*),INTENT(IN) :: *SECONDS*
INTEGER(4),OPTIONAL,INTENT(OUT) :: *SECLEFT*

プロセス実行を*SECONDS*指定の秒数の間、または、シグナルが届くまで中断します。*SECLEFT*には、残りのスリープ時間を秒数として設定されます。シグナルによって割り込まれていないときには0が設定されます。

SUBROUTINE SYSTEM(*STRING*,*STATUS*,*ERRNO*)
CHARACTER(*),INTENT(IN) :: *STRING*
INTEGER(4),OPTIONAL,INTENT(OUT) :: *STATUS*,*ERRNO*

コマンドプロセッサに*STRING*の文字列を渡します。*STATUS*には完了ステータスが設定されます。

*ERRNO*を指定したとき、正常に終了すると0が、失敗するとエラーコードが返却されます。*ERRNO*を指定しなかったとき、失敗するとエラーメッセージが出力され、プログラムの実行が終了されます。

SUBROUTINE WAIT(*STATUS*,*RETPID*,*ERRNO*)
INTEGER(4),OPTIONAL,INTENT(OUT) :: *STATUS*
INTEGER(4),OPTIONAL,INTENT(OUT) :: *RETPID*
INTEGER(4),OPTIONAL,INTENT(OUT) :: *ERRNO*

いずれかの子プロセスの終了を待ちます(既に終了しているプロセスがあれば即座にリタ

ーンします)。*STATUS*には、子プロセスの終了ステータスが設定されます。*RETPID*には、子プロセスのプロセス番号が設定されます。

*ERRNO*を指定したとき、正常に終了すると0が、失敗するとエラーコードが返却されます。*ERRNO*を指定しなかったとき、失敗するとエラーメッセージが出力され、プログラムの実行が終了されます。

11.4 その他のライブラリ

組込み関数/組込みサブルーチンの他に、以下に述べる関数/サブルーチンを使用できます。これらのルーチンは、Cライブラリで使用できるシステムの機能をFORTRANからも呼び出せるようにしたものです。

FORTRANライブラリは、組込み関数として扱われません。そのため、コンパイラはこれらの関数の型をIMPLICIT文の指定または暗黙の型宣言(先頭の英字I、J、K、L、MまたはNは整数型を表し、その他の英字は実数型を表す)にしたがって処理します。したがって、IMPLICIT文の指定または暗黙の型と関数の型が一致しない関数(例：CTIME)は型宣言が必要です。

11.4.1 ABORT()

- **機能**

  プログラムを異常終了します。

- **分類**

  サブルーチン

11.4.2 ACCESS(*PATH*,*MODE*)

- **機能**

  ファイルへのアクセス権をチェックします。

- **分類**

  関数

- **引数**

  PATH:　**INTENT(IN)**属性を持つ基本文字型のスカラ変数。ファイルパスを指定します。

  MODE:　**INTENT(IN)**属性を持つ基本文字型のスカラ変数。アクセス権を指定します。

- **結果の型、および、型パラメタ**

  整数型

- **結果の値**

アクセス権があれば0が、なければエラーコードが返却されます。

11.4.3 ALARM(*SECS*,*PROC*)

- **機能**

  プロセスのアラームクロックを設定します。

- **分類**

  関数

- **引数**

  SECS:　**INTENT(IN)**属性を持つ4バイト整数型。基準とする秒数を指定します。

  PROC:　アラームを設定する手続を指定します。

- **結果の型、および、型パラメタ**

  整数型

- **結果の値**

  呼出し時の残りの秒数の値。

11.4.4 CHDIR(*PATH*)

- **機能**

  作業ディレクトリを変更します。

- **分類**

  関数

- **引数**

  PATH:　INTENT(IN)属性を持つ基本文字型のスカラ変数。ディレクトリパスを指定します。

- **結果の型、および、型パラメタ**

  整数型

- **結果の値**

  正常に終了すると0が、失敗するとエラーコードが返却されます。

11.4.5 CHMOD(*NAME*,*MODE*)

- **機能**

  ファイルのアクセスモードを変更します。

- **分類**

  関数

- **引数**

  NAME:　**INTENT(IN)**属性を持つ基本文字型のスカラ変数。ファイルパスを指定します。

  MODE:　**INTENT(IN)**属性を持つ基本文字型のスカラ変数。アクセスモードを指定します。

- **結果の型、および、型パラメタ**

  整数型

- **結果の値**

  正常に終了すると0が、失敗するとエラーコードが返却されます。

11.4.6 CTIME(*I*)

- **機能**

  日付と時刻を文字列へ変換します。

- **分類**

  関数

- **引数**

  I:　**INTENT(IN)**属性を持つ4バイト整数型

- **結果の型、および、型パラメタ**

  長さ24文字の基本文字型

- **結果の値**

  *I*で指定された値をEpochからの経過時間に変換し、さらに地方時間に変換して以下の形式で返却されます。

```
Sun Jan 19 01:03:52 1992
```

11.4.7 DTIME(*TARRAY*)

- **機能**

  経過実行時間を求めます。

- **分類**

  関数

- **引数**

  TARRAY:　**INTENT(OUT)**属性を持つ配列要素2の4バイト実数型配列。直前のDTIME引用からのUSER時間を第1要素に、SYS時間を第2要素に設定されます。

- **結果の型、および、型パラメタ**

  4バイト実数型

- **結果の値**

  直前のDTIME引用からのUSER時間とSYS時間の合計値

11.4.8 ETIME(*TARRAY*)

- **機能**

  経過実行時間を求めます。

- **分類**

  関数

- **引数**

  TARRAY:　　**INTENT(OUT)**属性を持つ配列要素2の4バイト実数型配列。プログラム開始時からのUSER時間を第1要素に、SYS時間を第2要素に設定されます。

- **結果の型、および、型パラメタ**

  4バイト実数型

- **結果の値**

  プログラム開始時からのUSER時間とSYS時間の合計値（単位:秒）

- **備考**

  サブルーチンとして使用するときの詳細は、「11.1.33 ETIME(*D*)」を参照してください。

11.4.9 FDATE()

- **機能**

  現在の時刻の文字列を取得します。

- **分類**

  関数

- **結果の型、および、型パラメタ**

  長さ24文字の基本文字列型

- **結果の値**

  現在の時刻が以下の形式で返却されます。

```
Sun Jan 19 01:03:52 1992
```

- **備考**

  サブルーチンとしても、以下の形式で使用可能です。

```
call FDATE(A)
```

  引数Aは、**INTENT(OUT)**属性を持つ長さ24文字の基本文字列型です。

  現在の時刻が関数の結果と同様の形式で引数Aに設定されます。

```
Sun Jan 19 01:03:52 1992
```

11.4.10 FORK()

- **機能**

  新しいプロセスを作成します。

- **分類**

  関数

- **結果の型、および、型パラメタ**

  整数型

- **結果の値**

  正常に終了するとプロセスIDの値が、失敗するとエラーコードが返却されます。

11.4.11 FREE(*ADDR*)

- **機能**

  指定された領域を解放します。

- **分類**

  サブルーチン

- **引数**

  ADDR：**INTENT(IN)**属性を持つ8バイト整数型。**MALLOC**により割り当てられた領域のアドレスを指定します。

11.4.12 FREE2(*ADDR*)

- **機能**

  指定された領域を解放します。

- **分類**

  サブルーチン

- **引数**

  ADDR：**INTENT(IN)**属性を持つ8バイト整数型。**MALLOC2**により割り当てられた領域のアドレスを指定します。

11.4.13 FSEEK(*UNIT*,*OFFSET*,*WHENCE*)

- **機能**

  外部ファイル装置に接続されているファイルの位置を変更します。

- **分類**

  関数

- **引数**

  UNIT:　**INTENT(IN)**属性を持つ4バイト整数型。外部ファイル装置を指定します。

  OFFSET:　**INTENT(IN)**属性を持つ4バイト整数型。*WHENCE*を起点としたバイト単位のオフセットを指定します。

  WHENCE:　**INTENT(IN)**属性を持つ4バイト整数型。起点となるファイル位置として以下のいずれか一つを指定します。

Value	Position
0	ファイルの先頭
1	ファイルの現在位置
2	ファイルの末尾

- **結果の型、および、型パラメタ**

  4バイト整数型

- **結果の値**

  正常に終了すると0が、失敗するとエラーコードが返却されます。

- **備考**

  サブルーチンとしても、以下の形式で使用可能です。

```
call FSEEK(UNIT,OFFSET,WHENCE)
```

11.4.14 FSTAT(*UNIT*,*SXBUF*)

- **機能**

  外部ファイル装置に接続されているファイルの情報を取得します。

- **分類**

関数

- **引数**

  UNIT:　**INTENT(IN)**属性を持つ4バイト整数型。外部ファイル装置を指定します。

  SXBUF: **INTENT(OUT)**属性を持つ配列要素19の4バイト整数型配列。ファイル情報が設定されます。

- **結果の型、および、型パラメタ**

  整数型

- **結果の値**

  正常に終了すると0が、失敗するとエラーコードが返却されます。

- **備考**

  *SXBUF*によって取得できる情報を以下に示します。

  SXBUF(1)　ファイルがあるデバイスID

  SXBUF(2)　inode 番号

  SXBUF(3)　アクセス保護

  SXBUF(4)　ハードリンクの数

  SXBUF(5)　所有者のユーザ ID

  SXBUF(6)　所有者のグループ ID

  SXBUF(7)　0

  SXBUF(8)　全体のサイズ(bytes)

  SXBUF(9)　最終アクセス時刻

  SXBUF(10)　最終修正時刻

  SXBUF(11)　最終状態変更時刻

  SXBUF(12)～SXBUF(19)　予備

11.4.15 FTELL(*UNIT*)

- **機能**

  外部ファイル装置に接続されているファイルの現在位置を返却します。

- **分類**

  関数

- **引数**

  UNIT:　**INTENT(IN)**属性を持つ4バイト整数型。外部ファイル装置を指定します。

- **結果の型、および、型パラメタ**

4バイト整数型

- **結果の値**

  ファイルの先頭からのオフセット値をバイト単位で返却されます。負の値はエラーを示します。

11.4.16 FTELLI8(*UNIT*)

- **機能**

  外部ファイル装置に接続されているファイルの現在位置を返却します。

- **分類**

  関数

- **引数**

  UNIT:　**INTENT(IN)**属性を持つ4バイト整数型。外部ファイル装置を指定します。

- **結果の型、および、型パラメタ**

  8バイト整数型

- **結果の値**

  ファイルの先頭からのオフセット値をバイト単位で返却されます。負の値はエラーを示します。

11.4.17 GETARG(*POS*,*VAL*)

- **機能**

  プログラムに渡された引数を取得します。

- **分類**

  サブルーチン

- **引数**

  POS:　**INTENT(IN)**属性を持つ4バイト整数型。取得する引数の番号を指定します。

  VAL:　**INTENT(OUT)**属性を持つ基本文字型のスカラ変数。取得した引数が設定されます。

11.4.18 GETCWD(*PATH*)

- **機能**

  現在の作業ディレクトリのパス名を取得します。

- **分類**

  関数

- **引数**

  PATH:　**INTENT(OUT)**属性を持つ基本文字型のスカラ変数。現在の作業ディレクトリのパスが設定されます。

- **結果の型、および、型パラメタ**

  整数型

- **結果の値**

  正常に終了すると0が、失敗するとエラーコードが返却されます。

11.4.19 GETENV(*NAME*,*VAL*)

- **機能**

  環境変数を取得します。

- **分類**

  サブルーチン

- **引数**

  NAME:　**INTENT(IN)**属性を持つ基本文字型のスカラ変数。取得する環境変数の文字列を指定します。

  VAL:　　**INTENT(OUT)**属性を持つ基本文字型のスカラ変数。取得した環境変数の値が設定されます。

- **備考**

  関数としても、以下の形式で使用可能です。

  戻り値は、整数型で、一致する文字列があると1が、ないと0が返却されます。

```
INTEGER RESULT,GETENV
RESULT = GETENV(NAME,VAL)
```

11.4.20 GETGID()

- **機能**

  グループIDを取得します。

- **分類**

  関数

- **結果の型、および、型パラメタ**

  整数型

- **結果の値**

グループIDの値。

11.4.21 GETLOG(*NAME*)

- **機能**

  ログイン名を取得します。

- **分類**

  サブルーチン

- **引数**

  NAME:　**INTENT(OUT)**属性を持つ基本文字型のスカラ変数。ログイン名が設定されます。

11.4.22 GETPID()

- **機能**

  プロセスIDを取得します。

- **分類**

  関数

- **結果の型、および、型パラメタ**

  整数型

- **結果の値**

  プロセスIDの値。

11.4.23 GETPOS(*UNIT*)

- **機能**

  外部ファイル装置に接続されているファイルの現在位置を返却します。

- **分類**

  関数

- **引数**

  UNIT:　**INTENT(IN)**属性を持つ4バイト整数型。外部ファイル装置を指定します。

- **結果の型、および、型パラメタ**

  4バイト整数型

- **結果の値**

  ファイルの先頭からのオフセット値をバイト単位で返却されます。

11.4.24 GETPOSI8(*UNIT*)

- **機能**

  外部ファイル装置に接続されているファイルの現在位置を返却します。

- **分類**

  関数

- **引数**

  UNIT:　**INTENT(IN)**属性を持つ4バイト整数型。外部ファイル装置を指定します。

- **結果の型、および、型パラメタ**

  8バイト整数型

- **結果の値**

  ファイルの先頭からのオフセット値をバイト単位で返却されます。

11.4.25 GETUID()

- **機能**

  ユーザIDを取得します。

- **分類**

  関数

- **結果の型、および、型パラメタ**

  整数型

- **結果の値**

  ユーザIDの値。

11.4.26 GMTIME(*I,IA9*)

- **機能**

  日付と時刻を4バイト整数型配列へ変換します。

- **分類**

  サブルーチン

- **引数**

  I:　**INTENT(IN)**属性を持つ4バイト整数型

  IA9:　**INTENT(OUT)**属性を持つ配列要素9の4バイト整数型配列。*I*で指定された値をEpochからの経過時間に変換し、配列の各要素に設定されます。

11.4.27 HOSTNM(NAME)

- **機能**

  ホスト名を取得します。

- **分類**

  関数

- **引数**

  NAME: **INTENT(OUT)**属性を持つ基本文字型のスカラ変数。現在のホスト名が設定されます。

- **結果の型、および、型パラメタ**

  整数型

- **結果の値**

  正常に終了すると0が、失敗するとエラーコードが返却されます。

11.4.28 IARGC()

- **機能**

  プログラムに渡された引数の数を取得します。

- **分類**

  関数

- **結果の型、および、型パラメタ**

  整数型

- **結果の値**

  プログラムに渡されたコマンド行中の引数の数の値。

11.4.29 IDATE(*IA3*)

- **機能**

  日付を4バイト整数型配列へ変換します。

- **分類**

  サブルーチン

- **引数**

  IA3:　**INTENT(OUT)**属性を持つ配列要素3の4バイト整数型配列。日、月、年の値が、この順に、配列の各要素に設定されます。月は1～12の値、年は1969より大きい値です。

11.4.30 IERRNO()

- **機能**

  直前に検出したエラー番号を取得します。

- **分類**

  関数

- **結果の型、および、型パラメタ**

  4バイト整数型

- **結果の値**

  直前に検出されたエラー番号が返却されます。

11.4.31 ISATTY(*UNIT*)

- **機能**

  端末との接続をチェックします。

- **分類**

  関数

- **引数**

  UNIT:　**INTENT(IN)**属性を持つ4バイト整数型。外部ファイル装置を指定します。

- **結果の型、および、型パラメタ**

  整数型

- **結果の値**

  端末に接続されていると1が、接続されていないと0が返却されます。

11.4.32 ITIME(*IA3*)

- **機能**

  時刻を4バイト整数型配列へ変換します。

- **分類**

  サブルーチン

- **引数**

  IA3:　**INTENT(OUT)**属性を持つ配列要素3の4バイト整数型配列。時、分、秒の値が、この順に、配列の各要素に設定されます。

11.4.33 KILL(*PID*,*SIGNUM*)

- **機能**

  プロセスまたはプロセスグループへシグナルを発信する。

- **分類**

  関数

- **引数**

  PID:　　**INTENT(IN)**属性を持つ4バイト整数型。シグナルを送るプロセスIDを指定します。

  SIGNUM:　　**INTENT(IN)**属性を持つ4バイト整数型。送信するシグナル番号を指定します。

- **結果の型、および、型パラメタ**

  4バイト整数型

- **結果の値**

  正常に終了すると0が、失敗するとエラーコードが返却されます。

11.4.34 LINK(*PATH1*,*PATH2*)

- **機能**

  リンクを作成します。

- **分類**

  関数

- **引数**

  PATH1: **INTENT(IN)**属性を持つ基本文字型のスカラ変数。ファイルパスを指定します。

  PATH2: **INTENT(IN)**属性を持つ基本文字型のスカラ変数。リンクパスを指定します。

- **結果の型、および、型パラメタ**

  整数型

- **結果の値**

  正常に終了すると0が、失敗するとエラーコードが返却されます。

11.4.35 LSTAT(*PATH*,*SXBUF*)

- **機能**

  ファイルの情報を取得します。

- **分類**

  関数

- **引数**

  PATH:　**INTENT(IN)**属性を持つ基本文字型のスカラ変数。ファイルパスを指定します。

  SXBUF: **INTENT(OUT)**属性を持つ配列要素19の4バイト整数型配列。*PATH*のファイル情報が設定されます。*PATH*がリンクファイルのとき、リンクファイルの情報が設定されます。

- **結果の型、および、型パラメタ**

  整数型

- **結果の値**

  正常に終了すると0が、失敗するとエラーコードが返却されます。

- **備考**

  *SXBUF*によって取得できる情報を以下に示します。

  SXBUF(1)　ファイルがあるデバイスID

  SXBUF(2)　inode 番号

  SXBUF(3)　アクセス保護

  SXBUF(4)　ハードリンクの数

  SXBUF(5)　所有者のユーザ ID

  SXBUF(6)　所有者のグループ ID

  SXBUF(7)　0

  SXBUF(8)　全体のサイズ(bytes)

  SXBUF(9)　最終アクセス時刻

  SXBUF(10)　最終修正時刻

  SXBUF(11)　最終状態変更時刻

  SXBUF(12)～SXBUF(19)　予備

11.4.36 LTIME(*I,IA9*)

- **機能**

  ローカルの日付と時刻を4バイト整数型配列へ変換します。

- **分類**

  サブルーチン

- **引数**

  I:　**INTENT(IN)**属性を持つ4バイト整数型。

  IA9:　　**INTENT(OUT)**属性を持つ配列要素9の4バイト整数型配列。*I*で指定された値をEpochからの経過時間に変換し、さらに地方時間に変換して、配列の各要素に設定されます。

11.4.37 MALLOC(*SIZE*)

- **機能**

  領域を確保します。

- **分類**

  関数

- **引数**

  SIZE:　**INTENT(IN)**属性を持つ4バイト整数型。必要な領域の大きさをバイト単位で指定します。

- **結果の型、および、型パラメタ**

  8バイト整数型

- **結果の値**

  確保した領域の先頭アドレスの値。

11.4.38 MALLOC2(*SIZE*)

- **機能**

  領域を確保します。

- **分類**

  関数

- **引数**

  SIZE:　**INTENT(IN)**属性を持つ8バイト整数型。必要な領域の大きさをバイト単位で指定します。

- **結果の型、および、型パラメタ**

  8バイト整数型

- **結果の値**

  確保した領域の先頭アドレスの値。

11.4.39 PERROR(*A*)

- **機能**

  直前に検出したエラー番号のメッセージを標準エラーへ出力します。

- **分類**

  サブルーチン

- **引数**

  A:　**INTENT(IN)**属性を持つ基本文字型のスカラ変数。*A*に指定された文字列、コロン、空白に続いて、エラー番号のメッセージを標準エラーへ出力します。

11.4.40 RENAME(*FROM*,*TO*)

- **機能**

  ファイル名を変更します。

- **分類**

  関数

- **引数**

  FROM:　**INTENT(IN)**属性を持つ基本文字型のスカラ変数。変更元のファイルパスを指定します。

  TO:　　**INTENT(IN)**属性を持つ基本文字型のスカラ変数。変更先のファイルパスを指定します。

- **結果の型、および、型パラメタ**

  整数型

- **結果の値**

  正常に終了すると0が、失敗するとエラーコードが返却されます。

11.4.41 SECNDS(*T*)

- **機能**

  引数の値を基準時間とした経過時間を取得します。

- **分類**

  関数

- **引数**

  T:　**INTENT(IN)**属性を持つ4バイト実数型。基準とする秒数を指定します。

- **結果の型、および、型パラメタ**

4バイト実数型

- **結果の値**

  引数の値を基準時間とした経過時間(秒)の値。引数の値が0.0のときは、真夜中(0時)からの経過時間(秒)の値。

11.4.42 SIGNAL(*SIGNUM*,*HANDLER*)

- **機能**

  シグナル受信時の動作を指定する。

- **分類**

  関数

- **引数**

  SIGNUM:　**INTENT(IN)**属性を持つ4バイト整数型。対象となるシグナル番号を指定します。

  HANDLER:　**INTENT(IN)**属性を持つ外部手続名。シグナル受信時の処理手続名を指定します。

- **結果の型、および、型パラメタ**

  4バイト整数型

- **結果の値**

  正常に終了すると0が、失敗するとエラーコードが返却されます。

11.4.43 SLEEP(*SECS*)

- **機能**

  実行を中断します。

- **分類**

  サブルーチン

- **引数**

  SECS:　**INTENT(IN)**属性を持つ4バイト整数型。中断する秒数を指定します。

11.4.44 STAT(*PATH*,*SXBUF*)

- **機能**

  ファイルの情報を取得します。

- **分類**

  関数

- **引数**

  PATH:　**INTENT(IN)**属性を持つ基本文字型のスカラ変数。ファイルパスを指定します。

  SXBUF: **INTENT(OUT)**属性を持つ配列要素19の4バイト整数型配列。*PATH*のファイル情報が設定されます。*PATH*がリンクファイルのとき、リンク先のファイルの情報が設定されます。

- **結果の型、および、型パラメタ**

  整数型

- **結果の値**

  正常に終了すると0が、失敗するとエラーコードが返却されます。

- **備考**

  *SXBUF*によって取得できる情報を以下に示します。

  SXBUF(1)　ファイルがあるデバイスID

  SXBUF(2)　inode 番号

  SXBUF(3)　アクセス保護

  SXBUF(4)　ハードリンクの数

  SXBUF(5)　所有者のユーザ ID

  SXBUF(6)　所有者のグループ ID

  SXBUF(7)　0

  SXBUF(8)　全体のサイズ(bytes)

  SXBUF(9)　最終アクセス時刻

  SXBUF(10)　最終修正時刻

  SXBUF(11)　最終状態変更時刻

  SXBUF(12)～SXBUF(19)　予備

11.4.45 SYMLNK(*PATH1*,*PATH2*)

- **機能**

  シンボリックリンクの作成

- **分類**

  関数

- **引数**

  PATH1: **INTENT(IN)**属性を持つ基本文字型のスカラ変数。シンボリックリンク*PATH2*で使われるパスを指定します。

PATH2: **INTENT(IN)**属性を持つ基本文字型のスカラ変数。作成するファイル名(シンボリックリンク名)を指定します。

- **結果の型、および、型パラメタ**

  4バイト整数型

- **結果の値**

  正常に終了すると0が、失敗するとエラーコードが返却されます。

11.4.46 SYSTEM(*CMD*)

- **機能**

  コマンドプロセッサに文字列を渡します。

- **分類**

  関数

- **引数**

  CMD:　**INTENT(IN)**属性を持つ基本文字型のスカラ変数。コマンドプロセッサに渡す文字列を指定します。

- **結果の型、および、型パラメタ**

  整数型

- **結果の値**

  渡した文字列の完了ステータスを返却されます。

- **備考**

  サブルーチンとしても、以下の形式で使用可能です。

  サブルーチンを使用したとき、渡した文字列の完了ステータスは、受け取れません。

```
CALL SYSTEM(CMD)
```

11.4.47 TIME()

- **機能**

  日付/時刻（秒単位）を取得します。

- **分類**

  関数

- **結果の型、および、型パラメタ**

  整数型

- **結果の値**

  EPOCHを起点とする日付/時刻（秒単位）の値。

11.4.48 TTYNAM(*UNIT*)

- **機能**

  端末装置名を取得します。

- **分類**

  関数

- **引数**

  UNIT:　**INTENT(IN)**属性を持つ4バイト整数型。外部ファイル装置を指定します。

- **結果の型、および、型パラメタ**

  基本文字型

- **結果の値**

  外部ファイル装置に接続された端末の名称の値。

11.4.49 UNLINK(*PATH*)

- **機能**

  ファイルを削除します。

- **分類**

  関数

- **引数**

  PATH:　**INTENT(IN)**属性を持つ基本文字型のスカラ変数。削除するファイルパスを指定します。

- **結果の型、および、型パラメタ**

  整数型

- **結果の値**

  正常に終了すると0が、失敗するとエラーコードが返却されます。

11.4.50 WAIT(*STATUS*)

- **機能**

  子プロセスの停止または終了を待ち合わせます。

- **分類**

  関数

- **引数**

  STATUS:　　**INTENT(OUT)**属性を持つ4バイト整数型。子プロセスの状態が設定されます。

- **結果の型、および、型パラメタ**

  整数型

- **結果の値**

  成功すると子プロセスのプロセスIDが、失敗するとエラーコードがマイナスで返却されます。

11.5 注意事項

三角関数と指数関数は引数が特定の範囲の値であるとき、正しく結果を計算することができずエラー状態となります。エラー状態のとき、関数の値は”NaN”となります。
関数とそれに対応するエラー状態となる引数の範囲は以下のとおりです。

単精度:

関数	エラー状態となる引数の範囲
sin(x), cos(x), tan(x), cotan(x)	$\|x\| \geqq 2^{21} \times \pi$
sind(x), cosd(x)	$\|x\| \geqq 2^{21} \times 180$
csin(x+yi), ccos(x+yi)	$\|x\| \geqq 2^{21} \times \pi$
cexp(x+yi)	$\|y\| \geqq 2^{21} \times \pi$

倍精度:

関数	エラー状態となる引数の範囲
dsin(x), dcos(x), dtan(x), dcotan(x)	$\|x\| \geqq 2^{50} \times \pi$
dsind(x), dcosd(x)	$\|x\| \geqq 2^{50} \times 180$
cdsin(x+yi), cdcos(x+yi)	$\|x\| \geqq 2^{50} \times \pi$
cdexp(x+yi)	$\|y\| \geqq 2^{50} \times \pi$

4倍精度:

関数	エラー状態となる引数の範囲

qsin(x), qcos(x), qtan(x), qcotan(x)	$\|x\| \geqq 2^{110} \times \boldsymbol{\pi}$
cqsin(x+yi), cqcos(x+yi)	$\|x\| \geqq 2^{110} \times \boldsymbol{\pi}$
cqexp(x+yi)	$\|y\| \geqq 2^{110} \times \boldsymbol{\pi}$

第12章　メッセージ

12.1 診断メッセージ

コンパイラは、プログラムの最適化状況を示すメッセージを標準エラー出力、診断メッセージリストに出力します。本セクションでは、それらの形式、主なメッセージについて説明します。

12.1.1 メッセージの形式

診断メッセージは次の形式で出力されます。

種別（番号）: 位置情報: メッセージ本文［: ヒント］

種別 (番号):

メッセージの種別とメッセージ本文に割り当てられた番号が表示されます。種別には以下があります。

vec:　ベクトル化

opt:　最適化・ベクトル化

dtl:　最適化・ベクトル化のより詳細な情報

inl:　インライン展開

par:　OpenMP並列化・自動並列化

err:　主にOpenMP指示行指定時の問題

位置情報:

診断メッセージ対応するソースコードの行番号が出力されます。標準エラー出力に出力されたとき、行番号の行を含むファイル名も出力されます。

メッセージ本文:

診断メッセージの本文が出力されます。

ヒント:

診断メッセージによっては、手続名、変数名、配列名などが出力されます。

- モジュール手続を出力するとき、モジュール名と手続名を「::」で区切って出力します。
- 内部手続を出力するとき、親手続名と内部手続名を「::」で区切って出力します。

- 派生型の変数、配列の成分名を表示するとき、変数名、配列名と成分名を「%」で区切って出力します。
- 変数名、配列名が不明であるとき、型名が出力されることがあります。
- コンパイラが最適化のために生成した手続名、変数名としてベースとなった手続名、変数名に「$数値」が付加されて出力されることがあります。

12.1.2 メッセージ一覧

vec(101): Vectorized loop.

ループ全体がベクトル化された。

vec(102): Partially vectorized loop.

ループが部分ベクトル化された。

vec(103): Unvectorized loop.

ループがベクトル化されなかった。

vec(107): Iteration count is too small.

ループの繰返し数がベクトル化の閾値より小さいためベクトル化しない。ベクトル化の閾値は**-mvector-threshold=***n*で変更できる。

vec(108): Unvectorizable loop structure.

ループの繰返しを制御する変数やループ構造がベクトル化に適していないため、ベクトル化できない。主に次のような場合に出力される。

- ループの繰返しを制御する変数が別の型に型変換されている。**-mreplace-loop-induction**でベクトル化できる場合がある。
- ループの繰返しの終了判定式が、ループの繰返しを制御する変数とループ内不変の式の比較でない。
- ループ終了の条件式に、.AND.、.OR.、.EQV.、.NEQV.、.NOT.が現れている。
- ループ終了の条件式に、.EQ.、.NE.、＝=、/=が現れている。**-mreplace-loop-equation**でベクトル化できる場合がある。
- ループの中から外への分岐(飛出し)が2個以上含まれる。
- ループ外からループ内への分岐(飛込み)がある。**if**文と**goto**文からなるループの場合に考えられる。
- 部分ベクトル化しようとしたが、そのための作業ベクトルが作成できない。次の例で

は、部分ベクトル化するためにa(1)の作業ベクトルが必要だが、その型はベクトル化できない型であり作業ベクトルを作成できない。

```
      subroutine sub(a, b, c, d, n)
      complex(16) a(n)
      complex(8) b(n), c(n), d(n)
      do i=1,n
         a(1) = b(i) + d(i) + c(i)
         c(i) = a(1)
      enddo
      end
```

vec(109): Vectorization obstructive statement.

ベクトル化できない文である。

vec(110): Vectorization obstructive procedure reference : 手続名

ベクトル化できない手続の呼出しがある。

vec(111): “novector” is specified.

novector指示行が指定されたためベクトル化しない。

vec(112): “novwork” is specified.

novwork指示行が指定されたため部分ベクトル化しない。

vec(113): Overhead of loop division is too large.

ループ分割によるオーバーヘッドが大きくベクトル化の効果がないため、部分ベクトル化しない。

vec(115): Internal table overflow.

ベクトル化処理中に内部テーブルの大きさが足りなくなったため、ベクトル化できない。

vec(116): Unvectorizable procedure reference. : 手続名

ベクトル化できないユーザ手続の呼出しがある。手続ポインタを介して手続を呼び出しているとき手続名は出力されない。

vec(117): Unvectorizable statement.

ベクトル化できない文がある。

vec(118): Unvectorizable data type.

ベクトル化できないデータ型の参照がある。

vec(119): Array is not aligned. : 変数名

配列、または、ポインタの指示先が、ベクトル化できるメモリ境界に整列されていない。

vec(120): Unvectorizable dependency. : 変数名

ベクトル化不可の依存関係がある。

vec(121): Unvectorizable dependency.

ベクトル化不可の依存関係がある。

vec(122): Dependency unknown. Unvectorizable dependency is assumed. : 変数名

依存関係がわからないためベクトル化不可の依存関係が存在すると仮定する。変数名が不明であるとき、変数名は出力しない。ループに**ivdep**指示行を指定すると本依存関係をベクトル化できるものとみなしベクトル化を適用する。

vec(124): Iteration count is assumed. Iteration count=n

ループの繰返し数を最大*n*回であると仮定した。

vec(126): Idiom detected. : マクロ演算の種別

ベクトルマクロ演算が検出された。マクロ演算の種別は以下である。

Max/Min、List Vector、Sum、Product、Bit-op、Iteration、Search

vec(128): Fused multiply-add operation applied.

ベクトルFMA命令を使用した。

vec(129): Array is retained. : 配列名

retain指示行が適用された。

vec(130): Vector register is assigned.: 配列名

vreg指示行により、配列名にベクトルレジスタが割り当てられた。

vec(131): Too many statements.

文の数が多すぎるため、ベクトル化できない。

vec(132): Too many procedure calls.

手続呼出しの数が多すぎるためベクトル化できない。

vec(133): Too many memory refereneces.

メモリ参照の数が多すぎるためベクトル化できない。

vec(134): Too many branches.

分岐の数が多すぎるためベクトル化できない。

vec(139): Packed loop.

packed-vector命令を使用してベクトル化した。

vec(140): Unpacked loop. : 理由

-mvector-packed、または、**packed_vector**指示行が指定されたが、ループがpacked-vector命令を使用してベクトル化できなかった。

vec(141): “nopacked_vector” is specified.

nopacked_vector指示行が指定された。

vec(142): pvreg is used in vector loop.

pvreg指示行で指定された配列がpacked-vector命令を使用されずにベクトル化されたループに含まれる。

vec(143): vreg is used in packed vector loop.

vreg指示行で指定された配列がpacked-vector命令を使用しベクトル化されたループに含まれる。

vec(161): Structure assignment obstructs vectorization.

派生型の代入が含まれるためベクトル化できない。

vec(163): Exception handling obstructs vectorization.

C++例外処理に関わる処理が含まれるためベクトル化できない。

vec(180): I/O statement obstructs vectorization.

入出力に関わる処理が含まれるためベクトル化できない。

vec(181): Allocation obstructs vectorization.

メモリ確保に関わる処理が含まれるためベクトル化できない。

vec(182): Deallocation obstructs vectorization.

メモリ解放に関わる処理が含まれるためベクトル化できない。

vec(183): Run-time checking obstructs vectorization.

実行時チェックに関わる処理が含まれるためベクトル化できない。**-fcheck**で生成される処理の他に、メモリの確保、解放の成功、失敗などに関わる実行時チェックも対象に含まれる。

vec(184): Division obstructs vectorization.

ベクトル化できない型の除算が含まれるためベクトル化できない。

vec(185): Exponentiation obstructs vectorization.

ベクトル化できない型のべき乗算が含まれるためベクトル化できない。

opt(1011): Too large to optimize -- reduce program or loop size.

ループ、または、ルーチンが大きすぎるため最適化できない。ループ、ルーチンの分割を検討する。

opt(1019): Feedback of scalar value from one loop pass to another.

スカラ変数が異なる繰返しで定義された値を参照しているため最適化できない。

opt(1025): Reference to this procedure inhibits optimization.

最適化を阻害する手続呼出しがあるため最適化できない。

opt(1034): Multiple store conflict.

同一の配列要素が複数回定義されるため最適化できない。

opt(1037): Feedback of array elements.

異なる繰返しで同一の配列要素を定義・参照しているため最適化できない。

opt(1038): Loop too complex -- optimization of this loop halted.

ループが複雑すぎるため最適化できない。

opt(1056): Loop nest too deep for optimization.

ループのネストが深すぎるため最適化できない。

opt(1057): Complicated use of variable inhibits loop optimization.

変数の定義/参照が複雑で最適化できない。

opt(1059): Unable to determine last value of scalar temporary.

スカラ変数の終値が確定できないため最適化できない。

opt(1061): Use of scalar under different condition causes feedback.

スカラ変数が異なる条件下で参照されているため最適化できない。

opt(1062): Too many data dependency problems.

データ依存が多すぎるため最適化できない。

opt(1082): Backward transfers inhibit loop optimization.

逆方向分岐があるため最適化できない。

opt(1083): Last value of promoted scalar required.

作業配列化されたスカラ変数の終値が保証できないため最適化できない。

opt(1084): Branch out of the loop inhibits optimization.

ループからの飛出しがあるため最適化できない。

opt(1097): This statement prevents loop optimization.

最適化を阻害する文があり最適化できない。

opt(1117): Indirect branch inhibits to optimization of loop.

間接分岐があるため最適化できない。

opt(1118): This I/O statement inhibits to optimization of loop.

入出力文があるため最適化できない。

opt(1128): Branching too complex to optimize at this optimization level.

分岐が複雑で最適化できない。

opt(1130): Conditional scalar inhibits optimization of outer loop.

条件下で定義されるスカラ変数が外側ループの最適化を阻害している。

opt(1131): Function references in iteration count inhibits optimization.

ループの繰返し制御に現われる関数参照があり最適化できない。

opt(1166): Potential dependency due to pointer -- use restrict qualifier if ok.

ポインタの参照先の定義・参照関係に依存がある可能性があり最適化しない。ループに**ivdep**指示行を指定すると本依存関係を最適化できるものとみなしベクトル化を適用する。

inl(1214): Expansion routine is too big for automatic expansion.: ルーチン名

ルーチンが大きすぎるので自動インライン展開しない。**-finline-max-function-size=**

n、**-finline-max-times**=*n*で展開するルーチンの大きさを調整するとインライン展開できる場合がある。

inl(1219): Nesting level too deep for automatic expansion. : ルーチン名

インライン展開するルーチン呼のネストが深すぎるためインライン展開しない。

-finline-max-depth=*n*で展開するルーチンの呼出しの深さを調整するとインライン展開できる場合がある。

inl(1222): Inlined.: ルーチン名

ルーチン名がインライン展開された。

opt(1268): Use of pointer variable inhibits optimization.

ポインタ変数があるため最適化できない。

opt(1282): This store into array inhibits optimization of outer loop.

配列への代入が外側ループの最適化を阻害している。

opt(1285): Not enough work to justify concurrency optimization.

ループ中の作業量が少ないため自動並列化しない。

opt(1298): Use of induction variable outside the loop inhibits optimization.

インダクション変数がループ外で参照されているため最適化できない。

opt(1299): Redefinition of induction variable in loop inhibits optimization.

インダクション変数がループ内で再定義されているため最適化できない。

opt(1300): Assumed-size private arrays inhibit concurrency.

大きさ引継ぎ配列があるため並列化できない。

opt(1315): Iterations peeled from loop in order to avoid dependence.

ベクトル化不可の依存関係を除去するため、ループの前方/後方展開を行った。

opt(1376): User function reference inhibits optimization.

関数参照があるため最適化できない。

opt(1377): Must synchronize to preserve order of accesses.

同期制御が必要であるため最適化できない。

opt(1378): Many synchronizations needed.

同期制御が多数必要なため並列化しない。

opt(1380): User function references not ok without "cncall".

関数参照があるため並列化できない。**cncall**指示行を指定すると並列化できる場合がある。

opt(1382): Subroutine calls are handled only when "cncall" is used.

サブルーチン呼出しがあるため並列化できない。**cncall**指示行を指定すると並列化できる場合がある。

opt(1387): Overlapping EQUIVALENCEd variables inhibit concurrency.

EQUIVALENCEされた変数間に重なりがあるため最適化できない。

inl(1388): Inlining inhibited: OpenMP or parallel directive.

OpenMP指示行、自動並列化の指示行があるためインライン展開できなかった。

opt(1395): Inner loop stripped and strip loop moved outside outer loop.

外側ループストリップマイニングを適用した。

opt(1408): Loop interchanged.

ループを入れ換えた。

opt(1409): Alternate code is generated.

条件ベクトル化を適用した。

opt(1589): Outer loop moved inside inner loop(s).

外側ループを内側ループと入れ換えた。

opt(1590): Inner loop moved outside outer loop(s).

内側ループを外側ループと入れ換えた。

opt(1592): Outer loop unrolled inside inner loop.

外側ループをアンローリングした。

opt(1593): Loop nest collapsed into one loop.

ループを一重化した。

opt(1772): Loop nest fused with following nest(s).

後続のループと融合した。

opt(1800): Idiom detected (matrix multiply).

多重ループを行列積ライブラリ呼出しに置換した。

12.2 実行時エラーメッセージ

本セクションでは、Fortranコンパイラの実行時ルーチンの出力する主要なエラーメッセージについて説明します。

12.2.1 メッセージの形式

「Runtime Error:」に続けて、行番号、ファイル名、エラーの内容を示すメッセージ本文が出力されます。行番号、ファイル名が表示されない出力もあります。

```
Runtime Error:[ 行番号, ファイル名:] メッセージ本文
```

12.2.2 メッセージ一覧

ADVANCE= specifier must be 'YES' or 'NO'

READ文または**WRITE**文の**ADVANCE**指定子の値が不正です。**ADVANCE**指定子には 'YES' または 'NO' のいずれかを指定してください。

ALLOCATABLE *dimname* is not currently allocated

allocatable 属性の配列*dimname*に配列領域が割付けられていません。allocate文で配列領域を割付けてください。

ALLOCATE failed: Out of memory

メモリ不足のため割付けできません。使用しているメモリサイズを確認し、プログラムを見直してください。

Array constructor implied DO limit expression value *value* is out of range for index variable var type *type*

*type*型のindex変数*var*の終値*value*が範囲外です。*value*を見直してください。

Array constructor implied DO step expression value *value* is out of range for index variable var type *type*

*type*型のindex変数*var*の刻み幅の値*value*が範囲外です。*value*を見直してください。

ASYNCHRONOUS= specifier must be 'NO' or 'YES'

OPEN文の**ASYNCHRONOUS**指定子の値が不正です。**ASYNCHRONOUS**指定子には'YES' または 'NO'のいずれかを指定してください。

Buffer overflow on output

入出力文でレコードバッファがオーバーフローしました。**OPEN**文に指定した**RECL**指定子や環境変数**VE_FORT_FMTBUF**、環境変数**VE_FORT_RECORDBUF**に指定した値が、出力データのサイズより大きいことを確認してください。

Call to OMP_SET_MAX_ACTIVE_LEVELS from within a *name* region

*name*領域から**OMP_SET_MAX_ACTIVE_LEVELS**が呼ばれました。処理内容を見直してください。

Cannot allocate ALLOCATABLE variable - out of memory

割付け変数の割付けを行うために必要とするメモリが足りません。使用しているメモリサイズを確認し、プログラムを見直してください。

Cannot allocate array temporary - out of memory

メモリ不足のため配列の一時領域が確保できませんでした。使用しているメモリサイズを確認し、プログラムを見直してください。

Cannot allocate I/O buffer in OPEN processing UNIT=Unit-Number

この装置番号に対する**OPEN**文で、入出力バッファの領域が確保できませんでした。不要な外部ファイル装置を**CLOSE**文で閉じてください。または、環境変数**VE_FORT_SETBUF**で適切なサイズを指定してください。

Cannot allocate initial memory - out of memory

初期メモリを確保できません。使用しているメモリサイズを確認し、プログラムを見直してください。

Cannot allocate memory for asynchronous i/o

非同期入出力文の実行に必要な一時領域が確保できませんでした。並びの数や並びに指定している配列のサイズを見直してください。

Cannot allocate memory for environment variable VE_FMTIO_OFFLOAD

環境変数**VE_FMTIO_OFFLOAD**の実行に必要な一時領域が確保できませんでした。環境変数**VE_FMTIO_OFFLOAD**の指定を止めてください。

Cannot allocate memory for environment variable VE_FORT_UFMTADJUST

環境変数**VE_FORT_UFMTEADJUST**の実行に必要な一時領域が確保できませんでした。並びに指定している配列のサイズを見直してください。

Cannot allocate memory for environment variable VE_FORT_UFMTENDIAN

環境変数**VE_FORT_UFMTENDIAN**の実行に必要な一時領域が確保できませんでした。並びに指定している配列のサイズを見直してください。

Cannot allocate record buffer in OPEN processing UNIT=Unit-Number

この装置番号に対する**OPEN**文で、レコードバッファの領域が確保できませんでした。不要な外部ファイル装置を**CLOSE**文で閉じてください。または、環境変数**VE_FORT_RECORDBUF**で適切なサイズを指定してください。

Cannot BACKSPACE unformatted ACCESS='STREAM' unit Unit-Number

書式なしストリームファイルに対して**BACKSPACE**文は実行できません。書式なしストリームファイルを利用したいときは**BACKSPACE**文を削除してください。**BACKSPACE**文を実行したいときは、順番探査ファイルに変更してください。

Cannot find OLD file

STATUS='OLD'でオープンしようとしたファイルが存在しません。ファイル名を確認し、誤っているときは正しいファイル名に訂正してください。正しいときは**OPEN**文の**STATUS**指定子の値を見直してください。

Cannot get storage for automatic array - out of memory

メモリ不足のため自動配列の領域が確保できませんでした。使用しているメモリサイズを確認し、プログラムを見直してください。

Cannot get storage for variable - out of memory

メモリ不足のため変数の領域が確保できませんでした。使用しているメモリサイズを確認し、プログラムを見直してください。

Character string edit descriptor does not terminate before format end

文字列編集記述子が不正です。書式を見直してください。

- nH形編集で、n分の文字が存在しません
- 文字定数編集で、前の囲み文字と対となる後ろの囲み文字がありません

Character string edit descriptor used on input

入力文の書式に文字列編集記述子が指定されています。入力文の書式を見直してください。

DECIMAL= specifier must be 'POINT' or 'COMMA'

OPEN文、**READ**文または**WRITE**文の**DECIMAL**指定子の値が不正です。**DECIMAL**指定子には'POINT'または'COMMA'のいずれかを指定してください。

DELIM= specifier in OPEN for an UNFORMATTED file

OPEN文にFORM='UNFORMATTED'と一緒に**DELIM**指定子が指定されています。ファイルを書式なしファイルとしてオープンしたいときは**DELIM**指定子を削除してください。そうでないならば、**OPEN**文の**FORM**指定子の値を見直してください。

DIM argument (*value*) out of range 1:*rank* in intrinsic CSHIFT

組込み手続CSHIFTのDIM引数の値*value*が範囲外です。DIM引数の値*value*を見直してください。

DIM argument (*value*) out of range 1:*rank* in intrinsic EOSHIFT

組込み手続EOSHIFTのDIM引数の値*value*が範囲外です。DIM引数の値*value*を見直してください。

DIM argument (*value*) out of range 1:*rank* in intrinsic FINDLOC

組込み手続FINDLOCのDIM引数の値*value*が範囲外です。DIM引数の値*value*を見直してください。

DIM argument (*value*) out of range 1:*rank* in intrinsic LBOUND

組込み手続LBOUNDのDIM引数の値*value*が範囲外です。DIM引数の値*value*を見直してください。

DIM argument (*value*) out of range 1:*rank* in intrinsic MAXLOC

組込み手続MAXLOCのDIM引数の値*value*が範囲外です。DIM引数の値*value*を見直してください。

DIM argument (*value*) out of range 1:*rank* in intrinsic MAXVAL

組込み手続MAXVALのDIM引数の値*value*が範囲外です。DIM引数の値*value*を見直してください。

DIM argument (*value*) out of range 1:*rank* in intrinsic MINLOC

組込み手続MINLOCのDIM引数の値*value*が範囲外です。DIM引数の値*value*を見直してください。

DIM argument (*value*) out of range 1:*rank* in intrinsic MINVAL

組込み手続MINVALのDIM引数の値*value*が範囲外です。DIM引数の値*value*を見直してください。

DIM argument (*value*) out of range 1:*rank* in intrinsic SIZE

組込み手続SIZEのDIM引数の値*value*が範囲外です。DIM引数の値*value*を見直してください。

DIM argument (*value*) out of range 1:*rank* in intrinsic UBOUND

組込み手続UBOUNDのDIM引数の値*value*が範囲外です。DIM引数の値*value*を見直してください。

DIM argument (*value*) out of range 1:*rank*+1 in intrinsic SPREAD

組込み手続SPREADのDIM引数の値*value*が範囲外です。DIM引数の値*value*を見直してください。

Direct access is incompatible with the POSITION= specifier

OPEN文にACCESS='DIRECT'と一緒に**POSITION**指定子が指定されています。ファイルを直接探査ファイルとしてオープンしたいときは**POSITION**指定子を削除してください。そうでないならば、**OPEN**文の**ACCESS**指定子の値を見直してください。

DO limit expression value *value* is out of range for index variable *var* type *type*

*type*型のindex変数*var*の終値*value*が範囲外です。*value*を見直してください。

DO step expression value *value* is out of range for index variable *var* type *type*

*type*型のindex変数*var*の刻み幅の値*value*が範囲外です。*value*を見直してください。

Element *element* of ORDER argument (value *value*) to intrinsic RESHAPE is out of range (1:*rank*)

組込み手続RESHAPEのORDER引数の値*value*が範囲外です。ORDER引数の値*value*を見直してください。

ENDFILE applied twice to unit *Unit-Number* with no intervening file positioning

ENDFILE文の実行に続けて**ENDFILE**文を実行しようとしています。ファイル終了記録の後へはファイル終了記録を出力することはできません。2回目の**ENDFILE**文を削除してください。

EXECUTE_COMMAND_LINE has WAIT=.FALSE., but asynchronous execution is not supported

EXECUTE_COMMAND_LINEに**WAIT=.FALSE.**が設定されていますが、非同期実行はサポートされていません。処理内容を見直してください。

Expected decimal point in format specification

FORMAT文の中で編集記述子にドットが不足しています。**FORMAT**文の形式を確認してください。

Expected integer literal constant in format specification

書式内の記載が不正です。数値を記載するべき箇所に数値がありません。考えられるケースとしては下記があります。書式を見直してください。

- Iw.m, Zw.m, Ow.m, Bw.m 形式において、mの値が指定されていない(ピリオドは指定されている)
- Dw.d, Fw.d, Ew.d, ENw.d, ESw.d, Gw.d 形式において、dの値が指定されていな

い(ピリオドは指定されている)

- Ew.dEe, ENw.dEe, ESw.dEe, Gw.dEe 形式において、eの値が指定されていない(指数文字は指定されている)
- kP 形式において、kに符号(+-)を指定したあと、数字が指定されていない
- TLn, TRn, Tn 形式において、nの値が指定されていない

Expected P following signed integer constant in format specification

符号(+-)付きの数字のあとにP以外の文字が指定されいます。符号付きの数値が指定できるのはkP形編集のみです。書式を見直してください。

Exponent too large for Dw.d format

指数部の桁が大きすぎるため、指定されたDw.d形式の書式では出力できません。Ew.dEe形式の書式で指数の桁数を明示的に指定してください。Ew.dEe形式の書式に変更すると指数文字がDからEになるので注意してください。

Exponent too large for Ew.d format

指数部の桁が大きすぎるため、指定されたEw.d形式の書式では出力できません。Ew.dEe形式の書式で指数の桁数を明示的に指定してください。

F90_UNIX_DIR.GETCWD: Both NAME and LENNAME are not PRESENT

F90_UNIX_DIRモジュールの**GETCWD**手続で*NAME*と*LENNAME*が存在しません。処理内容を見直してください。

F90_UNIX_ENV.GETARG: Value of K (*value*) is out of range 0:*num*

F90_UNIX_ENVモジュールの**GETARG**手続のKの値が範囲外です。処理内容を見直してください。

F90_UNIX_ENV.GETENV(*var*): No such environment variable

F90_UNIX_ENVモジュールの**GETENV**手続(*var*)に指定された環境変数はありません。処理内容を見直してください。

F90_UNIX_ENV.ISATTY: LUNIT (*value*) is out of range

F90_UNIX_ENVモジュールの**ISATTY**手続に指定した論理装置*value*の値が範囲外です。処理内容を見直してください。

F90_UNIX_ENV.SYSCONF(*value*): Not a valid sysconf name

F90_UNIX_ENVモジュールの**SYSCONF**手続(*value*)に指定されたsysconf名は有効ではありません。処理内容を見直してください。

F90_UNIX_ENV.SYSCONF(*value*): Result (*value*) too large for VAL

F90_UNIX_ENVモジュールの**SYSCONF**手続(*value*)の結果の値が大きすぎます。処理内容を見直してください。

F90_UNIX_ENV.TTYNAME: LUNIT (*value*) is out of range

F90_UNIX_ENVモジュールの**TTYNAME**手続に指定した論理装置*value*の値が範囲外です。処理内容を見直してください。

F90_UNIX_FILE.FSTAT: LUNIT (*value*) is out of range

F90_UNIX_FILEモジュールの**FSTAT**手続に指定した論理装置*value*の値が範囲外です。処理内容を見直してください。

F90_UNIX_IO.FLUSH: LUNIT (*value*) is out of range

F90_UNIX_IOモジュールの**FLUSH**手続に指定した論理装置*value*の値が範囲外です。処理内容を見直してください。

Field/exponent width or repeat in format specification must be non-zero

書式指定の欄幅または反復数にゼロは指定できません。欄幅または反復数にゼロより大きい整数値を指定してください。

File name too long

オープンしようとしたファイルパス名が長すぎます。ファイルパス名は255バイト以内にしてください。

FILE= specifier on OPEN with STATUS='SCRATCH'

OPEN文にSTATUS='SCRATCH'と一緒に**FILE**指定子が指定されています。ファイルをSCRATCHファイルとしてオープンしたいときは**FILE**指定子を削除してください。そうでないならば、**OPEN**文の**STATUS**指定子の値を見直してください。

Floating overflow on real number input

実数型の入力で、対応する並びの実数の精度の範囲外の値を入力しようとしています。対応する並びの実数の精度を見直すか、入力データを見直してください。

FORALL limit expression value *value* is out of range for index variable *var* type *type*

*type*型のindex変数*var*の終値*value*が範囲外です。*value*を見直してください。

FORALL step expression value *value* is out of range for index variable *var* type *type*

*type*型のindex変数*var*の刻み幅の値*value*が範囲外です。*value*を見直してください。

FORALL step value is zero for index variable *var*

FORALL構文のステップ数がゼロです。ゼロ以外を指定してください。

Format specification does not end with a right parenthesis

書式指定の末尾が右括弧で終わっていません。書式の最後に右括弧を追加してください。

I/O error on unit *Unit-Number*: Disk quota exceeded

この装置番号を指定した**WRITE**文や**CLOSE**文がディスクのクォータ制限のため、書き込みに失敗しました。ファイルシステムのクォータ制限を確認してください。

I/O error on unit Unit-Number: Permission denied

この装置番号に接続されたファイルのパーミッションがないため、アクセスできません。指定したファイルのパーミッションを確認してください。

Illegal character '' in LOGICAL input field

論理データの入力で、論理データとして許されない文字を入力しようとしています。入力データを見直してください。

Incorrect unit for VE_FORT_MEM_BLOCKSIZE.

環境変数**VE_FORT_MEM_BLOCKSIZE**に不正な単位が指定されました。環境変数**VE_FORT_MEM_BLOCKSIZE**に指定した単位が、G、または、Mとなっていることを確認してください。

Input list bigger than record length in unformatted READ on unit *Unit-Number*

書式なし入力文の実行で、記録の長さを超えて入力しようとしています。記録の長さを超えて入力しないように書式なし入力文を修正してください。

Input value too large for default INTEGER type

基本整数型の入力で、基本整数型の範囲外の値を入力しようとしています。入力データを見直してください。

Input value too large for INTEGER(KIND=1)

1バイト整数型の入力で、1バイト整数型の範囲外の値を入力しようとしています。入力データを見直してください。

Input value too large for INTEGER(KIND=2)

2バイト整数型の入力で、2バイト整数型の範囲外の値を入力しようとしています。入力データを見直してください。

Internal file overflow

入出力文で指定した内部ファイルがオーバーフローしました。内部ファイルに指定した文字型変数のサイズが、出力データのサイズより大きいことを確認してください。

Invalid character in binary integer input field

2進データの入力で、2進データとして許されない文字を入力しようとしています。入力データを見直してください。

Invalid character in hexadecimal integer input field

16進データの入力で、16進データとして許されない文字を入力しようとしています。入力データを見直してください。

Invalid character in integer input field

整数データの入力で、整数データとして許されない文字を入力しようとしています。入力データを見直してください。

Invalid character in octal integer input field

8進データの入力で、8進データとして許されない文字を入力しようとしています。入力データを見直してください。

Invalid character in real input field

実数データの入力で、実数データとして許されない文字を入力しようとしています。入力データを見直してください。

Invalid character *value* in NAMELIST input

NAMELISTの入力に不正な文字*value*があります。入力データを見直してください。

Invalid edit descriptor beginning with 'edit character'

書式内に不正な文字があります。書式を見直してください。

Invalid edit descriptor for character i/o-list item

文字型の入出力並びに対して指定できない書式が指定されています。書式を見直してください。

Invalid edit descriptor for integer i/o-list item

整数型の入出力並びに対して指定できない書式が指定されています。書式を見直してください。

Invalid edit descriptor for logical i/o-list item

論理型の入出力並びに対して指定できない書式が指定されています。書式を見直してください。

Invalid edit descriptor for real i/o-list item

実数型の入出力並びに対して指定できない書式が指定されています。書式を見直してください。

Invalid edit descriptor G0.d for CHARACTER input/output item

文字型の入出力並びに対応する書式に欄幅がゼロのGw.d形編集が指定されました。欄幅に1以上の値を指定してください。

Invalid edit descriptor G0.d for INTEGER input/output item

整数型の入出力並びに対応する書式に欄幅がゼロのGw.d形編集が指定されました。欄幅に1以上の値を指定してください。

Invalid edit descriptor G0.d for LOGICAL input/output item

論理型の入出力並びに対応する書式に欄幅がゼロのGw.d形編集が指定されました。欄幅に1以上の値を指定してください。

Invalid exponent in real input field

実数データの入力で、実数データとして許されない指数データを入力しようとしています。入力データを見直してください。

Invalid input for character editing

文字型の入力で、不正な形式のデータを入力しようとしています。入力データを見直してください。

Invalid input for complex editing

複素数型の入力で、不正な形式のデータを入力しようとしています。入力データを見直してください。

Invalid input for integer editing

整数型の入力で、不正な形式のデータを入力しようとしています。入力データを見直してください。

Invalid input for logical editing

論理型の入力で、不正な形式のデータを入力しようとしています。入力データを見直してください。

Invalid input for real editing

実数型の入力で、不正な形式のデータを入力しようとしています。入力データを見直してください。

Invalid value for ACCESS= specifier

OPEN文の**ACCESS**指定子の値が不正です。**ACCESS**指定子には'SEQUENTIAL', 'DIREC

T', 'STREAM'または'APPEND'のいずれかを指定してください。

Invalid value for ACTION= specifier

OPEN文の**ACTION**指定子の値が不正です。**ACTION**指定子には'READWRITE', 'READ' または'WRITE'のいずれかを指定してください。

Invalid value for BLANK= specifier

OPEN文または**READ**文の**BLANK**指定子の値が不正です。**BLANK**指定子には'NULL'または'ZERO'のいずれかを指定してください。

Invalid value for DELIM= specifier

OPEN文または**WRITE**文の**DELIM**指定子の値が不正です。**DELIM**指定子には'NONE', 'APOSTROPHE'または'QUOTE'のいずれかを指定してください。

Invalid value for FORM= specifier

OPEN文の**FORM**指定子の値が不正です。**FORM**指定子には'FORMATTED'または'UNFORMATTED'のいずれかを指定してください。

Invalid value for PAD= specifier

OPEN文または**READ**文の**PAD**指定子の値が不正です。**PAD**指定子には'YES'または'NO'のいずれかを指定してください。

Invalid value for POS= specifier

READ文または**WRITE**文の**POS**指定子の値が不正です。**POS**指定子には1以上の整数値を指定してください。

Invalid value for POSITION= specifier

OPEN文の**POSITION**指定子の値が不正です。**POSITION**指定子には'ASIS', 'REWIND' または'APPEND'のいずれかを指定してください。

Invalid value for RECL= specifier (must be positive)

OPEN文の**RECL**指定子の値が不正です。正の整数にしてください。

Invalid value for ROUND= specifier

OPEN文、**READ**文または**WRITE**文の**ROUND**指定子の値が不正です。**ROUND**指定子には'PROCESSOR_DEFINED', 'UP', 'DOWN', 'ZERO', 'NEAREST' または 'COMPATIBLE' のいずれかを指定してください。

Invalid value for STATUS= specifier

OPEN文または**CLOSE**文のSTATUS指定子の値が不正です。**STATUS**指定子には'KEEP'または'DELETE'のいずれかを指定してください。

Invalid value for VE_FORT_MEM_BLOCKSIZE.

環境変数**VE_FORT_MEM_BLOCKSIZE**に不正な値が指定されました。環境変数**VE_FORT_MEM_BLOCKSIZE**に指定した値が、0、または、2のべき乗となっていることを確認してください。

Invalid value of VE_INIT_HEAP.

環境変数**VE_INIT_HEAP**に不正な値が指定されました。環境変数**VE_INIT_HEAP**に指定した値を確認してください。

Left-hand side of assignment has duplicate vector subscript value *value* for dimension *dim*

代入において左辺の*dim*次元目のベクトル添字が重複しています。処理内容を見直してください。

Left-hand side of assignment has vector subscript *name* with duplicate value *value*

代入において左辺のベクトル添字が重複しています。処理内容を見直してください。

LEN argument (*value*) out of range 0:*bitsize* in intrinsic IBITS

組込み手続IBCLRのLEN引数の値*value*が範囲外です。LEN引数の値*value*を見直してください。

Missing length of H edit descriptor

書式指定において、H形編集記述子「*n*H*string*」の*n*にあたる文字数の指定がありません。書式指定を見直してください。

Multiple assignment to scalar *var* in FORALL

FORALL構文内のスカラ変数*var*に対し複数回の代入がされています。スカラ変数への代入は一度のみとなるようにしてください。

Multiple assignment to scalar variable in FORALL

FORALL構文内のスカラ変数に対し複数回の代入がされています。スカラ変数への代入は一度のみとなるようにしてください。

Multiple assignment to whole array *var* in FORALL

FORALL構文内の配列の同一要素に対し複数回の代入がされています。同一要素への代入は一度のみとなるようにしてください。

Nested format-item-list is empty

入出力文の書式の()内に編集記述子が指定されていません。()内に編集記述子を指定するか、不要な()を削除してください。

NEW file already exists

STATUS='NEW'でオープンしようとしたファイルが既に存在します。ファイル名を確認し、誤っているときは正しいファイル名に訂正 してください。正しいときは**OPEN**文の**STATUS**指定子の値を見直してください。

NEWUNIT= specifier but no FILE= and STATUS= value is not 'SCRATCH'

OPEN文に**NEWUNIT**指定子が指定されていますが、**FILE**指定子が指定されておらず、**STATUS**指定子の値がSCRATCHでもありません。自動採番で装置をオープンするときは、**FILE**指定子でファイル名を指定するか、STATUS='SCRATCH'でSCRATCHファイルを指定してください。そうでないならば、**NEWUNIT**指定子ではなく**UNIT**指定子を利用してください。

No data edit descriptor in unlimited format item

入出力文の書式において、無制限反復が指定されましたが、対象の()内にデータ編集記述子が存在しません。無制限反復を指定するなら対象の()内にデータ編集記述子を指定してください。データ編集記述子が不要なら、無制限反復は使用しないように書式を修正してください。

No edit descriptor following repeat factor

書式指定において、反復数に続くデータ編集記述子が指定されていません。書式指定を見直してください。

No FILE= specifier with STATUS='REPLACE' or STATUS='NEW'

OPEN文の**STATUS**指定子にREPLACEまたはNEWが指定されましたが、**FILE**指定子が指定されていません。**OPEN**文の**STATUS**指定子にREPLACEまたはNEWを指定するときは**FILE**指定子も一緒に指定してください。そうでないならば、**STATUS**指定子の値を見直してください。

No left parenthesis after unlimited repeat factor '*'

入出力文の書式に不正な無制限反復が指定されました。無制限反復を指定する書式は必ず()で囲むようにしてください。

No unit available for NEWUNIT= specifier

OPEN文に**NEWUNIT**指定子が指定されましたが、自動採番でのファイルのオープン数が上限を超えました。不要なファイルはクローズしてください。

No value found in LOGICAL input field

論理データの入力で、論理データがありません。入力データを見直してください。

OPEN on connected unit *Unit-Number* has different ACCESS= specifier

既に接続されている装置に対する**OPEN**文の**ACCESS**指定子の値が以前と異なります。**ACCESS**指定子の値を同じ値に変更してください。新しい値で接続したいならば、一度装置をクローズしてから**OPEN**文を実行してください。

OPEN on connected unit *Unit-Number* has different ACTION= specifier

既に接続されている装置に対する**OPEN**文の**ACTION**指定子の値が以前と異なります。ACTION指定子の値を同じ値に変更してください。新しい値で接続したいならば、一度装置をクローズしてから**OPEN**文を実行してください。

OPEN on connected unit *Unit-Number* has different ASYNCHRONOUS= specifier

既に接続されている装置に対する**OPEN**文の**ASYNCHRONOUS**指定子の値が以前と異なります。**ASYNCHRONOUS**指定子の値を同じ値に変更してください。新しい値で接続したいなら、一度装置をクローズしてから**OPEN**文を実行してください。

OPEN on connected unit *Unit-Number* has different FORM= specifier

既に接続されている装置に対する**OPEN**文の**FORM**指定子の値が以前と異なります。**FORM**指定子の値を同じ値に変更してください。新しい値で接続したいならば、一度装置をクローズしてから**OPEN**文を実行してください。

OPEN on connected unit *Unit-Number* has different POSITION= specifier

既に接続されている装置に対する**OPEN**文の**POSITION**指定子の値が以前と異なります。**POSITION**指定子の値を同じ値に変更してください。新しい値で接続したいならば、一度装置をクローズしてから**OPEN**文を実行してください。

OPEN on connected unit *Unit-Number* has different RECL= specifier

既に接続されている装置に対する**OPEN**文の**RECL**指定子の値が以前と異なります。**RECL**指定子の値を同じ値に変更してください。新しい値で接続したいならば、一度装置をクローズしてから**OPEN**文を実行してください。

OPEN on connected unit *Unit-Number* with STATUS= specifier must have STATUS='OLD'

既に接続されている装置に対する**OPEN**文にSTATUS='OLD'以外が指定されています。**STATUS**指定子の値はOLDに変更してください。

Out of memory

一時領域を含む実行に必要なメモリが足りません。使用しているメモリサイズを確認し、プログラムを見直してください。

Out of memory in intrinsic ADJUSTL

組込み手続**ADJUSTL**において必要とするメモリが足りません。使用しているメモリサイズを確認し、プログラムを見直してください。

Out of memory in intrinsic ADJUSTR

組込み手続**ADJUSTR**において必要とするメモリが足りません。使用しているメモリサイズを確認し、プログラムを見直してください。

Out of memory in intrinsic EXECUTE_COMMAND_LINE

組込み手続**EXECUTE_COMMAND_LINE**において必要とするメモリが足りません。使用しているメモリサイズを確認し、プログラムを見直してください。

Out of memory in intrinsic PACK

組込み手続**PACK**において必要とするメモリが足りません。使用しているメモリサイズを確認し、プログラムを見直してください。

Out of memory in intrinsic RESHAPE

組込み手続**RESHAPE**において必要とするメモリが足りません。使用しているメモリサイズを確認し、プログラムを見直してください。

Out of memory in intrinsic SPREAD

組込み手続**SPREAD**において必要とするメモリが足りません。使用しているメモリサイズを確認し、プログラムを見直してください。

Out of range: substring ending position envpos is greater than length *len*

部分列の終了位置が文字列の長さ*len*より大きいです。範囲内の値を指定してください。

Out of range: substring starting position startpos is less than 1

部分列の開始位置が1未満です。1以上を指定してください。

POS argument (*value*) out of range 0:*bitsize* in intrinsic IBCLR

組込み手続IBCLRのPOS引数の値*value*が範囲外です。POS引数の値*value*を見直してください。

POS argument (*value*) out of range 0:*bitsize* in intrinsic IBITS

組込み手続IBITSのPOS引数の値*value*が範囲外です。POS引数の値*value*を見直してください。

POS argument (*value*) out of range 0:*bitsize* in intrinsic IBSET

組込み手続IBSETのPOS引数の値*value*が範囲外です。POS引数の値*value*を見直してください。

POS= specifier but unit *Unit-Number* is not open for STREAM i/o

ストリームファイル以外に対して、入出力文に**POS**指定子が指定されています。ストリームファイル以外のファイルを利用するなら**POS**指定子は削除してください。そうでないならば、ストリームファイルとして指定された装置を接続してください。

READ after WRITE with no intervening file positioning

順番探査ファイルに対して**WRITE**文の実行に続けて**READ**文を実行しようとしています。あるいは、ストリームファイルに対して**WRITE**文の実行に続けて**POS**指定子なしに**READ**文を実行しようとしています。入力文の前に**REWIND**文を実行してください。あるいは、ストリームファイルに対する**READ**文に**POS**指定子を指定してください。

READ/WRITE attempted after ENDFILE on unit *Unit-Number*

ENDFILE文の実行に続けて入出力文を実行しようとしています。あるいは、ファイル終了条件となった直後に、入出力文を実行しようとしています。ファイル終了記録の後へは記録を出力することはできません。入力文の前に**REWIND**文を実行してください。あるいは、ファイル終了記録の直前へ記録を追加するときには、ファイル終了記録の直前へ位置付けるために**BACKSPACE**文を1度実行してください。

RECL= specifier with ACCESS='STREAM'

OPEN文にACCESS='STREAM'と一緒に**RECL**指定子が指定されています。ファイルをストリームファイルとしてオープンしたいときは**RECL**指定子を削除してください。そうでないならば、**OPEN**文の**ACCESS**指定子の値を見直してください。

Record longer than 2GB not supported

順編成ファイルの書式なし記録の入出力で記録サイズが2ギガバイトを超えました。入力で本メッセージが出力されたときは、入力データのエンディアン形式を確認してください。エンディアン形式がビックエンディアンのときは、環境変数**VE_FORT_UFMTENDIAN**を指定してください。それ以外のときは、環境変数**VE_FORT_EXPRCW**、または、環境変数**VE_FORT_SUBRCW**を指定してください。

Record number *Record Number* out of range

READ文または**WRITE**文の**REC**指定子の値が不正です。**REC**指定子には1以上の整数値を指定してください。

Repeat factor given for blank-interpretation edit descriptor

書式指定において、空白解釈編集記述子に反復数が指定されています。書式指定を見直してください。

Repeat factor given for character string edit descriptor

書式指定において、文字列編集記述子に反復数が指定されています。書式指定を見直して

ください。

Repeat factor given for position edit descriptor

書式指定において、位置付け編集記述子に反復数が指定されています。書式指定を見直してください。

Repeat factor given for rounding edit descriptor

書式指定において、丸め編集記述子に反復数が指定されています。書式指定を見直してください。

Repeat factor given for sign edit descriptor

書式指定において、符号制御編集記述子に反復数が指定されています。書式指定を見直してください。

Scale factor *num* out of range for d=*num*

範囲外の桁移動数です。処理内容を見直してください。

SHIFT argument (*value*) out of range -*bitsize*:*bitsize* in intrinsic ISHFT

組込み手続ISHFTのSHIFT引数の値*value*が範囲外です。SHIFT引数の値*value*を見直してください。

SHIFT argument (*value*) out of range -*size*:*size* in intrinsic ISHFTC

組込み手続ISHFTCのSHIFT引数の値*value*が範囲外です。SHIFT引数の値*value*を見直してください。

Sign in a numeric input field not followed by any digits

整数または実数データの入力で、符号に続く数字データがありません。入力データを見直してください。

SIGN= specifier must be 'PROCESSOR_DEFINED', 'PLUS' or 'SUPPRESS'

OPEN文または**WRITE**文の**SIGN**指定子の値が不正です。**SIGN**指定子には'PROCESSOR_DEFINED', 'PLUS' または 'SUPPRESS'のいずれかを指定してください。

SIZE argument (*value*) out of range 1:*maxsize* in intrinsic ISHFTC

組込み手続ISHFTCのSIZE引数の値*value*が範囲外です。SIZE引数の値*value*を見直してください。

SIZE= is not valid without ADVANCE='NO'

READ文にADVANCE='NO' の指定なしに**SIZE**指定子が指定されています。停留入力文を利用するときは**SIZE**指定子を削除してください。**SIZE**指定子を指定するならADVANCE='NO'を指定してください。

STATUS='KEEP' is invalid for a SCRATCH file

CLOSE文で一時ファイルに対してSTATUS='KEEP'が指定されています。**CLOSE**文の**STATUS**指定子の値をDELETEに変更するか、**OPEN**文の**STATUS**指定子の値をSCRATCH以外に変更してください。

Subscript (*value*) out of range in input for object *objname* of NAMELIST/*namelist/*

NAMELISTの項目への入力で添字の値が範囲外です。添字の値を見直してください。

Subscript out of range for assumed-size array *name* - Access to element value but actual argument has only *value* elements

大きさ引継ぎ配列nameの添字の値が範囲外です。添字の値を見直してください。

Subscript *rank* of *dimname* (value *value*) is out of range (*lower*:*)

配列*dimname*のrank次元の添字の値*value*が範囲外です。配列*dimname*の*rank*次元の添字の値*value*を見直してください。

Subscript *rank* of *dimname* (value *value*) is out of range (*lower*:*upper*)

配列*dimname*の*rank*次元の添字の値*value*が範囲外です。配列*dimname*の*rank*次元の添字の値*value*を見直してください。

Substring (*lower*:*upper*) out of bounds in input for object *objname* of NAMELIST/*namelist/*

NAMELISTの項目への入力で部分列が範囲外です。部分列の値を見直してください。

Substring has zero length in input for object *objname* of NAMELIST/*namelist/*

NAMELISTの項目への入力で部分列の長さがゼロです。部分列の値を見直してください。

Sub-format groups nested too deeply

書式内の括弧のネストが40を超えています。ネスト数は40以内にしてください。

The RECL= specifier must be given for DIRECT access OPEN

OPEN文の**ACCESSS**指定子にDIRECTが指定されましたが、**RECL**指定子が指定されていません。**OPEN**文の**ACCESS**指定子にDIRECTを指定するときは**RECL**指定子も一緒に指定してください。そうでないならば、**ACCESS**指定子の値を見直してください。

Unexpected exponent for G0 edit descriptor

Gw.dEe 形編集において、欄幅にゼロが指定されました。欄幅をゼロにしたいときはGw.d形編集に変更してください。そうでないならば、欄幅に適切な値を指定してください。

Unexpected subscript for object *objname* of NAMELIST/*namelist*/

NAMELISTの項目に予期しない添字があります。添字の値を見直してください。

Unit number *Unit-Number* out of range

UNIT指定子の値が不正です。**UNIT**指定子には0から2147483647までの整数値か、**OPEN**文の**NEWUNIT**指定子に返却された値を指定してください。

Unit *Unit-Number* is not connected

指定された装置にファイルが接続されていません。指定された装置に対する入出力文の前にファイルをオープンするように見直してください。

Unit *Unit-Number* is not connected for DIRECT i/o

順番探査またはストリーム探査として接続した外部ファイルに対して、直接探査の入出力文を実行しようとしました。入出力文を順番探査またはストリーム探査の入出力文に訂正するか、ファイルを直接探査として接続してください。

Unit *Unit-Number* is not connected for FORMATTED i/o

書式なしファイルとして接続した外部ファイルに対して、書式付き入出力文を実行しようとしました。入出力文を書式なし入出力文に訂正するか、ファイルを書式付きファイルとして接続してください。

Unit *Unit-Number* is not connected for READ action

入力の実行しか許されていない装置に対して出力を実行しようとしています。装置番号が誤っていれば、それを修正する。そうでないならば、そのファイルから入力を行わないようにするか、**OPEN**文でその装置を入力可で接続してください。

Unit *Unit-Number* is not connected for SEQUENTIAL i/o

直接探査として接続した外部ファイルに対して、順番探査の入出力文を実行しようとしました。入出力文を直接探査の入出力文に訂正するか、ファイルを順番探査として接続してください。

Unit *Unit-Number* is not connected for UNFORMATTED i/o

書式付きァイルとして接続した外部ファイルに対して、書式なし入出力文を実行しようとしました。入出力文を書式付き入出力文に訂正するか、ファイルを書式なしファイルとして接続してください。

Unit *Unit-Number* is not connected for WRITE action

出力の実行しか許されていない装置に対して入力を実行しようとしています。装置番号が誤っていれば、それを修正する。そうでないならば、そのファイルから出力を行わないようにするか、**OPEN**文でその装置を入力可で接続してください。

Unit *Unit-Number* is not connected on OPEN with STATUS='OLD' and no FILE= specifier

OPEN文の**STATUS**指定子にOLDが指定されましたが、**FILE**指定子が指定されていません。**OPEN**文の**STATUS**指定子にOLDを指定するときは**FILE**指定子も一緒に指定してください。そうでないならば、**STATUS**指定子の値を見直してください。

VALUE argument (*value*) to intrinsic ATOMIC_ADD is out of range

組み込み手続**ATOMIC_ADD**のVALUE引数が範囲外です。処理内容を見直してください。

VALUE argument (*value*) to intrinsic ATOMIC_AND is out of range

組み込み手続**ATOMIC_AND**のVALUE引数が範囲外です。処理内容を見直してください。

VALUE argument (*value*) to intrinsic ATOMIC_FETCH_ADD is out of range

組み込み手続**ATOMIC_FETCH_ADD**のVALUE引数が範囲外です。処理内容を見直してください。

VALUE argument (*value*) to intrinsic ATOMIC_FETCH_AND is out of range

組み込み手続**ATOMIC_FETCH_AND**のVALUE引数が範囲外です。処理内容を見直してください。

VALUE argument (*value*) to intrinsic ATOMIC_FETCH_OR is out of range

組み込み手続**ATOMIC_FETCH_OR**のVALUE引数が範囲外です。処理内容を見直してください。

VALUE argument (*value*) to intrinsic ATOMIC_FETCH_XOR is out of range

組み込み手続**ATOMIC_FETCH_XOR**のVALUE引数が範囲外です。処理内容を見直してください。

VALUE argument (*value*) to intrinsic ATOMIC_OR is out of range

組み込み手続**ATOMIC_OR**のVALUE引数が範囲外です。処理内容を見直してください。

VALUE argument (*value*) to intrinsic ATOMIC_XOR is out of range

組み込み手続**ATOMIC_XOR**のVALUE引数が範囲外です。処理内容を見直してください。

Value *value* of KIND argument to OMP_SET_SCHEDULE is out of range 1:4

OMP_SET_SCHEDULEの引数KINDが指定可能な値の範囲外です。1から4の整数を指定してください。

Value *value* of MAX_LEVELS argument to OMP_SET_MAX_ACTIVE_LEVELS is negative

OMP_SET_MAX_ACTIVE_LEVELSに指定した値が負の値です。正の整数を指定してく

ださい。

Value *value* of NUM_THREADS argument to OMP_SET_NUM_THREADS is greater than maximum *num*

OMP_SET_NUM_THREADSに指定した値が最大値を超えています。範囲内の値を指定してください。

Value *value* of NUM_THREADS argument to OMP_SET_NUM_THREADS is not positive

OMP_SET_NUM_THREADSに指定した値が正の値ではありません。正の整数を指定してください。

***var* has not been assigned a branch target label**

*var*に文番号が指定されていません。プログラムを見直してください。

Vector subscript for rank *rank* of *name* has extent *value* instead of *value*

*rank*次元のベクトル添字のサイズが違います。添字の値を見直してください。

WRITE operation failed on unit Unit-Number: Disk quota exceeded

この装置番号を指定した**WRITE**文がディスクのクォータ制限のため、書き込みに失敗しました。ファイルシステムのクォータ制限を確認してください。

Zero repeat factor in list-directed input

並び入力で指定されたR*C形式のデータの反復数がゼロです。反復数を1以上に変更してください。

Zero stride value for subscript *num* of *name*

*name*の添字*num*の刻み幅にゼロが指定されています。ゼロは指定しないでください。

12.3 その他の実行時メッセージ

Compatibility Error: veos (older than v2.6.0) and ve_exec (vVEOS-verision) are not compatible

veosが古くve_execと互換がない。コンテナ上でVEプログラムを実行している場合は、ホストマシンに入っているveosのバージョンが古い可能性がある。最新のveosパッケージをホストマシンにインストールしてください。

Compatibility Error: veos (vVEOS-version-A) and ve_exec (vVEOS-verision-B) are not compatible

veosが古くve_execと互換がない。コンテナ上でVEプログラムを実行している場合は、

ホストマシンに入っているveosのバージョンが古い可能性がある。最新のveosパッケージをホストマシンにインストールしてください。

Failed to load EXEC DATA (fixed): *Error Message*

実行ファイルのデータ領域の読み込みに失敗した。VEメモリ不足の可能性がある。実行中の他のVEプロセスがあればそれを終了させるか、データ領域のサイズを減らしてください。なお、VE搭載メモリ量と現在のVEメモリ使用量は/opt/nec/ve/bin/free -hで取得できる。

Failed to load EXEC DATA (fixed, fileback): Error Message

実行ファイルのデータ領域の読み込みに失敗した。VEメモリ不足の可能性がある。実行中の他のVEプロセスがあればそれを終了させるか、データ領域のサイズを減らしてください。なお、VE搭載メモリ量と現在のVEメモリ使用量は/opt/nec/ve/bin/free -hで取得できる。

Unable to grow stack

スタックのサイズが足りない。環境変数**VE_LIMIT_OPT**を利用して、次の例のように利用可能なスタックのサイズ上限を増やしてください。

```
export VE_LIMIT_OPT="-s 8192"
```

現在のスタックサイズの上限は、ve_execコマンドの--show-limitで確認できる。

```
$ ve_exec --show-limit
core file size          (blocks, -c) 0                   0
data seg size           (kbytes, -d) unlimited           unlimited
pending signals                 (-i) 379523              379523
max memory size         (kbytes, -m) unlimited           unlimited
stack size              (kbytes, -s) unlimited           unlimited  <--
cpu time               (seconds, -t) unlimited           unlimited
virtual memory          (kbytes, -v) unlimited           unlimited
```

VE Node node-number is UNAVAILABLE

*node-number*のVEカードに障害が発生した。他のVEノードを利用して、ジョブを実行してください。

第13章　トラブルシューティング

13.1 プログラムのコンパイルに関するトラブルシューティング

“Fatal: License: Unknown host.” というコンパイルエラーが発生する。

コンパイラのライセンスチェック時にライセンスサーバーにアクセスできない問題が発生している可能性があります。以下のHPCソフトウェアライセンス発行のページに記載されているFAQを参照してください。解決しなければ、同ページよりお問い合わせください。
https://www.hpc-license.nec.com/aurora/

“Invalid #line directive” というコンパイルエラーが発生する。

#if、#includeなどのプリプロセッサディレクティブが使用されています。-fppを指定してコンパイルしてください。

“Cannot find module : ～“というコンパイルエラーが発生する。

モジュールが使用されていますが、そのモジュールファイル(*.mod)が見つかりません。コンパイラがモジュールファイルをサーチするディレクトリにモジュールファイルが存在するかを確認してください。コンパイラがモジュールファイルをサーチするディレクトリについては「1.6 モジュールファイルのサーチ」を参照してください。

“not a valid module information file” というコンパイルエラーが発生する。

モジュールファイルをコンパイルしたコンパイラが古いか、モジュールファイルが壊れている可能性があります。モジュールファイル(*.mod)を再作成してください。

指示行に対して、”Syntax error”というコンパイルエラーが発生する。

指示行の綴り、使い方が間違っていないかを確認してください。SXシリーズ向けコンパイラの指示行に対してエラーとなっているときは、コンパイラ指示行変換ツールなどでVEコンパイラの指示行に変換してください。コンパイラ指示行変換ツールについては「付録 C コンパイラ指示行変換ツール」を参照してください。

”Error: Invalid suffix”というアセンブルエラーが発生する。

binutils-veパッケージが古い可能性があります。binutils-veパッケージが最新版であるか確認してください。

モジュールファイル、ヘッダファイル、ライブラリを利用するときにコンパイラ、リンカが参照するディレクトリを確認したい。

「1.6 モジュールファイルのサーチ」、「1.7 INCLUDE行、および、#includeで取り込まれるファイルのサーチ」、「1.8 ライブラリのサーチ」、を参照してください。

”undefined reference to `ftrace_region_begin_' / `ftrace_region_end_'”というリンクエラーが発生する。

FTRACE 機能が使用されています。**-ftrace**を指定してリンクしてください。FTRACE 機能については「PROGINF/FTRACE ユーザーズガイド」を参照してください。

```
$ nfort a.o b.o -ftrace
```

”undefined reference to '__vthr$_barrier'”というリンクエラーが発生する。

-mparallel、または、**-fopenmp**を指定してリンクしてください。

”undefined reference to '__vthr$_pcall_va”というリンクエラーが発生する。

-mparallel、または、**-fopenmp**を指定してリンクしてください。

”cannot find -lveproginf”、および、”cannot find -lveperfcnt”というリンクエラーが発生する。

nec-veperfパッケージがインストールされていません。nec-veperfパッケージをインストールしてください。

コードサイズの大きいプログラムをコンパイルしたときにSIGSEGVでアボートする。

コンパイラが必要とするスタック領域の容量が設定の上限を超えている可能性があります。ulimitコマンドでスタックサイズの上限を拡大することで解消することがあります。以下のように「ulimit –s」でスタックサイズの上限値を確認できます。「ulimit –s (スタックサイズの上限値)」で上限値を変更できますので、「ulimit -s」で出力された上限値よりも大きな値を設定し、再度コンパイルしてください。

```
$ ulimit -s              (値の確認)
8192
```

```
$ ulimit -s 16384        （値の変更）
```

コンパイル時にSIGKILLが発生する。

コンパイルを行ったマシンのメモリが不足している可能性があります。**-O0**、**-O1**により最適化レベルを落とすことでメモリ使用量をある程度少なくすることができます。

VE向けの実行ファイルであることを確認したい。

「/opt/nec/ve/bin/nreadelf -h」に実行ファイルを指定して実行してください。「Machine:」の行に「NEC VE architecture」と出力されていれば、VE向けの実行ファイルであることを示します。

```
$ /opt/nec/ve/bin/nreadelf -h a.out
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           NEC VE architecture
  (...)
```

自動並列とOpenMP並列を混ぜてリンクするとき、-fopenmp、-mparallelのどちらを指定すればよいか。

-fopenmp、**-mparallel**のどちらか一方を指定してリンクしてください。

```
$ nfort -c -mparallel a.f90
$ nfort -c -fopenmp b.f90
$ nfort -fopenmp a.o b.o
```

-fcheckを指定すると、コンパイル時間が異常に長くなる。

コンパイル時に実行時のチェックコードが埋め込めまれるため長くなります。実行時のチェックが必要なルーチンを含むソースファイルのコンパイル時にのみ**-fcheck**を指定して

ください。

-fcheckを指定すると、実行時間が異常に長くなる。

実行時にチェックコードが実行されるため長くなります。実行時のチェックが必要なルーチンを含むソースファイルのコンパイル時にのみ**-fcheck**を指定してください。

-ftraceを指定すると、実行時間が異常に長くなる。

性能情報を取得するルーチンが実行されるため長くなります。このルーチンは手続の入口、出口、ユーザ指定リージョンの前後で呼び出されます。

性能情報を取得したい手続を含むソースファイルのみ**-ftrace**を指定してください。

OMP_NUM_THREADSで8より大きい値を指定しても、その数のスレッドが生成されない。

VEのコア数は8個であるため、8スレッドが上限です。

定義済みマクロの名前、値を知りたい。

「9.2.4 定義済みマクロ」を参照してください。

Fortranプログラムをプリプロセスしたい。

-fppを指定してコンパイルしてください。

FortranプログラムとC/C++プログラムのオブジェクトファイルを混在リンクしたい。

「10.6 プログラムのリンク」を参照してください。

SXシリーズで利用していたプログラムのオプションをVector Engine向けに変更したい。

「付録 B SXシリーズ向けコンパイラとの対応」を参照し、SXのコンパイラオプションに対応するVEのコンパイラオプションに変更してください。

SXシリーズで利用していたプログラムの指示行をVector Engine向けに変更したい。

SXのコンパイラ指示行をVEのコンパイラ指示行に変換するツールが利用できます。

「付録 C コンパイラ指示行変換ツール」を参照してください。

または、「付録 B SXシリーズ向けコンパイラとの対応」を参照し、SXのコンパイラ指示行に対応するVEのコンパイラ指示行に変更してください。

診断メッセージに、名前$1など、'$'の後に数字のある変数、ルーチン名が表示される。これらは何か？

ベクトル化、並列化のためにコンパイラが作成した変数、ルーチンです。

診断メッセージに、変数名ではなくDOUBLE、floatなどの型名のみが表示されることがある。これらは何か？

ベクトル化、並列化のためにコンパイラが作成した名前なし変数で、名前の代わりに型名を表示しています。

Internal error detected -- please report.というメッセージが表示された。

上記エラーが発生し、コンパイルが中断されなかった場合、コンパイラはエラーをリカバリし、コンパイルを継続しています。作成されたオブジェクトファイルはそのままご利用できます。

コンパイルが中断された場合、NECサポートポータルよりお問い合わせください。

ループ中にALLOCATE文、DEALLOCATE文がないのに次のメッセージが出力される。

```
vec(181): Allocation obstructs vectorization.
vec(182): Deallocation obstructs vectorization.
```

Fortranにおいて、言語仕様を実現するため、コンパイラが実行時に領域を暗黙的に確保、解放しなければならなかったとき、このメッセージが出力されます。特に、手続をインライン展開した際に引数、返却値の受け取りで発生することがあります。

-bssと-saveの違いについて知りたい。

SAVE属性をもつ変数の値は、前回当該ルーチンが呼び出され、リターンしたときの値が、本呼び出し時の最初の値となりますが、**-bss**の場合はそれが保証されません。

コンパイル時に指定した覚えのないコンパイラオプションが有効になっている。

コンパイラオプションがオプションファイルに指定されている可能性があります。オプションファイルの詳細については「1.5 コンパイラオプションの指定」を参照してください。

コンパイラのバージョンを確認したい。

--versionを指定してください。

-fpie、-fPIEオプションで位置独立実行形式の実行ファイルを作成したい。

位置独立実行形式の実行ファイルの作成はサポートしていません。

“Too many elements in array” というコンパイルエラーが発生する。

ALLOCATE文で確保しようとしている配列のサイズ、または、配列の宣言で確保しようとしている配列のサイズが1TiBを超えています。配列のサイズを見直してください。なお、コンパイル時に配列サイズの上限を1TiBでチェックしていますが、VEのメモリサイズは48 GBであるため、それより大きいサイズの配列を確保しようとすると、実行時エラー(Out of memory)となりますのでご注意ください。

モジュールソースファイルをコンパイルしたとき、.Lファイルが生成されない

モジュール手続を含まないモジュールソースファイルに対して.Lファイルが生成されないのは、仕様どおりです。

13.2 プログラムの実行に関するトラブルシューティング

実行時に“Node 'N' is Offline”というエラーが発生する。

ノード番号NのVEがOFFLINE状態になっています。VEをONLINE状態にしてください。

以下は0番のVEをONLINE状態にするときの例です。

```
# /opt/nec/ve/bin/vecmd -N 0 state set on
...
Result: Success
# /opt/nec/ve/bin/vecmd state get
...
------------------------------------------------------------------
VE0 [03:00.0] [ ONLINE ] Last Modif:2017/11/29 10:18:00
------------------------------------------------------------------
Result: Success
```

実行時に使用されているノードを確認したい。

/opt/nec/ve/bin/psを実行してください。psコマンドを実行するとVEのノードごとに現在実行されているプロセスのスナップショットを出力します。以下の例では、2番のVEノードでa.outというプログラムが実行中であることが確認できます。

```
/opt/nec/ve/bin/ps -a
VE Node: 3
  PID TTY          TIME CMD
VE Node: 1
  PID TTY          TIME CMD
VE Node: 2
  PID TTY          TIME CMD
50727 pts/1    00:01:36 a.out
VE Node: 0
  PID TTY          TIME CMD
```

実行時に"./a.out: error while loading shared libraries: libnfort.so.2: cannot open shared object file: No such file or directory"というエラーが出力される。

実行環境にパッケージnec-nfort-shared、nec-nfort-shared-instをインストールしてください。手順は「インストレーションガイド」を参照してください。

実行時にダイナミックリンクライブラリが見つからないというエラーが発生する。

共有ライブラリを配置しているディレクトリを環境変数**VE_LD_LIBRARY_PATH**に設定してください。環境変数**VE_LD_LIBRARY_PATH**については「2.2 実行時に参照される環境変数」を参照してください。

例外発生時に例外発生個所がソースファイルの何行目に対応しているか確認したい。

トレースバック情報から調べることができます。手順は「1.9.3トレースバック機能との連携」を参照してください。

例外発生時にトレースバック機能で表示される例外発生個所が正しくない。

HWによる命令の先行制御によってトレースバック機能で出力される例外発生個所が正しく表示されないことがあります。環境変数**VE_ADVANCEOFF**=YESを設定することで先行制御を無効にできます。先行制御を無効にすることで実行時間が大幅に長くなることが

ありますのでご注意ください。

```
$ export VE_ADVANCEOFF=YES
```

例外発生時にそれまでのバッファに書き込んだデバッグWRITE結果を出力するようにしたい。

WRITE文の実行後に**FLUSH**文を呼び出してください。

```
SUBROUTINE SUB()
  INTEGER :: U, X

  OPEN(NEWUNIT=U, FILE='debug.log', STATUS='replace')

  CALL SUB1(X)

#ifdef DEBUG
  WRITE(U, *) 'X=', X
  FLUSH(U)
#endif

END
```

未初期化変数を使用していないか確認したい。

倍精度浮動小数点型の未初期化変数については**-minit-stack**=snanを指定してコンパイル、環境変数**VE_INIT_HEAP**=SNANを指定して実行し、例外を検出することで確認できることがあります。単精度浮動小数点型の未初期化変数について確認するときは、snan、SNANの代わりにsnanf、SNANFをそれぞれ指定します。変数が浮動小数点型でないとき、この方法で確認することはできません。

未初期化変数の不定値を参照して、プログラムが異常終了するのを回避したい。

-minit-stack=zeroを指定してコンパイル、環境変数**VE_INIT_HEAP**=ZEROを指定して実行してください。未初期化の領域をゼロクリアすることで異常終了を回避できることがあります。潜在的な問題を解決するためにもプログラムの修正を推奨します。

自動並列化、OpenMP機能を使用したプログラムを実行したとき、"Unable to grow stack"、または、SIGSEGVでプログラムが異常終了する。

スタックの使用量が上限を超えている可能性があります。以下の方法でスタックの上限を拡張するか、スタックの使用量を削減してください。

- 環境変数**OMP_STACKSIZE**によって各スレッドが使用するスタックサイズの上限を拡張することができます。

```
export OMP_STACKSIZE=2G
```

- **-mno-stack-arrays**を指定することでスタックの使用量を削減することができます。ただし、**-mno-stack-arrays**を指定したとき実行時間が増加することがあります。

プログラムが実行時に何スレッドで動作したかを確認したい。

PROGINFのMax Active Threadsを参照してください。Max Active Threadsは環境変数**VE_PROGINF**にDETAILが設定されているとき、プログラムの実行終了時に標準エラー出力に出力されます。詳細は「PROGINF/FTRACE ユーザーズガイド」を参照してください。

以下の例では、Max Active Threadsが4であるため、4スレッドで動作したことが確認できます。

```
******** Program Information ********
(...)
Power Throttling (sec)          :        0.000000
Thermal Throttling (sec)        :        0.000000
Max Active Threads          :        4
Available CPU Cores         :        8
Average CPU Cores Used          :        3.323850
Memory Size Used (MB)           :        7884.000000
Start Time (date)               :        Mon Feb 19 04:43:34 2018 JST
End Time (date)             :        Mon Feb 19 04:44:08 2018 JST
```

自動並列化、OpenMP機能を使用したプログラムを実行したとき、スレッドがいつ生成、解放されるのか知りたい。

既定の動作では、環境変数**OMP_NUM_THREADS**、または、**VE_OMP_NUM_THREADS**によりスレッド数が指定されたときその個数、そうでないときプログラムで利用可能

なVEコア数と同じ個数のスレッドが、プログラムの実行開始前に生成され、終了時に解放されます。
詳細については、「7.3.2 スレッドの生成・解放」を参照してください。

13.3 プログラムのチューニングに関するトラブルシューティング

プログラムにどの最適化が適用されたかを確認したい

コンパイル時に出力される診断メッセージ、および、編集リストを参照してください。診断メッセージリストは**-report-diagnostics**、編集リストは**-report-format**を指定したとき出力されます。
詳細は、「第8章 コンパイルリスト」を参照してください。

ベクトル化を促進したにもかかわらず、性能が低下する。

ベクトル化対象のループの繰り返し数が少ない場合、ベクトル化のためのオーバーヘッドにより性能が低下する場合があります。このようなループは**novector**指示行で自動ベクトル化を抑止してください。

自動並列化、OpenMP機能を使用したプログラムを実行したとき、PROGINFとFTRACEの同項目で表示される値が異なる。

主プログラムの実行開始前に自動生成されるスレッドのスピンウェイト分の演算数等はPROGINFでは加算されますが、FTRACEでは加算されません。

!$omp parallel num_threads(4)を指定して、環境変数OMP_NUM_THREADS=4、OMP_NUM_THREADS=5でそれぞれ実行した場合、OMP_NUM_THREADS=5のほうが並列数が多いにもかかわらず、実行時間が長くなる。

num_threads句で渡される値が、環境変数OMP_NUM_THREADSで指定している値と違う場合には、スレッドの再生成による実行時間の増加が発生します。
主プログラムの実行開始前にスレッドが自動生成されます。この時生成されるスレッド数は、環境変数OMP_NUM_THREADSの値で決まります。プログラム中で、OpenMPのomp_set_thread_num()関数やnum_threads句の指定により、スレッド数が変更される場合、実行開始時に自動生成されたスレッドは解放され、新たにスレッドが再生成されます。

13.4 インストールに関するトラブルシューティング

正しくインストールできているかを確認したい

--versionを指定し、バージョンを確認します。表示されたバージョン番号がインストールした物件と同じであれば正しくインストールできています。以下のX.X.Xにバージョン番号が表示されます。

```
$ /opt/nec/ve/bin/nfort --version

nfort (NFORT) X.X.X (Build 14:10:47 Apr 23 2020)

Copyright (C) 2018,2020 NEC Corporation.
```

過去のバージョンのコンパイラをインストールしたい。

過去のバージョンのコンパイラのインストール手順は、SX-Aurora TSUBASA インストレーションガイドの「A.1.1 特定バージョンのコンパイラのインストール」を参照してください。

過去のバージョンのコンパイラを利用したい。

コンパイル時に /opt/nec/ve/bin/nfort-X.X.X、ncc-X.X.X、または、nc++-X.X.X(X.X.Xはコンパイラのバージョン番号)を起動してください。

詳細については「1.2 コンパイラの起動」を参照してください。

過去のバージョンのコンパイラをデフォルトにしたい。

各バージョンのncc/nc++/nfortコマンドの実体は以下のようにインストールされています。ここでは、X.X.Xはバージョン番号です。

```
/opt/nec/ve/ncc/X.X.X/bin/ncc
/opt/nec/ve/ncc/X.X.X/bin/nc++
/opt/nec/ve/nfort/X.X.X/bin/nfort
```

デフォルトにしたいバージョンの**bin**ディレクトリをコマンドサーチパス(環境変数**PATH**)に設定してください。

13.5 旧SXコンパイラからの移行に関するトラブルシューティング

数値記憶単位「-ew」を指定している。

以下について、プログラムの該当する記述がないか確認し修正してください。

(1) 個別名の組込み関数を利用しているときは、倍精度版か総称名に修正してください。

(2) プログラム中の型宣言、定数を以下に示す例のように修正してください。

Fortran90/SXコンパイラ	Vector Engineコンパイラ
INTEGER*2	INTEGER*8
INTEGER*4	INTEGER*8
INTEGER(KIND=2)	INTEGER(KIND=8)
INTEGER(KIND=4)	INTEGER(KIND=8)
LOGICAL*1	LOGICAL*8
LOGICAL*4	LOGICAL*8
LOGICAL(KIND=1)	LOGICAL(KIND=8)
LOGICAL(KIND=4)	LOGICAL(KIND=8)
REAL*4	REAL*8
REAL(KIND=4)	REAL(KIND=8)
COMPLEX*8	COMPLEX*16
COMPLEX(KIND=4)	COMPLEX(KIND=8)
定数1.23E1	定数1.23D1
定数1.23_4	定数1.23_8

(3) 種別指定のない型での宣言を種別指定ありにプログラム修正してください。コンパイル時にコンパイラオプション**-fdefault-real**=8と**-fdefault-integer**=8を指定することでも対処することができます。

精度拡張「-A dbl」を指定している。

以下について、プログラムの該当する記述がないか確認し修正してください。

(1) プログラム中の型宣言、定数を以下に示す例のように修正してください。

Fortran90/SXコンパイラ	Vector Engineコンパイラ
REAL*4	REAL*8
REAL*8	REAL*16
REAL(KIND=4)	REAL(KIND=8)
REAL(KIND=8)	REAL(KIND=16)
COMPLEX*8	COMPLEX*16
COMPLEX*16	COMPLEX*32
COMPLEX(KIND=4)	COMPLEX(KIND=8)
COMPLEX(KIND=8)	COMPLEX(KIND=16)
定数1.23E1	定数1.23D1
定数1.23D1	定数1.23Q1
定数1.23_4	定数1.23_8
定数1.23_8	定数1.23_16

(2)　種別指定のない型での宣言を種別指定ありにプログラム修正してください。コンパイル時にコンパイラオプション**-fdefault-real**=8と**-fdefault-double**=16を指定することでも対処することができます。

精度拡張「-A dbl4」を指定している。

以下について、プログラムの該当する記述がないか確認し修正してください。

(1) プログラム中の型宣言、定数を以下に示す例のように修正します。

Fortran90/SXコンパイラ	Vector Engineコンパイラ
REAL*4	REAL*8
REAL(KIND=4)	REAL(KIND=8)
COMPLEX*8	COMPLEX*16
COMPLEX(KIND=4)	COMPLEX(KIND=8)
定数1.23E1	定数1.23D1
定数1.23_4	定数1.23_8

(2)　種別指定のない型での宣言を種別指定ありにプログラム修正してください。コンパ

イル時にコンパイラオプション**-fdefault-real**=8を指定することでも対処することができます。

精度拡張「-A dbl8」を指定している。

以下について、プログラムの該当する記述がないか確認し修正してください。

(1) プログラム中の型宣言、定数を以下に示す例のように修正します。

Fortran90/SXコンパイラ	Vector Engineコンパイラ
REAL*8	REAL*16
REAL(KIND=8)	REAL(KIND=16)
COMPLEX*16	COMPLEX*32
COMPLEX(KIND=8)	COMPLEX(KIND=16)
定数1.23D1	定数1.23Q1
定数1.23_8	定数1.23_16

(2) 種別指定のない型での宣言を種別指定ありにプログラム修正してください。コンパイル時にコンパイラオプション**-fdefault-double**=16を指定することでも対処することができます。

精度指定(F_UFMTADJUST=TYPE2)を指定してバイナリ入力している。

SX-ACEで環境変数**F_UFMTADJUST**を指定して作成したバイナリデータの入力時には、環境変数**VE_FORT_UFMTADJUST**を指定します。

SX-ACEで作成したバイナリデータの入力している。

SX-ACEで作成したバイナリデータの入力時には、環境変数**VE_FORT_UFMTENDIAN**を指定します。

第14章　旧バージョンとの互換に関する注意事項

(1) NEC Fortran コンパイラのバージョン 1.X.X とバージョン 2.0.0 以降 には、互換性がありません。そのため、バージョン 1.X.X で作成したオブジェクトファイルはバージョン 2.0.0 以降で作成したオブジェクトファイルとリンクできません。

(2) バージョン 2.2.0 以降でコンパイラの実行時ルーチンを共有ライブラリでも提供しています。このため、バージョン 2.1.2 以前のコンパイラで作成した共有ライブラリは、2.2.X 以降のコンパイラで再コンパイルしてください。

(3) バージョン 2.2.*X* 以降のコンパイラで作成した実行ファイルを実行するには、バージョン 2.21-4 以降の glibc-ve パッケージに含まれるダイナミックリンカが必要です。実行がうまくいかないとき、glibc-ve パッケージのバージョンをご確認ください。

```
$ rpm -q glibc-ve
glibc-ve-2.21-4.el7.x86_64
```

(4) バージョン 2.2.*X* 以降のコンパイラは既定値で共有ライブラリをリンクするため、バージョン 2.1.2 以前に比べ、ダイナミックリンク処理のためのオーバーヘッドにより実行性能が低下することがあります。性能低下を回避するには、**-static**、または、**-static-nec** を指定して静的ライブラリをリンクしてください。

備考　**-static**、または、**-static-nec**を指定して作成した実行ファイルを実行するとき、結果不正や異常終了などにより正しく実行できないことがあります。

(5) バージョン 3.0.8 以降、NAMELIST 出力のフォーマットが変更されました。バージョン 3.0.7 以前の形式で出力するときは、プログラムの実行時に環境変数 **VE_FORT_NML_REPEAT_FORM** に”NO”を設定してください。

```
$ export VE_FORT_NML_REPEAT_FORM=NO
```

付録 A　コンフィギュレーションファイル

A.1　概要

コンフィギュレーションファイルを使用することで、コンパイラの既定の設定を変更できます。コンフィギュレーションファイルは、**-cf**で指定します。
コンフィギュレーションファイルは、以下の形式で記述します。

```
キーワード:値
```

コンフィギュレーションファイルには以下のキーワードを指定できます。

キーワード	説明
veroot	VEのルートディレクトリを指定します。 (既定値：/opt/nec/ve)
system	コンパイラシステムのルートディレクトリを指定します。 (既定値：/opt/nec/ve/nfort/*バージョン番号*)
as	アセンブラを指定します。 (既定値：<veroot>/bin/nas)
fcom	Fortranコンパイラを指定します。 (既定値：<system>/libexec/fcom)
ld	リンカを指定します。 (既定値：<veroot>/bin/nld)
fpp	Fortranプリプロセッサを指定します。 (既定値：<system>/libexec/fpp)
fc_pre_options **fc_post_options**	コンパイラオプションを指定します。 コンパイラへは以下の順で指定されます。 <fc_pre_options><user-specified-fc-options><fc_post_options>
as_pre_options **as_post_options**	アセンブラオプションを指定します。 アセンブラへは以下の順で指定されます。 <as_pre_options><user-specified-ld-options><as_post_options>
ld_pre_options **ld_post_options**	リンカオプションを指定します。リンカへは以下の順で指定されます。 <ld_pre_options><user-specified-ld-options><ld_post_options>
startfile	スタートアップファイルを指定します。

endfile	スタートアップファイルを指定します。リンカオプションの末尾に指定されます。

A.2　記述方法

- キーワードと値は、コロン(:)で区切ります。
- キーワードを記述しないときは既定値が使用されます。
- 区切り文字のコロンの前後は空白を指定できます。
- 値は複数行に指定でき、空白行、または、次のキーワードまでを値とみなします。複数行にわたり値を指定するときは改行部分に'¥'を指定します。

例

```
fc_pre_options: -I /tmp  ¥
-I /tmp2
```

- 同じキーワードを複数回指定したときは、最後のキーワードが有効になります。

A.3　使用例

コンフィギュレーションファイルの使用例を説明します。

- VEのルートディレクトリとコンパイラシステムのルートディレクトリを変更する。
  verootキーワードとsystemキーワードに変更後の値を指定します。

```
veroot: /foo/ve
system: /foo/ve/nfort/X.X.X
```

- 作成したコンフィギュレーションファイルを**-cf**で指定すると、指定した環境でコンパイルされます。ここでは、コンフィギュレーションファイル名をve.confとします。

```
$ nfort -cf=ve.conf  test.f90
```

- 使用するコンパイラだけを変更する。
  先の例は、コンパイル環境全体を変更しますが、使用するコンパイラだけを変更することもできます。この場合、fcomキーワードに値を指定します。

```
fcom: /foo/ve/nfort/X.X.X/libexec/fcom
```

作成したコンフィギュレーションファイルを**-cf**で指定すると、指定したコンパイラでコンパ

イルされます。アセンブラやリンカなども同様に変更することができます。

付録 B　SX シリーズ向けコンパイラとの対応

本節ではSXシリーズ向けのコンパイラとVector Engine向けコンパイラの主なコンパイラオプション、指示行環境変数の対応について説明します。

B.1　NEC Fortran 2003コンパイラオプション

NEC Fortran 2003のコンパイラオプションとVector Engine向けコンパイラのコンパイラオプションの対応表を示します。Vector Engineコンパイラ列の「なし」は対応するコンパイラオプションがないことを表します。

B.1.1　全体オプション

NEC Fortran 2003コンパイラ	Vector Engineコンパイラ
-Caopt	**-O4**
-Chopt	**-O3**
-Cvopt	**-O2**
-Csopt	**-O2 -mno-vector**
-Cvsafe	**-O1**
-Cssafe	**-O1 -mno-vector**
-Cnoopt	**-O0**
-S	**-S**
-NS	なし
-V (注) バージョン表示後にコンパイル処理を続けます。	**--version** (注) コンパイル処理を行わずにバージョン表示のみ行います。
-NV	なし
-c	**-c**
-Nc	なし

-cf ***文字列***	**-cf=*文字列***
-clear	**-clear**
-mod | -Nmod	なし
-o ***ファイル名***	**-o** ***ファイル名***
-size_t32	なし
-size_t64	なし (注) 常に**-size_t64**相当で動作します。
-syntax	**-fsyntax-only**
-Nsyntax	**-fno-syntax-only**
-tm ***ディレクトリ名***	なし
-to ***ディレクトリ名***	なし
-verbose	**-v**
-Nverbose	なし

B.1.2 最適化/ベクトル化オプション

NEC Fortran 2003コンパイラ	Vector Engineコンパイラ
-Ochg	**-fassociative-math** または、 **-faggressive-associative-math** (注) **-faggressive-associative-math**は、**-fassociative-math**より激しく最適化します。
-Onochg	**-fno-associative-math**
-Odiv	**-freciprocal-math**
-Onodiv	**-fno-reciprocal-math**
-Oextendreorder	**-msched-interblock**
-Onoextendreorder	なし

-Oignore_volatile	**-fignore-volatile**
-Onoignore_volatile	**-fno-ignore-volatile**
-Oiodo	**-marray-io**
-Onoiodo	**-mno-array-io**
-Omove	**-fmove-loop-invariants-unsafe**
-Onomovediv	**-fmove-loop-invariants**
-Onomove	**-fno-move-loop-invariants**
-Ooverlap	**-fnamed-alias**
-Onooverlap	**-fnamed-noalias**
-Oreorderrange=bblock	**-msched-insns**
-Ounroll	**-floop-unroll**
-Ounroll=*n*	**-floop-unroll** **-floop-unroll-max-times**=*n* (注) 2つのオプションを同時に指定します。
-Onounroll	**-fno-loop-unroll**
-dir { vec | novec }	なし
-ipa	**-fipa**
-Nipa	**-fno-ipa**
-math { errchk | noerrchk }	なし
-math { inline | noinline }	なし
-pvctl,altcode	**-mvector-dependency-test** **-mvector-loop-count-test** **-mvector-shortloop-reduction** (注) 3つのオプションを同時に指定します。
-pvctl,altcode=dep	**-mvector-dependency-test**
-pvctl,altcode=nodep	**-mno-vector-dependency-test**

-pvctl,altcode=loopcnt	**-mvector-loop-count-test**
-pvctl,altcode=noloopcnt	**-mno-vector-loop-count-test**
-pvctl,altcode=shortloop	**-mvector-shortloop-reduction**
-pvctl,altcode=noshortloop	**-mno-vector-shortloop-reduction**
-pvctl,noaltcode	**-mno-vecgtor-depencendy-test** **-mno-vector-loop-count-test** **-mno-vector-shortloop-reduction** (注) 3つのオプションを同時に指定します。
-pvctl,assoc	**-fassociative-math**
-pvctl,noassoc	**-fno-associative-math**
-pvctl { assume | noassume }	なし
-pvctl,chgpwr	**-mvector-power-to-explog** **-mvector-power-to-sqrt** (注) 2つのオプションを同時に指定します。
-pvctl,collapse	**-floop-collapse**
-pvctl,nocollapse	**-fno-loop-collapse**
-pvctl { compress | nocompress }	なし
-pvctl,cond_mem_opt	**-mvector-merge-conditional**
-pvctl,nocond_mem_opt	**-mno-vector-merge-conditional**
-pvctl { conflict | noconflict }	なし
-pvctl,divloop	なし
-pvctl,nodivloop	**-mwork-vector-kind**=none
-pvctl,expand=*n*	**-floop-unroll-complete**=*n*
-pvctl,noexpand	**-fno-loop-unroll-complete**
-pvctl,listvec	**-mlist-vector**
-pvctl,nolistvec	**-mno-list-vector**

-pvctl,loopchg	**-floop-interchange**
-pvctl,noloopchg	**-fno-loop-interchange**
-pvctl,loopcnt=*n*	**-floop-count**=*n*
-pvctl { lstval | nolstval }	なし
-pvctl,matmul	**-fmatrix-multiply**
-pvctl,nomatmul	**-fno-matrix-multiply**
-pvctl { neighbors | noneighbors }	なし
-pvctl,nodep	**-fivdep**
-pvctl,on_adb[=*カテゴリ***]**	なし
-pvctl,outerunroll=*n*	**-fouterloop-unroll** **-fouterloop-unroll-max-times**=*n* (注) 2つのオプションを同時に指定します。
-pvctl,outerunroll_lim=*n*	なし
-pvctl,split	**-floop-split**
-pvctl,nosplit	**-fno-loop-split**
-pvctl { vchg | novchg }	なし
-pvctl,vecthreshold=*n*	**-mvector-threshold**=*n*
-pvctl,verrchk	**-mvector-intrinsic-check**
-pvctl,noverrchk	**-mno-vector-intrinsic-check**
-pvctl { vlchk | novlchk }	なし
-pvctl,vwork={ static | stack | hybrid }	なし
-pvctl,vworksz=*n*	なし
-salloc	**-mstack-arrays**
-Nsalloc	**-mno-stack-arrays**
-v	**-mvector**

-Nv	**-mno-vector**
-xint	**-mno-vector-iteration**
-Nxint	**-mvector-iteration**

B.1.3 インライン展開オプション

NEC Fortran 2003コンパイラ	Vector Engineコンパイラ
-dir { inline | noinline }	なし
-pi,auto	**-finline-functions**
-pi,max_depth=*n*	**-finline-max-depth=*n***
-pi,max_size=*n*	**-finline-max-function-size=*n***
-pi,proc_size=*n*	なし
-pi,times=*n*	**-finline-max-times=*n***

B.1.4 並列化オプション

NEC Fortran 2003コンパイラ	Vector Engineコンパイラ
-dir { par | nopar }	なし
-Pauto	**-mparallel**
-Pmulti	なし
-Popenmp	**-fopenmp**
-Pstack	なし
-Pstatic	**-bss**
-pvctl,for[=*n*]	なし (注) 並列化のスケジュールは、**-mschedule-static**などで制御できます。「3.2 最適化・ベクトル化オプション」を参照してください。

-pvctl,by=*n*	なし (注) 並列化のスケジュールは、**-mschedule-static**などで制御できます。「3.2 最適化・ベクトル化オプション」を参照してください。
-pvctl,inner	**-mparallel-innerloop**
-pvctl,noinner	**-mno-parallel-innerloop**
-pvctl,outerstrip	**-mparallel-outerloop-strip-mine**
-pvctl,noouterstrip	**-mno-parallel-outerloop-strip-mine**
-pvctl,parcase	**-mparallel-sections**
-pvctl,noparcase	**-mno-parallel-sections**
-pvctl,parthreshold=*n*	**-mparallel-threshold**=*n*
-pvctl,noparthreshold	**-mno-parallel-threshold**
-pvctl,res={ whole | parunit | no }	なし
-reserve *n*	なし

B.1.5 コード生成オプション

NEC Fortran 2003コンパイラ	Vector Engineコンパイラ
-adv { on | off }	なし
-Nadv	なし
-mask { flovf | flunf | fxovf | inv | inexact | zdiv }	なし (注) 実行時の環境変数VE_FPE_ENABLEで制御できます。「1.9 **演算例外**」を参照してください。
-mask { setall | nosetall | setmain }	なし
-prec_complex_division	なし
-Nprec_complex_division	なし

-stkchk | -Nsckchk	なし

B.1.6　言語仕様制御オプション

NEC Fortran 2003コンパイラ	Vector Engineコンパイラ
-defacto_associated	なし
-Ndefacto_associated	なし
-default_double_size	**-fdefault-double**=*n*
-default_real_size	**-fdefault-real**=*n*
-default_integer_size	**-fdefault-integer**=*n*
-extend_source	**-fextend-source**
-fixed	**-ffixed-form**
-free	**-ffree-form**
-f2003 **-f2008** **-f95**	**-std**={ f2003 | f2008 | f95 }
-ignore_directive	なし
-Nignore_directive	なし
-small_integer | -Nsmall_integer	なし

B.1.7　性能測定オプション

NEC Fortran 2003コンパイラ	Vector Engineコンパイラ
-acct	**-proginf**
-Nacct	**-no-proginf**
-ftrace	**-ftrace**
-Nftrace	**-no-ftrace**
-p	**-p**

-Np	なし

B.1.8 デバッグオプション

NEC Fortran 2003コンパイラ	Vector Engineコンパイラ
-check	**-fcheck**=*keyword*
-init stack={ zero | nan | 0x*XXXX* **}**	**-minit-stack={ zero | snan | snanf |** 0x*XXXX* **}**
-mtrace [basic]	**-mmemory-trace**
-mtrace full	**-mmemory-trace-full**
B.1.9　-Nmtrace	**B.1.10**　なし
-traceback	**-traceback**
-Ntraceback	なし

B.1.11プリプロセッサオプション

NEC Fortran 2003コンパイラ	Vector Engineコンパイラ
-D*name*[=*def*]	**-D*name*[=*def*]**
-E	**-E**
-EP	なし
-Ep	**-fpp**
-NE	**-nofpp**
-H	なし
-I *ディレクトリ名*	**-I** *ディレクトリ名*
-M	**-M**
-U*name*	**-U*name***
-Wp,"*オプション列***"**	**-Wp,"***オプション列***"**

-ts *ディレクトリ名*	なし

B.1.12 リスト出力オプション

NEC Fortran 2003コンパイラ	Vector Engineコンパイラ
-Rappend	**-report-append-mode**
-Rnoappend	なし
-Rdiaglist	**-report-diagnostics**
-Rnodiaglist	なし
-Rfile={*ファイル名* **| stdout }**	**-report-file={***ファイル名* **| stdout }**
-Rfmtlist	**-report-format**
-Rnofmtlist	なし
-Robjlist	**-assembly-list**
-Rnoobjlist	なし
-R { summary | nosummary }	なし
-R { transform | notransform }	なし

B.1.13 メッセージオプション

NEC Fortran 2003コンパイラ	Vector Engineコンパイラ
-O { fullmsg | infomsg | nomsg }	なし
-pi { fullmsg | infomsg | nomsg }	**-fdiag-inline**={ 2 | 1 | 0 }
-pvctl { fullmsg | infomsg | nomsg }	**-fdiag-parallel**={ 2 | 1 | 0 } **-fdiag-vector**={ 2 | 1 | 0 }
-w all	**-Wall**
-w none	**-w**
-w { info | noinfo }	なし

-w extension	**-Wextension**
-w noextension	**-Wno-extension**
-w { observe | noobserve }	なし
-w obsolescent	**-Wobsolescent**
-w noobsolescent	**-Wno-obsolescent**
-w { unreffed | nounreffed }	なし
-w {unused | nounused }	なし

B.1.14アセンブラオプション

NEC Fortran 2003コンパイラ	Vector Engineコンパイラ
-Wa,"*オプション列***"**	**-Wa,"***オプション列***"**

B.1.15Cコンパイラオプション

NEC Fortran 2003コンパイラ	Vector Engineコンパイラ
-Wc,"*オプション列***"**	なし

B.1.16リンカオプション

NEC Fortran 2003コンパイラ	Vector Engineコンパイラ
-L*ディレクトリ名*	**-L***ディレクトリ名*
-l*ライブラリ名*	**-l***ライブラリ名*
-Wl,"*オプション列***"**	**-Wl,"***オプション列***"**

B.1.17ディレクトリオプション

NEC Fortran 2003コンパイラ	Vector Engineコンパイラ
-YI,*ディレクトリ名*	なし
-YL,*ディレクトリ名*	なし

-YM,*ディレクトリ名*	なし
-YS,*ディレクトリ名*	なし
-Ya,*ディレクトリ名*	なし
-Yf,*ディレクトリ名*	なし
-Yl,*ディレクトリ名*	なし
-Yp,*ディレクトリ名*	なし

B.2　Fortran90/SXコンパイラオプション

Fortran90/SX コンパイラオプションと Vector Engine 向けコンパイラのコンパイラオプションの対応表を示します。Vector Engine コンパイラ列の「なし」は対応するコンパイラオプションがないことを表します。

B.2.1　f90/sxf90コマンド基本オプション

Fortran90/SXコンパイラ	Vector Engineコンパイラ
-Chopt	**-O3**
-Cvopt	**-O2**
-Csopt	**-O2 –mno-vector**
-Cvsafe	**-O1**
-Cssafe	**-O1 –mno-vector**
-Cdebug	**-O0 -g**
-c	**-c**
-Nc	なし
-cf *文字列*	**-cf=***文字列*
-clear	**-clear**
-D*name*[=*def*]	**-D*name*[=*def*]**

-da	なし
-dC	**-fcheck=none**
-dD	なし
-dP	なし
-dR	**-fcheck=none**
-dW	なし (注) 常に**-dW**相当で動作します。
-dw	なし (注) 常に**-dw**相当で動作します。
-ea	なし
-eC	**-fbounds-check** または、**-fcheck=bounds**
-eD	なし
-eP	なし
-eR	**-fbounds-check** または、**-fcheck=bounds** (注) 配列添字の範囲のみチェックされます。
-eW	なし
-ew	なし (注) 移行の詳細については「13.5 旧SXコンパイラからの移行に関するトラブルシューティング」を参照してください。
-EP	なし
-Ep	**-fpp**
-NE	**-nofpp**
-f2003	なし

	(注) Fortran 2003機能は常に有効です。
-f2003 { cbind | nocbind }	なし
-f2003 { cptr_derive | cptr_i8 }	なし
-f2003 { opt_ieee | noopt_ieee }	なし
-Nf2003	なし
-f0	**-ffixed-form**
-f3	**-ffixed-form –fextend-source**
-f4	**-ffree-form**
-f5	**-ffree-form –fextend-source**
-ftrace	**-ftrace**
-Nftrace	**-no-ftrace**
-G { global | local }	なし
-g	**-g**
-gv	なし
-gw	なし
-Ng	**-g0**
-I*ディレクトリ名*	**-I***ディレクトリ名*
-L*ディレクトリ名*	**-L***ディレクトリ名*
-l*ライブラリ名*	**-l***ライブラリ名*
-o*ファイル名*	**-o***ファイル名*
-Pauto	**-mparallel**
-Pmulti	なし
-Popenmp	**-fopenmp**
-Pstack	なし

-Pstatic	**-bss**
-p	**-p**
-Np	なし
-pi argconsis={noexp|safe|unsafe}	なし
-pi auto	**-finline-functions**
-pi noauto	なし
-pi exp=*手続名*	なし
-pi noexp=*手続名*	なし
-pi expin={*ファイル名***|***ディレクトリ名***}**	**-finline-file=***ファイル名* または、 **-finline-directory=***ディレクトリ名* (注) **-finline-functions**の指定が必要です。
-pi { fullmsg | infomsg | nomsg }	**-fdiag-inline={ 2 | 1 | 0 }**
-pi { incdir | noincdir }	なし
-pi line=*n* (注) *n*は手続の最大行数を指定します。	**-finline-max-function-size=***n* (注) *n*は中間言語の量を指定します。**-finline-functions**の指定が必要です。
-pi { modout | nomodout }	なし
-pi nest=*n*	**-finline-max-depth=***n* (注) **-finline-functions**の指定が必要です。
-pi rexp=*手続名*	なし
-Npi	**-fno-inline-functions**
-R0	なし
-R1	なし
-R2	なし

-R3	なし
-R4	なし
-R5	**-report-diagnostics -report-format**
-S	**-S**
-NS	なし
-size_t32	なし
-size_t64	なし (注) 常に**-size_t64**相当で動作します。
-sx8 | -sx8r | -sx9 | -sxace	なし
-to ディレクトリ名	なし
-ts ディレクトリ名	なし
-U*name*	**-U***name*
-V (注) バージョン表示後にコンパイル処理を続けます。	**--version** (注) コンパイル処理を行わずにバージョン表示のみ行います。
-NV	なし
-verbose	**-v**
-Nverbose	なし
-Wa"*オプション列***"**	**-Wa,"***オプション列***"**
-Wc"*オプション列***"**	なし
-Wf"*オプション列***"** (注) Fortranコンパイラに対する詳細オプションの対応は次節以降を参照してください。	なし
-Wl"*オプション列***"**	**-Wl,"***オプション列***"**
-Wp"*オプション列***"**	**-Wp,"***オプション列***"**

-w	**-w**
-Nw	**-Wall**
-Yf," *ディレクトリ名* **"**	なし
-Yl," *ディレクトリ名* **"**	なし
-Yp," *ディレクトリ名* **"**	なし

B.2.2 f90/sxf90詳細オプション-最適化オプション

Fortran90/SXコンパイラ	Vector Engineコンパイラ
-ai | -Nai	なし
-fusion	**-floop-fusion**
-Nfusion	**-fno-loop-fusion**
-i { errchk | noerrchk }	なし
-O { aryinq | noaryinq }	なし
-O chg	**-fassociative-math** または、 **-faggressive-associative-math** （注）**-faggressive-associative-math** は、**-fassociative-math**より激しく最適化します。
-O nochg	**-fno-associative-math**
-O { compass | nocompass }	なし
-O darg	**-fargument-alias**
-O nodarg	**-fargument-noalias**
-O div	**-freciprocal-math**
-O nodiv	**-fno-reciprocal-math**
-O extendreorder	**-msched-interblock**
-O reorderrange=bblock	**-msched-insns**

-O { if | noif }	なし
-O iodo	**-marray-io**
-O noiodo	**-mno-array-io**
-O infomsg	なし
-O move	**-fmove-loop-invariants-unsafe**
-O nomovediv	**-fmove-loop-invariants**
-O nomove	**-fno-move-loop-invariants**
-O overlap	**-fnamed-alias**
-O nooverlap	**-fnamed-noalias**
-O { shapeprop | noshapeprop }	なし
-O unroll	**-floop-unroll**
-O unroll=*n*	**-floop-unroll** **-floop-unroll-max-times=***n* (注) 2つのオプションを同時に指定します。
-O nounroll	**-fno-loop-unroll**
-O wkary_opt	**-mstack-arrays**
-O nowkary_opt	**-mno-stack-arrays**
-O { zlpchk | nozlpchk }	なし
-prob_dir ***ディレクトリ名***	なし
-prob_file ***ファイル名***	なし
-prob_generate	なし
-prob_use	なし

B.2.3　f90/sxf90詳細オプション-ベクトル化・並列化オプション

Fortran90/SXコンパイラ	Vector Engineコンパイラ

-common { global | local }	なし
-moddata { global | local }	なし
-ompctl { condcomp | nocondcomp }	なし
-pvctl altcode	**-mvector-dependency-test** **-mvector-loop-count-test** **-mvector-shortloop-reduction** (注) 3つのオプションを同時に指定します。
-pvctl altcode=dep	**-mvector-dependency-test**
-pvctl altcode=nodep	**-mno-vector-dependency-test**
-pvctl altcode=loopcnt	**-mvector-loop-count-test**
-pvctl altcode=noloopcnt	**-mno-vector-loop-count-test**
-pvctl altcode=shortloop	**-mvector-shortloop-reduction**
-pvctl altcode=noshortloop	**-mno-vector-shortloop-reduction**
-pvctl noaltcode	**-mno-vecgtor-depencendy-test** **-mno-vector-loop-count-test** **-mno-vector-shortloop-reduction** (注) 3つのオプションを同時に指定します。
-pvctl assoc	**-fassociative-math**
-pvctl noassoc	**-fno-associative-math**
-pvctl { assume | noassume }	なし
-pvctl chgpwr	**-mvector-power-to-explog** **-mvector-power-to-sqrt** (注) 2つのオプションを同時に指定します。
-pvctl chgtanh	なし
-pvctl cncall=*手続名*	なし
-pvctl collapse	**-floop-collapse**
-pvctl nocollapse	**-fno-loop-collapse**

-pvctl { compress | nocompress }	なし
-pvctl cond_mem_opt	**-mvector-merge-conditional**
-pvctl nocond_mem_opt	**-mno-vector-merge-conditional**
-pvctl { conflict | noconflict }	なし
-pvctl divloop	なし
-pvctl nodivloop	**-mwork-vector-kind**=none
-pvctl expand=*n*	**-floop-unroll-complete**=*n*
-pvctl noexpand	**-fno-loop-unroll-complete**
-pvctl { farouter | nofarouter }	なし
-pvctl for[=*n***]**	なし (注) 並列化のスケジュールは、**-mschedule-static**などで制御できます。「3.2 最適化・ベクトル化オプション」を参照してください
-pvctl by=*n*	なし (注) 並列化のスケジュールは、**-mschedule-static**などで制御できます。「3.2 最適化・ベクトル化オプション」を参照してください
-pvctl { fullmsg | infomsg | nomsg }	**-fdiag-parallel={ 2 | 1 | 0 }** **-fdiag-vector={ 2 | 1 | 0 }** (注) 2つのオプションを同時に指定します。
-pvctl { ifopt | noifopt }	なし
-pvctl inner	**-mparallel-innerloop**
-pvctl noinner	**-mno-parallel-innerloop**
-pvctl listvec	**-mlist-vector**
-pvctl nolistvec	**-mno-list-vector**

-pvctl loopchg	**-floop-interchange**
-pvctl noloopchg	**-fno-loop-interchange**
-pvctl loopcnt=*n*	**-floop-count**=*n*
-pvctl lstval	なし
-pvctl nolstval	なし
-pvctl matmul	**-fmatrix-multiply**
-pvctl nomatmul	**-fno-matrix-multiply**
-pvctl matmulblass	なし
-pvctl { neighbors | noneighbors }	なし
-pvctl nodep	**-fivdep**
-pvctl on_adb[=*カテゴリ***]**	なし
-pvctl outerstrip	**-mparallel-outerloop-strip-mine**
-pvctl noouterstrip	**-mno-parallel-outerloop-strip-mine**
-pvctl outerunroll=*n*	**-fouterloop-unroll** **-fouterloop-unroll-max-times**=*n* (注) 2つのオプションを同時に指定します。
-pvctl outerunroll_lim=*n*	なし
-pvctl parcase	**-mparallel-sections**
-pvctl noparcase	**-mno-parallel-sections**
-pvctl parthreshold=*n*	**-mparallel-threshold**=*n*
-pvctl noparthreshold	**-mno-parallel-threshold**
-pvctl res={ whole | parunit | no }	なし
-pvctl shape=*n*	なし
-pvctl split	**-floop-split**
-pvctl nosplit	**-fno-loop-split**

-pvctl { vchg | novchg }	なし
-pvctl vecthreshold=*n*	**-mvector-threshold**=*n*
-pvctl verrchk	**-mvector-intrinsic-check**
-pvctl noverrchk	**-mno-vector-intrinsic-check**
-pvctl { vlchk | novlchk }	なし
-pvctl vregs=*n*	なし
-pvctl vsqrt	**-mvector-sqrt-instruction**
-pvctl novsqrt	**-mno-vector-sqrt-instruction**
-pvctl vwork={ static | stack | hybrid }	なし
-pvctl vworksz=*n*	なし
-reserve *n*	なし
-tasklocal { macro | micro }	なし
-v	**-mvector**
-Nv	**-mno-vector**

B.2.4 f90/sxf90詳細オプション-その他のオプション

Fortran90/SXコンパイラ	Vector Engineコンパイラ
-A { dbl | dbl4 | dbl8 | idbl | idbl4 | idbl8 }	**-A idbl : -fdefault-real=8 -fdefault-double=16** **-A idbl4 : -fdefault-real=8** **-A idbl8 : -fdefault-double=16** (注) 上記以外のオプション移行の詳細については「13.5 旧SXコンパイラからの移行に関するトラブルシューティング」を参照してください。
-acct	**-proginf**
-Nacct	**-no-proginf**

-adv { on | off }	なし
-Nadv	なし
-compatimod	なし
-const_ext | -Nconst_ext	なし
-cont	**-fassume-contiguous**
-Ncont	**-fno-assume-contiguous**
-dblprecision | -Ndblprecision	なし
-dir { vec | par | debug }	なし
-dir { novec | nopar | nodebug }	なし
-dollar | -Ndollar	なし
-esc | -Nesc	なし
-G | -NG	なし
-init stack={ zero | nan | 0xXXXX }	**-minit-stack={ zero | nan | 0xXXXX }**
-init heap={zero | nan | 0xXXXX }	なし (注) 実行時の環境変数**VE_INIT_HEAP**で制御できます。「2.2 実行時に参照される環境変数」を参照してください。
-K { a | Na }	なし
-K { b | Nb }	なし
-L { stdout | nostdout | filename=*ファイル名* **}**	**-report-file={ stdout |** *ファイル名* **}** (注) 既定では、**-Lnostdout**相当で動作します。
-L { eject | noeject }	なし
-L fmtlist	**-report-format**
-L nofmtlist	なし
-L { inclist | noinclist }	なし

-L { map | nomap }	なし
-L mrgmsg	なし
-L sepmsg	**-report-diagnostics**
-L objlist	**-assembly-list**
-L noobjlist	なし
-L { source | nosource }	なし
-L { summary | nosummary }	なし
-L { transform | notransform }	なし
-NL	なし
-M { zdiv | flovf | fxovf | inv | inexact }	なし (注) 実行時の環境変数**VE_FPE_ENABLE**で制御できます。「2.2 実行時に参照される環境変数」を参照してください。
-M { setall | setmain }	なし
-msg b	**-Wobsolescent**
-msg nb	**-Wno-obsolescent**
-msg { d | nd }	なし
-msg { f | nf }	なし (注) 常に**nf**相当で動作します。
-msg { o | no }	なし
-msg { w | nw }	なし (注) 常に**nw**相当で動作します。
-P { a | b | c | d | e | f | h | i | l | p | t | x | z }	なし
-P { b | nb }	なし

-P { c | nc }	なし
-P { d | nd }	なし
-P { e | ne }	なし
-P f	**-nofpp**
-P nf	なし (注) 既定の動作です。
-P h	なし (注) 既定の動作です。
-P nh	**-ff90-sign**
-P { i | ni }	なし
-P { l | nl }	なし
-P { p | np }	なし
-P { t | nt }	なし (注) 常に**nt**相当で動作します。
-P { x | nx }	なし (注) 常に**x**相当で動作します。
-P { z | nz }	なし
-ptr { byte | word }	なし (注) 常に**byte**相当で動作します。
-s | -Ns	なし
-stmtid | -Nstmtid	なし
-w { double16 | rdouble16 }	なし
-xint	**-mno-vector-iteration**
-Nxint	**-mvector-iteration**

B.3　コンパイラ指示行

SXシリーズ向けコンパイラとVector Engine向けコンパイラの指示行の対応表は「C.3 コンパイラ指示行」を参照してください。SXシリーズ向けコンパイラの指示行をVector Engine向けコンパイラに変換するツールを用意しています。詳細は「付録 C コンパイラ指示行変換ツール」を参照してください。

B.4　環境変数

SXシリーズ向けコンパイラの主要な実行時に参照される環境変数とほぼ同等の機能をもつVector Engine向けコンパイラの環境変数を以下に示します。

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
F_PROGINF	**VE_PROGINF**
F_TRACEBACK	**VE_TRACEBACK**
F_EXPRCW	**VE_FORT_EXPRCW**
F_FMTBUF	**VE_FORT_FMTBUF**
F_NORCW	**VE_FORT_NORCW**
F_PAUSE	**VE_FORT_PAUSE**
F_PARTRCW	**VE_FORT_PARTRCW**
F_SETBUF	**VE_FORT_SETBUF**
F_UFMTADJUST=TYPE1	**VE_FORT_UFMTADJUST=INT,LOG**
F_UFMTADJUST=TYPE2	**VE_FORT_UFMTADJUST=ALL**
F_UFMTENDIAN	**VE_FORT_UFMTENDIAN**
F_FF*n*	**VE_FORT***n*

B.5　その他のライブラリ

SXシリーズ向けコンパイラのその他のライブラリの手続とほぼ同等の機能をもつVector Engine向けコンパイラの手続を以下に示します。**USE**文の代わりに**-use**の使用も可能です。

SXシリーズ向けコンパイラ	Vector Engineコンパイラ
CALL ABORT()	**USE F90_UNIX** **CALL ABORT()**
RESULT = ACCESS(NAME,MODE)	**USE F90_UNIX_FILE** **CALL ACCESS(NAME,AMODE,RESULT)** (注) MODE(文字)がAMODE(整数)に変更されています。AMODE(整数)の詳細は「11.3.5 F90_UNIX_FILE」のパラメタを参照してください。
RESULT = ALARM(SECONDS,HANDLER)	**USE F90_UNIX_PROC** **CALL** **ALARM(SECONDS,HANDLER,RESULT,** **ERRNO)**
RESULT = CHDIR(NAME)	**USE F90_UNIX_DIR** **CALL CHDIR(NAME,RESULT)**
RESULT = CHMOD(NAME,MODE)	**USE F90_UNIX_FILE** **CALL CHMOD(PATH,AMODE,RESULT)** (注) MODE(文字)がAMODE(整数)に変更されています。AMODE(整数)の詳細は「11.3.5 F90_UNIX_FILE」のパラメタを参照してください。
CALL FLUSH(UNIT)	**FLUSH(UNIT)** (注)「11.1.42 FLUSH(*UNIT*)」を参照してください。
RESULT = FORK()	**USE F90_UNIX_PROC** **CALL FORK(RESULT,ERRNO)**
CALL FREE(PTR)	**USE F90_UNIX** **CALL FREE(PTR)**
RESULT = FSTAT(UNIT,BUFF)	**USE F90_UNIX_FILE** **CALL FSTAT(UNIT,BUFF,RESULT)**
CALL GETARG(POS,VALUE)	**USE F90_UNIX** **CALL GETARG(POS,VALUE)**

RESULT = GETCWD(DIRNAME)	USE F90_UNIX_DIR CALL GETCWD(DIRNAME,ERRNO=RESULT)
CALL GETENV(NAME,VALUE)	USE F90_UNIX CALL GETENV(NAME,VALUE)
RESULT = GETGID()	USE F90_UNIX RESULT = GETGID()
CALL GETLOG(NAME)	USE F90_UNIX_ENV CALL GETLOGIN(NAME)
RESULT = GETPID()	USE F90_UNIX RESULT = GETPID()
RESULT = GETUID()	USE F90_UNIX RESULT = GETUID()
RESULT = HOSTNM(NAME)	USE F90_UNIX_ENV CALL GETHOSTNAME(NAME,RESULT)
RESULT = IARGC()	USE F90_UNIX RESULT = IARGC()
RESULT = ISATTY(UNIT)	USE F90_UNIX_ENV CALL ISATTY(UNIT,RESULT,ERRNO)
RESULT = LINK(PATH1,PATH2)	USE F90_UNIX_DIR CALL LINK(PATH1,PATH2,RESULT)
RESULT = LSTAT(FILE,BUFF)	USE F90_UNIX_FILE CALL LSTAT(FILE,BUFF,RESULT)
PTR = MALLOC(SIZE)	USE F90_UNIX PTR = MALLOC(SIZE)
RESULT = RENAME(FROM,TO)	USE F90_UNIX_DIR CALL RENAME(FORM,TO,RESULT)
CALL SLEEP(SECONDS)	USE F90_UNIX_PROC CALL SLEEP(SECONDS)
RESULT = STAT(FILE,BUFF)	USE F90_UNIX_FILE CALL STAT(FILE,BUFF,RESULT)

RESULT = SYSTEM(COMMAND)	**USE F90_UNIX_PROC** **CALL** **SYSTEM(COMMAND,RESULT,ERRNO)**
RESULT = TIME()	**USE F90_UNIX_ENV** **CALL TIME(RESULT)**
RESULT = TTYNAM(UNIT)	**USE F90_UNIX_ENV** **CALL** **TTYNAME(UNIT,RESULT,ERRNO)**
RESULT = UNLINK(PATH)	**USE F90_UNIX_DIR** **CALL UNLINK(PATH,RESULT)**
RESULT = WAIT(I)	**USE F90_UNIX_PROC** **CALL WAIT(I,ERRNO=RESULT)**

B.6　処理系定義

SXシリーズ向けコンパイラの処理系定義とVector Engine向けコンパイラの処理系定義の対応表を以下に示します

B.6.1　型

型	**SXシリーズ向けコンパイラ**		**Vector Engineコンパイラ**	
	種別パラメタ	**対応するデータ宣言(*1)**	**種別パラメタ**	**対応するデータ宣言**
整数型	1 (*2)	1バイト整数型	1	1バイト整数型
整数型	2	2バイト整数型	2	2バイト整数型
整数型	4	4バイト整数型 (基本整数型)	4	4バイト整数型 (基本整数型)
整数型	8	8バイト整数型	8	8バイト整数型
実数型	4	実数型 (基本実数型)	4	実数型 (基本実数型)
実数型	8	倍精度実数型	8	倍精度実数型

型	SXシリーズ向けコンパイラ		Vector Engineコンパイラ	
	種別パラメタ	対応するデータ宣言(*1)	種別パラメタ	対応するデータ宣言
実数型	16	4倍精度実数型	16	4倍精度実数型
複素数型	4	複素数型 (基本複素数型)	4	複素数型 (基本複素数型)
複素数型	8	倍精度複素数型	8	倍精度複素数型
複素数型	16	4倍精度複素数型	16	4倍精度複素数型
論理型	1	1バイト論理型	1	1バイト論理型
論理型	4	4バイト論理型 (基本論理型)	4	4バイト論理型 (基本論理型)
論理型	8	8バイト論理型	8	8バイト論理型
文字型	1	文字型	1	文字型
文字型	2 (*3)	日本語型	なし	

(*1) Fortran90/SX コンパイラの場合、対応するデータ宣言の型はコンパイラオプションの指定によって変更可能です。

(*2) Fortran 90/SX コンパイラでは使用できません。

(*3) NEC Fortran 2003 コンパイラでは使用できません。

B.6.2 諸元

項目	Fortran90/SX コンパイラ	NEC Fortran 2003 コンパイラ	Vector Engine コンパイラ
INCLUDE行で取り込むファイルのネスト数	-	20	63
配列の次元数	7	31	31
継続行の行数	99行	511行	1023行

名前の長さ	63文字	199文字	199文字

B.6.3 組込み手続

組込み手続名	SXシリーズ向けコンパイラ	Vector Engineコンパイラ
SYSTEM_CLOCK	取得時間の起点は、プログラム開始時を基準時間(カウント 0)である。	取得時間の起点は、協定世界時(UTC)の 1970 年 1 月 1 日の 00:00 を基準時間(カウント 0)である。

付録 C　コンパイラ指示行変換ツール

本節ではsxf90/sxf03/sxcc/sxc++のコンパイラ指示行をnfort/ncc/nc++のコンパイラ指示行に変換するツールについて説明します。

C.1　nfdirconv

コマンド名

nfdirconv

書式

nfdirconv [*オプション*...] [*ファイル* | *ディレクトリ*]...

説明

ソースファイルに含まれるsxf90/sxf03/sxcc/sxc++のコンパイラ指示行をnfort/ncc/nc++のコンパイラ指示行に変換します。ソースファイルのかわりにディレクトリを指定することで、そのディレクトリの次の拡張子を持つファイルをまとめて変換できます。

```
.c  .i  .h  .C  .cc  .cpp  .cp  .cxx  .c++  .ii  .H  .hh  .hpp
.hp  .hxx  .h++  .tcc .F  .FOR  .FTN  .FPP  .F90  .F95  .F03  .f
.for  .ftn  .fpp  .f90  .f95  .f03  .i90
```

変換前のファイルは「*ファイル名*.bak」として保存されます。

sxf90/sxf03/sxcc/sxc++の指示行はオプションの指定により、変換後も残したり、削除したりできます。

オプション

オプション名	説明
-a,　--append	sxf90/sxf03/sxcc/sxc++のコンパイラ指示行を削除せずに、nfort/ncc/nc++のコンパイラ指示行を追加します。
-d,　--delete	sxf90/sxf03/sxcc/sxc++のコンパイラ指示行に対応するnfort/ncc/nc++のコンパイラ指示行が無い場合、sxf90/sxf03/sxcc/sxc++のコンパイラ指示行を削除します。
-f,　--force	入力ファイルの拡張子をチェックしません。
-h,　--help	本コマンドの利用方法を出力して終了します。
-o *ファイル名*,	出力ファイル名を指定します。入力ファイルを複数指定したと

--output *ファイル名*	き、または、ディレクトリを指定したとき、この指定は無視されます。
-p,　--preserve	sxf90/sxf03/sxcc/sxc++のコンパイラ指示行に対応するnfort/ncc/nc++のコンパイラ指示行が無い場合、sxf90/sxf03/sxcc/sxc++のコンパイラ指示行を削除しません。(既定の動作)
-q,　-quiet	コンパイラ指示行の変換に関するメッセージを出力しません。
-r,　--recursive	ディレクトリと同時に指定したとき、サブディレクトリを再帰的に走査します。ディレクトリのシンボリックリンクは無視します。
-v,　--version	バージョン情報を出力して終了します。

出力メッセージ

指示行を変換した場合やnfort/ncc/nc++で指示行が未サポートの場合、標準エラー出力に次の形式のメッセージを出力します。

ファイル名　: line *行番号*: メッセージ

ファイル名　: 入力ファイル名
行番号　　: 変換前のファイルの行番号
メッセージ　:

- converted "*SX用指示行*" to "*VE用指示行*" (Converted|Substitute)
  コンパイラ指示行を変換したことを表します。SXとVEのコンパイラ指示行が同等の機能を持つ場合、"Converted"が出力されます。完全に同じではないが、ほぼ同等の機能を持つ場合は"Substitute"が出力されます。
- "*SX用指示行*" is not supported [(Remained| Removed/Obsolescent)]
  sxf90/sxf03/sxcc/sxc++のコンパイラ指示行がVEではサポートしていないことを表します。将来実装予定のコンパイラ指示行は"Remained"が出力されます。サポート予定のないコンパイラ指示行は、"Removed/Obsolescent"が出力されます。

返却値

全ての変換に成功したときは0を返し、エラーが発生したときは0以外を返します。

注意事項

本コマンドは作業用の一時ファイルを/tmpに作成します。この一時ファイルは、コマンド終了時に自動的に削除されます。一時ファイルを作成するディレクトリは、環境変数**TMP**

DIRで変更できます。

C.2　使用例

指示行変換ツールの使用例を示します。ここでは、環境変数**PATH**に/opt/nec/ve/binが追加されていることを前提とします。

ファイルを指定した場合

ファイルに含まれるsxf90/sxf03/sxcc/sxc++のコンパイラ指示行をnfort/ncc/nc++のコンパイラ指示行に変換します。

```
$ cat sample.f90
program main
    integer s
!CDIR NOVECTOR
    do i=1, 1000
      s = s + i
    enddo
    print*,s
end program
```

```
$ nfdirconv sample.f90
sample.f90: line   3: converted 'NOVECTOR' to 'novector' (Converted)
```

```
$ cat sample.f90
program main
    integer s
!NEC$ novector
    do i=1, 1000
      s = s + i
    enddo
    print*,s
end program
```

ディレクトリを指定した場合

以下のようなディレクトリ構成とします。

```
dir/
    ├ Makefile
    ├ sample1.f90
    ├ sample2.f90
    └ subdir/
          ├ Makefile
          └ sample3.f90

$ nfdirconv dir
dir/sample1.f90: line   5: converted 'loopcnt=5' to 'loop_count(5)' (Converte
d)
dir/sample2.f90: line  16: converted 'nodep' to 'ivdep' (Substitute)
```

この場合、dir直下のsample1.f90とsample2.f90が変換対象となります。
Makefileは拡張子が無いため変換の対象外です。また、ディレクトリ直下のファイルが対象となるため、サブディレクトリsubdir配下のファイルも変換対象外です。

```
$ nfdirconv -r dir
dir/sample2.f90: line   5: converted 'nodep' to 'ivdep' (Substitute)
dir/sample1.f90: line  16: converted 'loopcnt=5' to 'loop_count(5)' (Converte
d)
dir/subdir/sample3.f90: line  12: converted 'loopcnt=5' to 'loop_count(5)' Con
verted)
```

サブディレクトリのファイルも含めて変換したい場合は、**-r**を指定します。**-r**を指定するとディレクトリを再帰的に処理するようになり、サブディレクトリにあるファイルも変換されるようになります。

C.3　コンパイラ指示行

sxf90/sxf03/sxcc/sxc++のコンパイラ指示行と変換後のコンパイラ指示行を以下に示します。「変換後」列の(Remained)は将来実装予定のコンパイラ指示行、(Removed/Obsolescent)はサポートしないコンパイラ指示行を表します。

SXシリーズ向けコンパイラ	変換後
alloc_on_vreg(*identifier***, *n*)**	**vreg(***identifier***)**
altcode	**dependency_test** **loop_count_test** **shortloop_reduction**
altcode=dep	**dependency_test**
altcode=loopcnt	**loop_count_test**
altcode=nodep	**nodependency_test**
altcode=noshort	**noshortloop_reduction**
altcode=short	**shortloop_reduction**
noaltcode	**nodependency_test** **noloop_count_test** **noshort_loop_reduction**
array(*c1*[,*c2*…])	(Removed/Obsolescent)
arraycomb	(Removed/Obsolescent)
assert	(Removed/Obsolescent)
assoc	**assoc**
noassoc	**noassoc**
assume	**assume**
noassume	**noassume**
atomic	**atomic**
cncall	**cncall**
collapse	**collapse**
compress	(Removed/Obsolescent)
nocompress	(Removed/Obsolescent)

concur	**concurrent**
concur(by=*m***)**	**concurrent schedule(**dynamic**,** *m***)**
concur(for=*n***)**	**concurrent**
noconcur	**noconcurrent**
data_prefetch	(Removed/Obsolescent)
delinearize	(Removed/Obsolescent)
nodelinearize	(Removed/Obsolescent)
divloop	**vwork**
nodivloop	**novwork**
end arraycomb	(Removed/Obsolescent)
end parallel sections	(Removed/Obsolescent)
expand	**unroll_complete**
expand=*n*	(Removed/Obsolescent)
noexpand	**nounroll**
extend	(Removed/Obsolescent)
extend_free	(Removed/Obsolescent)
fixed	(Removed/Obsolescent)
free	(Removed/Obsolescent)
gthreorder	**gather_reorder**
nogthreorder	(Removed/Obsolescent)
iexpand(function)	**inline**
noiexpand(function)	**noinline**
inline	**always_inline**
inner	**inner**
noinner	**noinner**
listvec	**list_vector**
nolistvec	**nolist_vector**

loopchg	**interchange**
noloopchg	**nointerchange**
loopcnt=*n*	**loop_count(***n***)**
lstval	**lstval**
nolstval	**nolstval**
move	**move_unsafe**
nomove	**nomove**
nomovediv	**move**
neighbors	(Removed/Obsolescent)
noneighbors	(Removed/Obsolescent)
nexpand	**inline_complete**
noconflict(*identifier***)**	(Removed/Obsolescent)
nodep	**ivdep**
on_adb(*identifier***)**	(Removed/Obsolescent)
outerunroll=*n*	**outerloop_unroll(***n***)**
noouterunroll	**noouterloop_unroll**
overlap	(Removed/Obsolescent)
nooverlap	(Removed/Obsolescent)
parallel do	**parallel do**
parallel do private(*identifier***)**	**parallel do private(***identifier***)**
parallel sections	(Removed/Obsolescent)
section	(Removed/Obsolescent)
select(*keyword***)**	**select_concurrent** **select_vector**
shape	(Removed/Obsolescent)
shortloop	**shortloop**
skip	(Removed/Obsolescent)
sparse	**sparse**

nosparse	**nosparse**
split	(Remained)
nosplit	(Remained)
sync	(Remained)
nosync	**nosync**
threshold	(Removed/Obsolescent)
nothreshold	(Removed/Obsolescent)
traceback	(Remained)
unroll=*n*	**unroll(***n***)**
nounroll	**nounroll**
unshared	(Removed/Obsolescent)
vecthreshold=*n*	**vector_threshold(***n***)**
vector	**vector**
novector	**novector**
verrchk	(Remained)
noverrchk	(Remained)
vlchk	(Removed/Obsolescent)
ovlchk	(Removed/Obsolescent)
vob	**vob**
novob	**novob**
vovertake(*identifier***)**	**vovertake**
novovertake	**novovertake**
vprefetch	(Remained)
novprefetch	(Removed/Obsolescent)
vreg(*identifier***)**	**vreg(***identifier***)**
vwork=keyword	(Removed/Obsolescent)
vworksz=*n*	(Removed/Obsolescent)

zcheck	(Removed/Obsolescent)
nozcheck	(Removed/Obsolescent)

C.4　留意事項

- ツール実行時に変換前のファイルが保存されます。「*ファイル名*.bak」が存在する場合は、".bak"を".bak2"にリネームし、新しいファイルを".bak"として保存します。*ファイル名*は最大5つまで保存されます。必要に応じてファイルを削除してください。
- 本ツールでは入力ファイルの書式をチェックしません。sxf90/sxf03/sxcc/sxc++のコンパイラ指示行の書式が誤っているときは正しく変換できない場合があります。
- 入力ファイルがシンボリックリンクファイルの場合は、シンボリックリンク先のファイルが更新されます。保存用の*ファイル名*.bakは通常のファイルとして作成されます。
- 範囲開始行/範囲終了行は未サポートの指示行として扱います。

付録 D　ファイル入出力解析情報

本章では、NEC Fortranコンパイラがサポートするファイル入出力解析情報の出力情報について説明します。

D.1　出力例

環境変数**VE_FORT_FILEINF**に"DETAIL"を設定したときの出力例です。

```
              ******  File Information  ******
Unit No.   : 10
File Name  : fort.10
Named      : YES
Current Directory    : /usr/uhome/xxxxxxxx
TMPDIR               : /tmp


I/O Exec. Count      :       READ        WRITE       OPEN      CLOSE    INQUIRE
                                1            1          0          1          0
                            REWIND    BACKSPACE    ENDFILE
                                1            0          0
                              WAIT        FLUSH
                                0            0


Format               :          FORMATTED    Access               :    SEQUENTIAL
Blank(OPEN)          :               NULL    Blank(READ)          :          NULL
Delim(OPEN)          :               NONE    Delim(WRITE)         :          ----
Pad(OPEN)            :                YES    Pad(READ)            :           YES
Decimal(OPEN)        :              POINT    Decimal(R/W)         :         POINT
Sign(OPEN)           :          PROCESSOR    Sign(WRITE)          :     PROCESSOR
Round(OPEN)          :          PROCESSOR    Round(R/W)           :     PROCESSOR
Asynchronous         :                 NO    Encoding             :       DEFAULT
Position             :             REWIND
Recl (Byte)          :              65536
File Size (Byte)     :                 13    File Descriptor      :             5
File System Type     :    NFS(0x00006969)    Open Mode            :     READWRITE
Terminal Assignment  :                 NO    Shrunk File          :           YES
Max File Size(Byte)  :                600
```

```
I/O Buffer Size (KByte)    :          512
Record Buffer Size (Byte)  :        65536


                              Total(In/Out)          Input          Output
Total Data Size (Byte)     :           25,             13,              12
Max   Data Size (Byte)     :                           13,              12
Min   Data Size (Byte)     :                           13,              12
Ave   Data Size (Byte)     :           12,             13,              12
Transfer Rate (KByte/sec)  :       18.789,         19.261,          18.303


                            Total(In/Out/Aux)          Input          Output
Real Time (sec)            :     0.004284,       0.000659,        0.000640
User Time (sec)            :     0.002874,       0.000062,        0.000129


Environment Variable List :
```

D.2　出力項目

Unit No.

外部装置識別子を示します。

File Name

ここで出力するファイル名は**FILE**指定子あるいは事前接続の際に指定した名前であり、ホームディレクトリあるいはカレントディレクトリからの名前ではありません。SCRATCHファイルのときもシステムによってつけられたファイル名を出力します。

Named

名前付きファイルか否かを出力します。

Current Directory

現在作業しているディレクトリ名を出力します。

TMPDIR

SCRATCHファイルが作成されるディレクトリ名を出力します。この情報はSCRATCHファイルのときのみ出力されます。

I/O Exec Count

各入出力文の実行回数を出力します。直編成ファイルのとき、**REWIND** / **BACKSPACE** / **ENDFILE** の情報は出力されません。

Format

FORM指定子を示します。

Access

ACCESS指定子を示します。

Blank (OPEN)

OPEN文に指定された値を出力します。この情報は書式付きのときのみ出力されます。

Blank (READ)

READ文に指定された値を出力します。**READ**文がないときは”----“が出力されます。**READ**文が複数あり、それぞれの**READ**文に異なる値が指定されたときは“MIXED”が出力されます。この情報は書式付きのときのみ出力されます。

Delim (OPEN)

OPEN文に指定された値を出力します。この情報は書式付きのときのみ出力されます。

Delim (WRITE)

WRITE文に指定された値を出力します。**WRITE**文がないときは”----“が出力されます。**WRITE**文が複数あり、それぞれの**WRITE**文に異なる値が指定されたときは“MIXED”が出力されます。この情報は書式付きのときのみ出力されます。

Pad (OPEN)

OPEN文に指定された値を出力します。この情報は書式付きのときのみ出力されます。

Pad (READ)

READ文に指定された値を出力します。**READ**文がないときは”----“が出力されます。**READ**文が複数あり、それぞれの**READ**文に異なる値が指定されたときは“MIXED”が出力されます。この情報は書式付きのときのみ出力されます。

Decimal (OPEN)

OPEN文に指定された値を出力します。この情報は書式付きのときのみ出力されます。

Decimal (R/W)

READ文、および、**WRITE**文に指定された値を出力します。いずれもないときは”----“が出力されます。**READ**文や**WRITE**文が複数あり、それぞれの**READ**文、**WRITE**文に異なる値が指定されたときは“MIXED”が出力されます。この情報は書式付きのときのみ出力さ

れます。

Sign（OPEN）

OPEN文に指定された値を出力します。この情報は書式付きのときのみ出力されます。

Sign（WRITE）

WRITE文に指定された値を出力します。**WRITE**文がないときは”----“が出力されます。**WRITE**文が複数あり、それぞれの**WRITE**文に異なる値が指定されたときは“MIXED”が出力されます。この情報は書式付きのときのみ出力されます。

Round（OPEN）

OPEN文に指定された値を出力します。この情報は書式付きのときのみ出力されます。

Round（R/W）

READ文、および、**WRITE**文に指定された値を出力します。いずれもないときは”----“が出力されます。**READ**文や**WRITE**文が複数あり、それぞれの**READ**文、**WRITE**文に異なる値が指定されたときは“MIXED”が出力されます。この情報は書式付きのときのみ出力されます。

Asynchronous

ASYNCHRONOUS指定子の有無を出力します。

Encoding

OPEN文に指定された値を出力します。この情報は書式付きのときのみ出力されます。

Position

OPEN文に指定された値を出力します。この情報は直編成ファイル以外のときに出力されます。

Recl

OPEN文の**RECL**指定子の値をバイト単位で出力します。**RECL**指定子の指定が省略されたとき、既定値をバイト単位で出力します。この情報はストリームファイル以外のときに出力されます。

Max Record No.

ファイルの大きさからの最大レコード番号ではなく、実際に入出力を行った中での最大のレコード番号を示します。この情報は直編成ファイルのときのみ出力されます。

File Size

クローズ時のファイルの大きさをバイト単位で出力します。この値は順番探査出力時にプログラムが付与する情報も含んでいます。

File Descriptor

ファイル識別子を示します。

File System Type

ファイルが属するファイルシステムを示します。

Open Mode

ファイルをオープンしたときのモードを示します。

Terminal Assignment

端末接続の有無を示します。

Shrunk File

ファイル縮小機能が働いたか否かを示します。ファイル縮小機能とは、オープン時のファイルの大きさ、あるいは、プログラム実行中に最も大きくなったときのファイルの大きさに比べて、クローズ時のファイルの大きさが小さいとき、残り領域を解放する機能です。この情報は順編成ファイルのときのみ出力されます。

Max File Size

プログラム実行中に最も大きくなったときのファイルの大きさをバイト単位で示します。この情報はShrunk Fileが”YES”のときのみ出力されます。この情報はI/Oバッファの大きさを決めるための有効な情報となります。

I/O Buffer Size

入出力のために準備しているI/Oバッファサイズの大きさをキロバイト(1024バイト)単位で示します。

Record Buffer Size

入出力のために準備しているレコードバッファサイズの大きさをバイト単位で示します。

Total Data Size

入出力を行った総転送量を、入出力の合計、入力の合計、出力の合計の順にバイト単位で出力します。順編成ファイルでプログラムが付与する情報はこの中に含みません。

Max Data Size

入出力の中で最も大きい記録長の値を、入力、出力の順にバイト単位で出力します。

Min Data Size

入出力の中で最も小さい記録長の値を、入力、出力の順にバイト単位で出力します。

Ave Data Size

平均記録長を、入出力の合計、入力の合計、出力の合計の順にバイト単位で出力します。

これにより、このファイルが小さいI/Oが主か、大きいI/Oが主であるかを知ることができます。

Transfer Rate

ファイル転送速度を秒あたりのキロバイト(1024バイト)単位で示します。転送速度は、Total Data SizeをReal Timeで割った値をキロバイト(1024バイト)単位で示した値です。”DETAIL”を指定したときのみ表示します。

Real Time

経過時間を示します。”DETAIL”を指定したときのみ表示します。

User Time

ユーザ時間を示します。”DETAIL”を指定したときのみ表示します。

Environment Variable List

このファイルに対して、有効となった環境変数の一覧をアルファベット順に出力します。”DETAIL”を指定したときのみ表示します。

付録 E　追加・変更点詳細

前版(第25版、2021年11月発行)からの更新内容は以下のとおりです。

- 第13章に以下のトラブルシューティングを追加しました。
  - “Too many elements in array” というコンパイルエラーが発生する。
  - 過去のバージョンのコンパイラをインストールしたい。

索　引

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

い

え

お

か

き

く

け

こ

さ

し

す

せ

そ

た

て

と

な

は

ひ

ふ

へ

ほ

ま

む

め

も

る

ろ

SX-Aurora TSUBASA システムソフトウェア

Fortranコンパイラ　ユーザーズガイド

2022年3月 26版

日本電気株式会社

東京都港区芝五丁目７番１号

TEL(03)3454-1111（大代表）