FUJITSU

Fujitsu Software
Technical Computing Suite V4.0L20

Development Studio
Fortran文法書

J2UL-2557-01Z0(09)
2024年3月

まえがき

本書の目的

本書では、富士通Fortran システムでサポートしているFortran の文法について説明します。富士通Fortran システムの使用方法については、“Fortran 使用手引書”を参照してください。

本書に記述されている言語仕様は、Fortran 2008 言語仕様および規格からの拡張言語仕様です。

Fortran 2008 言語仕様は、以下のISO 規格です。

ISO/IEC 1539-1:2010 Information technology - Programming languages - Fortran

本書の読者

本書は、Fortranプログラミングする方が対象です。

本書の構成

本書は、次の構成になっています。

“第1章 Fortran の基本事項”

Fortran の構文素、データ型、データの使用方法、およびプログラムの構造など、基本的な項目について説明します。

“第2章 文および手続の詳細”

Fortran の各文、構文、組込み手続、およびサービスルーチンの構文規則および使用方法について説明します。

“付録A 組込み手続一覧”

本処理系で使用できる組込み手続の一覧表です。

“付録B サービスルーチン一覧”

本処理系で使用できるサービスルーチンの一覧表です。

“付録C 拡張仕様一覧”

本処理系で使用できるFortran 2008 規格からの拡張仕様の一覧表です。

“付録D Fortran 2018サポート仕様一覧”

本処理系が実現しているFortran 2018 規格仕様の一覧表です。

“付録E 用語集”

本書で使用している用語の定義です。

“付録F ASCII コード表”

ASCIIコード表です。本処理系において、文字型データの内部表現は、ASCIIコード系で表されます。

記述上の約束

本書は、次の形式に従って説明します。

青文字	Fortran 2008 規格からの拡張言語仕様を意味します。
“青文字”	ハイパーリンクを意味します。
Program	Fortran の形式、およびプログラム例を意味します。
Italic	構文規則中のこの形式(斜体)で記述された部分は、実際のプログラムでは、あるプログラムの構成要素によって置き換えることを示します。置き換えられる構成要素については、その記述のあとで定義します。
KEYWORD	構文規則中のこの形式で記述された部分は、実際のプログラムにおいても、そのまま記述することを意味します。
[ ]	括弧の中の記述が省略可能であることを意味します。
...	直前の項目を0個以上連続して指定してもよいことを意味します。
abc-list	*abc* [,*abc*] ... と同じであることを意味します。

■ 構文規則が継続していることを意味します。

輸出管理規制について

本ドキュメントを輸出または第三者へ提供する場合は、お客様が居住する国および米国輸出管理関連法規等の規制をご確認のうえ、必要な手続きをおとりください。

商標

OpenMPは、OpenMP Architecture Review Boardの商標です。

そのほか、本マニュアルに記載されている会社名および製品名は、それぞれ各社の商標または登録商標です。

本資料に掲載されているシステム名、製品名などには、必ずしも商標表示(TM、(R))を付記しておりません。

出版年月および版数

版数	マニュアルコード
2024年 3月 第1.9版	J2UL-2557-01Z0(09)
2023年 9月 第1.8版	J2UL-2557-01Z0(08)
2023年 3月 第1.7版	J2UL-2557-01Z0(07)
2022年 9月 第1.6版	J2UL-2557-01Z0(06)
2021年11月 第1.5版	J2UL-2557-01Z0(05)
2021年 7月 第1.4版	J2UL-2557-01Z0(04)
2021年 3月 第1.3版	J2UL-2557-01Z0(03)
2021年 1月 第1.2版	J2UL-2557-01Z0(02)
2020年 6月 第1.1版	J2UL-2557-01Z0(01)
2020年 2月 初版	J2UL-2557-01Z0(00)

著作権表示

変更履歴

変更内容	変更箇所	版数
付録Cの記事を各章に追加しました。	1.5.11.2 1.5.12 1.6.2 1.12.6.1.4 1.17 2.20 2.57 2.361	第1.9版
GETLOGサービスサブルーチンの注意事項を追加しました。	2.211	
索引を修正しました。	索引	
説明文を見直しました。	―	
SYSTEM_CLOCK組込みサブルーチンの引数COUNTの最大値を変更しました。	付録 A	第1.8版
拡張仕様に関する記事を修正しました。	1.5.11.2 1.5.12 1.6.2 1.12.6.1.4	第1.7版

変更内容	変更箇所	版数
	1.17 2.20 2.57 2.361 付録 C	
ラージページ機能使用時の注意事項を追加しました。	2.20	
RANK組込み関数の説明を追加しました。	2.406 付録 A	
「Fortran 2018サポート仕様一覧」を追加しました。	付録 D	
注釈行の記事を修正しました。	1.4.1 1.4.2	第1.6版
内部手続の記事を修正しました。	1.12 1.12.7.2 2.62 2.389	
表「数学関数」の誤記を修正しました。	付録 A	第1.5版
説明文を見直しました。	－	第1.4版
RANDOM_SEED組込みサブルーチンの引数にSIZEが指定された場合の返却値を変更しました。	2.404	第1.3版
表「日付および時刻サブルーチン」を修正しました。	付録 A	第1.2版
文字型の種別型パラメタの記事を修正しました。	1.5.2 1.5.4.5 2.3 2.287 2.372 2.427	第1.1版

目　次

第1章 Fortran の基本事項

この章では、Fortran の構文素、データ型、データの使用方法、およびプログラムの構造など、基本的な項目について説明します。構文素とは、プログラムを構成する要素であり、定数表現、演算子、文番号、区切り記号などをいいます。

1.1 文字集合

Fortran 文字集合は、英字、数字、下線、および特殊文字から構成されます。

- 英字

  A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
  a b c d e f g h i j k l m n o p q r s t u v w x y z

- 数字

  0 1 2 3 4 5 6 7 8 9

- 下線

  _

- 特殊文字

  （空白） = + - * / ¥ ( ) [ ] { } , . : ; ! ″ % & ~ < > ? ' ` ^ | $ # @

英小文字は、文字文脈の中を除いて、対応する英大文字と等価です。

下線は名前の一部として使用することができます。ただし、名前の第1文字には記述できません。

特殊文字は、演算子、括弧、他の構文素の分離記号などとして使用できます。

1.2 名前

名前は、変数、プログラム単位、仮引数、名前付き定数、派生型などのプログラムの構成要素を識別するために使用します。名前は英字または‘$’で始まる240文字までの英字、数字、下線、および‘$’で構成されます。

名前の例:

```
NAME_LENGTH2        Name_Length2        name_length2
```

これら3つの名前は等価です。

1.3 文番号

文は5文字までの数字で構成される文番号をもつことができます。文番号の少なくとも1つの数字は0以外でなければなりません。1つの有効域において、2つ以上の文に同じ文番号を与えることはできません。文番号中の先行する数字0は、文番号を区別する上で意味がありません。

文番号は、個々の文を参照するのに使用します。文番号を用いて参照することができるのは、飛び先文、FORMAT 文、およびDO 端末文だけです。飛び先文は、ELSE 文、ELSE IF 文、CASE 文、END FORALL 文、ELSEWHERE 文、END WHERE 文、および型保持文を除く実行文です。

文番号の例:

```
123
5000
10
010
```

この例において、文番号10と010は等価です。

1.4 プログラム形式

Fortran のプログラム形式には、自由形式と固定形式があります。

文字文脈とは、文字定数表現(“1.5.4.5 文字定数表現”参照)または文字列編集記述子(“1.8.1.4 文字列編集記述子”参照)の中の文字です。

1.4.1 自由形式

自由形式の行は、最大255個の文字を含むことができ、行中のどこに文を書いてもかまいません。

空白は、文字文脈中に現れる場合を除いて、意味をもちません。

文字‘!’は、文字文脈中に指定される場合を除いて、注釈の開始を指示し、その行の終わりまでを注釈とします。

文字‘;’は、文字文脈または注釈の中にある場合を除いて、行の中での文の区切りを指示します。

注釈内でない空白でない最後の文字‘&’は、注釈行でない次の行に継続することを指示します。継続行の空白でない最初の文字が‘&’のとき、その文は‘&’に続く文字に継続し、それ以外のとき、継続行の第1けたに継続します。

名前、定数、キーワードなどの構文素が行の終わりで分断される場合、継続行の空白でない最初の文字は‘&’でなければならず、その直後に分断された構文素の続きの文字を書かなければなりません。

文字文脈を継続する場合、‘&’はその行の空白でない最後の文字でなければならず、注釈を続けることはできません。継続行の空白でない最初の文字は‘&’でなければならず、その文はその‘&’に続く文字に継続します。

自由形式の継続行の数は、無制限です。

注釈行は継続することはできませんが、継続される行は文字‘!’による注釈を含むことができます。

空白でない文字が1つの‘&’だけの行、または注釈の前の空白でない文字が1つの‘&’だけの行を書くことはできません。

第1けたの文字‘”’、‘*’、または‘+’は、注釈行を指示します。ただし、第1けたから第8けたが‘*include’の行は注釈行ではありません。また、注釈内を除いて、その行の空白でない最後の文字‘-’、および文字文脈内を除いて、注釈の前の空白でない最後の文字‘-’は、次の行の第1けた目に継続することを指示します。

1.4.2 固定形式(廃止予定事項)

固定形式では、1つの行の中で文を書くことのできる位置に制限があります。

空白は、文字文脈中に現れる場合を除いて意味をもちません。

注釈行を除いて、行の中のカラム位置には以下の意味があります。

- 第1けたから第5けたまでは、文番号を指定します。文番号を指定しない場合、空白でなければなりません。継続行の第1けたから第5けたまでは、空白でなければなりません。
- 第6けたは、継続を指示するために使用します。第6けたが空白または‘0’なら、その行は第7けたから始まる新しい文の開始行とします。第6けたが空白でも‘0’でもないとき、その行の第7けたから第72けたは、先行する注釈行でない行の継続行とします。継続行の行数は無制限です。
- 第7けたから第72けたは、Fortran の文を指定します。
- 第73けた以降は無視されます。

文字‘!’は、文字文脈の中または第6けたに現れる場合を除いて、注釈の開始を指示し、その行の終わりまでを注釈とします。第1けたが文字‘C’、‘c’、または‘*’で始まる行は注釈行とします。注釈行は継続することはできませんが、継続行は文字‘!’による注釈を含むことができます。END 文は継続してはなりません。ただし、第1けたから第8けたが‘*include’の行は注釈行ではありません。

文字‘;’は文字文脈の中、注釈の中、または第6けたに現れる場合を除いて、行の中での文の区切りを指示します。1つの行の中で文字‘;’に続く文は、文番号をもつことはできません。

1.5 データ

データは、データ型をもち、そのデータ型で表現できる値をもつことができます。データ型は、値の集合、それらの値の表現方法、およびそれらの値を解釈し操作する演算の集合によって特徴付けられます。定数、変数、および部分定数をデータ実体といいます。データ実体、式の評価結果、および関数結果をデータ要素といいます。データ要素は、データ型をもち、不定である変数の場合を除いて、データ値を

もちます。すべてのデータ要素は次元数をもちます。したがって、データ要素はスカラまたは配列のいずれかです。名前付きデータ実体は、型宣言文の属性指定子または属性宣言文で宣言することにより、属性をもつことができます。

型は、型指定子によって指定します。型指定子には、組込み型指定子および派生型指定子があります。

組込み型指定子については、“2.3 型宣言文”を参照してください。また、派生型指定子については、“1.5.11.8 派生型指定子”を参照してください。

宣言型指定子は、以下の形式です。

```
組込み型指定子              または
TYPE( 組込み型指定子 )      または
TYPE( 派生型指定子 )        または
CLASS( 派生型指定子 )       または
CLASS( * )                  または
TYPE( * )
```

型指定子は、配列構成子、SELECT TYPE構文、ALLOCATE文、DO CONCURRENT構文、またはFORALL構文で使用します。それら以外で宣言型指定子を使用します。

TYPE(*)を使って宣言した実体は、型引継ぎであり、かつ無制限多相的(“1.5.13 クラス”参照)です。型をもつと宣言されません、かつ、他の無制限多相的を含め、他の実体と同じ宣言時の型をもちません。

型引継ぎ実体は、ALLOCATABLE、CODIMENSION、INTENT(OUT)、POINTER、VALUE属性をもってはならず、形状明示配列であってはならず、仮引数でなければなりません。

型引継ぎ実体の変数名は、以下に現れることができます。

- 型引継ぎ仮引数に対応する実引数、または
- 組込み関数IS_CONTIGUOUS、LBOUND、PRESENT、RANK、SHAPE、SIZE、またはUBOUNDの第1引数、または
- 組込みモジュールISO_C_BINDINGのC_LOC関数

次元引継ぎ仮引数に対応する型引継ぎ実引数は、形状引継ぎまたは次元引継ぎでなければなりません。

型引継ぎは、言語間結合で使用します。言語間結合については、“Fortran使用手引書”の“言語間結合”を参照してください。

1.5.1 組込みデータ型

組込みデータ型には、整数型(INTEGER)、実数型(REAL)、複素数型(COMPLEX)、論理型(LOGICAL)、および文字型(CHARACTER)があります。整数型、実数型、および複素数型を数値型とします。論理型および文字型を非数値型とします。

1.5.2 種別型パラメタ

組込みデータ型は、種別型パラメタをもちます。本処理系において、整数型、実数型、論理型、および文字型の種別型パラメタの値は、各データのメモリ占有バイト数と同じです。複素数型の種別型パラメタの値は、実部および虚部のそれぞれのメモリ占有バイト数と同じです。

種別型パラメタの指定がない場合、種別型パラメタ値は暗黙的に選択されます。暗黙の種別型パラメタ値は、整数型、実数型、複素数型、および論理型の場合は4、文字型の場合は1です。これらの型を、それぞれ基本整数型、基本実数型、基本複素数型、基本論理型、および基本文字型とします。

問合せ組込み関数SELECTED_CHAR_KIND(“2.428 SELECTED_CHAR_KIND組込み関数”参照)、SELECTED_INT_KIND(“2.429 SELECTED_INT_KIND組込み関数”参照)、およびSELECTED_REAL_KIND(“2.430 SELECTED_REAL_KIND組込み関数”参照)は、指定された範囲および精度を表現可能な種別値を返却します。これらの関数を使用して種別を決定することは、他のFortranシステムへの移植性を保証します。

データの型と種別値の対応を以下の表に示します。

表1.1 データの型と種別値の対応

型指定子	種別型パラメタ	型名	表現可能な値
INTEGER	1	1バイトの整数型	-128 から 127
	2	2バイトの整数型	-32768 から 32767

型指定子	種別型パラメタ	型名	表現可能な値
	4(基本整数型)	4バイトの整数型	-2147483648 から 2147483647
	8	8バイトの整数型	-9223372036854775808 から 9223372036854775807
REAL	2	半精度実数型	絶対値が0または 2^{-14} から 2^{15}
	4(基本実数型)	単精度実数型	絶対値が0または 2^{-126} から 2^{127}
	8	倍精度実数型	絶対値が0または 2^{-1022} から 2^{1023}
	16	4倍精度実数型	絶対値が0または 2^{-16382} から 2^{16383}
COMPLEX	2	半精度複素数型	絶対値が0または 2^{-14} から 2^{15}
	4(基本複素数型)	単精度複素数型	絶対値が0または 2^{-126} から 2^{127}
	8	倍精度複素数型	絶対値が0または 2^{-1022} から 2^{1023}
	16	4倍精度複素数型	絶対値が0または 2^{-16382} から 2^{16383}
LOGICAL	1	1バイトの論理型	.TRUE. または .FALSE.
	2	2バイトの論理型	.TRUE. または .FALSE.
	4(基本論理型)	4バイトの論理型	.TRUE. または .FALSE.
	8	8バイトの論理型	.TRUE. または .FALSE.
CHARACTER	1(基本文字型)	ASCII 文字型	ASCII 文字集合

1.5.3 文字型の長さ

文字型の長さは、文字列中の文字の個数です。例えば、文字定数'character data' の長さは、14です。

1.5.4 定数表現

定数表現はFortran の各型の値を表現します。

1.5.4.1 整定数表現

整定数表現は、1つ以上の数字の列で表します。数字列の前に符号を指定することもできます。数字列の後に下線‘_’と種別を指定することもできます。種別を省略した場合、その定数は基本整数型（本処理系では、4バイトの整数型）になります。

整定数表現の例:

```
34      -256      354_4      +78_mykind
```

この例で、34 および -256 は基本整数型（本処理系では、4バイトの整数型）です。354_4は4バイトの整数型です。+78_mykind で、mykind はこれ以前に宣言された名前付き定数で、整数型に指定可能な種別の値（1、2、4、または8）をもたなければなりません。

種別指定のない整定数は-2147483648であってはなりません。

1.5.4.2 実定数表現

実定数表現は、1つ以上の数字の列と、小数点‘.’で表します。小数点は数字列の前、途中、または後ろのどこでもかまいません。数字列の前に符号を指定することもできます。数字列の後に指数部英字と指数部を指定することもできます。数字列または指数部の後に下線‘_’と種別を指定することもできます。指数部英字には、‘E’(単精度)、‘D’(倍精度)、および‘Q’(4倍精度)があります。指数部の後に種別を指定する場合、指数部英字は‘E’でなければなりません。指数部英字を指定した場合、小数点は省略してもかまいません。種別を省略した場合、その定数は基本実数型(本処理系では、単精度実数型)になります。

実定数表現の例:

```
-3.45       .0001       34.e-4       1.4_8
```

この例で、-3.45および.0001は基本実数型(本処理系では、単精度実数型)です。
34.e-4は、単精度実数型です。1.4_8は、倍精度実数型です。

1.5.4.3 複素定数表現

複素定数表現は、コンマで区切られた2つの実定数表現、整定数表現または名前付き定数を括弧でくくったものです。名前付き定数は、実数型または整数型でなければなりません。1番目の実定数表現、整定数表現、または名前付き定数は複素定数表現の実部を表し、2番目の実定数表現、整定数表現、または名前付き定数は虚部を表します。複素定数表現の種別は、実部または虚部のどちらか一方が4倍精度実数型なら16、どちらか一方が倍精度実数型なら8、そうでなければ4(基本複素数型)となります。

複素定数表現の例:

```
(3.4,-5.45)       (-1,-3)       (3.4,-5)       (-3.d13,6._8)       (p,q)
```

この例で、(3.4,-5.45)、(-1,-3)、および(3.4,-5)は、基本複素数型(本処理系では、単精度複素数型)です。(-3.d13,6._8)および(p,q)は、倍精度複素数型です。p およびq は、倍精度実数型の名前付き定数です。

1.5.4.4 論理定数表現

論理定数表現は、‘.TRUE.’または‘.FALSE.’のいずれかであり、その後に下線‘_’と種別を指定することもできます。種別を省略した場合、その定数は基本論理型(本処理系では、4バイトの論理型)となります。

論理定数表現の例:

```
.false.       .true.       .false._4       .true._mykind
```

この例で、.false. および.true. は、基本論理型(本処理系では、4バイトの論理型)です。

.false._4は、4バイトの論理型です。.true._mykind で、mykind はこれ以前に宣言された名前付き定数で、論理型に指定可能な種別の値(1、2、4、または8)をもたなければなりません。

1.5.4.5 文字定数表現

文字定数表現は、任意の文字列をアポストロフィ‘’’または引用符‘”’でくくったものであり、その前に種別と下線‘_’を指定することもできます。アポストロフィでくくられた形の文字定数中にアポストロフィを入れる場合は、連続した2つのアポストロフィを指定します。この場合、連続した2つのアポストロフィは、1つの文字(アポストロフィ)と解釈されます。同様に、引用符でくくられた形の文字定数中に引用符を入れる場合は、連続した2つの引用符を指定します。

文字定数表現の例:

```
"Hello world"
'don''t give up'
1_"Fortran"
mykind_'FUJITSU LIMITED'
''
""
```

この例で、mykind は、これ以前に宣言された名前付き定数で、文字型の種別を現す値(1) をもたなければなりません。
'' および"" は、長さ0の文字定数です。

文字定数表現のもう1つの形として、ホレリス定数があります。ホレリス定数は、符号なし整定数表現の後に英字‘H’または‘h’を書き、その後に整定数表現の値分の、任意の文字列を書いたものです。ホレリス定数は、以下の個所に記述することはできません。

- STOP 文およびPAUSE 文のオペランド
- PRINT 文およびUNIT 指定子をもたないREAD 文の書式識別子

ホレリス定数中に特殊文字列は指定できません。

原始プログラムが、自由形式である場合、ホレリス定数内に指定された文字‘&’または‘-’がその行の空白でない最後の文字である場合、継続の指定がされたとして扱われます。

1.5.4.5.1 特殊文字列

特殊文字列とは、文字定数表現の文字列、およびアポストロフィ編集の文字列として使用される場合の逆斜線‘¥’およびそれに続く1文字の計2文字からなる文字列です。特殊文字列はホレリス定数中およびH 形編集（廃止事項）に指定することはできません。

特殊文字列は翻訳時オプションにより、有効にするかどうかを指定することができます。翻訳時オプションの詳細については、“Fortran使用手引書”を参照してください。

以下に特殊文字列とその意味を示します。

- ¥0 :ヌル文字（null）
- ¥b :後退（back space）
- ¥t :水平タブ（horizontal tabulation）
- ¥n :改行（line feed）
- ¥f :書式送り（form feed）
- ¥" :引用符（quotation）
- ¥' :アポストロフィ（apostrophe）
- ¥¥ :逆斜線（back slash）
- ¥*x* :上記の文字を除く任意の文字

これらの特殊文字列は、1文字の文字列として扱われます。

特殊文字列の例:

```
i = len('abc¥n')    ! i には、4が設定されます
```

1.5.4.6 非10進定数表現

非10進定数表現には、2進定数表現、8進定数表現、および16進定数表現があります。非10進定数表現は、次の箇所に指定できます。

- 代入文の右辺式、またはPARAMETER文の右辺式
- DATA 文の初期値指定、または型宣言文の初期値指定
- 組込み関数DBLE（“2.115 DBLE組込み関数”参照）、REAL（“2.408 REAL組込み関数”参照）、またはINT（“2.288 INT組込み関数”参照）の引数A
- 組込み関数CMPLX（“2.79 CMPLX組込み関数”参照）の引数X またはY
- 組込み関数BGE（“2.49 BGE組込み関数”参照）、BGT（“2.50 BGT組込み関数”参照）、BLE（“2.56 BLE組込み関数”参照）、BLT（“2.59 BLT組込み関数”参照）、DSHIFTL（“2.126 DSHIFTL組込み関数”参照）、DSHIFTR（“2.127 DSHIFTR組込み関数”参照）、IAND（“2.229 IAND組込み関数”参照）、IEOR（“2.275 IEOR組込み関数”参照）、IOR（“2.293 IOR組込み関数”参照）、MERGE_BITS（“2.350 MERGE_BITS組込み関数”参照）の引数I またはJ

代入文の右辺式、DATA 文、PARAMETER 文、および型宣言文の初期値指定に指定する場合、非10 進定数に対応する変数の型は、組込み型でなければなりません。また、非10進定数は、対応する型の変数領域の即値として設定されます。非10進定数が、対応する型の変数領域の大きさより小さい場合、先頭に0が補われます。対応する型の変数領域の大きさより大きい場合、先頭の余分な部分は無視されます。

組込み関数の非10進定数の実引数が、対応する関数結果の型の領域の大きさより小さい場合、関数結果の先頭に0が補われます。対応する関数結果の型の領域の大きさより大きい場合、関数結果の先頭の余分な部分は無視されます。

1.5.4.6.1 2進定数表現

2進定数表現は、以下の形式です。

```
B'digits'          または
B"digits"
```

digits は数字‘0’または‘1’からなる数字列です。

2進定数表現の例:

```
data i /b'0001'/       ! 初期値1の設定
parameter (j=b'0010')  ! 定数2の指定
```

1.5.4.6.2 8進定数表現

8進定数表現は、以下の形式です。

```
O'digits'          または
O"digits"          または
'digits'O          または
"digits"O
```

digits は、‘0’から‘7’までの数字からなる数字列です。

8進定数の例:

```
data i /o'001'/        ! 初期値1の指定
data j /o"010"/        ! 初期値8の指定
parameter (k='015'o)   ! 定数13の指定
```

1.5.4.6.3 16進定数表現

16進定数は、以下の形式です。

```
Z'digits'          または
Z"digits"          または
X'digits'          または
X"digits"          または
'digits'X          または
"digits"X
```

digits は、‘0’から‘9’までの数字および‘A’から‘F’までの英字からなる文字列です。文字列の中の英字は、英小文字でもかまいません。

16進定数の例:

```
data i /z'01'/         ! 初期値1の指定
data j /z"10"/         ! 初期値16の指定
parameter (k='1a'x)    ! 定数26の指定
```

1.5.5 名前付きデータ実体

変数、名前付き定数、関数結果などの名前付きデータ実体は型をもちます。名前付きデータ実体の型は、型宣言文によって明示的に指定するか、またはその名前の第1文字によって、暗黙的に決定します。また、実体の使用法を決定するその他の属性をもつこともできます。これらの属性は、型宣言文の属性指定子によって与えるか、または各属性宣言文によって与えることができます。

1.5.5.1 暗黙の型規則

明示的な型宣言文がない場合、名前付きデータ実体の型は、その名前の第1文字によって、暗黙的に決定されます。英字‘I’から‘N’で始まる名前は基本整数型となり、その他の英字および‘$’で始まる名前は基本実数型となります。これらの暗黙の型規則は、IMPLICIT 文により変更および無効にすることが可能です。IMPLICIT NONE 文は、有効域内の暗黙の型規則を無効にします。

1.5.5.2 明示的な型宣言

型宣言文は、名前付きデータ実体および関数の型、型パラメタ、および属性を宣言します。型宣言文では、すべての組込み型および派生型(“1.5.11 派生型”参照)の名前付きデータ実体の宣言が可能です。

1.5.5.3 属性

名前付きデータ実体および関数は型や型パラメタの他に、属性をもつことができます。これらの属性は、型宣言文または属性宣言文により、指定することができます。1つの名前付きデータ実体には、1つの有効域内で、型および型パラメタを含め同じ属性を2回以上明示的に指定することはできません。

以下に名前付きデータ実体および関数がもつことのできる属性を示します。

- ALLOCATABLE 属性

  データ要素がプログラムの実行時に任意に割り付けることのできる変数であることを指定します(“2.19 ALLOCATABLE文”参照)。

- ASYNCHRONOUS 属性

  変数が、非同期入出力で扱われることを指定します(“2.31 ASYNCHRONOUS文”参照)。

- AUTOMATIC 属性

  データ要素をスタックに割り付けることを指定します(“2.41 AUTOMATIC文”参照)。

- BIND 属性

  変数または共通ブロックが、外部結合をもつC言語の変数と相互利用可能であることを指定します(“2.52 BIND文”参照)。

- CHANGEENTRY 属性

  手続がFortran 以外の言語で記述された外部手続であることを指定します(“2.67 CHANGEENTRY文”参照)。

- CODIMENSION 属性

  共配列であることを指定します(“2.80 CODIMENSION文”参照)。

- CONTIGUOUS 属性

  形状引継ぎ配列、ポインタ配列、または次元引継ぎが CONTIGUOUS であることを宣言します(“2.88 CONTIGUOUS文”参照)。

- DIMENSION 属性

  データ要素が配列、または次元引継ぎであることを指定します(“2.119 DIMENSION文”参照)。

- EXTERNAL 属性

  手続が外部手続、仮手続、または手続ポインタであり、その名前を実引数として使用できることを指定します(“2.179 EXTERNAL文”参照)。

- INTENT 属性

  仮引数の授受特性を指定します。INTENT(IN) は、仮引数を再確定したり不定にしたりしてはならないことを指定します。INTENT(OUT) は、仮引数と結合する実引数が確定可能でなければならず、かつ引用するより前に確定しなければならないことを指定します。INTENT(INOUT) は、データを受け取ったり、データを返したりできることを指定します(“2.290 INTENT文”参照)。

- INTRINSIC 属性

  手続が組込み手続であることを指定します(“2.292 INTRINSIC文”参照)。

- OPTIONAL 属性

  仮引数が手続の引用において必ずしも実引数と結合しなくてもよいことを指定します(“2.373 OPTIONAL文”参照)。

- PARAMETER 属性

  データ要素が名前付き定数であることを指定します(“2.375 PARAMETER文”参照)。

- POINTER 属性

  データ要素がポインタであることを指定します(“2.379 POINTER文”参照)。

- PRIVATE 属性

  モジュール内のデータ要素、関数、および型が非公開であり、そのモジュール内でのみ参照可能であることを指定します(“2.387 PRIVATE文”参照)。

- PROTECTED 属性

  モジュールの宣言部で名前付き変数に指定し、その変数を参照結合した有効範囲で引用できる箇所を限定します(“2.394 PROTECTED文”参照)。

- PUBLIC 属性

  モジュール内のデータ要素、関数、および型が公開であり、そのモジュールを参照している可視単位でも参照可能であることを指定します(“2.395 PUBLIC文”参照)。

- SAVE 属性

  RETURN 文またはEND 文の実行後も結合状態、割付け状態、定義状態、および値を保持することを指定します(“2.422 SAVE文”参照)。

- STATIC 属性

  データ要素をメモリ上に割り付けることを指定します(“2.459 STATIC文”参照)。

- TARGET 属性

  データ要素がポインタと結合できることを指定します(“2.476 TARGET文”参照)。

- VALUE 属性

  仮引数が、実引数の値を初期値とする確定可能である一時的な領域と結合することを指定します。仮引数の値および定義状態のその後の変更は、実引数に影響を与えません。(“2.498 VALUE文”参照)。

- VOLATILE 属性

  データ要素を最適化の対象としないことを指定します(“2.500 VOLATILE文”参照)。

1.5.6 スカラ

スカラは、そのデータ型の単一な値によって表現される、配列ではないデータ要素です。スカラの次元数は0です。

1.5.6.1 スカラポインタ

スカラポインタは、必要な時にALLOCATE 文によって割り付け、DEALLOCATE 文によって解放することができます。あるいは、ポインタ代入文によって指示先と結合できます。スカラポインタの次元数は0です。

1.5.6.2 スカラ割付け変数

スカラ割付け変数は、必要な時にALLOCATE 文によって割り付け、DEALLOCATE 文によって解放することができます。スカラ割付け変数の次元数は0です。スカラ割付け変数の結合状態は、割付け配列と同様です。“1.5.9.1.1 割付け配列結合状態”を参照してください。

1.5.7 部分列

文字列は文字の列です。文字列中の文字には、文字列の左から右に1から順に番号がつけられます。部分列は、文字列の連続した一部分です。部分列は、以下の形式です。

string ([*starting-point*] : [*ending-point*])

string は、文字型の名前付きデータ実体、またはホレリス定数の形式を除く文字定数表現でなければなりません。

starting-point は、文字列 *string* 中の部分列の開始位置です。*starting-point* を省略した場合、開始位置は1とします。

ending-point は、文字列 *string* 中の部分列の終了位置です。*ending-point* を省略した場合、終了位置は、*string* の長さとします。

部分列の長さは、その部分列中の文字の個数です。*starting-point* が *ending-point* を超える場合、部分列の長さは0です。

部分列の例:

```
! char_string が値'abcdefg' をもっている場合、
char_string(2:4)    ! は、'bcd' です。
char_string(2: )    ! は、'bcdefg' です。
char_string( :4)    ! は、'abcd' です。
char_string( : )    ! は、'abcdefg' です。
char_string(3:3)    ! は、'c' です。
'abcdefg'(2:4)      ! は、'bcd' です。
'abcdefg'(3:2)      ! は、長さ0の文字列です。
```

1.5.8 配列

配列は、すべてが同じ型および同じ型パラメタをもつスカラデータの集合であり、それらの要素を四角い形に配置したものです。配列は次元数をもち、次元数は最大30次元です。配列の形状は、各次元の大きさです。配列の大きさは、配列要素の数です。

配列の例:

```
integer,dimension(3,2) :: i
```

この例で、配列 i の次元数は2、形状は(3,2)、大きさは6となります。

1.5.8.1 配列引用

全体配列は、配列の名前によって引用します。配列要素や部分配列は、部分配列添字を使用して引用します。配列引用は以下の形式です。

part[%*part*]...　　または
part[. *part*]...

part は以下の形式です。

name [(*subscript-list*)]

name は名前です。(*subscript-list*)を含む場合、*name* は配列でなければなりません。

'%' 演算子および '.' 演算子の使用は構造体成分の引用を意味します。詳細については、"1.5.11.6 構造体成分"を参照してください。

subscript-list は配列引用の添字であり、コンマで区切られた以下の形式です。

element-subscript　　または
subscript-triplet　　または
vector-subscript

element-subscript は、スカラ整数式です。

subscript-triplet は添字三つ組であり、以下の形式です。

[*element-subscript*]:[*element-subscript*][: *stride*]

stride は刻み幅であり、スカラ整数式です。

vector-subscript はベクトル添字であり、1次元の整数配列式です。

subscript-list 中の各添字は、配列の各次元を参照します。添字は左から順に、第1次元、第2次元の順に参照します。

subscript-list に *subscript-triplet* または *vector-subscript* を含む場合、他の *part* の *subscript-list* に *subscript-triplet* および *vector-subscript* を含むことはできません。また、他の *name* が(*subscript-list*)の指定をもたない配列であってはなりません。

(*subscript-list*)の指定をもたない *name*が配列である場合、他の *name*が(*subscript-list*)の指定をもたない配列であってはなりません。また、他の *part* の *subscript-list* に *subscript-triplet* および *vector-subscript* を含むことはできません。

1.5.8.2 配列要素

配列引用の各添字がすべて *element-subscript* の場合、配列引用は1つの配列要素を引用します。配列要素はスカラです。配列引用の各添字に *subscript-triplet* または *vector-subscript* が指定されている場合、部分配列となります("1.5.8.4 部分配列"参照)。

1.5.8.3 配列要素順序

1つの配列中の要素は、配列要素順序と呼ばれる列を形成します。この列中での配列要素の位置は、以下の式で表されます。

$$(1 + (s_1 - j_1)) + ((s_2 - j_2) \times d_1) + \ldots + ((s_n - j_n) \times d_{n-1} \times d_{n-2} \times \ldots \times d_1)$$

s_i は第 i 次元の添字です。
j_i は第 i 次元の下限値です。
d_i は第 i 次元の大きさです。
n は配列の次元数です。

配列順序の例:

```
integer ,dimension(2,3) :: a
```

この例で、配列要素順序はa(1,1)、a(2,1)、a(1,2)、a(2,2)、a(1,3)、a(2,3) になります。
入出力並びに配列名が指定された場合、配列要素順序が使用されます。

1.5.8.4 部分配列

部分配列によって、配列の一部を別の配列のように引用することができます。部分配列は、添字に少なくとも1つ、*subscript-triplet* または *vector-subscript* を含まなければなりません。部分配列は、その要素が1つであっても、スカラではありません。

1.5.8.5 添字三つ組

添字三つ組は以下の形式です。

```
[ element-subscript ] : [ element-subscript ] [ : stride ]
```

添字三つ組の1番目の*element-subscript* は下限値を、2番目の*element-subscript* は上限値を、*stride* は添字の値の増分値を示します。これらはそれぞれ省略可能であり、3つすべてを省略してもかまいません。1番目の*element-subscript*、下限値を省略した場合、その配列の下限の値を指定したのと等価です。2番目の*element-subscript*、上限値を省略した場合、その配列の上限の値を指定したのと等価です。*stride* を省略した場合、1を指定したのと等価です。

添字三つ組の例:

```
a(2:8:2)      ! a(2),a(4),a(6),a(8)
b(1,3:1:-1)   ! b(1,3),b(1,2),b(1,1)
c(:,:,:)      ! c
```

1.5.8.6 ベクトル添字

ベクトル添字は、以下の形式です。

```
vector-subscript
```

vector-subscript は、1次元の整数配列式でなければなりません。

ベクトル添字は、その配列式の要素の値に対応した添字の列を指定します。

ベクトル添字の例:

```
integer :: vector(3) = (/3,8,12/)
real :: whole(3,15)
```

```
  ...
print *,whole(3,vector)
```

この例で、PRINT文では、whole(3,3)、whole(3,8)、およびwhole(3,12)の要素が出力されます。

1.5.8.7 部分列をもつ配列引用

文字型の配列要素および部分配列は、配列添字の後に部分列範囲を指定することができます。部分配列が部分列範囲の指定をもつ場合、それぞれの要素は、部分配列中の対応する要素中の部分列です。全体配列に部分列範囲は指定できません。

部分列範囲をもつ部分配列の例:

```
character (len=10), dimension(10) :: string
string(3:8)(2:4) = 'abc'
```

この例で、'abc' は、文字型配列string の3番目から8番目の要素の、2文字目から4文字目に代入されます。

1.5.9 動的配列

動的配列は、配列の大きさや形状を翻訳時に確定せず、実行時に動的に確定させる配列です。

- 割付け配列および配列ポインタは、必要な時にALLOCATE 文によって割り付け、DEALLOCATE 文によって解放することができます(“1.5.9.1 割付け配列”および“1.5.9.2 配列ポインタ”参照)。割付け配列および配列ポインタは形状無指定配列でなければなりません。
- 形状引継ぎ配列は、結合される実引数配列から形状を引き継ぐ、仮引数配列です(“1.5.9.3 形状引継ぎ配列”参照)。
- 大きさ引継ぎ配列は、結合される実引数配列から大きさを引き継ぐ、仮引数配列です(“1.5.9.4 大きさ引継ぎ配列”参照)。
- 自動割付け配列は、定数式でない配列宣言添字をもつ、仮引数でない配列です(“1.5.9.5 自動割付け配列および寸法可変の形状明示配列”参照)。

1.5.9.1 割付け配列

ALLOCATABLE 属性は配列に対しても指定でき、ALLOCATABLE 属性をもつ配列を割付け配列といいます。割付け配列は、形状無指定配列(“2.119 DIMENSION文”参照)でなければなりません。

割付け配列の配列宣言の各次元は、以下の形式(コロンのみ)です。

```
:
```

割付け配列の宣言の例:

```
integer , allocatable :: a(:) , b(:,:,:)
```

この例では、2つの割付け配列を宣言し、a は1次元、b は3次元の配列です。

割付け配列の形状と大きさは、ALLOCATE 文によって割り付けられた時に決まります。

割付け配列の割付けの例:

```
allocate (a(3) , b(1,3,-3:3))
```

この例では、1次元の配列a と3次元の配列b を割り付けており、b の大きさは21、第3次元の下限値は-3です。

ALLOCATE 文によって割り付けられた領域は、DEALLOCATE 文によって解放します。

1.5.9.1.1 割付け配列結合状態

割付け配列の割付け状態は、以下のいずれかであり、組込み関数ALLOCATED(“2.21 ALLOCATED組込み関数”参照)によって知ることができます。

- 割り付けられていない:この状態の割付け配列は、引用も確定もしてはなりませんが、ALLOCATE 文で割り付けることができます。DEALLOCATE 文によってこの領域を解放しようとすることは、DEALLOCATE 文の誤り条件となります。この状態の割付け配列に対して、組込み関数ALLOCATED は偽を返します。
- 割り付けられている:この状態の割付け配列は、引用、確定、または解放することができます。ALLOCATE 文で割り付けようとすることは、ALLOCATE 文の誤り条件となります。この状態の割付け配列に対して、組込み関数ALLOCATED は真を返します。

ALLOCATE 文により生成された派生型実体の末端割付け成分は、次の条件の場合を除き、“割り付けられていない”という割付け状態をもちます。

- SOURCE= に派生型が指定されている、かつ
- 派生型の成分が割り付けられている。

SAVE 属性をもった割付け配列は、“割り付けられていない”という初期状態をもちます。その配列が割り付けられると、その状態は“割り付けられている”になります。“割り付けられている”という状態は、その配列が解放されるまで、保持されます。

手続の仮引数である割付け配列は、手続の入口で結合し、実引数の割付け状態を受け取ります。手続の仮引数である派生型の末端成分の割付け配列は、手続の入口で、実引数の対応する成分の割付け状態を受け取ります。

SAVE 属性をもたない割付け配列のうち、手続内で局所的な変数またはその末端成分であって、仮引数またはその部分実体でないもの、および参照結合または親子結合によって参照できないものは、手続の呼出し時には“割り付けられていない”という割付け状態をもちます。この状態は、手続の実行中に変更することができます。“割り付けられている”という状態になった配列は、手続の結果変数またはその部分実体を除いて、手続がRETURN 文またはEND 文の実行によって終了した時に解放されます。

SAVE 属性をもたない割付け配列のうち、参照結合によって参照するものは、“割り付けられていない”という初期状態をもちます。この状態はプログラムの実行中に変更することができます。“割り付けられている”という状態をもつ配列が、RETURN 文またはEND 文の実行の結果として参照できるモジュールの有効域をもたなくなっても、この状態は保持されます。

1.5.9.2 配列ポインタ

POINTER 属性は配列に対しても指定することができます。配列ポインタは、割付け配列と同様に、形状無指定配列(“2.119 DIMENSION 文”参照)でなければなりません。

配列ポインタの配列宣言の各次元は、以下の形式(コロンのみ)です。

```
:
```

配列ポインタの宣言の例:

```
integer , pointer , dimension(:,:) :: c
```

この例では、次元数2の配列ポインタ c を宣言しています。

配列ポインタは、割付け配列と同様にALLOCATE 文によって割り付けることができます。あるいは、ポインタ代入文によって指示先と結合した場合も、配列ポインタの形状は決まります。

配列ポインタの結合の例:

```
integer , pointer , dimension(:,:) :: c
integer , target  , dimension(2,4) :: d
c => d
```

この例で、配列ポインタ c は配列 d と結合し、その形状は(2,4) です。

1.5.9.3 形状引継ぎ配列

形状引継ぎ配列は、結合される実引数配列から形状を引き継ぐポインタでない仮引数配列です。形状引継ぎ配列の下限値は宣言することができ、その値は結合される実引数配列の下限値と同じである必要はありません。省略した場合の下限値は1です。

形状引継ぎ配列の配列宣言の各次元は、以下の形式です。

```
[ lower-bound ] :
```

形状引継ぎ配列の例:

```
integer :: a(3,4)
call zee(a)
contains
  subroutine zee(x)
    integer,dimension(-1:,:) :: x
      ...
```

```
  end subroutine zee
end
```

この例で、仮引数 x は形状引継ぎ配列であり、結合する実引数配列 a の形状を引き継ぎます。
x の第1次元の下限値は -1です。

手続の仮引数に形状引継ぎ配列を含む場合、引用仕様は明示的でなければなりません(“1.12.7.1 明示的引用仕様”参照)。

1.5.9.4 大きさ引継ぎ配列

大きさ引継ぎ配列は、結合される実引数からその大きさを引き継ぐ仮引数配列です。仮引数配列の次元数および寸法は、結合される実引数の次元数および寸法と同じである必要はありません。

大きさ引継ぎ配列の配列宣言は、以下の形式です。

[*explicit-shape-spec-list*] [*lower-bound* :] *

大きさ引継ぎ配列は最後の次元の寸法および形状をもたないため、寸法および形状を必要とする個所で全体配列の引用をしてはなりません。配列値関数の関数結果は大きさ引継ぎ配列として宣言してはなりません。INTENT(OUT) 属性をもつ大きさ引継ぎ配列は、次のいずれかであってはなりません。

- 暗黙的初期値指定をもつ派生型、または
- 多相的、または
- 割付け成分をもつ派生型、または
- 後始末可能な派生型

大きさ引継ぎ配列の例:

```
integer,dimension(4) :: a
call zee(a)
end program
subroutine zee(x)
  integer,dimension(-1:*) :: x
    ...
end subroutine
```

この例で、x は大きさ引継ぎ配列であり、x の大きさは不明です。

1.5.9.5 自動割付け配列および寸法可変の形状明示配列

形状明示配列の上下限は、仮引数、共通ブロック要素、親子結合、および参照結合された変数の値に依存する式であってもかまいません。そのような配列は、仮引数、関数結果、または自動割付け配列でなければなりません。自動割付け配列とは、副プログラム内で宣言された非定数式の上下限をもつ仮引数でない形状明示配列です。

自動割付け配列および寸法可変の形状明示配列の例:

```
subroutine foo(i,k)
integer :: i,j,k
dimension :: i(k,3),j(k)
```

この例で i および j の上下限は仮引数 k の値に依存する非定数式であり、j は自動割付け配列です。

1.5.10 配列構成子

配列構成子は以下の形式です。

(/ *ac-spec* /)　　または
left-square-bracket　*ac-spec*　*right-square-bracket*

ac-spec は以下の形式です。

[*type-spec*::] [*ac-value-list*]

type-spec は、型指定子です。型指定子には、組込み型指定子および派生型指定子があります。
組込み型指定子については、“2.3 型宣言文”を参照してください。また、派生型指定子については、“1.5.11.8 派生型指定子”を参照してください。

type-spec が組込み型を指定するとき、1つの配列構成子中の配列構成項目の式の型は、型指定子の型の変数と型が適合する組込み型でなければなりません。

type-spec が派生型を指定するとき、1つの配列構成子中の配列構成項目の式の型は、その派生型であって、型指定子で指定した型と同じ種別型パラメタ値をもたなければなりません。

type-spec が省略されたとき、配列構成子中のすべての配列構成項目の式は同じ長さ型パラメタをもたなければなりません。このとき、配列構成子の型および型パラメタは、配列構成項目の式と同じです。

*type-spec*を書いたとき、配列構成子の型および型パラメタを指定します。配列構成子中の配列構成項目の式は、その型および型パラメタの変数への組込み代入と型が適合しなければなりません。それぞれの値は、組込み代入の規則に従って配列構成子の型パラメタに変換されます。

left-square-bracket は、左角括弧です。左角括弧は、[です。

right-square-bracket は、右角括弧です。右角括弧は、] です。

ac-value-list は、コンマで区切られた配列構成項目の並びです。*ac-value* は、以下の形式です。

expr または
ac-implied-do

expr は式です。

ac-implied-do は、以下の形式です。

(*ac-value-list* , *ac-implied-do-control*)

ac-implied-do-control は以下の形式です。

[*integer-type-spec* ::] *ac-do-variable* = *scalar-int-expr* , *scalar-int-expr* [, *scalar-int-expr*]

integer-type-spec は、整数型指示子で以下の形式です。

INTEGER [*kind-selector*]

ac-do-variable は、名前付きスカラ整変数です。

配列構成DO 形反復の*ac-do-variable* に、その配列構成DO 形反復に含まれる配列構成DO 形反復の*ac-do-variable* と同じ変数を指定してはなりません。

scalar-int-expr はスカラ整数式です。

配列構成子は、1次元配列です。*ac-value* がスカラ式の場合、スカラ式の値は配列構成子の要素を指定します。*ac-value* が配列式の場合、配列式の値は、配列要素順序(“1.5.8.3配列要素順序”参照)に対応して、配列構成子の要素の列を指定します。*ac-value* が*ac-implied-do* の場合、DO 構文と同じ*ac-do-variable* の制御に従って展開され、配列構成子の要素の列を形成します(“2.120 DO構文”参照)。

type-spec を省略した場合、1つの配列構成子中の、配列構成項目の式の型および種別型パラメタは、すべて同じでなければなりません。

配列構成子の例:

```
integer,dimension(3) :: a, b=(/1,2,3/) ,c=(/(i,i=4,6)/)
a = b + c + (/7,8,9/)
```

この例で、a には(/12,15,18/) が代入されます。

配列構成子は1次元配列ですが、RESHAPE 組込み関数によって、2次元以上の配列値を構成することも可能です。

配列構成子とRESHAPE 組込み関数の例:

```
integer,dimension(2,2) :: a = reshape((/1,2,3,4/),(/2,2/))
```

この例で、aには配列要素順序に従ってaの形状に変形された配列値(/1,2,3,4/)が設定されます。

1.5.11 派生型

派生型は、組込みデータ型(整数型、実数型、複素数型、論理型、および文字型)および他の派生型から構成される利用者定義の型です。派生型のスカラ要素を構造体と呼びます。

1.5.11.1 派生型定義

データ要素を派生型として明示的に宣言する場合、その派生型はそれ以前に定義されているか、親子結合または参照結合により参照可能になっていなければなりません。派生型として宣言されたデータ要素の成分は、派生型定義の対応する成分定義文によって指定された型であると宣言されます。

派生型定義は以下の形式です。

```
TYPE [ [ , type-attr-spec-list ] :: ] type-name[ ( type-param-name-list ) ]
    [ type-param-def-statement ]...
    [ private-or-sequence ]...
    [ component-def-statement ]...
    [ type-bound-procedure ]
END TYPE [ type-name ]
```

*type-attr-spec-list*は、型属性指定子並びで、PUBLIC、PRIVATE、EXTENDS(*parent-type-name*)、ABSTRACT、およびBIND(C)を指定できます。PUBLIC とPRIVATE は、同時に指定できません。親型名は、それ以前に定義されている拡張可能型(“1.5.11.4 型の拡張”参照)の名前でなければなりません。型定義が遅延束縛を含むか、または遅延束縛を継承する場合は、ABSTRACT を指定しなければなりません。ABSTRACT を指定する場合は、その型は拡張可能でなければなりません。EXTENDS を指定する場合は、SEQUENCE を指定してはなりません。

*type-name*は、定義される派生型の名前です。*type-param-name-list*に型パラメタ名並びを指定します(“1.5.11.2 派生型パラメタ”参照)。

*type-param-def-statement*は、型パラメタ定義文です(“1.5.11.2 派生型パラメタ”参照)。

*private-or-sequence*は、以下の形式です。

```
PRIVATE         または
SEQUENCE
```

*component-def-statement*は、以下の形式です。

```
component-data-def-statement        または
component-proc-def-statement
```

*component-def-statement*は、データ成分定義文です。データ成分定義文は、型宣言文(“2.3 型宣言文”参照)と同じ形式で、その属性指定子には、POINTER、ALLOCATABLE、CODIMENSION、CONTIGUOUS、DIMENSION、PUBLIC、およびPRIVATE が指定できます。

POINTER 属性またはALLOCATABLE 属性の指定がある場合、配列形状指定は無指定上下限並びでなければなりません。POINTER 属性およびALLOCATABLE 属性の指定がない場合、配列形状指定は定数式の明示上下限並びでなければなりません。1つの成分定義文に、POINTER 属性とALLOCATABLE 属性の両方を指定してはなりません。CONTIGUOUS 属性の指定がある場合、データ成分定義文はPOINTER 属性をもつ配列でなければなりません。

POINTER 属性またはALLOCATABLE 属性の指定がない場合、データ成分定義文の宣言型指定子(“2.3 型宣言文”参照)に組込み型または、それ以前に定義した派生型を指定しなければなりません。

CODIMENSION 属性の指定がある、または、データ成分に共配列指定(“1.17.1 共配列指定”参照)の指定がある場合:

- データ成分定義文にALLOCATABLE 属性を指定しなければなりません。かつ、
- その割付け末端成分の実体は、POINTER 属性、ALLOCATABLE 属性をもたないスカラでなければなりません。また、共配列(“1.17 共配列”参照)、または関数結果であってはなりません。

component-proc-def-statement は、手続成分定義文で、以下の形式です。

```
PROCEDURE ( [ proc-interface] ), proc-component-attr-spec-list :: proc-decl-list
```

proc-interface は、手続引用仕様です。

proc-component-attr-spec-list は、手続成分属性指定子並びです。

proc-decl-list は、手続宣言並びです。

手続成分属性指定子には、POINTER、PASS [(*arg-name*)]、NOPASS、PUBLIC、および PRIVATEが指定でき、POINTER は常に指定しなければなりません。手続ポインタ成分が、暗黙の引用仕様をもつか、または引数をもたない場合は、NOPASS を指定しなければなりません。PASS(*arg-name*)を指定する場合、引用仕様はその引数名を名前とする仮引数をもたなければなりません。その引数名をもった仮引数が、手続ポインタ成分の当該実体仮引数になります。

同じ手続成分属性指定子並びにPASS およびNOPASS を同時に指定してはなりません。

type-bound-procedure は、型束縛手続部です(“1.5.11.3 型束縛手続”参照)。

END TYPE 文に*type-name* を指定する場合、対応するTYPE 文に指定された*type-name* と同じでなければなりません。

SEQUENCE 文の指定のある型は、連続型といいます。連続型は、成分が指定された順に記憶列を使用することを指定します。SEQUENCE 文が指定された型定義の成分定義文で指定された派生型は、すべて連続型でなければなりません。

基本整数型、基本実数型、倍精度実数型、基本複素数型、または基本論理型であってポインタでも割付け変数でもない末端成分だけからなる連続型を、数値連続型といいます。

基本文字型であってポインタでも割付け変数でもない末端成分だけからなる連続型を、文字連続型といいます。

型定義中のPRIVATE 属性および文は、モジュールの宣言部の型定義にだけ指定することができます。型がPUBLIC 属性をもつ公開の型であっても、そのPRIVATE 属性の成分名は、その型定義のあるモジュール内だけで参照することができます。その型がPRIVATE 属性をもつ場合、構造体構成子はその型定義のあるモジュール内だけで参照することができます。

派生型はSTRUCTURE 文によっても定義することができます。STRUCTURE 文による構造形定義は、以下の形式です。

```
STRUCTURE [ / type-name / ] [ component-list ]
   struct-elem-def-statement
   [ struct-elem-def-statement ]...
END STRUCTURE
```

STRUCTURE 文の / *type-name* / は、派生型定義が入れ子になっている場合の内側のSTRUCTURE 文に対してだけ、省略することができます。*component-list*は、入れ子になっている場合の内側のSTRUCTURE 文に対してだけ指定することができます。*component*はそのSTRUCTURE 文で定義される型をもつ成分です。

struct-elem-def-statement は、以下の形式です。

```
component-def-statement      または
union-declaration            または
structure-definition
```

union-declaration は、共用体を宣言します。*union-declaration* は、以下の形式です。

```
UNION
   map-declaration
   [ map-declaration ]...
END UNION
```

map-declaration は、共用体において記憶域を共有するブロックを宣言します。*map-declaration* は以下の形式です。

```
MAP
   struct-elem-def-statement
   [ struct-elem-def-statement ]...
END MAP
```

それぞれの*map-declaration* は、同じ記憶域を共有します。共用体の記憶域の大きさは、その共用体に含まれる*map-declaration* の中で記憶域の大きさが最大になるものと同じです。

structure-definition はSTRUCTURE 文による派生型の定義です。

STRUCTURE 文によって定義される派生型は、連続型です。

ポインタでない成分に対して初期値指定がある場合、その型のすべての実体のその成分は、初期値をもちます。割付け変数や、自動割付け変数の成分も、割り付けられた時点で初期値をもちます。ポインタである成分に初期値指定がある場合、その成分のポインタの結合状態は、空状態となります。これを暗黙的初期値指定といいます。

割付け変数成分に対して、初期値指定は指定できません。

暗黙的初期値指定をもつ成分が派生型であって、その型の成分もまた暗黙的初期値指定をもつ場合、その暗黙的初期値指定は上書きされ、上書きした初期値指定だけが有効になります。型宣言文による明示的な初期値指定は、暗黙的初期値指定を上書きし、明示的な初期値指定だけが有効になります。

派生型定義の例:

```
type coordinates
  real :: latitude=0.0, longitude=0.0
end type coordinates

type place
  character(len=20) :: name
  type(coordinates) :: location
end type place

type link
  integer :: j
  type (link) , pointer :: next=>null( )
end type link
```

この例で、派生型 coordinates は、2つの実数型の成分 latitude と longitude をもち、それぞれの成分は、暗黙の初期値 0.0をもちます。派生型place は、長さ20の文字型の成分 nameと派生型 coordinates の成分 location の2つをもちます。派生型 link は、整数型の成分jと、派生型 link の成分 next の2つをもちます。next は自身の型 link へのポインタであり、暗黙の初期値として空状態をもちます。

同じ派生型定義を参照して宣言された2つのデータ要素は、同じ型をもちます。派生型定義は親子結合または参照結合により参照することも可能です。異なる有効域内で定義される2つの派生型が、同じ型名および SEQUENCE 文またはBIND(C) の指定をもち、PRIVATE 文の指定をもたず、かつ順序、名前、および属性の一致する型パラメタおよび成分をもつ場合、それらは同じ型です。

1.5.11.2 派生型パラメタ

型パラメタ定義文は、派生型定義における型パラメタ名を宣言します。

型パラメタ定義文は以下の形式です。

```
INTEGER [ kind-selector ] , type-param-attr-spec :: type-param-decl-list
```

kind-selector は、種別型パラメタです。

type-param-attr-spec は、型パラメタ属性指定子です。

type-param-decl-list は、型パラメタ宣言並びです。型パラメタ宣言は以下の形式です。

```
type-param-name [ = scalar-int-initialization-expr ]
```

type-param-name は、型パラメタ名です。

scalar-int-initialization-expr は、スカラ整数定数式です。

派生型定義の型パラメタ定義文における型パラメタ名は、その派生型定義のTYPE 文中の型パラメタ名の1つでなければなりません。

派生型定義のTYPE 文内のそれぞれの型パラメタ名は、その派生型定義の型パラメタ定義文中の型パラメタ名として現れなければなりません。

型パラメタ属性指定子は以下の形式です。

KIND　　または
LEN

型パラメタ属性指定子は、型パラメタが種別型パラメタであるか長さ型パラメタであるかを明示的に指定します。KIND属性指定子は種別型パラメタ、LENは長さ型パラメタであることを指定します。

長さ型パラメタは、その型に対する派生型定義の中の宣言式で指定できますが、定数式では指定してはなりません。成分初期値を指定した場合、成分のそれぞれの型パラメタおよび配列上下限は定数式でなければなりません。

直接拡張型の型パラメタ順序は、親型の型パラメタ順序に続けて、その直接拡張型の定義の型パラメタ並びにおける順序を追加したものです。

派生型パラメタの値は、派生型指定子で指定します。

長さ型パラメタをもつ派生型の仮引数は、INTENT(OUT)属性を指定できません。

長さ型パラメタをもつ派生型の定義、実体宣言、または成分宣言をBLOCK構文の有効域で指定できません。

ゼロでない次元数の部分参照より右にある構造体成分は、配列上下限または文字長の指定で長さ型パラメタを指定できません。

長さ型パラメタが指定された末端成分をもつ派生型の実体または式は、入出力項目またはNAMELIST文に指定することはできません。

引継ぎ長さ型パラメタをもつ仮引数に対応する実引数として、結果がパラメタ付き派生型である組込み手続は指定できません。

ALLOCATE文においてMOLD指定子を指定する場合、割付け実体の型はパラメタ付き派生型であってはなりません。

パラメタ付き派生型となる式は、選択子に指定できません。

長さ型パラメタをもつ派生型には、成分初期値を指定できません。

パラメタ付き派生型、またはパラメタ付き派生型を成分としてもつ型は、共配列の型指定子に指定できません。

派生型パラメタの例:

```
type struct(dim,lg)
  integer, kind :: dim
  integer, len :: lg
  real :: array(dim)
  character(len=lg) :: ch
end type
```

1.5.11.3 型束縛手続

型束縛手続は、派生型定義におけるTYPE 文中で、その派生型を操作するための手続を指定します。

型束縛手続部は以下の形式です。

```
CONTAINS 文
[ binding-private-stmt ]
proc-binding-stmt
[ proc-binding-stmt ] ...
```

binding-private-stmt は、束縛PRIVATE 文です。以下の形式です。

```
PRIVATE
```

束縛PRIVATE 文は、型定義がモジュールの宣言部にあるときだけ指定することができます。

proc-binding-stmt は、手続束縛文です。

手続束縛文は、個別束縛、総称束縛または後始末束縛のいずれかでなければなりません。

個別束縛は、以下の形式です。

```
PROCEDURE [ ( interface-name ) ] [ [ , binding-attr-list ] :: ] binding-assignment-list
```

interface-name は、引用仕様名です。

binding-attr-list は、束縛属性並びです。

binding-assignment-list は、コンマで区切られた束縛手続名宣言の並びです。束縛手続名宣言は、以下の形式です。

binding-name[=> *procedure-name*]

binding-name は、束縛名です。

procedure-name は、手続名です。

“=> *procedure-name*”を指定したとき、区切りの2 連コロンを省略してはなりません。

“=> *procedure-name*”を指定したとき、引用仕様名を指定してはなりません。

手続名は、参照可能なモジュール手続名かまたは明示的引用仕様をもった外部手続名でなければなりません。

“=>*procedure-name*”も引用仕様名も指定しないとき、“=> *procedure-name*”の手続名に束縛名が指定された場合と同じになります。

総称束縛は以下の形式です。

GENERIC [, *access-attr*] :: *generic-spec* => *binding-name-list*

access-attr は、参照許可属性です。

generic-spec は、総称指定です。

binding-name-list は、束縛名並びです。

モジュールの宣言部内で、それぞれの総称束縛は、その派生型において同じ総称指定をもつ他のすべての総称束縛と同じ参照許可属性を、明示的または暗黙的に指定しなければなりません。

束縛名並びの束縛名は、その型の個別束縛の名前でなければなりません。

総称指定が総称名でないとき、それぞれの個別束縛は、当該実体仮引数をもたなければなりません。総称指定は、“1.12.7.3 総称引用仕様”を参照してください。

総称指定がOPERATOR(*.operator-name.*) であるとき、それぞれの束縛の引用仕様は“1.12.7.3.2 利用者定義演算”の指定方法に従っていなければなりません。

総称指定がASSIGNMENT(=) であるとき、それぞれの束縛の引用仕様は“1.12.7.3.3 利用者定義代入”の指定方法に従っていなければなりません。

総称指定が派生型入出力総称指定（“2.291 INTERFACE文”参照）であるとき、それぞれの束縛の引用仕様は“1.7.4.2 利用者定義派生型入出力手続”で示す指定方法に従っていなければなりません。仮引数*dtv_arg*（“1.7.4.2 利用者定義派生型入出力手続”参照）の型は、その派生型定義の型でなければなりません。

参照許可属性は以下の形式です。

```
PUBLIC                  または
PRIVATE
```

参照許可属性は、モジュールの宣言部にだけ指定することができます。

束縛属性は、以下の形式です。

```
PASS [ ( arg-name ) ]   または
NOPASS                  または
NON_OVERRIDABLE         または
DEFERRED                または
access-attr
```

arg-name は、引数名です。

access-attr は、参照許可属性です。

同じ束縛属性は、1つの束縛属性並び中に2回以上指定してはなりません。

その束縛の引用仕様が定義している型の仮引数をもたないとき、NOPASS を指定しなければなりません。

PASS(*arg-name*) を指定したとき、その束縛の引用仕様には、引数名をもった仮引数を指定しなければなりません。その引数名をもった仮引数が、型束縛手続の当該実体仮引数になります。

1つの束縛属性並びにPASS とNOPASS を同時に指定してはなりません。

1つの束縛属性並びにNON_OVERRIDABLE とDEFERRED を同時に指定してはなりません。

引用仕様名を指定した場合、DEFERRED を指定しなければなりません。DEFERRED を指定した場合、引用仕様名を指定しなければなりません。

上書きする型束縛手続が遅延するときだけ、DEFERRED 属性を指定しなければなりません。

NON_OVERRIDABLE 属性をもった継承束縛を上書きしてはなりません。

手続束縛文の各束縛は、型束縛手続を指定します。型束縛手続は、当該実体仮引数をもつことができます。総称束縛は、その個別束縛に対する型束縛の総称引用仕様を指定します。DEFERRED 属性をもつ束縛は、遅延束縛です。遅延束縛は、抽象型の定義中にだけ指定することができます。

型束縛手続は、型定義の有効範囲の中の束縛名で識別できます。この名前は、個別束縛に対しては束縛名とし、総称指定が総称名である総称束縛に対しては総称名です。後始末束縛または総称指定が総称名と異なる総称束縛または、後始末束縛は、束縛名をもちません。

個別束縛の引用仕様は、手続名によって指定された手続の引用仕様または引用仕様名によって指定された手続の引用仕様です。

型および型束縛手続の例:

```
module mod
type two_real
  real :: x,y
  contains
    procedure, pass :: two_real_fun=> fun
end type two_real
contains
  real function fun (a, b)
    class (two_real), intent (in) :: a, b
    fun = a%x + b%x
  end function fun
end module
```

1つの派生型定義内の複数の総称束縛に、同じ総称指定を指定できます。同じ総称指定をもつ追加された総称束縛は、総称引用仕様を拡張します。

総称型束縛手続名は、それと同じ型をもつ個別型束縛手続名と同じ名前であってはなりません。

型定義が束縛PRIVATE 文を含むとき、型の手続束縛の暗黙の参照許可属性は非公開です。束縛PRIVATE 文を含まないとき、公開です。手続束縛の参照許可属性は、明示的に指定できます。

型束縛手続の参照許可属性は、成分定義部のPRIVATE 文の影響を受けません。データ成分の参照許可属性は、型束縛手続部のPRIVATE 文の影響を受けません。

後始末束縛は以下の形式です。

```
FINAL [ :: ] final-subroutine-name-list
```

final-subroutine-name-list は、後始末サブルーチン名並びです。

後始末サブルーチン名は、仮引数を1つだけもつモジュール手続の名前です。その引数は、省略不可であり、定義しようとする派生型の、ポインタでもなく、割付け変数でもなく、多相的でもない変数です。仮引数は、INTENT(OUT) またはVALUE 属性であってはなりません。同じ型で、同じ後始末サブルーチン名を2回以上指定してはなりません。また、同じ次元数をもった仮引数をもつ後始末サブルーチン名を2回以上指定してはなりません。

派生型が後始末可能であるとは、次のいずれかの場合です。

- 後始末サブルーチンをもっている、または
- ポインタでもなく割付け成分でもない、後始末可能である型の成分をもっている

次のときに、後始末されます。

- ポインタが解放されるとき、その指示先は後始末されます。

- 割付け要素が解放されるとき、その要素は後始末されます。
- 次の条件のすべてを満たす実体は、RETURN 文またはEND 文で後始末されます。
  - 主プログラム単位でない、かつ
  - SAVE 属性をもたない、かつ
  - 仮引数でない、かつ
  - 関数結果でない、かつ
  - ポインタでない、かつ
  - モジュール内で定義されていない
- 実行可能な構造構文が関数を参照しているとき、関数結果は、その参照を含む最も内側の実行可能な構造構文の実行後に後始末されます。
- 手続が呼び出されるとき、INTENT(OUT) 属性をもつ仮引数と結合する実引数であり、ポインタでない実体は、後始末されます。
- 組込み代入が実行されるとき、変数は式の評価後かつその変数の確定前に後始末されます。

1.5.11.4 型の拡張

拡張可能型は、BIND 属性をもたない非連続の派生型です。

直接拡張型は、EXTENDS 属性をもつ型です。直接拡張型の親型は、EXTENDS 属性に指定する型名です。

抽象型は、ABSTRACT 属性をもつ型です。

1.5.11.4.1 継承

直接拡張型は、親型を継承します。直接拡張型は、親型のすべての型パラメタ、すべての成分、および上書きも後始末もしない手続束縛を含みます。直接拡張型は、親型がもっていたすべての属性を保持します。直接拡張型の派生型定義によって、型パラメタ、成分および手続束縛を加えることができます。

直接拡張型は、親型の型および型パラメタをもつ、スカラであってポインタでも割付け成分でもない親成分をもちます。この成分の名前は、親型名です。この成分は、親型の参照許可属性をもちます。親成分の成分は、親型から継承された対応する成分と継承結合されています。直接拡張型で宣言された成分または型パラメタの名前は、親型の参照可能な成分または型パラメタの名前と同じであってはなりません。

継承の例:

```
type t1                    ! 基底型
  real :: x, y
end type t1
type, extends(t1) ::t2     ! 型t1の直接拡張型
                           ! 成分x、y および成分名t1は親型から継承されています。
  real :: z
end type t2
type(t2) :: a
```

1.5.11.4.2 型束縛手続の上書き

型定義で指定された総称束縛ではない束縛が、親型の束縛と同じ束縛名をもつとき、その型定義に指定した束縛が、親型からの束縛を上書きします。

型定義で指定された総称束縛が、継承束縛と同じ総称指定をもつとき、総称引用仕様が拡張されます。上書きする束縛手続および上書きされる束縛手続は、次のすべての条件を満たさなければなりません。

- 両方とも当該実体仮引数をもっているか、または、両方ともももっていないかでなければなりません。
- 上書きされる束縛手続が純粋な場合、上書きする束縛手続も純粋でなければなりません。
- 両方とも要素別処理手続であるか、または、両方とも要素別処理手続であってはなりません。
- 両方とも仮引数の個数は同じでなければなりません。
- 当該実体仮引数が現れる場合、名前と位置が対応していなければなりません。

- 位置が対応する仮引数は、当該実体仮引数の型を除いて、同じ名前および特性をもたなければなりません。
- 両方ともサブルーチンであるか、または、同じ結果の特性をもった関数でなければなりません。
- 上書き束縛される手続がPUBLIC 属性をもつとき、上書きする束縛手続は、PRIVATE 属性をもってはなりません。

手続上書きの例:

```
module mod
type t1
  real :: x, y
  contains
  procedure, pass :: fun => afun
end type t1
type, extends(t1) ::t2
  real:: z
  contains
  procedure, pass :: fun => bfun
end type t2
contains
  function afun (a, b) result(r)
    class (t1), intent (in) :: a, b
    real :: r
    r = a%x + b%x + a%y + b%y
  end function afun
  function bfun (a, b) result(r)
    class (t2), intent (in) :: a
    class (t1), intent (in) :: b
    real :: r
    select type(b)
    type is(t2)
      r = a%x + b%x + a%y + b%y + a%z +b%z
    end select
  end function bfun
end module mod
```

1.5.11.5 派生型変数の宣言

派生型の変数は、型宣言文の宣言型指定子にTYPE (*type-name*) を指定して宣言します。

派生型変数宣言の例:

```
type(coordinates) :: my_coordinates
type(place) :: my_town
type(place) , dimension(10,10) :: cities
type(link) :: head
```

この例で使用している型は、“派生型定義の例:”で定義されている型です。

1.5.11.6 構造体成分

構造体の成分は、‘%’演算子または‘.’演算子によって参照することができます。

構造体成分の引用は、以下の形式です。

part % *part* [% *part*]...　　または
part . *part* [. *part*]...

part は、以下の形式です。

name [(*subscript-list*)] [(*substrings*)]

*name*は、名前です。(*subscript-list*) の指定を含む場合は、*name*は配列でなければなりません。配列引用については、“1.5.8.1 配列引用”を参照してください。(*substrings*)の指定を含む場合は、*name*は文字型でなければなりません。部分列引用については、“1.5.7 部分列”を参照してください。

name は右端のものを除いて、派生型でなければなりません。また、左端のものを除いて直前の*name* の型である派生型の成分の名前でなければなりません。

構造体の成分を手続引用する場合、右端の*name* は手続ポインタ成分名でなければなりません。

‘.’演算子を使用した形の場合、成分の名前は、TRUE、FALSE、EQ、NE、GT、GE、LT、LE、NOT、AND、OR、EQV、およびNEQV であってはなりません。また、利用者定義演算子の英字の並びと同じであってはなりません。

構造体成分の例:

```
my_coordinates%latitude
my_town%location%latitude
cities(:,:)%name
cities%name
cities(1,1:2)%location%latitude
```

この例は、“1.5.11.5 派生型変数の宣言”の例で宣言されている派生型変数の成分を引用します。
my_coordinates%latitude は、my_coordinates の成分latitude を参照します。
my_town%location%latitude は、my_town 中の成分location 中の成分latitude を参照します。
cities(:,:)%name は、cities のすべての要素の成分name を参照します。これは、cities%name と書いても同じ意味です。
cities(1,1:2)%location%latitude は、cities(1,1)およびcities(1,2) の末端成分latitude を参照します。

1.5.11.7 列挙体および列挙子

列挙体定義は、列挙体およびその対応する基本整数型の列挙子の集合を指定します。

列挙体定義は、以下の形式です。

```
ENUM , BIND ( C )
    ENUMERATOR [ :: ] named-constant[ = scalar-int-initialization-expr]
    [ ENUMERATOR [ :: ] named-constant[ = scalar-int-initialization-expr] ]...
END ENUM
```

named-constant は列挙子並びであり、基本整数型の名前付き定数です。

scalar-int-initialization-expr はスカラ整数定数式です。

列挙子に= を指定したとき、列挙子並びの前に、区切りの2連コロンを指定しなければなりません。

列挙子には、PARAMETER 属性が暗黙的に指定されます。列挙子は、組込み代入の規則に従って定義され、値は次のとおりに決定されます。

- スカラ整数定数式が指定されている場合、列挙子の値はスカラ整数定数式の結果の値です。
- スカラ整数定数式が指定されていない場合、列挙子が列挙体定義の最初の列挙子のとき、列挙子の値は0です。
- スカラ整数定数式が指定されていない場合、列挙子が列挙体定義の最初の列挙子でないとき、列挙子の値は、列挙体定義中の直前の列挙子の値に1を加えた値です。

列挙体定義の例:

```
enum, bind(c)
  enumerator :: desktoppc = 2007, notepc = 2009
  enumerator netbookpc
end enum
```

次の宣言は、上記の宣言と等価です。

```
integer, parameter :: desktoppc = 2007, notepc = 2009, netbookpc = 2010
```

1.5.11.8 派生型指定子

派生型指定子は、派生型および型パラメタを指定します。派生型指定子は以下の形式です。

type-name [(*type-param-spec-list*)]

type-name は、派生型の型名であり、その型名は有効域内で前もって定義されているか、参照結合または親子結合により参照可能でなければなりません。

type-param-spec は、型パラメタ指定であり、以下の形式です。

[*keyword=*] *type-param-value*

(*type-param-spec-list*) は、その派生型がパラメタをもつときだけ指定できます。

その型の各パラメタに対応して、1つの型パラメタ指定が存在しなければなりません。型パラメタが暗黙の値をもっていないとき、その型パラメタに対応する型パラメタ指定が存在しなければなりません。

型パラメタ指定に先行する“*keyword=*” を省略するときだけ、型パラメタ指定から“*keyword=*” を省略できます。

keyword はキーワードであり、その型のパラメタ名です。

type-param-value はパラメタの値であり、以下の形式です。

scalar-int-expr または
* または
:

scalar-int-expr はスカラ整数式です。

種別型パラメタに対する型パラメタ値は、定数式でなければなりません。

型パラメタ値の“:” は、POINTER 属性またはALLOCATABLE 属性をもつ要素または成分の宣言内でだけ使用できます。

型パラメタ値の“:” は、無指定型パラメタを指定します。無指定型パラメタは、プログラムの実行中にその値を変えることができる長さ型パラメタです。

型パラメタ値の“*” は、引継ぎ型パラメタを指定します。引継ぎ型パラメタは、仮引数、結合名の宣言、または仮引数の割付けのいずれかの場合、指定することができます。

1.5.12 構造体構成子

構造体構成子は、派生型の各成分に対応する値の列から構成された派生型のスカラ値を表します。

構造体構成子は、以下の形式です。

derived-type-spec ([*expr-list*])

*derived-type-spec*は、派生型指定子です。派生型指定子については、“1.5.11.8 派生型指定子”を参照してください。型名には抽象型を指定してはなりません。

派生型指定子の型パラメタ指定は、その派生型がパラメタをもつときだけ指定でき、その派生型がパラメタをもつときは省略できません。

expr-list は成分指定並びです。成分指定は、以下の形式です。

[*keyword=*] *component-data-source*

keyword は成分キーワードであり、その成分の名前です。

component-data-source は、成分データソースであり、以下の形式です。

expr または
data-target または
proc-target

expr は式であり、派生型の成分の値を指定します。*data-target* はデータ指示先です。*proc-target*は手続指示先です。

派生型の1つの成分には、継承結合している場合を含め、2つ以上の成分指定を指定してはなりません。成分が暗黙の初期値をもたない場合、成分指定をもたなければなりません。構造体構成子中の先行する各成分指定から“*keyword=*”を省略するときだけ、成分指定から“*keyword=*”を省略できます。

型名および成分指定が指定された型のすべての成分は、構造体構成子を含んでいる有効域内で参照可能でなければなりません。

総称名と同じ型名のとき、成分指定並びは、総称引用として解決される関数引用の有効な実引数指定並びであってはなりません。

データ指示先は、手続ポインタではないポインタ成分に対応しなければなりません。手続指示先は、手続ポインタである成分に対応しなければなりません。

データ指示先は、対応する成分と同じ次元数でなければなりません。

“*keyword=*”が存在しない場合、対応する成分データソースが成分順序に従って代入されます。“*keyword=*”が存在する場合、キーワードによって名付けられた成分に代入されます。ポインタではない成分に対して、その成分および式の宣言時の型および型パラメタは、組込み代入文での変数および式と同様に適合しなければなりません。ポインタでもなく割付け成分でもない成分に対しては、式の形状は成分の形状と適合しなければなりません。

ポインタ成分に対して、対応する成分データソースは、ポインタ代入文で許されるデータ指示先または手続指示先でなければなりません。

派生型の成分が割付け要素の場合、対応する成分データソースは、引数を省略した組込み関数NULL（“2.368 NULL組込み関数”参照）の引用、同じ次元数の割付け変数、または同じ次元数の実体と評価されなければなりません。式が組込み関数NULLの引用の場合、対応する成分は、“割り付けられていない”という割付け状態をもちます。式が割付け変数の場合、対応する成分はその割付け変数と同じ割付け状態をもち、“割り付けられている”場合は同じ上下限および値をもちます。配列と評価される式の場合、対応する成分は“割り付けられている”という割付け状態をもち、式と同じ上下限および値をもちます。

暗黙の初期値設定が存在する成分が、対応する成分データソースをもたないとき、暗黙の初期値設定が、その成分に対して設定されます。割付け成分が対応する成分データソースをもたないとき、その成分は未割付け状態をもちます。

構造体構成子は、引用している型を定義する前に指定してはなりません。

構造体構成子の例:

```
type my_type                        ! 派生型の定義
  integer :: i,j
  character(len = 40) :: string
end type my_type
type (my_type) :: a                 ! 派生型変数の宣言
a = my_type(4,5.0*2.3,'abcdefg')
```

この例で、構造体構成子内の2番目の式は、基本整数型に変換され、代入されます。

1.5.12.1 成分キーワード

成分キーワード“*keyword=*”を指定すると、式は構造体構成子並び中の位置に関係なく、*keyword*に指定された成分と結合します。

成分キーワードの指定がない場合、式は、派生型の成分並びの対応する位置の成分と結合します。

成分キーワードは、その式が成分並びの対応する位置の成分と結合しない場合、および式の並びの先行する式に成分キーワードの指定がある場合に、指定しなければなりません。

成分キーワードの例:

```
type ty1
  integer :: a,b,c
end type
type(ty1):: t
t = ty1(c=1,b=2,a=3)
end
```

この例で、式は指定されたのと逆の順に結合します。

1.5.12.2 省略可能な成分

成分が暗黙的初期値指定をもっている、または割付け成分の場合、対応する式を省略することができます。

成分並びの最後でない省略可能な成分を省略する場合、それ以降の実引数並びには、成分キーワードの指定が必要です。

省略可能な成分の例:

```
type ty1
  integer :: a = 1
  integer :: b
  integer :: c = 1
end type
type (ty1) :: t
t = ty1(b = 3)          ! a および c を省略。b に対して成分キーワードが必要です。
t = ty1(2, 3)           ! c を省略。成分キーワードは指定しなくてもかまいません。
t = ty1(b = 3, c = 8)   ! a を省略。b および c に対して成分キーワードが必要です。
end
```

1.5.12.3 派生型の直接拡張型

派生型が親型から直接拡張された型である場合、親型の名前をもつ親型の変数を先頭の成分として扱います。構造体構成子中に親型の変数と結合する式がない場合、親型の成分に対応する式を指定しなければなりません。

派生型の直接拡張型の例:

```
type ty1
  integer :: a
end type
type, extends(ty1) :: ty2
  integer :: b
end type
type (ty1) :: t1
type (ty2) :: t2
  ...
t2 = ty2(1, b = 1)
t2 = ty2(ty1 = ty1(1), b = 1)
t2 = ty2(a = 1, b = 1)
end
```

1.5.13 クラス

多相的データ要素は、プログラムの実行中に異なる型をもつことができるデータ要素です。多相的データ要素の実行時の型は、プログラム実行中の型です。データ要素の宣言時の型とは、明示的または暗黙的に宣言された型です。

CLASS 指定子は、多相的実体を宣言します。CLASS 指定子が型名を含むとき、その型名が多相的実体の宣言時の型になります。

CLASS(*) 指定子は、無制限多相的な実体を宣言します。無制限多相的データ要素は、型をもつ実体として宣言されません。更に、別の無制限多相的データ要素を含む、すべての他の要素と宣言時の型が同じではありません。

データ要素が多相的でないとき、同一の宣言時の型をもつデータ要素だけと型が適合します。多相的データ要素が無制限多相的でなければ、型が適合するデータ要素は、同一の宣言時の型をもつ場合か、またはその直接拡張型の場合です。無制限多相的データ要素は、宣言時の型をもちません。ただし、すべてのデータ要素と型が適合します。データ要素がある型と“型が適合している”とは、その型のデータ要素と型が適合している場合です。2つのデータ要素の型が“適合しない”とは、双方とも、互いに型が適合していない場合です。

データ要素が対象となるデータ要素と型が適合し、両者の種別型パラメタが同一であり、更に次元数が一致している場合、データ要素は対象となるデータ要素とTKR 適合します。

多相的割付け実体は、適合する型のうち任意の型の実体を割り付けることができます。多相的ポインタまたは多相的仮引数は、プログラムの実行中、型が適合している実体と結合することができます。

割り付けられた割付け多相的実体の実行時の型は、割付け時の型です。

結合状態にある多相的ポインタの実行時の型は、指示先の実行時の型です。

割付け実体でなく、かつポインタでない多相的仮引数の実行時の型は、それが結合している実引数の実行時の型です。

割付け状態にない割付け実体であるかまたは空状態のポインタであるとき、その実行時の型は、その宣言時の型です。

結合名をもつ実体の実行時の型は、それが結合された選択子の実行時の型です。

多相的でない実体の実行時の型は、その宣言時の型です。

1.5.14 ポインタ

ポインタは、POINTER 属性をもつ変数です。ポインタには、データポインタまたは手続ポインタがあります。

データポインタは、引用可能または確定可能な指示先実体と結合されない限り、引用も確定もしてはなりません。

手続ポインタは、EXTERNAL 属性およびPOINTER 属性をもつ手続です。指示先の手続とポインタ結合されない限り、引用してはなりません。

1.5.14.1 ポインタ結合

データポインタは、ポインタ代入(“2.2 ポインタ代入文”参照)または、割付け(“2.20 ALLOCATE文”参照)によって指示先と結合することができます。指示先は、TARGET 属性をもつ変数またはポインタでなければなりません。結合されたポインタは、引用または確定することができ、それらは指示先を間接的に参照します。ポインタが多相的である場合、その実行時の型は、指示先の実行時の型です。

手続ポインタは、ポインタ代入(“2.2 ポインタ代入文”参照)によって外部手続、モジュール手続、組込み手続、または手続ポインタではない仮手続と結合することができます。

1.5.14.2 ポインタ結合状態

ポインタの結合状態は、“結合している”、“空状態”、または“不定”のいずれかです。ポインタの結合状態は、組込み関数ASSOCIATED(“2.30 ASSOCIATED組込み関数”参照)によって知ることができます。

ポインタの初期の結合状態は、明示的または暗黙的に初期化しない限り、不定です。初期化されたポインタは、空状態です。ポインタ結合状態が空状態または不定である場合、そのポインタは引用または解放してはなりません。ポインタがどの結合状態にあっても、そのポインタに対して、NULLIFY 文を実行したり、ALLOCATE 文で割り付けたり、ポインタ代入したりすることができます。NULLIFY 文を実行すると、そのポインタは空状態になります。ALLOCATE 文によりポインタが割り付けられると、そのポインタは結合している状態になります。ポインタがポインタ代入されると、その結合状態は、その指示先によって決まります。

1.5.14.3 データポインタおよび指示先の宣言

データポインタは、型宣言文のPOINTER 属性またはPOINTER 文により宣言することができます。データポインタの指示先は、型宣言文のTARGET 属性または、TARGET 文により宣言することができます。ポインタ配列を宣言する場合、形状無指定配列でなければなりません。

ポインタおよび指示先の宣言の例:

```
integer ,pointer :: a , b(:,:)
integer , target :: c
a => c                  ! ポインタ代入文により、ポインタ a は指示先 c と結合します
allocate (b(3,2))       ! ALLOCATE 文により形状が(3,2) の暗黙的に
                        ! TARGET 属性をもつ実体が生成され、ポインタ b は
                        ! その実体と結合します
```

1.5.15 暗黙形状配列

暗黙形状配列は、宣言時の定数式と同じ形状になる名前付き定数です。省略した場合の下限値は1です。暗黙形状配列の配列宣言の各次元は、以下の形式です。

[*lower-bound* :] *

暗黙形状配列は、名前付き定数でなければなりません。

暗黙形状配列の例:

```
integer,dimension(0:*),parameter :: x=[1,2,3]
```

この例で、名前付き定数x は暗黙形状配列であり、形状は定数式と同じです。

x の第1次元の下限値は 0で、上限値は2です。

1.5.16 次元引継ぎ

次元引継ぎ実体は、その有効実引数から引き継いだ次元数をもつ仮実体です。この次元数をゼロにすることができます。次元引継ぎ実体は、配列形状指定の*assumed-rank-spec*で宣言します。配列形状指定については、“2.119 DIMENSION文”を参照してください。

assumed-rank-spec は、以下の形式(2つのピリオド)です。

..

次元引継ぎ実体は、ALLOCATABLE、CODIMENSION、POINTERまたはVALUE属性をもたない仮引数でなければなりません。

手続言語束縛指定子をもつ手続では、次元引継ぎ実体は、CONTIGUOUS属性をもたない仮引数でなければなりません。

次元引継ぎの変数名は、以下に現れることができます。

- 次元引継ぎ仮引数に対応する実引数、または
- 組込みモジュールISO_C_BINDINGのC_LOC関数、またはC_SIZEOF関数の引数、または
- 組込み問合せ関数の第1引数

割付けでなくポインタでない次元引継ぎである実引数が、INTENT(OUT)属性をもつ次元引継ぎの仮引数に対応する場合、実引数は多相的、後始末可能、割付け可能な末端成分をもつ型、長さ型パラメタをもつパラメタ付き派生型、または暗黙的初期値指定をもつ型であってはなりません。

手続が手続言語束縛指定子をもつ場合、その手続の次元引継ぎ仮引数の下限は0です。

手続が手続言語束縛指定子をもたない場合、その手続の次元引継ぎ仮引数の下限は1です。

次元引継ぎは、言語間結合で使用します。言語間結合については、“Fortran使用手引書”の“言語間結合”を参照してください。

1.5.17 特定子

特定子は、実体名(“1.5.17.1 実体名”参照)、配列要素(“1.5.8.2 配列要素”参照)、部分配列(“1.5.8.4 部分配列”参照)、共添字付き実体(“1.5.17.2 共添字付き実体”参照)、虚実部特定子(“1.5.17.3 虚実部特定子”参照)、構造体成分(“1.5.11.6 構造体成分”参照)、または部分列(“1.5.7 部分列”参照)です。

1.5.17.1 実体名

実体名は、データ実体の名前です。

1.5.17.2 共添字付き実体

共添字付き実体は、像選択子(“1.17.5 像選択子”参照)の指定をもつ実体です。

1.5.17.3 虚実部特定子

虚実部特定子は、以下の形式です。

```
特定子%RE  または
特定子%IM
```

*特定子*は、複素数型でなければなりません。

特定子%REは*特定子*の実部、*特定子*%IMは*特定子*の虚部です。虚実部特定子の型は実数型で、その種別型パラメタと形状は、*特定子*のものです。

虚実部特定子の例:

```
complex :: c=(2.0 , 3.0)
real :: r
```

r= c%re ! r には 2.0 が代入されます

1.6 式

式はデータ引用または計算処理を表現するものであり、演算対象、演算子、および括弧で構成されます。式の評価結果は、型、型パラメタ（種別および、文字型の長さ）、および形状をもちます。

式の例:

```
5
n
(n+1)*y
"Fujitsu " // 'Fortran ' // text(1:23)
(-b + (b**2-4*a*c)**.5) / (2*a)
b%a - a(1:1000:10)
sin(a) .le. .5
l .operator. r + .operator. m
```

この例で、最後の式は、利用者定義演算子.operator.を使用しています（“1.12.7.3.2 利用者定義演算”参照）。

1つの式において、すべての配列の演算対象は同じ形状でなければなりません。スカラはどんな形状の配列とも形状適合します。配列式は、対応する要素ごとに演算の評価が行われます。演算対象の一方が配列であり、他方がスカラである場合、そのスカラは配列演算対象と同じ形状をもち、かつすべての要素がそのスカラ値に等しい配列として扱われます。

配列演算の例:

```
a(2:4) + b(1:3) + 5
```

この式では、以下の演算が実施されます。

```
a(2) + b(1) + 5
a(3) + b(2) + 5
a(4) + b(3) + 5
```

式は、演算子の評価順序（“1.6.3 組込み演算”参照）に従って評価されます。評価順序の同じ演算子が隣接している場合、べき乗演算は右から左へ、その他の演算子は左から右へ順に評価されます。括弧でくくられた要素は、それが他の演算子と結合される前に評価されます。

括弧を使った式の例:

```
a+(b-c)
```

この例では、b-c の演算がまず評価され、次にその結果とa の加算が評価されます。

演算子‘**’、‘/’、‘*’、‘+’、または‘-’に続けて、単項演算子‘+’または‘-’を記述することはできません。括弧でくくる必要があります。

記述できない例:

```
K + -1      ! K + (-1) に修正してください
K ** -2     ! K ** (-2) に修正してください
```

1.6.1 宣言式

宣言式は、長さ型パラメタ、配列の上下限の宣言に用いるスカラ整数式です。このスカラ整数式は制限式です。制限式は、その中のすべての演算が組込みであり、かつ次のいずれかで構成されなければなりません。その中のそれぞれの添字、部分配列添字、部分列開始位置、部分列終了位置および型パラメタ値は、制限式であり、呼び出される後始末サブルーチンは純粋でなければなりません。

- 定数または部分定数
- OPTIONAL 属性およびINTENT(OUT) 属性のいずれももたない仮引数
- 共通ブロック内の実体
- 参照結合または親子結合によって参照可能となった実体
- BLOCKの外側の局所変数である実体
- 配列構成子。その中のすべての要素および配列構成DO 制御の各スカラ整数式が制限式
- 構造体構成子。その中のすべての成分が制限式

- 組込みモジュールIEEE_ARITHMETIC、IEEE_EXCEPTIONS、またはISO_C_BINDINGの変形関数の引用。その中のすべての引数が制限式
- 関数引数が次のいずれかである宣言問合せ
    1. 制限式、または
    2. その性質の問合せができる変数。ただし、その性質は、次のいずれでもない。
        - 大きさ引継ぎ配列の最後の次元の上限、または
        - 無指定
        - 制約式以外で確定される式
- 組込み関数の引用。すべての引数が制限式
- 宣言関数の引用。すべての引数が制限式
- 定義される派生型の型パラメタ
- 括弧でくくられた制限式

宣言問合せは、次のいずれかの引用でなければなりません。

- 組込み問合せ関数
- 型パラメタ問合せ
- 組込みモジュールIEEE ARITHMETIC、およびIEEE EXCEPTIONSの問合せ関数
- 組込みモジュールISO_C_BINDINGのC_SIZEOF
- 組込みモジュールISO_FORTRAN_ENVのCOMPILER_VERSION、またはCOMPILER_OPTIONS

宣言関数は、純粋な関数であり、標準組込み関数でなく、内部関数でなく、文関数でなく、仮手続引数をもたない関数です。

宣言関数の評価は、呼び出される副プログラムによって定義される手続を、直接的にも間接的にも呼び出してはなりません。

1.6.2 定数式

定数式は、翻訳時に評価可能な式です。定数式は、次のいずれかで構成されなければなりません。ただし、その中のすべての添字、部分配列添字、部分列開始位置、部分列終了位置および型パラメタ値は、定数式でなければなりません。

- 配列構成子

  その配列構成子の中のすべての要素、DO 形反復の上下限および刻み幅が、定数式でなければなりません。

- 構造体構成子

  その構造体構成子の中のすべての割付け成分に対応する成分指定は、変形組込み関数NULLの引用であり、それ以外の成分指定が、定数式でなければなりません。

- 要素別処理組込み関数の引用

  すべての引数が定数式でなければなりません。

- 変形関数の引用

  次のいずれかでなければなりません。

  ー COMMAND_ARGUMENT_COUNT、NULL、NUM_IMAGES、THIS_IMAGE、TRANSFER以外の変形標準組込み関数の引用で、すべての引数が定数式。

  ー 変形組込み関数NULL の引用で、定数式以外の式で定義されるか、または仮定される型パラメタをもつ引数が指定されていない。

  ー 組込みモジュールIEEE_ARITHMETIC、IEEE_EXCEPTIONS、またはISO_C_BINDINGの変形関数の引用。その中のすべての引数が定数式。

- 宣言問合せ

  次のいずれかでなければなりません。

— 定数式の問合せ。

— 引き継がれるもの、無指定および定数式以外の式で確定されるものを除く問合せでなければなりません。

- 配列構成子の中のDO 変数

  対応するDO 形反復の上下限およびDO 形反復の刻み幅が、定数式でなければなりません。

- 括弧でくくられた定数式

定数式が同じ宣言部の中で宣言された実体の型パラメタまたは配列の上下限に関する問合せ関数を引用する場合は、その型パラメタまたは配列の上下限は、宣言部の先行する部分に宣言されていなければなりません。なお、その宣言は、同じ文の中のその問合せ関数の左側にあってもかまいませんが、同じデータ要素宣言中にあってはなりません。

定数式には、演算結果の型が半精度実数型または半精度複素数型のべき乗(**)演算子は指定できません。

定数式では、組込み関数MAXVAL、MINVAL、NORM2、またはSPACINGの引数は、半精度実数型であってはなりません。

定数式では、組込み関数ABS、DOT_PRODUCT、またはMATMULの引数は、半精度複素数型であってはなりません。

定数式では、組込み関数CSHIFT、EOSHIFT、PACK、PRODUCT、SPREAD、SUM、TRANSFER、またはUNPACKの引数は、半精度実数型または半精度複素数型であってはなりません。

1.6.3 組込み演算

組込み演算子の評価順序を以下の表に示します。

表1.2 組込み演算子の評価順序

演算子	意味	演算対象	評価の優先順序
**	べき乗	2つの数値型	最も高い
*,/	乗算、除算	2つの数値型	↑
単項+,単項-	単項加算、単項減算	1つの数値型	・
+,-	加算、減算	2つの数値型	・
//	連結演算	2つの文字型	・
.EQ. ,== , .NE. ,/= , .LT. ,< , .LE. ,<= , .GT. ,> , .GE. ,>=	関係演算(等価・非等価) 関係演算(大小比較)	2つの数値型または2つの文字型 2つの数値型(複素数型を除く)または2つの文字型	・
.NOT.	論理否定演算	1つの論理型	・
.AND.	論理積演算	2つの論理型	・
.OR.	論理和演算	2つの論理型	↓
.EQV. , .NEQV.	論理等価演算、 論理非等価演算	2つの論理型	最も低い

単項演算(単項+、単項-、.NOT.)の結果の型および種別は、演算対象の型および種別となります。関係演算の結果の型は、基本論理型です。関係演算以外の演算において、演算対象が同じ型の場合、演算結果の型は演算対象の型となり、種別は大きい方の値となります。演算対象が異なる数値型の場合、結果の型および種別は以下のようになります。

- どちらか一方の演算対象が複素数型の場合、結果の型は複素数型となり、他方が整数型の場合は、種別は複素数型のそれとなり、他方が実数型の場合は、種別は大きい方の値となります。
- 複素数型の演算対象がなく、どちらか一方の演算対象が実数型の場合、結果の型は実数型となり、種別は実数型のそれとなります。

連結演算の結果の長さは、演算対象の長さの和となります。

1.6.4 演算および演算対象の評価

要素が演算子によって結合される場合、どちらの要素を先に評価するかは、規定されません。例えば、func を関数とする時、式func(x)+func(y) において、func(x) とfunc(y) のどちらが先に評価されるかを規定したプログラムを、作成してはなりません。

文の中で関数の引用がある場合、その文の中の、他の要素の値を更新してはなりません。例えば、式func(i).and.i==1 において、関数の引用func(i) で、i の値を更新してはなりません。
ただし、IF 文(“2.278 IF文”参照)の論理式、単純WHERE 文(“2.505 単純WHERE文”参照)の選別式、または単純FORALL 文(“2.189 単純FORALL文”参照)の添字もしくは刻み幅における関数引用の実行は、選択的に実行される文の中の変数を更新してもかまいません。

式の値を得る場合、式の一部が評価されないときがあります。例えば、論理式l1.and.l2 において、l1 が偽であれば、l2 を評価しなくても式の値は偽であることがわかります。したがって、このような評価されない部分に関数の引用を書く場合は、十分に注意しなければなりません。

1.7 入出力文

入出力文は、外部媒体または内部ファイルと、内部記憶の間でデータを転送します。

入出力文には、OPEN 文(“2.372 OPEN文”参照)、CLOSE 文(“2.78 CLOSE文”参照)、READ 文(“2.407 READ文”参照)、WRITE 文(“2.506 WRITE文”参照)、PRINT 文(“2.386 PRINT文”参照)、BACKSPACE 文(“2.42 BACKSPACE文”参照)、ENDFILE 文(“2.139 ENDFILE文”参照)、REWIND 文(“2.416 REWIND文”参照)、INQUIRE 文(“2.287 INQUIRE文”参照)、FLUSH 文(“2.186 FLUSH文”参照)、およびWAIT 文(“2.502 WAIT文”参照)があります。

1.7.1 Fortran 記録

Fortran 記録は、値の列または文字の列であり、1つのFortran 記録は必ずしも1つの物理記録に対応するものではありません。Fortran 記録には以下のものがあります。

- 書式付きFortran 記録
- 書式なしFortran 記録
- 並びによるFortran 記録
- 変数群Fortran 記録
- ファイル終了記録
- BINARY Fortran 記録
- STREAM Fortran 記録

1.7.1.1 書式付きFortran 記録

書式付きFortran 記録は、任意の文字列で構成され、書式仕様によって定義されます。この書式付きFortran 記録は、書式付き順番探査入出力文、書式付き直接探査入出力文および内部ファイル入出力文で扱うことができます。

1.7.1.1.1 書式付き順番探査入出力文で扱うFortran 記録

書式付き順番探査入出力文で扱うファイルの記録形式は不定長記録であり、1つのFortran 記録が1つの論理記録に対応します。不定長記録の記録長は、扱うFortran 記録の長さにより可変です。記録の最大の長さは、OPEN 文のRECL 指定子で指定することができます。論理記録の終端には改行文字(¥n)が入っています。書式付き順番探査WRITE 文における書式仕様に$ 編集記述子または¥ 編集記述子を指定した場合、改行文字を含まないFortran 記録を出力することができます。

1.7.1.1.2 書式付き直接探査入出力文で扱うFortran 記録

書式付き直接探査入出力文で扱うファイルの記録形式は固定長記録です。1つのFortran 記録は1つの論理記録を形成します。論理記録の長さはOPEN 文のRECL 指定子で与えなければなりません。出力では、論理記録よりFortran 記録が短いときは残りの部分に空白が詰められます。ただし、論理記録よりFortran 記録が長くてはなりません。この固定長記録形式はFortran 固有の形式です。

1.7.1.1.3 内部ファイル入出力文で扱うFortran 記録

内部ファイル入出力文で扱うファイルの形式は固定長記録であり、内部ファイルに存在します。内部ファイルがスカラ文字型変数、文字型配列要素、または文字部分列のとき、それはただ1つのFortran 記録を含みます。Fortran 記録の長さはスカラ文字型変数、文字型配

列要素、または文字部分列の長さを超えてはなりません。内部ファイルが文字型配列のとき、1つのFortran 記録の長さは配列要素の長さを超えてはなりません。

出力で、Fortran 記録の長さがスカラ文字型変数、文字型配列要素、または文字部分列の長さより短いときには、それぞれの残りの部分に空白が詰められます。入力では、Fortran 記録の長さはそれぞれの長さに等しくなります。

1.7.1.2 書式なしFortran 記録

書式なしFortran 記録は、値(文字および文字以外のデータをともに含んでもよいし、データを含まなくてもよい)の列で構成され、長さは原始プログラムの入出力並びに依存(長さはゼロであってもよい)します。この書式なしFortran 記録は、書式なし順番探査入出力文および書式なし直接探査入出力文で扱うことができます。

1.7.1.2.1 書式なし順番探査入出力文で扱うFortran 記録

書式なし順番探査入出力文で扱うファイルの記録形式は可変長記録であり、1つのFortran 記録が1つの論理記録に対応します。可変長記録形式の記録の長さは、扱うFortran 記録の長さにより可変であり、論理記録の先頭および最後の各4バイトには1つのFortran 記録の長さが格納されています。この最大の長さは、OPEN 文のRECL 指定子で指定することができます。

先頭の長さ域は、書式なし入力文が実行される時に使用され、最後の長さ域は、BACKSPACE 文が実行される時に使用されます。この可変長記録は、Fortran 固有の形式です。

1.7.1.2.2 書式なし直接探査入出力文で扱うFortran 記録

書式なし直接探査入出力文で扱うファイルの記録の形式は固定長記録であり、1つ以上の論理記録が対応します。固定長記録の形式については“1.7.1.1.2 書式付き直接探査入出力文で扱うFortran 記録”を参照してください。

記録長はOPEN 文のRECL 指定子で与えなければなりません。1つのFortran 記録は1つ以上の論理記録を形成します。出力では、論理記録の途中でFortran 記録が終了したとき、残りの部分にバイナリゼロが詰められます。また、Fortran 記録の長さが論理記録長より長いときは、出力並びの残りの部分がFortran 記録として次の記録番号に出力されます。この固定長記録形式はFortran 固有の形式です。

1.7.1.3 並びによるFortran 記録

並びによるFortran 記録は、並びによる入出力文、PRINT 文および内部ファイル入出力文で扱うことができ、データ項目と値区切り子で構成されます。データ項目とは、入出力並び項目に割り当てられる文字の列です。出力では、出力並び項目の個数および型によってFortran 記録の長さが決定されます。入力では、論理記録の先頭から入力並び項目の処理を終了するまでのデータ項目を1つのFortran 記録として扱います。

並びによる入出力文で扱うファイルの記録形式は不定長記録および固定長記録です。不定長記録形式については“1.7.1.1.1 書式付き順番探査入出力文で扱うFortran 記録”と同じです。固定長記録の形式については“1.7.1.1.3 内部ファイル入出力文で扱うFortran 記録”と同じです。

1.7.1.4 変数群Fortran 記録

変数群Fortran 記録は、変数群入出力文および内部ファイル入出力文で扱うことができ、‘& 変数群名’から‘/’または‘&end’までのデータ項目(変数群名によって指定された要素に割り当てられた文字の列)で構成されます。変数群Fortran 記録と論理記録との対応は、並びによる入出力文で扱うFortran 記録と同じです。

変数群入出力文で扱うファイルの記録形式は不定長記録です。不定長記録の形式については“1.7.1.1.1 書式付き順番探査入出力文で扱うFortran 記録”と同じです。また、内部ファイル入出力文で扱うファイルの記録形式は固定長記録です。固定長記録の形式については“1.7.1.1.3 内部ファイル入出力文で扱うFortran 記録”と同じです。

1.7.1.5 ファイル終了記録

ファイル終了記録は、1つのファイルの最後の記録としてだけ存在することができ、長さに関する属性はもちません。ファイル終了記録はENDFILE 文によって書かれます。ファイルは順編成ファイルに接続されていなければなりません。ファイル終了記録は、ファイルが順編成ファイルに接続されていて、かつ、最後の処理がENDFILE 文以外で、WRITE 文実行後の以下のいずれかの条件のときにも暗黙に出力されます。

- REWIND 文が実行された場合。
- BACKSPACE 文が実行された場合。
- CLOSE 文が実行された場合。

1.7.1.6 BINARY Fortran 記録

BINARY Fortran 記録は、値（文字および文字以外のデータをともに含んでもよいし、データを含まなくてもよい）の列で構成され、長さは原始プログラムの入出力並びに依存（長さはゼロであってもよい）します。このFortran 記録は、書式なし順番探査入出力文および書式なし直接探査入出力文で扱うことができます。

BINARY Fortran 記録の形式は、固定長形式です。

1.7.1.7 STREAM Fortran 記録

STREAM Fortran 記録は、記憶単位で構成され、正の整数によって記録位置が一意に識別できます。このFortran 記録は、流れ探査入出力文で扱うことができます。

1.7.2 ファイル

入出力文で扱うファイルには、内部ファイルと外部ファイルがあります。内部ファイルは、内部記憶領域にあり、スカラ文字型変数、文字型配列要素、文字型配列、または文字部分列のいずれかです。内部ファイルは内部ファイル入出力文で入出力することができます。外部ファイルは、Fortranプログラムの外部媒体上にあり、順編成ファイルと直接編成ファイルの2つに分けられます。順編成ファイルとは、Fortran における探査法の順番探査に接続されるファイルです。直接編成ファイルとは、Fortran における探査法の直接探査に接続されるファイルです。

Fortran が扱うことができるファイルを次に示します。

- 標準入力ファイル（stdin）
- 標準出力ファイル（stdout）
- 標準エラー出力ファイル（stderr）
- 通常ファイル

標準ファイルは、順番探査でだけ入出力できます。ただし、書式なし入出力で探査することはできません。通常ファイルは、順番探査または直接探査のいずれかで入出力できます。

1.7.2.1 ファイルの存在

ファイルは、実行可能プログラムの実行開始時に存在しているものと、実行可能プログラムの実行中に新しく存在するものとがあります。存在しているファイルまたは新たに存在するファイルは、入出力文に指定した装置番号と結びつけられることにより、入出力することができます。この結びつきについては、“1.7.3.1 装置番号とファイルの接続”を参照してください。

1.7.2.2 ファイル位置

入出力文の実行により、外部ファイルの位置に影響を与える場合があります。

データ転送が行われる直前のファイル位置は、直接探査の場合、REC 指定子で指定された記録番号の先頭に位置づけられます。順番探査の場合、最後に入出力された記録の後に位置づけられます。最後に実行された入出力文がADVANCE 指定子を含む停留入出力文の場合、ファイル位置は変わりません。

ENDFILE 文の実行により、ファイル終了記録が最後に入出力された記録の後に出力され、ファイル位置はファイル終了記録の後に位置づけられます。REWIND 文の実行により、ファイル位置はファイルの先頭に位置づけられます。BACKSPACE 文の実行により、ファイル位置は現在記録の1つ前の記録に位置づけられます。

誤り条件が発生したとき、ファイル位置は不定になります。

誤り条件が発生せず、ファイル終了記録が入力された場合、ファイル位置はファイル終了位置の後ろに位置づけられます。このファイルに対して再び入出力を行う場合、REWIND 文またはBACKSPACE 文によってファイル位置を変更しなければなりません。

1.7.2.3 内部ファイル

内部ファイルは、常に書式付き順番探査ファイルであり、文字型の変数です。変数が配列である場合、各配列要素が1つの記録となります。部分配列の場合、ベクトル添字をもってはなりません。ファイルの中の、記録の順序は、配列要素順序と同じです。

1.7.3 装置とファイルの接続

装置は、ファイルを参照するための手段を与えます。

装置は、次のいずれかです。

- 外部ファイル装置
- *
- 内部ファイル装置

外部ファイル装置は、スカラ整数式です。外部ファイル装置または‘*’は、外部ファイルを参照するために使用します。内部ファイル装置は、文字型変数で、内部ファイルを参照するために使用します。

1.7.3.1 装置番号とファイルの接続

入出力文を実行するためには、ファイルと装置番号が結びつけられていなければなりません。これをファイルの接続といいます。また、CLOSE 文の実行で、ファイルと装置番号の結びつきが解除されます。これを、ファイルの接続の解除といいます。

1つの装置番号は同時に2つ以上のファイルと接続できません。また、1つのファイルは同時に2つ以上の装置番号と接続できません。CLOSE 文の実行によって外部ファイルの接続を解除した後で、そのファイルを同じプログラム中で同じ装置番号または別の装置番号にふたたび接続してもかまいません。

1.7.3.2 事前接続

事前接続とは、プログラムの実行の始めに装置がファイルに接続されていて、前もってOPEN 文を実行しなくても、入出力文で装置を参照できることをいいます。装置番号0、5、および6は、それぞれ標準エラー出力ファイル、標準入力ファイル、および標準出力ファイルに事前接続されています。

装置識別子‘*’は、常に標準入力ファイルまたは標準出力ファイルに指定されています。装置識別子をもたないREAD 文およびPRINT 文は、‘*’で指定される装置を指定します。

1.7.4 利用者定義派生型入出力

利用者定義派生型入出力は、利用者定義派生型入出力手続によって、派生型の並び項目についての実際のデータ転送処理を制御します。

利用者定義派生型入出力手続は、派生型入出力総称指定(“2.291 INTERFACE文”参照)によって参照可能な手続です。

1.7.4.1 利用者定義派生型入出力データ転送の実行

派生型入出力が選択されると、その有効域内で実行されるデータ転送入出力文に対して、選択された利用者定義派生型入出力手続が呼び出されます。

派生型並び項目をもち、利用者定義派生型入出力手続を呼び出すデータ転送文は親データ転送文です。親データ転送文の処理中に実行され、利用者定義派生型入出力手続に渡された装置を指定したデータ転送文は子データ転送文です。

子データ転送文は、次の仕様です。

- 子データ転送文の実行は、データ転送の前にファイルの位置付けを行いません。
- 子データ転送文に指定したADVANCE指定子は、無視されます。
- 書式なし子データ転送文は、データ転送の完了後にファイルの位置付けを行いません。
- 子データ転送文にREC指定子、POS指定子またはID指定子は、指定できません。

1.7.4.2 利用者定義派生型入出力手続

利用者定義派生型入出力手続には4つの可能な特性の組があり、それは書式付き入力、書式付き出力、書式なし入力、および書式なし出力です。手続は以下の仕様で指定します。

- 派生型入出力総称指定をもつ引用仕様宣言(“2.291 INTERFACE文”参照)

・ 派生型入出力総称指定をもつ総称束縛(“1.5.11.3 型束縛手続”参照)

利用者定義派生型入出力手続の特性の4つの引用仕様は、次の構文規則が適用されます。

派生型入出力総称指定がREAD(FORMATTED)ならば、その特性は、次の引用仕様と同じでなければなりません。

```
SUBROUTINE  formatted_read_subroutine  (          &
dtv_arg, unit_arg, iotype_arg, value_list_arg,    &
iostat_arg, iomsg_arg)
dtv-type-spec, INTENT(INOUT) :: dtv_arg
INTEGER, INTENT(IN) :: unit_arg
CHARACTER (LEN=*), INTENT(IN) :: iotype_arg
INTEGER, INTENT(IN) :: value_list_arg(:)
INTEGER, INTENT(OUT) :: iostat_arg
CHARACTER (LEN=*), INTENT(INOUT) :: iomsg_arg
END
```

派生型入出力総称指定がREAD(UNFORMATTED)ならば、その特性は、次の引用仕様と同じでなければなりません。

```
SUBROUTINE unformatted_read_subroutine  (          &
dtv_arg, unit_arg,                                 &
iostat_arg, iomsg_arg)
dtv-type-spec, INTENT(INOUT) :: dtv_arg
INTEGER, INTENT(IN) :: unit_arg
INTEGER, INTENT(OUT) :: iostat_arg
CHARACTER (LEN=*), INTENT(INOUT) :: iomsg_arg
END
```

派生型入出力総称指定がWRITE(FORMATTED)ならば、その特性は、次の引用仕様と同じでなければなりません。

```
SUBROUTINE  formatted_write_subroutine  (          &
dtv_arg, unit_arg, iotype_arg, value_list_arg,    &
iostat_arg, iomsg_arg)
dtv-type-spec, INTENT(IN) :: dtv_arg
INTEGER, INTENT(IN) :: unit_arg
CHARACTER (LEN=*), INTENT(IN) :: iotype_arg
INTEGER, INTENT(IN) :: value_list_arg(:)
INTEGER, INTENT(OUT) :: iostat_arg
CHARACTER (LEN=*), INTENT(INOUT) :: iomsg_arg
END
```

派生型入出力総称指定がWRITE(UNFORMATTED)ならば、その特性は、次の引用仕様と同じでなければなりません。

```
SUBROUTINE  unformatted_write_subroutine  (        &
dtv_arg, unit_arg,                                 &
iostat_arg, iomsg_arg)
dtv-type-spec, INTENT(IN) :: dtv_arg
INTEGER, INTENT(IN) :: unit_arg
INTEGER, INTENT(OUT) :: iostat_arg
CHARACTER (LEN=*), INTENT(INOUT) :: iomsg_arg
END
```

斜体で示される手続名および仮引数名は、任意の名前です。

*dtv-type-spec*は、派生型変数型指定です。以下の形式です。

```
TYPE (派生型指定子)  または、
CLASS (派生型指定子)
```

派生型指定子(“1.5.11.8 派生型指定子”参照)が拡張可能型のとき、キーワードCLASSを指定しなければなりません。拡張可能型でないなら、キーワードTYPEを指定しなければなりません。

派生型指定子の長さ型パラメタは、引継ぎでなければなりません。

利用者定義派生型入出力手続が呼び出されるとき、仮引数*unit_arg*は、次の値をもちます。

- 親データ転送文にファイル装置番号の指定がある場合、仮引数*unit_arg*は、その装置番号の値です。
- 親データ転送文が装置番号に星印を指定したWRITE文またはPRINT文の場合、仮引数*unit_arg*は、ISO_FORTRAN_ENV組込みモジュールで定義されているOUTPUT_UNITの値です。
- 親データ転送文が装置番号に星印を指定したREAD文またはデータ転送指定子並びのないREAD文の場合、仮引数*unit_arg*は、ISO_FORTRAN_ENV組込みモジュールで定義されているINPUT_UNITの値です。
- 親データ転送文が内部ファイルを参照する場合、仮引数*unit_arg*は負の値です。利用者定義派生型入出力手続は、この仮引数*unit_arg* の値を装置番号とするINQUIRE文を実行してはなりません。

書式付きデータ転送の場合、仮引数*iotype_arg*は、次の値をもちます。

- 親データ転送文に並び書式の指定がある場合、LISTDIRECTED
- 親データ転送文に変数群書式の指定がある場合、NAMELIST
- 親データ転送文に書式仕様の指定があり、並び項目に対応する編集記述子がDT 形編集記述子(“1.8.1 書式仕様”参照)である場合、DT に編集記述子の文字定数表現を連結した値

親データ転送文がREAD文の場合、仮引数*dtv_arg*は、有効並び項目と引数結合されます。親データ転送文がWRITE文またはPRINT文の場合、仮引数*dtv_arg*は、有効並び項目の値をもちます。

親データ転送文にDT編集記述子(“1.8.1 書式仕様”参照)の値指定並びが現れる場合、仮引数*value_list_arg*は、値指定並びと同じ順序で同じ個数の要素をもちます。編集記述子に値指定並びがないか、データ転送文に並び書式または変数群書式を指定している場合、仮引数*value_list_arg*は、大きさゼロの配列です。

誤り条件が発生した場合、利用者定義派生型入出力手続は、仮引数*iostat_arg*に正の値を代入しなければなりません。ファイル終了条件が発生した場合、利用者定義派生型入力手続は、仮引数*iostat_arg*にISO_FORTRAN_ENV組込みモジュールで定義されているIOSTAT_ENDの値を代入しなければなりません。記録終了条件が発生した場合、利用者定義派生型入力手続は、仮引数*iostat_arg*にISO_FORTRAN_ENV組込みモジュールで定義されているIOSTAT_EORの値を代入しなければなりません。これら以外の場合、利用者定義派生型入出力手続は、仮引数*iostat_arg*に値ゼロを代入しなければなりません。

利用者定義派生型入出力手続が仮引数*iostat_arg*にゼロでない値を返す場合、仮引数*iomsg_arg*に説明の通知を返さなければなりません。これ以外の場合、仮引数*iomsg_arg*の値を変更してはなりません。

親READ文が活動状態にあるとき、入出力文は仮引数*unit_arg*で指定した装置以外の外部装置から読んだり、いずれかの外部装置に書いたりしてはなりません。

親WRITE 文または親PRINT 文が活動状態にあるとき、入出力文は仮引数*unit_arg*で指定した装置以外の外部装置に書いたり、いずれかの外部装置から読んだりしてはなりません。

親データ転送文が活動状態にあるとき、OPEN文、CLOSE文、BACKSPACE文、ENDFILE文、またはREWIND文を実行してはなりません。

親データ転送文が活動状態にあるとき、内部ファイルを指定したデータ転送文は実行できます。

子データ転送文は、データ転送の前にファイルを位置付けしません。子データ転送文は、親データ転送文で直前に処理した有効並び項目または位置付け編集記述子(“1.8.1 書式仕様”参照)によってファイルが位置付けられた場所からデータ転送を開始します。

子データ転送文による位置付け編集記述子(“1.8.1 書式仕様”参照)は、記録位置を利用者定義派生型入出力手続が呼び出されたときの記録位置よりも前に位置付けてはなりません。

親データ転送文も子データ転送文も、非同期であってはなりません。

仮引数*dtv_arg*を経由する場合を除いて、利用者定義派生型入出力手続およびそこから呼び出した手続は、活動状態にある親データ転送文の入出力並び項目、対応する書式または指定子が参照する記憶場所を、確定にしたり不定にしたりする原因となってはなりません。

1.7.4.3 利用者定義派生型入出力手続参照の解決

データ転送中に派生型である有効項目に出会うと、次の条件がともに真であれば、定義した派生型入出力が行われます。

1. 次のいずれかである。
   - 転送は、並び入出力、変数群入出力、または書式なし入出力で開始している。
   - 入出力文に対して書式仕様が指定され、有効項目に対応する編集記述子がDT 形編集記述子(“1.8.1 書式仕様”参照)である。
2. 次のいずれかである。
   - 有効項目の宣言した型が、適切な総称束縛(“1.5.11.3 型束縛手続”参照)をもっている。
   - 適切な総称引用仕様(“2.291 INTERFACE文”参照)が参照可能である。

1.8 入出力編集

書式は、入出力文と組み合わせて使用し、データの内部表現と書式付き記録列としての文字列との間の編集を指示する情報を与えます。

書式は、書式仕様を明示的に与えてもよいし、並び書式(“1.8.2 並び書式”参照)、変数群書式(“1.8.3 変数群書式”参照)でもかまいません。書式仕様はFORMAT 文またはFMT 指定子に文字列として指定できます。

1.8.1 書式仕様

書式仕様は、以下の形式です。

([*format-item-list*])

format-item-list は、コンマで区切られた書式項目の並びです。コンマは以下の場合には省略してもかまいません。

- P 形編集記述子と、その直後のF 形、E 形、EN 形、ES 形、D 形、G 形、またはQ*w.d*形編集記述子との間
- 書式反復数をもたない斜線編集記述子の前
- 斜線編集記述子の後
- コロン編集記述子の前後

format-item は、以下の形式です。

```
[ r ] data-edit-descriptor          または
control-edit-descriptor             または
char-string-edit-descriptor         または
[ r ] ( format-item-list )          または
Q
```

無制限書式項目は、以下の形式です。

```
* ( format-item-list )
```

r は、1 から2147483647までの、種別指定のない正の整定数表現でなければならず、書式反復数といいます。

data-edit-descriptor はデータ編集記述子であり、以下の形式です。

```
I[w [.m ]]                          または
B[w [.m ]]                          または
O[w [.m ]]                          または
Z[w [.m ]]                          または
F[w [.d ]]                          または
```

```
E[w [.d [Ee ]]]                          または
E[w [.d [De ]]]                          または
ENw.d [Ee ]                              または
ESw.d [Ee ]                              または
G[w [.d [Ee ]]]                          または
L[w ]                                    または
A[w ]                                    または
D[w [.d ]]                               または
DT[char-literal-constant][(value-list)]  または
Qw.d
```

DT形編集記述子の*char-literal-constant*は種別指定のない文字定数表現でなければなりません。

DT形編集記述子の*value-list*は値指定並びです。値指定は、指定種別のない任意符号付き整定数表現でなければなりません。

A 形編集記述子を除き、*w*、*m*、*d*、および*e*は、0から255までの、種別指定のない整定数表現でなければなりません。A 形編集記述子において、*w*は0から65000までの、種別指定のない整定数表現でなければなりません。

control-edit-descriptor は制御編集記述子であり、以下の形式です。

```
position-edit-descriptor          または
[ r ]/                            または
:                                 または
sign-edit-descriptor              または
kP                                または
blank-interp-edit-descriptor      または
round-edit-descriptor             または
decimal-edit-descriptor           または
$                                 または
¥                                 または
[ n ]R
```

*k*は、-127から127までの、種別指定のない任意符号付き整定数表現でなければなりません。

*n*は、2から36までの、種別指定のない正の整定数表現でなければなりません。

position-edit-descriptor は位置付け編集記述子であり、以下の形式です。

```
Tn      または
TLn     または
TRn     または
nX
```

*n*は、1から32760までの、種別指定のない正の整定数表現でなければなりません。

sign-edit-descriptor は符号制御編集記述子であり、以下の形式です。

```
S       または
SP      または
SS
```

blank-interp-edit-descriptor は空白解釈編集記述子であり、以下の形式です。

```
BN      または
BZ
```

char-string-edit-descriptor は文字列編集記述子であり、アポストロフィ‘’’または引用符‘”’でくくった形の文字定数表現、またはH 形編集記述子（廃止事項）です。

round-edit-descriptor は、丸め編集記述子であり、以下の形式です。

```
RU    または
RD    または
```

```
RZ      または
RN      または
RC      または
RP
```

decimal-edit-descriptor は、小数点編集記述子であり、以下の形式です。

```
DC      または
DP
```

1.8.1.1 書式制御

書式反復数*r*の付いた書式項目を除けば、書式仕様は左から右へと解釈されます。書式反復数*r*の付いた書式項目は、その書式項目から書式反復数を除いた部分を*r*個コンマで区切って並べたものとして処理されます。

書式仕様中の1つの*data-edit-descriptor*には、入出力項目並びで指定される1つの有効項目が対応します。ただし、複素数型の1つの入出力項目には、2つのF 形、E 形、EN 形、ES 形、D 形、G 形、およびQ*w.d*形の編集記述子が対応します。
control-edit-descriptor および*char-string-edit-descriptor* には、対応する入出力項目はありません。

書式制御が書式仕様全体を閉じる右括弧に到達し、かつ有効入出力項目が残っていなければ、書式制御は終了します。有効入出力項目が残っていれば、書式制御はDT 形編集記述子を除いて最後から2番目の右括弧で終わっている書式項目の始めに戻ります。そのような右括弧がなければ、書式制御は書式仕様の先頭の左括弧に戻ります。書式制御の戻りが起こる場合、書式仕様の再使用部分は、データ編集記述子を少なくとも1つは含んでいなければなりません。書式制御が書式反復数*r*のついた括弧に戻る場合、その書式反復数も再使用されます。書式制御の戻りは、変更可能なモードに対して何の影響も与えません。書式制御が無制限書式項目の開始でない括弧に戻る場合、ファイルは斜線編集記述子(“1.8.1.3.2 斜線編集”参照)が処理されたときと同じ位置付けがされます。

1.8.1.2 データ編集記述子

データ編集記述子は、内部表現との間のデータ変換を引き起こします。すべてのデータ編集記述子はすべての型の入出力に拡張して指定できます。

1.8.1.2.1 数値編集

I 形、B 形、O 形、Z 形、F 形、E 形、EN 形、ES 形、D 形、G 形、およびQ*w.d*形の編集記述子は、整数型、実数型、および複素数型のデータの入出力を指定します。これらの編集記述子は以下の共通規定があります。

- 入力において、欄中の先行する空白は意味がありません。
- 入力において、入力欄中の小数点は編集記述子による小数点位置よりも優先して解釈されます。
- 入力において、欄幅はゼロであってはなりません。
- 出力において、外部表現は欄中で右寄せした形となります。
- 出力において、編集によって生成される文字の個数が欄幅を超える場合、欄全体が星印‘*’で埋められます。
- 出力において、内部データの値がゼロであるときは、負符号は出力しません。

1.8.1.2.2 整数型の編集

I*w*形、I*w.m*形、B*w*形、B*w.m*形、O*w*形、O*w.m*形、Z*w*形、およびZ*w.m*形の編集記述子は、*w*が0の場合を除き、編集する欄が*w*けたを占めることを指示します。*w*が0の場合は、出力する文字の幅を欄幅とします。入力において*w*は0であってはなりません。*w*が省略された場合の欄の幅を、以下の表に示します。

編集記述子	入出力項目の型			
	1バイトの整数型	2バイトの整数型	4バイトの整数型	8バイトの整数型
I形編集記述子	4	6	11	20
B形編集記述子	9	17	33	65
O形編集記述子	4	7	12	23
Z形編集記述子	3	5	9	17

出力編集において、*m*が指定された場合、定数表現の数字部分が*m*けた未満の場合には、*m*けたになるように先行0が付けられます。*m*の値は、*w*が0である場合を除いて、*w*の値を超えてはなりません。*m*が0で、かつ内部データの値が0の場合、出力される欄にはなにも出力されません。*m* および*w* がともに0で、内部データの値も0の場合、欄の幅は1となり、何も出力されません。

m の指定は入力においては効果をもちません。

1.8.1.2.3 実数型および複素数型の編集

F*w.d*形、E*w.d*形、D*w.d*形、E*w.d*E*e*形、E*w.d*D*e*形、EN 形、ES 形、およびQ*w.d*形の編集記述子は、実数型データおよび複素数型データの編集を指定します。複素数型の1つの入出力項目には、2つの編集記述子が対応します(“1.8.1.2.4 複素数型の編集”参照)。

入力においては、F、E、D、EN、ES、Q*w.d*形編集の解釈は同じです。欄が*w*けたを占め、そのうちの小数部が*d*けたからなることを指示します。*w* は0であってはなりません。*w*が省略された場合の欄の幅を、以下の表に示します。

編集記述子	入出力項目の型		
	単精度実数型、 単精度複素数型	倍精度実数型、 倍精度複素数型	4倍精度実数型、 4倍精度複素数型
F形編集記述子	15	22	43
E形編集記述子	15	22	43
D形編集記述子	15	22	43
Q*w.d*形編集記述子	15	22	43

入力欄が小数点を含む場合、*d*は効果をもちません。小数点を含まない場合、数字列の右側の*d*けたが小数部とみなされます。入力においては、*e* は効果をもちません。入力欄には以下のいずれかの形式の指数部を含むことができます。

- 符号付き数字列
- 英字‘E’、‘D’、または‘Q’に続いて任意符号付き数字列

F 形編集による出力欄の構成は、必要な数の空白に続いて負符号‘-’または符号制御編集(“1.8.1.3.4 符号制御編集”参照)に従った正符号‘+’、続いて小数点を含んだ1つ以上の数字列です。この数字列は、値をけた移動数(“1.8.1.3.5 P 形編集”参照)で修正し、小数点以下*d*けたに丸めたものです。*w*が0の場合は、出力する符号と数字列の幅を欄幅とします。

E 形、D 形、およびQ*w.d*形編集による出力編集は、欄が*w*けたを占め、そのうちの小数部が*d*けたからなり、指数部の整数が*e*けたからなることを指示します。出力欄は、以下の順に構成されます。

- 0個以上の空白
- 負符号‘-’または符号制御編集(“1.8.1.3.4 符号制御編集”参照)に従った正符号‘+’
- 0または、けた移動(“1.8.1.3.5 P 形編集”参照)による有効数字
- 小数点

  小数点編集モードが'POINT' である場合、小数点記号はピリオド'.' ですが、小数点編集モードが'COMMA' である場合、小数点記号はコンマ',' です。
- *d*けた(けた移動(“1.8.1.3.5 P 形編集”参照)がある場合それより小さいけた数)に丸められた有効数字
- E*w.d*E*e*形、E*w.d*D*e*形、および指数部の数字列が2 けた以下の場合、‘E’、‘D’、‘Q’
- 指数部の符号‘-’または‘+’
- 指数部の数字列(*e*の指定がある場合、そのけた数。*e*の指定がなければ2けたまたは3けた)

指数部の符号は常に生成されます。指数の値が0であるときは、正符号が生成されます。指数部の値が999より大きい場合、E*w.d*形、D*w.d*形、およびQ*w.d*形編集を使用してはなりません。

E 形、D 形、およびQ*w.d*形編集において、けた移動数*k*(“1.8.1.3.5 P 形編集”参照)は、10進正規化を制御します。-*d*< *k*<= 0 ならば、出力欄には、小数点の右に|*k*| 個の先行0、続いて*d*-|*k*|けたの有効数字が来ます。0 < *k*< *d*+2 ならば、小数点の左にちょうど*k*けたの有効数字、小数点の右に *d*-*k*+1 けたの有効数字が来ます。これ以外の*k* の値を指定してはなりません。

EN 形編集記述子は、実数値を工学表記の形で出力します。すなわち、出力値が0でないとき、10進指数は3の倍数となり、有効数字部の絶対値は1以上1000未満となります。出力において、けた移動数は効果をもちません。EN 形編集記述子は、欄が*w*けたを占め、そのうちの小数部が*d*けたからなり、指数部の整数が*e*けたからなることを指示します。

ES 形編集記述子は、実数値を科学表記の形で出力します。すなわち、出力値が0でないとき、有効数字部の絶対値が1以上10未満となります。けた移動数は効果をもちません。ES 形編集記述子は、欄が*w*けたを占め、そのうちの小数部が*d*けたからなり、指数部の整数が*e*けたからなることを指示します。

IEEE 例外仕様である入力欄は、省略可能な空白に続いて次のいずれかがあり、更に省略可能な空白が続きます。

1. 省略可能な符号に続いて、文字列'INF' または文字列'INFINITY'。
2. 省略可能な符号に続く文字列'NAN'。括弧に囲まれたゼロ個以上の英数字がその後に続いてもかまいません。

形式 1. で規定した値は、IEEE 無限大です。形式 2. で規定した値は、IEEE 非数です。入力欄の空白以外の文字が'NAN' または'NAN()' だけのとき、非数の値は、例外非通知非数とします。

内部の値がIEEE 無限大であるときの出力欄の構成は、必要ならば空白、続いて負の無限大のとき負符号それ以外のとき省略可能な正符号、続いて文字列'Inf' または'Infinity' であり、欄の中で右詰めとなります。*w*が3より小さい場合は、欄を星印で埋め、*w*が3以上で8未満の場合は、'Inf' を生成します。

内部の値がIEEE 非数であるときの出力欄の構成は、必要ならば空白、続いて文字列'NaN'、続いて省略可能で引用符で囲んだ1文字以上*w*-5 文字以下の英数字であり、欄の中で右詰めとなります。*w*が3未満の場合は、欄を星印で埋めます。

内部の値がIEEE 無限大でもIEEE 非数でもないとき、出力欄の構成は、必要ならば空白、続いて内部の値が負のとき負符号、それ以外のとき任意正符号、続いて小数点記号を1つ含む数字列となります。この数字列は、内部の値の絶対値をけた移動数で修正し、小数点記号以下*d*けたに丸めたものとなります。

1.8.1.2.4 複素数型の編集

複素数型データの編集は、実数型データの編集に用いる編集記述子を2つ用いて指定します。1番目の編集記述子が実部を指定し、2番目の編集記述子が虚部を指定します。2つの編集記述子は異なっていてもかまいません。2つの編集記述子の間に制御編集記述子（“1.8.1.3 制御編集記述子”参照）および文字列編集記述子（“1.8.1.4 文字列編集記述子”参照）があってもかまいません。

1.8.1.2.5 論理型の編集

L*w*形編集記述子は、欄が*w*けたを占めることを指示します。

入力欄の構成は、省略可能な空白列、続いて省略可能な1つのピリオド‘.’、続いて英字‘T’または‘F’とします。‘T’は真を表し、‘F’は偽を表します。英字‘T’または‘F’は、英小文字‘t’または‘f’でもかまいません。‘T’または‘F’の後に任意の文字があってもかまいません。例えば、‘.TRUE.’や‘.FALSE.’は、入力文字列として許されます。

出力欄の構成は、*w*-1 個の空白、続いて‘T’または‘F’となります。‘T’は真を表し、‘F’は偽を表します。

*w*が省略された場合の欄の幅を、以下の表に示します。

編集記述子	入出力項目の型			
	1バイトの論理型	2バイトの論理型	4バイトの論理型	8バイトの論理型
L形編集記述子	2	2	2	2

1.8.1.2.6 文字型の編集

A*w*形編集記述子は文字型の入出力項目に対応します。

A 形編集記述子に欄幅*w*を指定すると、欄は、*w*個の文字からなります。欄幅*w*を指定しない場合、欄は入出力項目の長さと同じ個数の文字からなります。

入出力項目の長さを*len*とした場合、A 形編集の入力において、指定した欄幅*w*が*len*以上であれば、入力欄の右側の*len*個が読み込まれます。指定した欄幅*w*が*len*未満であれば、その*w*個の文字は、内部表現中で左寄せされ、*len*-*w*個の空白が補われます。

出力において、指定した欄幅*w*が*len*より大きければ、出力欄の構成は、*w*-*len*個の空白、続いて*len*個の文字となります。*w*が*len*以下であれば、出力欄の構成は、内部表現の左端の*w*文字となります。

1.8.1.2.7 G 形編集

G*w*形、G*w.d*形、およびG*w.d*E*e*形編集記述子は、すべての組込み型の入出力項目に対応させて用いることができます。これらの編集記述子は、欄が*w*けたを占め、そのうちの小数部が最大*d*けたからなり、指数部の整数が*e*けたからなることを指示します。整数型、論理型、または文字型データの入出力の指定に用いた場合、*d*および*e*は効果をもちません。*w*が省略された場合の欄の幅を、以下の表に示します。

編集記述子	入出力項目の型										
	1バイトの整数型	2バイトの整数型	4バイトの整数型	8バイトの整数型	単精度実数型、単精度複素数型	倍精度実数型、倍精度複素数型	4倍精度実数型、4倍精度複素数型	1バイトの論理型	2バイトの論理型	4バイトの論理型	8バイトの論理型
G形編集記述子	4	6	11	20	15	22	43	2	2	2	2

対応する入出力項目が整数型データである場合、I*w*形編集記述子と同じです。

対応する入出力項目が実数型および複素数型の場合、入力においては、F*w*形編集と同じです。出力においては、データの絶対値がF形編集で編集可能な範囲であれば、F 形編集と同じです。そうでなければ、E 形編集と同じです。データの絶対値がF 形編集の編集可能範囲内である場合、けた移動数(“1.8.1.3.5 P 形編集”参照)は効果をもちません。G*w*形編集は、*w*が0でないならば、G*w*.0と指定した場合と同じです。*w*が0ならば、G*w*.*d*E*e*編集記述子と同じです。*w*、*d*(G0形編集記述子である場合)、および*e*の値は、欄がアスタリスクで埋められない最小の値となります。

対応する入出力項目が論理型データである場合、L*w*形編集記述子と同じです。*w*が0ならば、L1形編集記述子と同じです。

対応する入出力項目が文字型データである場合、A*w*形編集記述子と同じです。*w*が0ならば、A形編集記述子と同じです。

1.8.1.3 制御編集記述子

制御編集記述子は、内部表現との間でのデータの転送、およびデータの変換は行わず、後のデータ編集記述子によって実行される変換に影響を及ぼします。

1.8.1.3.1 位置付け編集

T*n*形、TL*n*形、TR*n*形、および*n*X 形編集記述子は、記録へ、または記録から転送される次の文字の位置を指定します。指定する位置は現在の位置の前でも後でもかまいません。これにより、入力において記録の一部分を異なる編集で複数回処理することも可能です。また、記録の一部分を読み飛ばすことも可能です。

T*n*形編集記述子は、次の文字の位置を現在の位置から*n*番目の文字位置にします。TL*n*形編集記述子は、次の文字の位置を現在の位置から*n*文字だけ左へ戻った文字位置にします。TR*n*形編集記述子は、次の文字の位置を現在の位置から*n*文字だけ右へ進んだ文字位置にします。*n*X 形編集は、TR*n*形編集と同じ効果をもちます。

出力において、これらの編集記述子が記録の長さに影響を及ぼすことはありません。これらの編集記述子によって文字位置を移動し、データを転送する場合、以前に文字が埋め込まれていない位置には、空白が埋め込まれます。

1.8.1.3.2 斜線編集

斜線編集記述子(/編集記述子)は、現在記録への、または現在記録からのデータ転送の終了を指示します。ファイルは直後の記録の先頭に位置付けられます。順番探査として接続されているファイルへの出力では、新しい記録が作られ、それがファイルの最後の記録となります。

1.8.1.3.3 コロン編集

コロン編集記述子(:編集記述子)は、入出力項目並び中に有効項目が残っていない場合に、書式制御を終了させます。コロン編集記述子は、入出力項目並び中に有効項目が残っている場合には、効果をもちません。

1.8.1.3.4 符号制御編集

S 形、SP 形、およびSS 形編集記述子は、数値出力欄の正符号文字‘+’の出力を制御します。SP形編集は、正符号文字‘+’を出力することを指示します。SS 形編集は、正符号文字‘+’を出力しないことを指示します。S 形編集は、標準の状態(正符号文字‘+’を出力しない)に戻すことを指示します。

1.8.1.3.5 P 形編集

*k*P 形編集記述子は、けた移動数として*k*を設定します。けた移動数は、数値編集(F 形、E 形、EN形、ES 形、D 形、G 形、およびQ*w*.*d*形編集記述子)にだけ、以下に示す効果をもちます。

- 入力において、編集記述子がF 形、E 形、EN 形、ES 形、D 形、G 形、およびQ*w*.*d*形の場合、欄中に指数部がなければ、内部表現の数値は外部表現の数値の1/(10**k*) 倍に等しくなります。欄中に指数部がある場合、けた移動数は効果をもちません。

- 出力において、編集記述子がF 形編集の場合、外部表現の数値は内部表現の数値の10**_k_ 倍に等しくなります。
- 出力において、編集記述子がE 形、D 形、およびQ*w.d*形編集の場合、出力する値の有効数字部の値は10**_k_ 倍にされ、指数部の値は_k_ だけ減らされます。
- 出力において、編集記述子がG 形の場合、編集を受けるデータの絶対値がF 形編集を適応される範囲を超えなければ、けた移動数は効果をもちません。E 形編集を適応される場合には、けた移動数は、E 形の出力編集の場合と同じ効果をもちます。
- 出力において、編集記述子がEN 形およびES 形の場合、けた移動数は効果をもちません。

1.8.1.3.6 空白解釈編集

BN 形編集およびBZ 形編集は、数値入力欄中の、先行空白以外の空白の解釈方法を指示します。

BN 編集は、それらの空白を無視することを指示します。BZ 編集は、それらの空白を0として扱うことを指示します。

1.8.1.3.7 $ 編集

$ 編集記述子は、書式付き順番探査出力文において、現在の記録を終了せずに、次の項目の出力を同じ記録に出力することを指示します。

1.8.1.3.8 ¥ 編集

¥ 編集記述子は、$ 編集記述子（“1.8.1.3.7 $ 編集”参照）と同じ効果をもちます。

1.8.1.3.9 R 編集

R 形編集記述子は、10以外の基数を指定します。

R または、*n*R と指定し、*n* は2から36までの符号なし整定数でなければなりません。*n* が省略されると、デフォルトの10進基数が復元されます。

1.8.1.3.10 RU 形編集、RD 形編集、RZ 形編集、RN 形編集、RC 形編集およびRP 形編集

丸め編集記述子は、接続の入出力丸めモードを一時的に変更します。

RU形、RD形、RZ形、RN形、RC形およびRP形の丸め編集記述子は、それぞれROUND 指定子の値である'UP'、'DOWN'、'ZERO'、'NEAREST'、'COMPATIBLE' および'PROCESSOR_DEFINED' に対応する入出力丸めモードを設定します。入出力丸めモードは、書式付き入出力での実数および複素数の値の変換に影響を及ぼします。また、D 形、E 形、EN 形、ES 形、F 形およびG 形の編集に対してだけ影響を及ぼします。

入出力丸めモードが'UP' である場合、変換結果の値は、元の値以上で最小の表現可能な値とします。入出力丸めモードが'DOWN' である場合、変換結果の値は、元の値以下で最大の表現可能な値とします。入出力丸めモードが'ZERO' である場合、変換結果の値は、元の値に最も近く、絶対値で元の値未満の値とします。入出力丸めモードが'NEAREST' である場合、変換結果の値は、最も近い表現可能な値とし、丸め結果として表現可能な2つの値の中間値であった場合には、偶数となる値とします。入出力丸めモードが'COMPATIBLE' である場合、変換結果の値は、最も近い表現可能な値とし、丸め結果として表現可能な2つの値の中間値であった場合には、ゼロから遠い値とします。入出力丸めモードが'PROCESSOR_DEFINED' である場合、接続の入出力丸めモードの指定を省略したものとみなされます。入出力丸めモードの指定を省略した場合には、浮動小数点の丸めモードに依存します。

1.8.1.3.11 利用者定義派生型の編集

DT 形編集記述子では、派生型の並び項目の処理において、暗黙の入出力編集の代わりに利用者の用意した手続を使用することができます。

1.8.1.3.12 DC 形編集およびDP 形編集

小数点編集記述子は、接続の小数点編集モードを一時的に変更します。

DC 形およびDP 形の編集記述子は、それぞれDECIMAL 指定子の値である'COMMA' および'POINT' に対応する小数点編集モードを設定します。

小数点編集モードとは、書式付き入出力での実数および複素数の値の変換において、小数点記号の表現を制御するためのものです。小数点記号は、内部ファイル中または外部ファイル中の実数の10 進表現で、整数部と小数部とを分ける文字です。

小数点編集モードは、D 形、E 形、EN 形、ES 形、F 形およびG 形の編集に対してだけ影響を及ぼします。小数点編集モードがPOINT であるとき、小数点記号は、ピリオドです。小数点編集モードが'COMMA' であるとき、小数点記号は、コンマです。

並び入出力において小数点編集モードが'COMMA' であると、値区切り子として使用される文字は、コンマの替わりにセミコロンとなります。

1.8.1.4 文字列編集記述子

文字列編集記述子は、最大65000文字までの文字列を出力します。文字列編集記述子は、入力に使用してはなりません。

文字列編集記述子は、アポストロフィ‘’’または引用符‘”’でくくった文字定数表現の形式、またはH形編集記述子で、その編集記述子中の文字(空白も含みます)の列を出力します。欄の幅はその中に含まれる文字の個数です。両端の囲み記号(アポストロフィ‘’’または引用符‘”’)は文字の個数に含めません。欄の中での2連の囲み記号は、1文字として数えます。

1.8.1.4.1 H 形編集(廃止事項)

H 形編集記述子は以下の形式です。

*c*H*char*[*char*]...

c は1から65000までの、種別指定のない正の整定数表現です。

char は、文字です。

c はH の後に続く文字の個数を指定します。H 形編集記述子の欄の幅は、*c* です。

1.8.1.5 残余文字編集

Q 形編集記述子は、現在の入力レコードの残余文字数を返却します。対応する入力並び項目は、整数型でなければなりません。

1.8.2 並び書式

書式識別子が‘*’である入出力文を、並び入出力文といい、これらの入出力文の入出力編集は並び書式となります。

並び書式入出力文の例:

```
read *,a
print *,x,y,z
write (unit=*,fmt=*) i,j,k
```

1.8.2.1 並び入力

並び記録は、値および値区切り子の列で構成されます。値は、空値または、以下のいずれかの形です。

```
c               または
r * c           または
r *
```

c は、種別型パラメタをもたない、任意符号付き定数表現または囲みなし文字定数です。

r は、種別型パラメタをもたない、0でない正の整定数表現です。

r * *c* の形式は、定数*c* を*r* 個連続して書いたものと同じです。

r * の形式は、空値を*r* 個連続して書いたものと同じです。

値区切り子は、0個以上の空白が前後にあるコンマ‘,’、斜線‘/’、改行文字、または空白でない値の間の、1つ以上の空白または水平タブ文字です。斜線‘/’は、並び入出力を終了させる区切り子であり、斜線‘/’の後に続く値は入力されません。

区切り子は、小数点編集モードが'POINT' である場合はコンマとし、小数点編集モードが'COMMA' である場合はセミコロンとします。

文字列の囲み記号を省略したとき、その文字列は最初の空白、コンマ(小数点編集モードが'POINT' である場合)、セミコロン(小数点編集モードが'COMMA' である場合)、斜線または記録の終わりで終了し、データ中のアポストロフィまたは引用符は2連にしません。

並び入力の各値は、対応する入力項目に従って、以下の表のように編集されます。

入力項目の型	編集
整数型	I
実数型	F

入力項目の型	編集
複素数型	複素定数表現の形式
論理型	L
文字型	文字列。文字列はいくつもの記録にまたがってもかまいません。文字列が記録の境界をまたがず、文字列中に値区切り子および囲み記号を含まない場合、囲み記号であるアポストロフィ‘’’または引用符‘”’は、省略することができます。

1.8.2.2 並び出力

区切り子は、小数点編集モードが'POINT' の場合はコンマとし、小数点編集モードが'COMMA'の場合はセミコロンとします。

並び出力は、出力項目の型に従って、以下の表に示す編集が行われます。

出力項目の型	編集
整数型	I*w*
実数型	F*w* または E*w.d*E*e*
複素数型	(F*w* またはE*w.d*E*e* , F*w* またはE*w.d*E*e*)
論理型	真に対しては‘T’、偽に対しては‘F’
文字型	DELIM 指定子の指定に従って、必要ならば囲み記号であるアポストロフィ‘’’または引用符‘”’を付加した文字列

1.8.3 変数群書式

NML 指定子が指定された入出力文を変数群入出力文といい、これらの入出力文の入出力編集は変数群書式となります。変数群入出力文による記録の構成は、以下のようになります。

- 省略可能な空白列、および変数群注釈
- 文字‘&’と、その直後にNAMELIST 文で指定された変数群名
- 1つ以上の空白
- 0個以上の‘名前- 値対応’を、値区切り子で区切った列
- 変数群記録の終了を示す斜線‘/’または‘&end’

‘名前 - 値対応’は、実体名または部分実体、続いて等号‘=’、続いて値および値区切り子の列で構成されます。名前は変数群要素並びで指定した名前でなければなりません。変数群書式中の値は、並び書式(“1.8.2 並び書式”参照)での形式と同じです。

文字定数表現の中にある場合を除いて、値区切り子の直後、または変数群入力記録の最初の空白でない位置にある文字‘!’は、注釈の開始を意味します。注釈の範囲は、現在の入力記録の最後までです。

有効項目が複素数型である場合、入力の形の構成は、左括弧、続いてコンマ(小数点編集モードが'POINT' である場合)またはセミコロン(小数点編集モードが'COMMA' である場合)で区切られた数値入力欄の順序対、続いて右括弧とします。区切り子は、小数点編集モードが'POINT'である場合はコンマとし、小数点編集モードが'COMMA' である場合はセミコロンとします。

変数群入出力文の例:

```
integer :: i,j(10)
real :: n(5)
namelist /my_namelist/ i,j,n
read (*,nml=my_namelist)
```

この例で、入力記録が以下の形式の場合、i には5 が、n(3) には4.5 が、j(1:4) には0が、それぞれ読み込まれます。

```
&my_namelist i=5, n(3)=4.5,
j(1:4)=4*0/
```

1.9 文

Fortran の文は実行文と非実行文に分類されます。実行文は、動作を規定または制御し、プログラムの実行順序を決めます。実行文として分類されない文は、すべて非実行文です。非実行文は、プログラム環境を指定するために使用します。

ここでは、文の種類と、概略を説明します。各文の詳細については、“第2章 文および手続の詳細”を参照してください。

1.9.1 実行文

型保持文

型保持文は、SELECT TYPE 構文中に指定し、直後にあるブロックの実行される条件を指定します(“2.431 SELECT TYPE構文”参照)。

代入文

代入文は、等号‘=’の右側の式の評価結果を、等号‘=’の左側の変数に代入します(“2.1 代入文”参照)。

ポインタ代入文

ポインタ代入文は、ポインタを指示先と結合します(“2.2 ポインタ代入文”参照)。

ALLOCATE 文

ALLOCATE 文は、ポインタ指示先および割付け変数を動的に生成します(“2.20 ALLOCATE文”参照)。

ASSIGN 文(廃止事項)

ASSIGN 文は、文番号をスカラ整変数に割り当てます(“2.28 ASSIGN文(廃止事項)”参照)。

ASSOCIATE 文

ASSOCIATE 文は、名前付き要素に対して式または変数を結合します(“2.29 ASSOCIATE構文”参照)。

BACKSPACE 文

BACKSPACE 文は、ファイルに現在記録が存在するならば、現在記録の前に位置付けます。現在記録がなければ、直前記録の前に位置付けられます(“2.42 BACKSPACE文”参照)。

BLOCK 構文

BLOCK 構文は、宣言を含むことができる実行構文です(“2.57 BLOCK構文”参照)。

CALL 文

CALL 文は、サブルーチンを引用し、指定された実引数を渡します(“2.62 CALL文”参照)。

CASE 文

CASE 文は、CASE 構文中に指定し、直後にあるブロックの実行される条件を指定します(“2.63 CASE構文”および“2.64 CASE文”参照)。

CLOSE 文

CLOSE 文は、外部ファイルと装置との接続を解除します(“2.78 CLOSE文”参照)。

CONTINUE 文

CONTINUE 文の実行は、効果をもちません(“2.89 CONTINUE文”参照)。

CRITICAL 構文

CRITICAL 構文は、複数の像が同時にブロックを実行することを制限します(“2.102 CRITICAL構文”参照)。

CYCLE 文

CYCLE 文は、DO ループ中に指定し、残りのループ範囲の実行を1回飛び越します(“2.105 CYCLE文”参照)。

DEALLOCATE 文

DEALLOCATE 文は、割付け変数またはポインタ指示先を解放し、ポインタを空状態にします(“2.116 DEALLOCATE文”参照)。

DO 文

DO 文は、DO 構文の開始を示します。DO 構文は、実行構文の列の繰返し実行を指定します(“2.120 DO構文”参照)。

ELSE 文

ELSE 文はIF 構文中に指定し、それ以前にあるすべてのブロックが実行されていない場合に、直後にあるブロックを実行することを指定します(“2.277 IF構文”および“2.129 ELSE文”参照)。

ELSE IF 文

ELSE IF 文はIF 構文中に指定し、直後にあるブロックの実行される条件を指定します(“2.277 IF構文”および“2.130 ELSE IF文”参照)。

ELSEWHERE 文

ELSEWHERE文はWHERE構文中に指定し、それ以前にある構造WHERE文およびELSEWHERE文の選別式の値が偽である要素の振る舞いを指定します(“2.503 WHERE構文”および“2.131 ELSEWHERE文”参照)。

END 文

END 文は、プログラム単位および副プログラムの終了を指定します(“2.132 END文”参照)。

END ASSOCIATE 文

END ASSOCIATE 文は、ASSOCIATE 構文の終了を指定します(“2.29 ASSOCIATE構文”および“2.133 END ASSOCIATE文”参照)。

END BLOCK 文

END BLOCK 文は、BLOCK 構文の終了を指定します(“2.57 BLOCK構文”および“2.134 END BLOCK文”参照)。

END CRITICAL 文

END CRITICAL 文は、CRITICAL 構文の終了を指定します(“2.102 CRITICAL構文”および“2.136 END CRITICAL文”参照)。

END DO 文

END DO 文は、DO 構文の終了を指定します(“2.120 DO構文”および“2.137 END DO文”参照)。

ENDFILE 文

ENDFILE 文は、そのファイルの直後記録として、ファイル終了記録を書き、ファイルをファイル終了記録の後ろに位置付けます(“2.139 ENDFILE文”参照)。

END FORALL 文

END FORALL 文は、FORALL 構文の終了を指定します(“2.187 FORALL構文”および“2.140 END FORALL文”参照)。

END FUNCTION 文

END FUNCTION 文は、関数副プログラムの終了を指定します(“2.141 END FUNCTION文”および“1.12.1 関数副プログラム”参照)。

END IF 文

END IF 文は、IF 構文の終了を指定します(“2.277 IF構文”および“2.142 END IF文”参照)。

END PROGRAM 文

END PROGRAM 文は、主プログラムの終了を指定します(“2.147 END PROGRAM文”および“1.11.1 主プログラム”参照)。

END SELECT 文

END SELECT 文は、CASE 構文またはSELECT TYPE 構文の終了を指定します(“2.63 CASE構文”、“2.148 END SELECT文”、および“2.431 SELECT TYPE構文”参照)。

END SUBROUTINE 文

END SUBROUTINE 文は、サブルーチン副プログラムの終了を指定します(“2.151 END SUBROUTINE文”および“1.12.2 サブルーチン副プログラム”参照)。

END WHERE 文

END WHERE 文は、WHERE 構文の終了を指定します(“2.503 WHERE構文”および“2.154 END WHERE文”参照)。

ERROR STOP 文

ERROR STOP 文は、プログラムの実行を誤り終了します(“2.164 ERROR STOP文”参照)。

EXIT 文

EXIT 文は、DO 構文の実行の終了を指定します(“2.172 EXIT文”参照)。

FLUSH 文

FLUSH 文は、実行すると、外部ファイルに書き込まれたデータを他の手続で引用したり、Fortran以外の手段で外部ファイルに置かれたデータをREAD 文で利用したりすることができます(“2.186 FLUSH文”参照)。

単純FORALL 文

単純FORALL 文は、代入文およびポインタ代入文の実行を制御します(“2.189 単純FORALL文”参照)。

構造FORALL 文

構造FORALL 文は、FORALL 構文の開始を示します。FORALL 構文は、複数の代入、配列選別代入(WHERE)および入れ子になったFORALL 構文または単純FORALL 文を制御します(“2.187 FORALL構文”参照)。

GO TO 文

GO TO 文は、指定された文番号をもつ飛び先文に制御移行します(“2.221 GO TO文”参照)。

計算形GO TO 文(廃止予定事項)

計算形GO TO 文は、指定された文番号並びのいずれか1つの飛び先文または、直後の文に制御移行します(“2.222 計算形GO TO文(廃止予定事項)”参照)。

割当て形GO TO 文(廃止事項)

割当て形GO TO 文は、ASSIGN 文で変数に割り当てられた文番号をもつ飛び先文に制御移行します(“2.223 割当て形GO TO文(廃止事項)”参照)。

IF 文

IF 文は、1つの実行文の実行を制御します(“2.278 IF文”参照)。

IF THEN 文

IF THEN 文は、IF 構文の開始を示します。IF 構文は、それを構成するブロックのうち、多くても1つの実行を選択します(“2.277 IF構文”参照)。

算術IF 文(廃止予定事項)

算術IF 文は、数値式の評価結果により、指定されたいずれかの文番号をもつ飛び先文に制御移行します(“2.280 算術IF文(廃止予定事項)”参照)。

INQUIRE 文

INQUIRE 文は、ファイルまたは装置の接続の性質について、問い合わせます(“2.287 INQUIRE文”参照)。

LOCK 文

LOCK 文は、ロック変数をロックします(“2.327 LOCK文”参照)。

NULLIFY 文

NULLIFY 文は、ポインタの結合状態を空状態にします(“2.369 NULLIFY文”および“1.5.14 ポインタ”参照)。

OPEN 文

OPEN 文は、外部ファイルと装置を接続したり、接続を修正したりします(“2.372 OPEN文”参照)。

PAUSE 文(廃止事項)

PAUSE 文は、プログラムの実行の一時中断を指定します(“2.377 PAUSE文(廃止事項)”参照)。

PRINT 文

PRINT 文は、出力項目並びおよび書式仕様で指定されたデータ要素から、ファイルに値を転送します(“2.386 PRINT文”参照)。

READ 文

READ 文は、ファイルから入力項目並びで指定されたデータ要素に、または変数群要素に値を転送します(“2.407 READ文”参照)。

RETURN 文

RETURN 文は、関数副プログラムまたはサブルーチン副プログラムの実行を終了します(“2.415 RETURN文”参照)。

REWIND 文

REWIND 文は、指定されたファイルをその始点に位置付けます(“2.416 REWIND文”参照)。

SELECT CASE 文

SELECT CASE 文は、CASE 構文の開始を示します。CASE 構文は、それを構成するブロックのうち、多くても1つの実行を選択します（“2.63 CASE構文”参照）。

SELECT TYPE 文

SELECT TYPE 文は、SELECT TYPE 構文の開始を示します。SELECT TYPE 構文は、それを構成するブロックのうち、多くても1つの実行を選択します（“2.431 SELECT TYPE構文”参照）。

STOP 文

STOP 文は、プログラムの実行を終了します（“2.460 STOP文”参照）。

SYNC ALL 文

SYNC ALL 文は、すべての像で同期を取ります（“2.467 SYNC ALL文”参照）。

SYNC IMAGES 文

SYNC IMAGES 文は、像と同期を取ります（“2.468 SYNC IMAGES文”参照）。

SYNC MEMORY 文

SYNC MEMORY 文は、セグメントを終了し、新しいセグメントを開始します（“2.469 SYNC MEMORY文”参照）。

UNLOCK 文

UNLOCK 文は、ロック変数をロック解除します（“2.494 UNLOCK文”参照）。

WAIT 文

WAIT 文は、指定した非同期データ転送操作に対して待機操作を行います（“2.502 WAIT文”参照）。

単純WHERE 文

単純WHERE 文は、論理配列式の値に従って、配列代入文における式の評価および代入を選別します（“2.505 単純WHERE文”参照）。

構造WHERE 文

構造WHERE 文は、WHERE 構文の開始を示します。WHERE 構文は、複数の配列代入および入れ子になった単純WHERE 文またはWHERE 構文の評価および代入を選別します（“2.503 WHERE構文”参照）。

WRITE 文

WRITE 文は、出力項目並びおよび書式仕様で指定されたデータ要素から、または変数群からファイルに値を転送します（“2.506 WRITE文”参照）。

1.9.2 非実行文

型宣言文

型宣言文は、データ実体の型、型パラメタおよびその他の属性を宣言します（“2.3 型宣言文”参照）。

ALLOCATABLE 文

ALLOCATABLE 文は、割付け変数を宣言します（“2.19 ALLOCATABLE文”参照）。

ASYNCHRONOUS 文

ASYNCHRONOUS 文は、並び中の実体に対して、ASYNCHRONOUS 属性を指定します（“2.31 ASYNCHRONOUS文”参照）。

AUTOMATIC 文

AUTOMATIC 文は、変数をスタックに割り付けることを宣言します（“2.41 AUTOMATIC文”参照）。

BIND 文

BIND 文は、並び中の変数や共通ブロックに対して、BIND 属性を指定します（“2.52 BIND文”参照）。

BLOCK DATA 文

BLOCK DATA 文は、初期値設定プログラム単位を開始します（“2.58 BLOCK DATA文”および“1.11.4 初期値設定プログラム単位”参照）。

BYTE 型宣言文

BYTE 型宣言文は、データ実体の型を1バイトの整数型として宣言します(“2.3 型宣言文”および“2.61 BYTE型宣言文”参照)。

CHANGEENTRY 文

CHANGEENTRY 文は、外部手続名の加工方法を変更します(“2.67 CHANGEENTRY文”参照)。

CHARACTER 型宣言文

CHARACTER 型宣言文は、データ実体の型を文字型として宣言します(“2.3 型宣言文”および“2.69 CHARACTER型宣言文”参照)。

CLASS 型宣言文

CLASS 型宣言文は、多相的実体を宣言します(“2.3 型宣言文”および“2.72 CLASS型宣言文”参照)。

CODIMENSION 文

CODIMENSION 文は、共配列を宣言します(“2.80 CODIMENSION文”および“1.17 共配列”参照)。

COMMON 文

COMMON 文は、共通ブロックを宣言します。共通ブロックは、プログラム中のどの有効域からも参照可能な物理的な記憶場所のブロックです。(“2.82 COMMON文”参照)。

COMPLEX 型宣言文

COMPLEX 型宣言文は、データ実体の型を複素数型として宣言します(“2.3 型宣言文”および“2.85 COMPLEX型宣言文”参照)。

CONTAINS 文

CONTAINS 文は、主プログラム、モジュールまたは副プログラムの本体を、それらが含む内部副プログラムまたはモジュール副プログラムから分離します(“2.87 CONTAINS文”参照)。

CONTIGUOUS 文

CONTIGUOUS文は、形状引継ぎ配列、ポインタ配列、または次元引継ぎがCONTIGUOUSであることを宣言します(“2.88 CONTIGUOUS文”参照)。

DATA 文

DATA 文は、変数に初期値を与えます(“2.112 DATA文”参照)。

DIMENSION 文

DIMENSION 文は、配列の形状を宣言します(“2.119 DIMENSION文”参照)。

DOUBLE PRECISION 型宣言文

DOUBLE PRECISION 型宣言文は、データ実体の型を倍精度実数型として宣言します(“2.3 型宣言文”および“2.123 DOUBLE PRECISION型宣言文”参照)。

END BLOCK DATA 文

END BLOCK DATA 文は、初期値設定プログラム単位の終了を指定します(“2.135 END BLOCK DATA文”および“1.11.4 初期値設定プログラム単位”参照)。

END ENUM 文

END ENUM 文は、列挙体宣言の終了を指定します。(“2.138 END ENUM文”参照)。

END INTERFACE 文

END INTERFACE 文は、引用仕様宣言の終了を指定します(“2.143 END INTERFACE文”および“1.12.7.2 手続引用仕様宣言”参照)。

END MAP 文

END MAP 文は、共用体宣言のブロックの終了を指定します(“1.5.11.1 派生型定義”および“2.144 END MAP文”参照)。

END MODULE 文

END MODULE 文は、モジュールの終了を指定します(“2.145 END MODULE文”および“1.11.2 モジュール”参照)。

END MODULE PROCEDURE副プログラム文

END MODULE PROCEDURE副プログラム文は、MODULE PROCEDURE副プログラムの終了を指定します(“2.146 END MODULE PROCEDURE副プログラム文”および“1.11.5 MODULE PROCEDURE副プログラム”参照)。

END STRUCTURE 文

END STRUCTURE 文は、STRUCTURE 文による派生型定義の終了を指定します(“1.5.11.1 派生型定義”および“2.149 END STRUCTURE文”参照)。

END SUBMODULE文

END SUBMODULE文は、サブモジュールの終了を指定します(“2.150 END SUBMODULE文”および“1.11.3 サブモジュール”参照)。

END TYPE 文

END TYPE 文は、TYPE 文による派生型定義の終了を指定します(“1.5.11.1 派生型定義”および“2.152 END TYPE文”参照)。

END UNION 文

END UNION 文は、共用体宣言の終了を指定します。(“1.5.11.1 派生型定義”および“2.153 END UNION文”参照)。

ENTRY 文

ENTRY 文は、追加の関数または追加のサブルーチンを定義します(“2.155 ENTRY文(廃止予定事項)”参照)。

ENUM 文

ENUM 文は、列挙体宣言の開始を指定します。(“2.156 ENUM文”参照)。

ENUMERATOR 文

ENUMERATOR 文は、列挙体を宣言します。(“2.157 ENUMERATOR文”参照)。

EQUIVALENCE 文

EQUIVALENCE 文は、有効域内で2つ以上の実体が同じ記憶単位を共有することを宣言します(“2.160 EQUIVALENCE文”参照)。

EXTERNAL 文

EXTERNAL 文は、外部手続名、仮手続名、および初期値設定プログラム単位名であることを宣言します(“2.179 EXTERNAL文”参照)。

FINAL 文

FINAL 文は、後始末束縛を定義します。(“2.182 FINAL文”参照)。

FORMAT 文

FORMAT 文は、データの内部表現と書式付き記録列としての文字列との間の明示的な編集情報を用意します。(“2.191 FORMAT文”および“1.8.1 書式仕様”参照)。

FUNCTION 文

FUNCTION 文は、関数副プログラムを開始し、関数および関数結果の特性を宣言します(“2.201 FUNCTION文”および“1.12.1 関数副プログラム”参照)。

IMPLICIT 文

IMPLICIT 文は、有効域内において、この文で指定された英字を第1文字としてもつ名前のデータ要素が、暗黙的に型宣言されるときの型および型パラメタを指定します。または、暗黙の型規則を適用しないことを指定します(“2.282 IMPLICIT文”参照)。

IMPORT 文

IMPORT 文は、引用仕様本体において、親有効域の名前付き要素を親子結合によって参照可能にすることを宣言します(“2.283 IMPORT文”参照)。

INTEGER 型宣言文

INTEGER 型宣言文は、データ実体の型を整数型として宣言します(“2.3 型宣言文”および“2.289 INTEGER型宣言文”参照)。

INTENT 文

INTENT 文は、仮引数の授受特性を宣言します(“2.290 INTENT文”参照)。

INTERFACE 文およびABSTRACT INTERFACE 文

INTERFACE 文およびABSTRACT INTERFACE 文は、引用仕様宣言を開始します。引用仕様宣言は、その手続を呼び出すときの引用の形を定義します。また、手続の総称引用仕様、利用者定義演算、および利用者定義代入を指定します(“2.291 INTERFACE文”および“1.12.7.2 手続引用仕様宣言”参照)。

INTRINSIC 文

INTRINSIC 文は、組込み手続であることを確認し、指定された個別組込み関数名を実引数として使用できるようにします(“2.292 INTRINSIC文”参照)。

LOGICAL 型宣言文

LOGICAL 型宣言文は、データ実体の型を論理型として宣言します(“2.3 型宣言文”および“2.332 LOGICAL型宣言文”参照)。

MAP 文

MAP 文は、STRUCTURE 文による派生型定義内において、共用体宣言のブロックを開始します(“1.5.11.1 派生型定義”および“2.341 MAP文”参照)。

MODULE 文

MODULE 文は、モジュールを開始します(“2.356 MODULE文”および“1.11.2 モジュール”参照)。

MODULE PROCEDURE副プログラム文

MODULE PROCEDURE副プログラム文は、MODULE PROCEDURE副プログラムを開始します(“2.357 MODULE PROCEDURE副プログラム文”および“1.11.5 MODULE PROCEDURE副プログラム”参照)。

NAMELIST 文

NAMELIST 文は、変数群入出力文において、変数群名で参照できる名前付きデータ実体群を宣言します(“2.361 NAMELIST文”参照)。

OPTIONAL 文

OPTIONAL 文は、手続の引用において、仮引数が必ずしも実引数と結合しなくてもよいことを宣言します(“2.373 OPTIONAL文”参照)。

PARAMETER 文

PARAMETER 文は、名前付き定数を宣言します(“2.375 PARAMETER文”参照)。

POINTER 文

POINTER 文は、ポインタを宣言します(“2.379 POINTER文”および“1.5.14 ポインタ”参照)。

POINTER 文(CRAY 仕様)

POINTER 文(CRAY 仕様)は、アドレス保持変数と、そのアドレスによって指されるポインタ変数の組を宣言します(“2.380 POINTER文(CRAY 仕様)”参照)。

PRIVATE 文

モジュールの宣言部のPRIVATE 文は、言語要素がそのモジュール内だけで参照可能であることを宣言します(“2.387 PRIVATE文”参照)。派生型定義の中のPRIVATE 文は、その型の成分の名前が、そのモジュール内でだけ参照可能であることを宣言します(“1.5.11.1 派生型定義”参照)。

PROCEDURE 文

PROCEDURE 文は、手続成分定義、個別束縛、総称引用仕様をもつモジュール手続宣言、手続宣言定義します(“2.389 PROCEDURE文”および“1.12.7.2 手続引用仕様宣言”参照)。

PROGRAM 文

PROGRAM 文は、主プログラムを開始します(“2.392 PROGRAM文”および“1.11.1 主プログラム”参照)。

PROTECTED 文

PROTECTED 文は、モジュールの宣言部で名前付き変数を指定し、その変数を参照結合した有効範囲で引用できる箇所を限定します(“2.394 PROTECTED文”参照)。

PUBLIC 文

PUBLIC 文は、モジュール内に指定し、言語要素が他のプログラム単位からUSE 文によって参照可能であることを宣言します(“2.395 PUBLIC文”)。

REAL 型宣言文

REAL 型宣言文は、データ実体の型を実数型として宣言します(“2.3 型宣言文”および“2.409 REAL型宣言文”参照)。

RECORD 文

RECORD 文は、派生型のデータ実体を宣言します(“2.410 RECORD文”参照)。

SAVE 文

SAVE 文は、RETURN 文またはEND 文の実行後も結合状態、割付け状態、定義状態および値を保持することを宣言します(“2.422 SAVE文”参照)。

SEQUENCE 文

SEQUENCE 文は、派生型定義中に指定し、その型の成分を指定された順に格納することを宣言します(“1.5.11.1 派生型定義”および“2.432 SEQUENCE文”参照)。

STATIC 文

STATIC 文は、変数をメモリ上に割り付け、かつSAVE 属性をもつことを宣言します(“2.459 STATIC文”参照)。

STRUCTURE 文

STRUCTURE 文は、派生型の定義を開始します(“1.5.11.1 派生型定義”および“2.462 STRUCTURE文”参照)。

SUBMODULE文

SUBMODULE文は、サブモジュールを開始します(“2.463 SUBMODULE文”および“1.11.3 サブモジュール”参照)。

SUBROUTINE 文

SUBROUTINE 文は、サブルーチン副プログラムを開始し、サブルーチンの特性を宣言します(“2.464 SUBROUTINE文”および“1.12.2 サブルーチン副プログラム”参照)。

TARGET 文

TARGET 文は、ポインタと結合できるTARGET 属性をもつ実体を宣言します(“2.476 TARGET文”参照)。

TYPE 文(派生型定義)

TYPE 文は、派生型の定義を開始します(“1.5.11.1 派生型定義”および“2.487 TYPE文(派生型定義)”参照)。

TYPE 型宣言文

TYPE 型宣言文は、データ実体の型を指定された派生型として宣言します(“2.3 型宣言文”および“2.488 TYPE型宣言文”参照)。

UNION 文

UNION 文は、STRUCTURE 文による派生型定義内において、共用体宣言を開始します(“1.5.11.1 派生型定義”および“2.492 UNION文”参照)。

USE 文

USE 文は、その有効域内から指定されたモジュール内の公開要素を参照可能にします(“2.496 USE文”参照)。

VALUE 文

VALUE 文は、仮引数が、実引数の値を初期値とする確定可能である一時的な領域と結合することを指定します。仮引数の値および定義状態のその後の変更は、実引数に影響を与えません(“2.498 VALUE文”参照)。

VOLATILE 文

VOLATILE 文は、実体を最適化の対象としないことを宣言します(“2.500 VOLATILE文”参照)。

文関数定義文(廃止予定事項)

文関数定義文は、文関数を定義します。文関数は、単一の文によって定義される関数です(“2.4 文関数定義文(廃止予定事項)”参照)。

1.9.3 文の順序

プログラム単位および副プログラムの中での、文の出現順序には規定があります。

- USE 文は、宣言部の先頭に記述します。
- USE 文とCONTAINS 文の間では、非実行文は実行文より前に記述します。ただし、FORMAT 文、ENTRY 文、およびDATA 文は、実行文の間に指定することができます。
- モジュール手続および内部手続は、CONTAINS 文の後に記述します。

文の出現順序を、以下の表に示します。この表において、縦線は混在できる文の種類を区切り、横線は混在できない文の種類を区切ります。

<table>
<tr><td colspan="3">PROGRAM文、FUNCTION文、SUBROUTINE文、MODULE文、SUBMODULE文、またはBLOCK DATA文</td></tr>
<tr><td colspan="3">USE文</td></tr>
<tr><td colspan="3">IMPORT文</td></tr>
<tr><td rowspan="4">FORMAT文およびENTRY文</td><td colspan="2">IMPLICIT NONE</td></tr>
<tr><td>PARAMETER文</td><td>IMPLICIT文</td></tr>
<tr><td>PARAMETER文および
DATA文</td><td>派生型定義、引用仕様宣言、型宣言文、列挙体定義、手続宣言、単純宣言文、および文関数定義文</td></tr>
<tr><td>DATA文（廃止予定事項）</td><td>実行構文</td></tr>
<tr><td colspan="3">CONTAINS文</td></tr>
<tr><td colspan="3">内部副プログラムまたはモジュール副プログラム</td></tr>
<tr><td colspan="3">END文</td></tr>
</table>

それぞれの有効域（“1.14 有効範囲”参照）で許される文を以下の表に示します。

有効域の種類	主プログラム	モジュール、およびサブモジュール	初期値設定プログラム単位	外部副プログラム	モジュール副プログラム	内部副プログラム	引用仕様本体
USE文	○	○	○	○	○	○	○
IMPORT文	×	×	×	×	×	×	○
ENTRY文	×	×	×	○	○	×	×
FORMAT文	○	×	×	○	○	○	×
PARAMETER文、IMPLICIT文、型宣言文、および単純宣言文	○	○	○	○	○	○	○
DATA文	○	○	○	○	○	○	×
派生型定義	○	○	○	○	○	○	○
引用仕様宣言	○	○	×	○	○	○	○
実行文	○	×	×	○	○	○	×
CONTAINS文	○	○	×	○	○	×	×
文関数定義文(廃止予定事項)	○	×	×	○	○	○	×

1.10 構造構文

構造構文は、文の実行または実行順序を制御します。

- IF 構文は、それを構成するブロックのうち、どのブロックを実行するかを制御します(“2.277 IF構文”参照)。
- CASE 構文は、それを構成するブロックのうち、どのブロックを実行するかを制御します(“2.63 CASE構文”参照)。
- DO 構文は、実行構文の列の実行を制御します(“2.120 DO構文”参照)。
- WHERE 構文は、配列代入文の評価する要素を制御します(“2.503 WHERE構文”参照)。
- FORALL 構文は、複数の代入、配列選別代入(WHERE)および入れ子になったFORALL 構文または単純FORALL 文を制御します(“2.187 FORALL構文”参照)。
- ASSOCIATE 構文は、ブロックの実行中に、名前付き要素に対して式または変数を結合します(“2.29 ASSOCIATE構文”参照)。
- SELECT TYPE 構文は、式の実行時の型に基づいた実行を選択できるようにします(“2.431 SELECT TYPE構文”参照)。
- BLOCK 構文は、宣言を含むことができる実行構文です(“2.57 BLOCK構文”参照)。
- CRITICAL 構文は、複数の像が同時にブロックを実行することを制限します(“2.102 CRITICAL構文”参照)。

1.10.1 構文名

構造構文には、構文名を指定することができます。構文名を指定する場合は、構造構文の最初の文と、対応する最後の文の両方に指定しなければなりません。構造構文中の文は、構文名の指定がなければ、最も内側の構造構文に属し、構文名の指定があれば、その名前の構造構文に属します。例えば、CYCLE 文およびEXIT 文は構文名を指定することにより、最も内側のDO 構文以外の、すなわち、入れ子の外側のDO 構文に属し、そのDO 構文のループ範囲を飛び越したり、ループを終了させたりすることができます。

1.11 プログラム単位

プログラム単位は、分割して翻訳できるFortran プログラムの最小単位です。プログラム単位には、以下の7つが有ります。

- 主プログラム
- 外部関数副プログラム
- 外部サブルーチン副プログラム
- モジュール
- サブモジュール
- 初期値設定プログラム単位
- MODULE PROCEDURE副プログラム

外部関数副プログラムについては“1.12.1 関数副プログラム”を、外部サブルーチン副プログラムについては、“1.12.2 サブルーチン副プログラム”を参照してください。

1.11.1 主プログラム

Fortran プログラムは必ず1つの主プログラムを含まなければなりません。プログラムの実行は、主プログラムの最初の実行文から始まり、プログラム中のSTOP 文、または主プログラムのEND 文で終了します。

主プログラムは以下の形式です。

```
[ PROGRAM program-name ]
    [ use-stmts ]
    [ specification-part ]
    [ execution-part ]
    [ internal-subprogram-part ]
END [ PROGRAM [ program-name ] ]
```

program-name は主プログラム名です。

use-stmts は1つ以上のUSE 文です。

specification-part は、OPTIONAL 文、INTENT 文、PUBLIC 文、PRIVATE 文およびVALUE 文を除く1つ以上の宣言文、派生型定義、または引用仕様宣言です。

execution-part は、RETURN 文およびENTRY 文を除く1つ以上の実行文です。

internal-subprogram-part は、CONTAINS 文とそれに続く省略可能な内部手続です。

END PROGRAM文に*program-name*を指定する場合、対応するPROGRAM文に指定された*program-name*と同じでなければなりません。

主プログラム中に、自動割付け変数は宣言できません。

1.11.2 モジュール

モジュールは、その中にある宣言および定義を、他のプログラム単位から参照できるようにします。モジュールには、名前付き変数宣言、名前付き定数宣言、派生型定義、引用仕様宣言、モジュール副プログラム、および他のモジュール引用を含むことができます。モジュール内で定義または宣言された名前は、PUBLIC 文またはPRIVATE 文に指定することにより、そのモジュールを参照する他のプログラム単位内で参照可能かどうかを制御することができます。モジュール宣言部の変数、共通ブロック、または手続ポインタは暗黙のSAVE属性をもちます。

モジュールの使用例としては、以下のものがあります。

- プログラム内で共通に使用するデータの宣言および初期化
- 外部副プログラムの明示引用仕様の宣言
- 派生型の定義、およびその派生型の利用者定義演算の宣言

モジュールは、以下の形式です。

```
MODULE module-name
    [ use-stmts ]
    [ specification-part ]
    [ module-subprogram-part ]
END [ MODULE [ module-name ] ]
```

module-name は、モジュール名です。

use-stmts は、1つ以上のUSE 文です。

specification-part は、OPTIONAL 文、INTENT 文、およびVALUE 文を除く1つ以上の宣言文、派生型定義、または引用仕様宣言です。

module-subprogram-part は、CONTAINS 文とそれに続く省略可能なモジュール手続です。

END MODULE 文に*module-name* を指定する場合、対応するMODULE文に指定された*module-name* と同じでなければなりません。

モジュールの例:

```
module example
  implicit none
  integer,dimension(2,2) :: bar1=1, bar2=2
  type phone_number                          ! 派生型定義
    integer :: area_code, number
  end type phone_number
  interface                                  ! 引用仕様宣言
    function test(sample,result)
      implicit none
      real :: test
      integer , intent(in) :: sample,result
    end function test
    function count(total)
      implicit none
      integer :: count
      real , intent(in) :: total
    end function count
  end interface
  interface swap
```

```
    module procedure swap_reals, swap_integers
  end interface swap
  contains
    subroutine swap_reals(r1,r2)          ! モジュール手続
      real , intent(inout) :: r1,r2
      real :: t
    t = r1 ; r1 = r2 ; r2 = t
    end subroutine swap_reals
    subroutine swap_integers(i1,i2)       ! モジュール手続
      integer ,intent(inout) :: i1,i2
      integer :: t
      t = i1 ; i1 = i2 ; i2 = t
    end subroutine swap_integers
end module example
```

1.11.2.1 モジュール手続

モジュール手続は、外部手続と同じ構成で定義されます。手続の構成については、“1.12.1 関数副プログラム”および“1.12.2 サブルーチン副プログラム”を参照してください。モジュール手続は、その親プログラムのモジュールを引用しているプログラム単位内で、参照可能になります。モジュール手続内では、親プログラムのモジュール内の名前はPRIVATE 属性をもつ名前も含めて、参照することができます。

1.11.2.2 モジュール引用

モジュール内の情報は、USE 文によってモジュールを引用することにより、そのプログラム単位内で有効になります。これを参照結合といいます。

USE 文の例:

```
use mod_sample
```

この例では、モジュールmod_sample 内で宣言および定義された名前が、参照可能になります。また、以下のような仮称指定を行うことにより、モジュール内の名前を別の名前で参照することも可能です。

仮称指定の例:

```
use mod_sample ,a => b
```

この例では、モジュールmod_sample 内の実体b をa という名前で参照することを宣言します。また、USE 文にONLY 句を指定することにより、モジュール内の特定な名前だけを参照することも可能です。

ONLY 句指定の例:

```
use mod_sample , only : c , d
```

この例では、モジュールmod_sample 内の名前のc およびd だけが、参照可能になります。

1.11.3 サブモジュール

サブモジュールは、モジュールまたはほかのサブモジュールを拡張します。サブモジュールをUSE文による参照結合で使用することはできません。

拡張するプログラム単位は祖先モジュール名および親サブモジュール名で指定します。モジュールの非公開成分は、そのモジュールを拡張するサブモジュール(子孫サブモジュール)から参照可能です。モジュール内の引用仕様本体において手続接頭辞MODULEが指定された(分離引用仕様)手続は、定義時に手続接頭辞MODULEを指定することによって子孫サブモジュール中で定義できます。その際、子孫サブモジュールにおいてMODULE PROCEDURE副プログラム文を指定すると(MODULE PROCEDURE 副プログラム)、分離引用仕様により宣言された手続の特性および引数と同じであるとみなされ、記述を省略できます。モジュール副プログラム部において手続接頭辞MODULEが指定された関数、手続接頭辞MODULEが指定されたサブルーチン、またはMODULE PROCEDURE副プログラムを分離モジュール手続といいます。分離引用仕様本体は、IMPORT文を指定しなくても親有効域の要素を親子結合で引用することができます。分離引用仕様本体中にIMPORT文を指定することはできません。

サブモジュールは以下の形式です。

```
SUBMODULE ( parent-identifier ) submodule-name
  [ use-stmts ]
```

```
  [ specification-part ]
  [ module-subprogram-part ]
END [ SUBMODULE [ submodule-name ] ]
```

*parent-identifier*は、親識別子です。

*submodule-name*は、サブモジュール名です。

*use-stmts*は、1つ以上のUSE文です。

*specification-part*は、1つ以上の宣言です。

module-subprogram-part は、CONTAINS 文とそれに続く省略可能なモジュール手続です。

親識別子は以下の形式です。

ancestor-module-name [: *parent-submodule-name*]

ancestor-module-name は、祖先モジュール名です。

parent-submodule-name は、親サブモジュール名です。

サブモジュールは、祖先モジュール名とサブモジュール名の組によって識別されます。

END SUBMODULE文に*submodule-name*を指定する場合、対応するSUBMODULE文に指定された*submodule-name*と同じでなければなりません。

サブモジュールからは、祖先(親サブモジュール、その先祖親サブモジュール、祖先モジュール)の要素を親子結合により引用できます。

サブモジュールの例:

```
module mod
  interface
    module subroutine sub1(val)  ! 分離引用仕様本体
      integer, intent(in) :: val
    end subroutine sub1

    module subroutine sub2(val)  ! 分離引用仕様本体
      integer, intent(in) :: val
    end subroutine sub2
  end interface
end module

submodule(mod)submod
  contains
    module subroutine sub1(val)   ! 分離モジュール手続
      integer, intent(in) :: val  ! 特性および仮引数名は上記のsub1と一致させる
    end subroutine sub1

    module procedure sub2          ! MODULE PROCEDURE 文を使用すると
    end procedure sub2              ! 特性および仮引数名を省略できる
end submodule submod
```

1.11.4 初期値設定プログラム単位

初期値設定プログラム単位は、名前付き共通ブロック中のデータ実体に初期値を与えます。初期値設定プログラム単位には、実行文は指定できません。

初期値設定プログラム単位は以下の形式です。

```
BLOCK DATA [ block-data-name ]
    [ use-stmts ]
    [ specification-part ]
END [ BLOCK DATA [ block-data-name ] ]
```

block-data-name は初期値設定プログラム単位名です。BLOCK DATA 文の*block-data-name* は指定しなくてもかまいませんが、1つのプログラムに、2つ以上の名前のない初期値設定プログラム単位があってはなりません。

use-stmts は、1つ以上のUSE 文です。

specification-part は、ALLOCATABLE 文、INTENT 文、PUBLIC 文、PRIVATE 文、OPTIONAL 文、EXTERNAL 文、およびVALUE 文を除く、1つ以上の宣言文または派生型定義です。

END BLOCK DATA 文に*block-data-name* を指定する場合、対応するBLOCK DATA 文に指定された*block-data-name* と同じでなければなりません。

1.11.5 MODULE PROCEDURE副プログラム

MODULE PROCEDURE 副プログラムは、分離モジュール手続(“1.11.3 サブモジュール”参照)の1つです。

MODULE PROCEDURE 副プログラムは、以下の形式です。

```
MODULE PROCEDURE procedure-name
  [ specification-part ]
  [ execution-part ]
  [ internal-subprogram-part ]
END [ PROCEDURE [ procedure-name ] ]
```

*procedure-name*は、サブルーチン名または関数名です。

*specification-part*は、1つ以上の宣言です。

*execution-part*は、1つ以上の実行文です。

*internal-subprogram-part*は、CONTAINS文とそれに続く省略可能な内部手続です。

MODULE PROCEDURE 副プログラムの引用仕様(特性、仮引数名)は、分離引用仕様本体で宣言されたものと同じになります。

1.12 手続

手続は、任意の処理の列をまとめたものであり、プログラムの実行中に直接呼び出すことができます。手続は関数またはサブルーチンのいずれかです。関数は、式の中で関数引用または利用者定義演算を通して呼び出される手続です。サブルーチンは、CALL 文で、または利用者定義代入文によって呼び出される手続です。

手続は、その定義の方法により、以下のように分類されます。

- 組込み手続

  組込み手続は、処理系の一部としてあらかじめ用意されている手続です。各組込み手続の詳細については、“第2章 文および手続の詳細”を参照してください。また、組込み手続の一覧表が、“付録A 組込み手続一覧”に用意されていますので併せて参照してください。

- 外部手続

  外部手続は、外部副プログラムによって、またはFortran 以外の手段によって定義される手続です。外部手続は、主プログラムから、またはプログラムの任意の手続から呼び出すことが可能です。

- モジュール手続

  モジュール手続は、モジュール副プログラムによって定義される手続です。モジュール手続は、そのモジュール中の他のモジュール副プログラムからも、そのモジュール手続を参照結合している有効域からも呼び出すことが可能です。

- 内部手続

  内部手続は、内部副プログラムによって定義される手続です。その内部副プログラムを含む主プログラムまたは副プログラムを、その内部手続の親プログラムと呼び、内部手続は親プログラムの有効域、および親プログラムの他のすべての内部手続から呼び出すことが可能です。その他の場所からは呼び出すことはできません。

- 仮手続

  仮手続は、手続として指定した仮引数、および手続引用に書いた仮引数です。仮手続と結合する実引数は、外部手続、モジュール手続、内部手続、仮手続、または組込み手続の個別名でなければなりません。仮手続を手続引用すると、結合している実引数の手続が呼び出されます。

- 仮引数でない手続ポインタ

  手続ポインタは、EXTERNAL 属性およびPOINTER 属性をもつ手続です。手続ポインタは、外部手続、モジュール手続、内部手続、組込み手続、または手続ポインタではない仮手続とポインタ結合が可能です。

- 文関数(廃止予定事項)

  単一の文によって定義される関数のことを、文関数と呼びます。

1.12.1 関数副プログラム

関数副プログラムは、外部関数副プログラム、モジュール関数副プログラム、または内部関数副プログラムを定義します。

関数副プログラムは、以下の形式です。

```
[ prefix-spec ]... FUNCTION function-name ( [ dummy-arg-name-list ] ) [ suffix ]
    [ use-stmts ]
    [ specification-part ]
    [ execution-part ]
    [ internal-subprogram-part ]
END [ FUNCTION [ function-name ] ]
```

prefix-spec は、以下の形式です。

```
type-spec          または
ELEMENTAL          または
IMPURE             または
MODULE             または
PURE               または
RECURSIVE
```

type-spec は宣言型指定子です。宣言型指定子については、“2.3 型宣言文”を参照してください。

*prefix-spec*にMODULEを指定した関数副プログラムは、分離モジュール手続です(“1.11.3 サブモジュール”参照)。

function-name は、関数名です。

dummy-arg-name-list は、コンマで区切られた仮引数名の並びです。

suffix は、次のいずれかです。

```
手続言語束縛指定子      [ RESULT ( result-name ) ]        または
RESULT ( result-name ) [ 手続言語束縛指定子]
```

result-name は結果名であり、関数の結果変数を指定します。RESULT 句を指定しない場合、結果変数は関数名です。

手続言語束縛指定子は、以下の形式です。

```
BIND ( C [, NAME = スカラ文字定数式] )
```

use-stmts は1つ以上のUSE 文です。

specification-part は、PUBLIC 文およびPRIVATE 文を除く1つ以上の宣言文、派生型定義、または引用仕様宣言です。

execution-part は、1つ以上の実行文です。

internal-subprogram-part は、CONTAINS 文とそれに続く省略可能な内部手続です。

END FUNCTION 文に*function-name*を指定する場合、対応するFUNCTION 文に指定された*function-name*と同じでなければなりません。

内部関数副プログラムは、ENTRY 文および*internal-subprogram-part* を含むことはできません。

関数副プログラムおよび関数引用の例:

```
program main
  implicit none
  interface                    ! 手続引用仕様宣言
```

```
    function square(x)
      implicit none
      real , intent(in) :: x
      real :: square
    end function square
  end interface
  real :: a , b=3.6 , c=3.8
  a = 3.7 + b + square(c) + sin(4.7)
  print *,a
  stop
end program main

function square(x)
  implicit none
  real, intent(in) :: x
  real :: square
  square = x*x
  return
end function square
```

この例で、squareは外部関数副プログラムであり、square(c)およびsin(4.7)は関数引用です。

1.12.2 サブルーチン副プログラム

サブルーチン副プログラムは、外部サブルーチン副プログラム、モジュールサブルーチン副プログラム、または内部サブルーチン副プログラムを定義します。

サブルーチン副プログラムは、以下の形式です。

```
[ prefix-spec ]... SUBROUTINE subroutine-name [ ( [ dummy-arg-list ] ) proc-language-binding-spec ]
    [ use-stmts ]
    [ specification-part ]
    [ execution-part ]
    [ internal-subprogram-part ]
END [ SUBROUTINE [ subroutine-name ] ]
```

prefix-spec は、以下の形式です。

```
ELEMENTAL          または
IMPURE             または
MODULE             または
PURE               または
RECURSIVE
```

*prefix-spec*にMODULEを指定したサブルーチン副プログラムは、分離モジュール手続です(“1.11.3 サブモジュール”参照)。

subroutine-name は、サブルーチン名です。

dummy-arg-list は、コンマで区切られた仮引数の並びです。

dummy-arg は、以下の形式です。

```
dummy-arg-name          または
*
```

dummy-arg-name は、仮引数名です。

proc-language-binding-spec は手続言語束縛指定子であり、以下の形式です。

```
BIND ( C [, NAME = スカラ文字定数式] )
```

use-stmts は1つ以上のUSE 文です。

specification-part は、PUBLIC 文およびPRIVATE 文を除く1つ以上の宣言文、派生型定義、または引用仕様宣言です。

execution-part は、1つ以上の実行文です。

internal-subprogram-part は、CONTAINS 文とそれに続く省略可能な内部手続です。

END SUBROUTINE 文に*subroutine-name*を指定する場合、対応するSUBROUTINE 文に指定された*subroutine-name*と同じでなければなりません。

内部サブルーチン副プログラムは、ENTRY 文および*internal-subprogram-part*を含むことはできません。

サブルーチン副プログラムおよびサブルーチン引用の例:

```
program main
  implicit none
  interface                    ! 手続引用仕様宣言
    subroutine multiply(x,y)
      implicit none
      real, intent(inout) :: x
      real, intent(in) :: y
    end subroutine multiply
  end interface
  real :: a,b
  a = 4.0
  b = 12.0
  call multiply(a,b)
  print *,a
end program main

subroutine multiply(x,y)
  implicit none
  real, intent(inout) :: x
  real, intent(in) :: y
  x = x*y
end subroutine
```

この例で、multiply は外部サブルーチン副プログラムであり、CALL 文によって、呼び出されます。

1.12.3 再帰的引用

手続は、直接または間接に、その手続自身またはその同じ副プログラム内のENTRY 文で定義された手続を呼び出すことができます。このような呼出しを再帰的呼出しといいます。このような場合には、FUNCTION 文またはSUBROUTINE 文の*prefix-spec*に‘RECURSIVE’を指定しなければなりません(“2.201 FUNCTION文”および“2.464 SUBROUTINE文”参照)。また、関数が直接再帰呼出しを行う場合、呼び出す関数を定義するFUNCTION 文またはENTRY 文にRESULT 句も指定しなければなりません。

1.12.4 純粋手続

FUNCTION 文およびSUBROUTINE 文の*prefix-spec*に‘PURE’、または‘ELEMENTAL’を指定し、‘IMPURE’をもたない場合、その手続は純粋手続となります。純粋手続は副作用をもたないことを意味します。純粋手続には、以下の制約があります。

- 純粋関数の仮引数は、手続引数およびPOINTER 属性をもつ引数を除いて、INTENT(IN) 属性、またはVALUE属性をもたなければなりません。
- 純粋サブルーチンの仮引数は、手続引数およびPOINTER 属性をもつ引数を除いて、INTENT 属性をもたなければなりません。
- 純粋手続の結果は多相的割付け可能、または多相的割付け可能成分をもってはなりません。
- 純粋サブルーチンのポインタでない仮データオブジェクトは、INTENT属性を指定し、VALUE属性をもってはなりません。
- 純粋手続のINTENT(OUT)の仮引数は多相的、または多相的割付け可能成分をもってはなりません。
- 純粋手続の手続引数はすべて純粋でなければなりません。
- 純粋手続の局所変数は明示的初期値指定およびSAVE 属性をもってはなりません。
- 純粋手続中のすべての内部手続は、純粋でなければなりません。

- 純粋手続にVOLATILE属性の変数の特定子が現れてはなりません。
- 純粋手続において、共通ブロック内の変数、親子結合または参照結合によって参照可能な変数、純粋関数の仮引数、INTENT(IN)属性をもつ純粋サブルーチンの仮引数、共添字付き実体、またはそれらと記憶域結合している実体は、値を確定、結合状態を変更または引継ぎ、および割付け状態を変更するような個所に指定してはなりません。
- 純粋手続中で引用する手続は、すべて純粋手続でなければなりません。
- 純粋手続中で引用する定義代入、定義演算、定義入出力、または後始末手続は、純粋手続でなければなりません。
- 純粋手続は、内部ファイル入出力文以外のすべての入出力文を含んではなりません。
- 純粋手続は、STOP 文またはERROR STOP 文を含んではなりません。
- 純粋手続は、像制御文(“1.18 像制御文”参照)を含んではなりません。

1.12.5 要素別処理手続

要素別処理手続は、引数に指定された配列の要素ごとに、処理される手続です。

1.12.5.1 要素別処理手続宣言

要素別処理手続は、要素別処理組込み手続または要素別処理副プログラムによって定義された手続です。

FUNCTION 文およびSUBROUTINE 文の*prefix-spec* に、手続接頭辞ELEMENTAL を指定した場合、その手続は要素別処理手続となります。

手続接頭辞ELEMENTAL は、手続接頭辞IMPUREをもつ場合を除いて、純粋であることを包含するので、手続接頭辞PURE を書く必要はありません。要素別処理手続には、以下の制約があります。

- 要素別処理手続のすべての仮引数は、スカラ仮データ実体でなければならず、POINTER 属性またはALLOCATABLE 属性をもってはなりません。共配列であってはなりません。
- 要素別処理関数の結果変数はスカラでなければならず、POINTER属性もALLOCATABLE 属性ももってはなりません、かつ、型パラメタが定数式でない式の定義をもってはなりません。
- 仮引数は、仮手続であってはなりません。
- 仮引数は、‘*’であってはなりません。

要素別処理手続は、純粋手続です。純粋手続の制約もすべて適用されます(“1.12.4 純粋手続”参照)。

1.12.5.2 要素別処理関数の実引数および結果

要素別処理関数を総称名または個別名で参照した場合、結果の形状は、実引数の指定がないか、またはすべての実引数がスカラであるとき、スカラです。配列が実引数に現れるなら、その形状になります。2つ以上の引数をもつ要素別処理関数の場合、すべての実引数は、形状適合していなければなりません。配列の場合、結果の各要素の値は、それぞれに対応する各引数の要素に、スカラの関数を(順序によらず)個別に適用して得られる値になります。

1.12.5.3 要素別処理サブルーチンの実引数

要素別処理サブルーチンを引用する場合、次のいずれかでなければなりません。

- すべての実引数がスカラである。
- INTENT(OUT) 属性またはINTENT(INOUT) 属性をもつ仮引数と結合した実引数は、すべて同じ形状の配列であり、他の引数がそれらと形状適合している。

INTENT(OUT) 属性またはINTENT(INOUT) 属性をもつ仮引数と結合した実引数が配列であるとき、各要素の値は、それぞれに対応する各引数の要素にスカラ値の関数を(順序によらず)個別に適用して得られる値になります。

1.12.6 手続引用

関数引用は以下の形式です。

procedure-designator ([*actual-arg-spec-list*])

procedure-designator は関数を特定しなければなりません。

関数は、利用者定義演算としても引用できます(“1.12.7.3.2 利用者定義演算”参照)。関数は、式の評価の間に呼び出されます。関数を呼び出すと、すべての実引数の式が評価され、次に引数が結合され、最後に関数が実行されます。関数の実行が終了したとき、関数結果の値は、それを呼び出した式において、使用可能となります。

サブルーチン引用は以下の形式です。

CALL *procedure-designator* [([*actual-arg-spec-list*])]

procedure-designator は手続特定子です。以下の形式です。

procedure-name または
proc-component-ref または
data-ref % *binding-name*

procedure-name は、手続名です。

proc-component-ref は、手続成分の引用です。手続成分は、手続ポインタ成分の名前でなければなりません。

data-ref %*binding-name* は、データ参照%束縛名です。

サブルーチンは、利用者定義代入としても引用することができます(“1.12.7.3.3 利用者定義代入”参照)。

data-ref はデータ参照です。

binding-name は束縛名です。

データ参照が配列である場合、引用する型束縛手続は、PASS 属性をもたなければなりません。

actual-arg-spec-list はコンマで区切られた実引数指定子の並びです。

actual-arg-spec は以下の形式です。

[*keyword* =] *actual-arg*

keyword は引数キーワードであり、仮引数名でなければなりません。

actual-arg は実引数であり、以下の形式です。

expr または
variable または
procedure-name または
proc-component-ref または
**label* または
%VAL (*expr*)

expr は、式です。

variable は変数です。

procedure-name は手続名です。組込み手続でない要素別処理手続は、実引数に指定することはできません。実引数の手続名は、文関数の名前であってはならず、その名前が個別名でもある場合を除いて手続の総称名であってはなりません。

proc-component-ref は手続成分引用です。

**label* は選択戻り指定子(廃止予定事項)です。*label* は文番号であり、CALL 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。選択戻り指定子は、関数引用の実引数には指定できません。

%VAL は、実引数を値渡しにすることを指定します。

1.12.6.1 手続の引数

引数は、手続を呼び出す側から、呼び出される手続に情報を渡すのに使用します。呼び出される手続で仮引数を宣言し、呼び出す側の手続引用において、仮引数に対応する実引数を指定します。実引数の型、型パラメタ、および形状は、その手続の仮引数の特性に適合しなければなりません。

実引数の並びは、指定した実引数と、手続の仮引数の間の対応を識別します。引数キーワードを省略した場合には、実引数は仮引数並びの対応する位置の仮引数と結合します。すなわち、1番目の実引数は1番目の仮引数と、2番目の実引数は2番目の仮引数と、というように順に結合します。引数キーワードを書いた場合には、実引数は、引数キーワードと同じ名前をもつ仮引数と結合します。この場合、手続引用がある有効域内で参照可能な引用仕様から仮引数名を使用します。それぞれの省略可能でない仮引数には、1つの実引数が結合しなければなりません。それぞれの省略可能な仮引数には、実引数は結合しなくてもかまいませんし、1つの実引数が結合してもかまいません。それぞれの実引数は、仮引数と結合しなければなりません。

1.12.6.1.1 仮引数の授受特性

仮引数にINTENT 属性(“2.290 INTENT文”参照)を指定することにより、仮引数の使用意図を明確にします。詳細については“2.290 INTENT文”を参照してください。

INTENT 属性は、INTENT 文または、型宣言文のINTENT 属性により指定することができます。

1.12.6.1.2 引数キーワード

引数キーワード‘*keyword*=’を指定すると、実引数は実引数並び中の位置に関係なく、*keyword*に指定された仮引数と結合します。引数キーワードの指定がない場合、実引数は、仮引数並びの対応する位置の仮引数と結合します。

引数キーワードは、手続の引用仕様が明示的(“1.12.7 手続引用仕様”参照)である場合に、指定することができます。引数キーワードは、その実引数が仮引数並びの対応する位置の仮引数と結合しない場合、および実引数並びの先行する実引数に引数キーワードの指定がある場合に、指定しなければなりません。

引数キーワードの例:

```
interface
  subroutine zee(a,b,c)
    integer :: a,b,c
  end subroutine
end interface
call zee(c=1,b=2,a=3)
```

この例で、実引数は指定されたのと逆の順に結合します。

1.12.6.1.3 省略可能な仮引数

仮引数がOPTIONAL 属性をもっている場合、対応する実引数を省略することができます。

次の条件のとき、仮引数は実引数と結合しません。

- 仮引数に対応する実引数が存在しない。または、
- 仮引数に対応する実引数が存在し、仮引数がポインタまたは割付け変数でない。かつ、
  - 対応する実引数は割付け実体であり、未割付けである。または、
  - 対応する実引数はポインタであり、空状態である。

OPTIONAL 属性は、OPTIONAL 文または、型宣言文のOPTIONAL 属性により指定することができます。

手続が省略可能な仮引数をもっている場合、その手続の引用仕様は明示的でなければなりません(“1.12.7 手続引用仕様”参照)。

省略可能な仮引数が実引数と結合しているかどうかは、組込み関数PRESENTにより確認することができます。省略可能な仮引数が実引数と結合していない場合、組込み関数PRESENT の引数および、省略可能な仮引数に対応する実引数として以外は、引用することはできません。

仮引数並びの最後でない省略可能な仮引数を省略する場合、それ以降の実引数並びには、引数キーワード(“1.12.6.1.2 引数キーワード”参照)の指定が必要です。

省略可能な仮引数の例:

```
interface
  subroutine zee(a,b,c)
    implicit none
    real, intent(in), optional :: a,c
    real, intent(in) :: b
  end subroutine
end interface
call zee(b=3.0)             ! a およびc を省略。b に対して引数キーワードが必要です。
call zee(2.0,3.0)           ! c を省略。引数キーワードは指定しなくてもかまいません。
call zee(b=3.0,c=8.5)       ! a を省略。b およびc に対して引数キーワードが必要です。
```

1.12.6.1.4 仮データ実体

実引数の型パラメタ値は、仮引数の型パラメタが引継ぎでもなく無指定でもないなら、対応する仮引数の型パラメタ値と一致しなければなりません。ただし、形状引継ぎでない仮引数と結合する文字型の実引数の文字長パラメタは、一致しなくてもかまいません。

仮引数がポインタである場合には、実引数、その型、型パラメタ、および次元数は同じでなければなりません。

仮ポインタがINTENT(IN)をもたないなら、実引数はポインタでなければなりません。仮ポインタがINTENT(IN)をもつなら、実引数は以下のいずれかでなければなりません。

- ポインタ
- 仮ポインタに対して、ポインタ代入文として正しいデータ指示先指定(“2.2 ポインタ代入文”参照)

手続を引用すると、仮ポインタは実引数の結合状態を受け取ります。実引数が指示先に結合していれば、仮引数は、実引数と同じ指示先に結合されます。結合状態は、手続の実行の間に変更してもかまいません。手続の実行が終了した時、実引数のポインタ結合状態は、仮引数の結合状態と同じになります。

仮引数がOPTIONAL属性でなくかつポインタでなく、対応する実引数がポインタである場合、実引数は指示先と結合していなければならず、仮引数はその指示先と引数結合します。

仮引数がOPTIONAL属性でなくかつ割付け変数でなく、対応する実引数が割付け実体である場合、実引数は割り付けられていなければなりません。

仮引数がTARGET 属性もPOINTER 属性ももたない場合、手続呼出し時に実引数と結合しているポインタは、対応する仮引数と結合しません。

仮引数がTARGET 属性をもつスカラまたは形状引継ぎ配列であり、対応する実引数がTARGET属性をもち、かつベクトル添字をもつ部分配列でない場合、手続の呼出し時に実引数と結合しているポインタは、対応する仮引数と結合し、手続の実行終了時に仮引数と結合しているポインタは実引数と結合したままとなります。

仮引数がTARGET 属性をもつ形状明示配列または大きさ引継ぎ配列であり、対応する実引数がTARGET 属性をもち、かつベクトル添字をもつ部分配列でない場合、手続の呼出し時に実引数と結合しているポインタは対応する仮引数と結合し、手続の実行終了時に仮引数と結合しているポインタは実引数と結合したままとなります。

仮引数がTARGET 属性をもち、対応する実引数がTARGET 属性をもたないか、またはベクトル添字をもつ部分配列である場合には、手続の実行終了時に仮引数と結合しているすべてのポインタは不定となります。

実引数がスカラである場合には、その実引数が形状引継ぎ配列でもポインタ配列でもない配列の要素であるとき、およびそのような要素の部分列であるとき、および文字型スカラであるときを除いて、対応する仮引数はスカラでなければなりません。手続が要素別処理でなく、総称名によって引用される場合、および利用者定義演算子または利用者定義代入として引用される場合には、実引数の次元数と対応する仮引数の次元数とは、一致しなければなりません。

仮引数が形状引継ぎ配列である場合には、実引数は、大きさ引継ぎ配列でもスカラでもあってはなりません。

実引数が共添字付きのスカラの場合、仮引数はスカラでなければなりません。

非要素別処理手続のスカラ仮引数は、スカラ実引数とだけ結合することができます。

仮データ実体の特性には、型、型パラメタ、形状、INTENT 属性、OPTIONAL 属性、ALLOCATABLE 属性、VALUE 属性、ASYNCHRONOUS 属性、VOLATILE 属性、多相的、POINTER属性、CONTIGUOUS属性、およびTARGET 属性があります。実体の型パラメタまたは配列の上下限が、定数式である場合には、その式の要素への依存の仕方も特性です。形状または大きさが引き継がれた場合、または型パラメタが無指定の場合には、それも特性になります。

仮引数が割付けまたはポインタで多相的である場合、結合する実引数は多相的でなければなりません。また、以下のいずれかを満たさなければなりません。

- 実引数および仮引数は無制限多相的でなければなりません。
- 実引数の宣言時の型は仮引数の宣言時の型と同じでなければなりません。

仮引数が割付けまたはポインタの場合、結合する実引数は、その仮引数と同じ型パラメタが無指定でなければなりません。

仮引数がポインタの場合、実引数はポインタでなければならず、無指定でない型パラメタおよび次元数は同じでなければなりません。

仮引数がCONTIGUOUS属性をもつポインタ配列の場合、実引数はCONTIGUOUS属性のポインタでなければなりません。

仮引数が割付けの場合、実引数は割付けでなければならず、無指定でない型パラメタおよび次元数は同じでなければなりません。実引数の割付け状態は、“割り付けられていない”という状態であってもかまいません。

仮引数が共配列の場合、実引数はすべての像で同じ共配列でなければなりません。仮引数が割付け共配列の場合、実引数の共次元数は、仮引数と同じでなければなりません。

実引数が部分配列または形状引継ぎ配列であり、対応する仮引数がVOLATILE 属性またはASYNCHRONOUS 属性のいずれかをもつ場合、その仮引数は、形状引継ぎ配列でなければなりません。

以下の条件を満たすとき、仮引数はCONTIGUOUS属性をもたない形状引継ぎ配列でなければなりません。

- 実引数が単純CONTIGUOUS（“2.88 CONTIGUOUS文”参照）でなく、ポインタでなく、VOLATILE属性またはASYNCHRONOUS属性のいずれかをもっている。かつ、
- 対応する仮引数がVOLATILE属性またはASYNCHRONOUS属性のいずれかをもっている。

実引数がポインタ配列であり、対応する仮引数がVOLATILE 属性またはASYNCHRONOUS 属性のいずれかをもつ場合、その仮引数は、形状引継ぎ配列またはポインタ配列でなければなりません。

以下の条件を満たすとき、仮引数はCONTIGUOUS属性をもたないポインタ配列または形状引継ぎ配列でなければなりません。

- 実引数がVOLATILE属性またはASYNCHRONOUS属性のいずれかをもち、CONTIGUOUS属性をもたないポインタ配列である。かつ、
- 対応する仮引数がVOLATILE属性またはASYNCHRONOUS属性のいずれかをもっている。

共添字付き実体の実引数は、ポインタ末端成分または割付け末端成分をもってはなりません。実引数が共添字付き実体なら、仮引数は、INTENT(IN)属性をもたなければなりません。

仮引数が、CONTIGUOUS属性をもつ配列共配列、または形状引継ぎの配列共配列でない場合、対応する実引数は単純CONTIGUOUSでなければなりません。

1.12.6.1.5 仮手続

仮手続は、EXTERNAL 文または引用仕様本体で手続として指定した仮引数および手続として引用した仮引数です。

仮手続と結合する実引数は、外部手続、モジュール手続、仮手続、または組込み手続の個別名でなければなりません。

仮引数が手続ポインタである場合には、結合される実引数は、以下のいずれかでなければなりません。

- 手続ポインタ
- 手続ポインタを返す関数引用
- 組込み関数NULL の引用
- 仮ポインタに対して、ポインタ代入文として正しい手続指示先指定（“2.2 ポインタ代入文”参照）

1.12.6.1.6 選択戻り指定子（廃止予定事項）

仮引数が星印‘*’の場合、対応する実引数は選択戻り指定子‘ *label* ’でなければなりません。
label は、そのCALL 文と同じ有効域中内にある飛び先文の文番号でなければなりません。選択戻り指定子は、CALL 文から呼び出したサブルーチン実行後に制御の移行する文を指定します。選択戻り指定子は、関数引用の実引数には指定できません。

1.12.7 手続引用仕様

手続の引用仕様は、その手続を呼び出すときの引用の形を定義します。引用仕様は、手続の特性、手続の名前、それぞれの仮引数の名前と特性、および、もしあれば手続の総称識別子からなります。手続の特性は、関数であるかサブルーチンであるか、純粋手続かどうか、要素別であるかどうか、手続の引数の特性、および関数である場合には結果の値の特性からなります。引数の特性は、仮データ実体、仮手続、または星印'*'の仮引数の種類、仮データ実体の場合には、型、型パラメタ、形状、INTENT 属性、OPTIONAL 属性があるかどうか、割付けであるかどうか、ASYNCHRONOUS属性、CONTIGUOUS属性、VALUE 属性、VOLATILE 属性があるかどうか、ポインタであるかどうか、多相的であるかどうか、および指示先であるかどうかです。仮手続の特性は、その引用仕様が明示的であるかどうか、引用仕様が明示的であればその手続としての特性、およびOPTIONAL 属性があるかどうかです。関数結果の特性は、型、型パラメタ、次元数、およびポインタであるかまたは割付けであるかどうかです。

手続が有効域内で参照可能な場合、手続の引用仕様は、その有効域において明示的または暗黙的のどちらかです。内部手続、モジュール手続、および組込み手続の引用仕様は、常に明示的です。再帰サブルーチンおよび結果名をもつ再帰関数の引用仕様は、それを定義している副プログラムの中では明示的です。外部手続および仮手続の引用仕様は、その手続に対する引用仕様宣言が、与えられているかまたは参照可能であれば、その手続の引用仕様は明示的です。それ以外の場合は、手続の引用仕様は暗黙的です。

1.12.7.1 明示的引用仕様

以下の場合、手続は明示的引用仕様をもたなければなりません。

手続の引用を書いたとき、

- 引数キーワードがある場合
- 総称名によって引用している場合
- 利用者定義代入として現れている場合
- 利用者定義演算として式中に現れている場合
- 純粋手続であることが要求される文脈に現れている場合

手続が、

- 要素別処理手続である場合
- BIND 属性をもつ手続である場合
- 省略可能な仮引数をもっている場合
- 形状引継ぎ配列、割付け、ポインタ、または指示先である仮引数をもっている場合
- 配列値の結果を返す場合
- ポインタまたは割付けである結果を返す場合
- 引継ぎでも定数式でもない文字長パラメタ、または型パラメタをもつ結果を返す場合
- VOLATILE 属性、VALUE 属性、またはASYNCHRONOUS 属性をもつ仮引数をもっている場合
- パラメタ付き派生型の仮引数をもっている場合
- 多相的な仮引数をもっている場合
- 共配列の仮引数をもっている場合
- 次元引継ぎ仮引数をもっている場合

1.12.7.2 手続引用仕様宣言

手続引用仕様は以下の形式です。

```
INTERFACE [ generic-spec ]
    [ interface-body ]...
    [ [MODULE] PROCEDURE [::] procedure-name-list ]...
END INTERFACE [ generic-spec ]
```

または

```
ABSTRACT INTERFACE
    [ interface-body ]...
END INTERFACE
```

generic-spec は、総称指定であり、以下の形式です。

```
generic-name                        または
OPERATOR ( defined-operator )       または
ASSIGNMENT ( = )                    または
dtio-generic-spec
```

generic-name は総称名です。

defined-operator は利用者定義演算子であり、以下の形式です。

```
intrinsic-operator                  または
.operator-name.
```

intrinsic-operator は組込み演算子です。

operator-name は、利用者が定義した240文字までの利用者定義演算子の名前です。

ASSIGNMENT(=) は、利用者定義代入です。

dtio-generic-spec は派生型出力手続であり、以下の形式です。

```
READ ( FORMATTED )                  または
READ ( UNFORMATTED )                または
WRITE ( FORMATTED )                 または
WRITE ( UNFORMATTED )
```

interface-body は引用仕様本体です。

procedure-name-list は、コンマで区切られた手続名の並びです。
手続名は、明示的な引用仕様をもたなければなりません。また、その手続名は、手続ポインタ、外部手続、仮手続、モジュール手続、または内部手続でなければなりません。

PROCEDURE 文にMODULE を指定する場合、その文に指定した手続名は、モジュール手続でなければなりません。

END INTERFACE 文に*generic-spec*を指定する場合、対応するINTERFACE 文に指定された*generic-spec*と同じでなければなりません。
ABSTRACT INTERFACE には*generic-spec* を指定できません。

interface-body は、以下の形式です。

```
[ prefix-spec ]... FUNCTION function-name ( [ dummy-arg-name-list ] ) [ suffix ]
    [ use-stmts ]
    [ import-stmts ]
    [ specification-part ]
END [ FUNCTION [ function-name ] ]
```

または

```
[ prefix-spec ]... SUBROUTINE subroutine-name [ ( [ dummy-arg-list ] ) proc-language-binding-spec ]
    [ use-stmts ]
    [ import-stmts ]
    [ specification-part ]
END [ SUBROUTINE [ subroutine-name ] ]
```

prefix-spec は、以下の形式です。

```
type-spec                           または
ELEMENTAL                           または
```

```
IMPURE                          または
MODULE                          または
PURE                            または
RECURSIVE
```

*prefix-spec*にMODULEを指定した引用仕様は、分離引用仕様です(“1.11.3 サブモジュール”参照)。

type-spec は宣言型指定子です。SUBROUTINE 文の*prefix-spec* には、*type-spec* は指定できません。
宣言型指定子については、“2.3 型宣言文”を参照してください。

function-name は、関数名です。

suffix は、次のいずれかです。

```
手続言語束縛指定子 [ RESULT ( result-name ) ]        または
RESULT ( result-name ) [ 手続言語束縛指定子]
```

result-name は結果名であり、関数の結果変数を指定します。RESULT 句を指定しない場合、結果変数は関数名です。

手続言語束縛指定子は、以下の形式です。

```
BIND ( C [, NAME = スカラ文字定数式] )
```

subroutine-name は、サブルーチン名です。

dummy-arg-name-list は、コンマで区切られた仮引数名の並びです。

dummy-arg-list は、コンマで区切られた仮引数の並びです。

dummy-arg は、以下の形式です。

```
dummy-arg-name                  または
*
```

proc-language-binding-spec は手続言語束縛指定子です。

use-stmts は1つ以上のUSE 文です。

import-stmts は1つ以上のIMPORT 文です。

*specification-part*は、1つ以上の宣言文、派生型定義、または引用仕様宣言です。*specification-part*には、ENTRY 文、DATA 文、および FORMAT 文を含んではなりません。

END FUNCTION 文に*function-name*を指定する場合、対応するFUNCTION 文に指定された*function-name*と同じでなければなりません。

END SUBROUTINE 文に*subroutine-name*を指定する場合、対応するSUBROUTINE 文に指定された*subroutine-name*と同じでなければなりません。

引用仕様宣言中の引用仕様本体は、外部手続および仮手続に対して、明示的な引用仕様を指定します。引用仕様宣言内のFUNCTION 文またはSUBROUTINE 文に書いた名前が、この引用仕様宣言を含む副プログラムの仮引数の名前と同じである場合には、その仮引数が、指定した引用仕様をもつ仮手続であることを宣言します。それ以外の場合には、その名前が、指定した手続引用仕様をもつ外部手続の名前であることを宣言します。

引用仕様本体は、手続の特性のすべてを指定します。これらは、手続の定義と一致していなければなりません。ただし、手続が純粋として定義されている場合に、引用仕様でその手続を純粋でないと宣言してもかまいません。引用仕様宣言は、ENTRY 文を含んではなりませんが、入口名を引用仕様宣言内で手続の名前として使用することによって、入口の引用仕様を指定してもかまいません。1つの有効域内では、1つの手続が2つ以上の明示的引用仕様をもっていてはなりません。

INTERFACE 文がABSTRACT INTERFACE である場合、FUNCTION文の関数名またはSUBROUTINE 文のサブルーチン名は、組込み型を指定するキーワードと同じであってはなりません。

ABSTRACT INTERFACE による引用仕様宣言は抽象引用仕様宣言です。抽象引用仕様宣言内の引用仕様本体は、抽象引用仕様を宣言します。
総称指定をもつ引用仕様宣言は、総称引用仕様宣言です。
ABSTRACT および総称指定のいずれの指定もない引用仕様宣言は、個別引用仕様宣言です。

引用仕様宣言の例:

```
interface
  subroutine x (a,b,c)
    implicit none
    real, intent(in), dimension(2,8) :: a
    real, intent(out), dimension(2,8) :: b,c
  end subroutine x
  function y (a,b)
    implicit none
    integer :: y
    logical, intent(in) :: a,b
  end function y
end interface
```

この例では、手続 x と y の明示引用仕様を宣言しています。

1.12.7.3 総称引用仕様

INTERFACE 文(“2.291 INTERFACE文”参照)および手続束縛文(“1.5.11.3 型束縛手続”参照)には、総称指定を指定することができます。

総称指定をもった引用仕様宣言は、引用仕様宣言内のそれぞれの手続に対して、総称引用仕様を指定します。INTERFACE 文に総称指定がある場合、PROCEDURE 文を含むことができます。PROCEDURE 文中の手続名は、明示的な引用仕様をもたなければなりません。また、参照可能な手続ポインタ、外部手続、仮手続またはモジュール手続でなければなりません。

総称手続の引用と一致する個別手続は、1つでなければなりません。総称手続が引用されたとき、その引用と一致する個別手続を引用します。

1.12.7.3.1 総称名

INTERFACE 文(“2.291 INTERFACE文”参照)に指定された総称名は、その引用仕様宣言中のすべての手続を引用できる1つの名前を指定します。総称名は、引用仕様中の手続の名前と同じであってもかまいませんし、参照可能な他の総称名と同じであってもかまいません。

1つの有効域内において、同じ総称名をもつ2つの手続は、双方ともサブルーチンまたは双方とも関数でなければならず、次のいずれかでなければなりません。

- 双方の仮引数の個数が異なる。
- ある仮引数が他方とは異なる型、異なる種別型パラメタ、または異なる次元数をもつ。
- 双方が当該実体仮引数をもち、かつその当該実体仮引数が区別可能である。
- ある仮引数が手続に対し、他方はデータ実体である。
- ある仮引数がALLOCATABLE属性をもつのに対し、他方はPOINTER属性をもちINTENT(IN)属性をもたない。
- ある仮引数がゼロ次元でない関数に対し、他方は関数になっていない。

総称名が総称組込み手続の名前と同じ場合、引用仕様宣言中の手続および組込み手続がすべて関数またはすべてサブルーチンでなければ、総称組込み手続を参照することはできません。総称呼出しが、引用仕様宣言中の個別手続および参照可能な総称組込み手続の個別手続の両方に当てはまる場合には、引用仕様宣言中の個別手続が引用されます。

総称名の例:

```
interface swap                          ! 総称名swap
  subroutine real_swap(x,y)
    implicit none
    real, intent(inout) :: x,y
  end subroutine real_swap
  subroutine int_swap(x,y)
    implicit none
    integer, intent(inout) :: x,y
  end subroutine int_swap
end interface
```

この例で、総称名swap は、実数型および整数型の引数の両方に使用できます。

1.12.7.3.2 利用者定義演算

INTERFACE 文の総称指定にOPERATOR を指定することにより、組込み演算子を拡張したり、新たな利用者定義演算子を定義することができます。その引用仕様宣言に含まれる個別手続は、すべて関数でなければなりません。2つの引数をもつ関数の場合には、2項演算の規則が適用されます。1つの引数をもつ関数の場合には、単項演算の規則が適用されます。仮引数は、INTENT(IN) 属性をもつ省略可能でない仮データ実体でなければなりません。利用者定義演算は、指定された関数の引用とみなされます。利用者定義単項演算に対しては、演算対象は、関数の仮引数に対応します。利用者定義2項演算に対しては、左側の演算対象は、関数の1番目の仮引数に対応し、右側の演算対象は、2番目の仮引数に対応します。

利用者定義演算は以下の形式です。

defined-unary-operator operand または
operand defined-binary-operator operand

defined-unary-operator は、単項演算子です。

defined-binary-operator は、2項演算子です。

operand は演算対象です。

defined-unary-operator および*defined-binary-operator* は、組込み演算子または利用者定義演算子のいずれかです。

利用者定義演算子は以下の形式です。

. *operator-name*.

operator-name は、240文字までの英字です。

次のいずれかでなければなりません。

- OPERATOR(.*operator-name*.) の総称指定をもつ関数を指定しなければなりません。
- 演算対象の宣言時の型にOPERATOR(.*operator-name*.) の総称指定をもつ総称束縛があり、かつその実行時の型の中に、関数に対応する束縛がなければなりません。また、関数の仮引数が配列である場合、実引数と形状適合しなければなりません。

関数の仮引数の型は、実引数の実行時の型と適合しなければなりません。

利用者定義演算の評価順序は、演算子が組込み演算子であれば、その評価順序に従います。利用者定義単項演算子の評価順序は、他のすべての演算子よりも優先されます。利用者定義2項演算子の評価順序は、他のすべての演算子よりも後になります。

総称指定が組込み演算を拡張する場合、引用仕様宣言内の手続は、その組込み演算として許される型、種別型パラメタ、および次元数の組合せと同じであってはなりません。1つの有効域内において、同じ総称演算子および同じ個数の引数をもつ2つの手続は、仮引数並び中のいずれかの対応する位置に、異なる型、異なる種別型パラメタ、または異なる次元数をもつ仮引数をもたなければなりません。

利用者定義演算(利用者定義2項演算)の例:

```
use type_define_mod           ! 派生型set の定義を参照結合
type(set) :: a,b,c
  interface operator (.intersection.)
    function set_intersection (a,b)
      use type_define_mod      ! 派生型set の定義を参照結合
      implicit none
      type (set) , intent(in) :: a,b
      type (set) :: set_intersection
    end function set_intersection
  end interface operator(.intersection.)
a = b.intersection.c
```

この例で、b.instruction.c は、関数引用set_intersection(b,c) になります。

利用者定義演算(組込み演算子の拡張)の例:

```
use type_define_mod           ! 派生型set の定義を参照結合
type(set) :: a,b,c
  interface operator (*)
    function set_intersection (a,b)
      use type_define_mod    ! 派生型set の定義を参照結合
```

```
      implicit none
      type (set) , intent(in) :: a,b
      type (set) :: set_intersection
    end function set_intersection
  end interface operator(*)
a = b*c
```

この例で、b*c は、関数引用set_intersection(b,c) になります。

1.12.7.3.3 利用者定義代入

INTERFACE 文の総称指定にASSIGNMENT を指定することにより、利用者定義代入を宣言します。その引用仕様宣言に含まれる個別手続は、すべてサブルーチンでなければなりません。これらのサブルーチンはちょうど2つの引数をもたなければなりません。1番目の仮引数は、INTENT(OUT) またはINTENT(INOUT) 属性をもつ省略可能でない仮データ実体でなければなりません。2番目の仮引数は、INTENT(IN) 属性をもつ省略可能でない仮データ実体でなければなりません。利用者定義代入は、1番目の引数として左辺をもち、2番目の引数として右辺を括弧でくくったものをもつようなサブルーチンの引用とみなされます。

総称指定が利用者定義代入を宣言する場合、引用仕様宣言内の手続は、組込み代入として許される型、種別型パラメタ、および次元数の組合せと同じであってはなりません。1つの有効域内において、代入を定義する2つの手続は、仮引数並び中のいずれかの対応する位置に、異なる型、異なる種別型パラメタ、または異なる次元数をもつ仮引数をもたなければなりません。

利用者定義代入を定義するサブルーチンは、次のいずれかでなければなりません。

- 総称引用仕様がASSIGNMENT(=) の総称指定をもつサブルーチンを指定する。
- ASSIGNMENT(=) の総称指定によって、代入対象の宣言時の型に総称束縛があり、かつその実行時の型に対応するサブルーチンの束縛を指定する。

サブルーチンの仮引数の型は、代入対象の実行時の型と適合しなければなりません。

利用者定義代入の例:

```
interface assignment(=)
  subroutine integer_to_logical_array(b,n)
    implicit none
    logical, intent(out) :: b(:)
    integer, intent(in) :: n
  end subroutine integer_to_logical_array
end interface
```

この例では、整数型データから論理型の配列への代入を拡張します。

1.12.7.4 型束縛手続引用の解決

手続特定子の束縛名が個別型束縛手続の束縛名である場合、引用される手続は、データ参照の実行時の型において選択された個別束縛と同じ名前に束縛されている手続です。

手続特定子の束縛名が総称型束縛手続の束縛名である場合、データ参照の宣言時の型においてその名前をもつ総称束縛から、次のように個別束縛が選択されます。

- 引用が総称束縛の1つの個別束縛と一致する場合、その個別束縛が選択されます。
- そうでない場合、その引用は、総称束縛の1つの個別束縛の要素別処理引用と一致する個別束縛が選択されます。

1.12.8 サービスルーチン

サービスルーチンは、外部手続として提供される関数群およびサブルーチン群です。また、サービスルーチンの引用仕様を定義したモジュール‘SERVICE_ROUTINES’も合わせて提供されます。

サービス関数は、関数引用の形式で引用することができます。サービス関数は純粋関数ではありません。

サービスサブルーチンは、CALL 文にサービスサブルーチン名を指定することにより引用することができます。サービスサブルーチンは純粋サブルーチンではありません。

サービスルーチンは、手続引用として指定するだけでも使用できますが、各サービスルーチンの引用仕様を定義したモジュール‘SERVICE_ROUTINES’を引用することにより、引数の数、引数の型などの正当性の検査が実施されます。モジュール

‘SERVICE_ROUTINES’には、すべてのサービスルーチンの引用仕様宣言が含まれます。引用する場合はUSE 文にONLY 句を指定し、実際に引用するサービスルーチンの引用仕様だけを参照結合することをお勧めします。モジュール‘SERVICE_ROUTINES’を引用しない場合、サービスルーチン引用の正当性の検査は実施されません。

各サービスルーチンの詳細については、“第2章 文および手続の詳細”を参照してください。また、サービスルーチンの一覧表が、“付録B サービスルーチン一覧”に用意されていますので併せて参照してください。

1.13 組込みモジュール

組込みモジュールは、処理系に備っているモジュールです。組込みモジュールには、標準組込みモジュールと非標準組込みモジュールがあります。

1.13.1 標準組込みモジュール

標準組込みモジュールには、C プログラムとの相互利用のためのモジュール(“2.303 ISO_C_BINDING組込みモジュール”参照)、例外処理またはIEEE 算術を支援するモジュール(“1.15 IEEE 例外およびIEEE 算術”参照)、およびFortran 環境モジュール(“2.305 ISO_FORTRAN_ENV組込みモジュール”参照)があります。
標準組込みモジュールの中で定義されている関数を組込みモジュール関数、サブルーチンを組込みモジュールサブルーチンといいます。

1.13.2 非標準組込みモジュール

非標準組込みモジュールには、サービスルーチンの手続引用仕様を定義したモジュールSERVICE_ROUTINES(“2.433 SERVICE_ROUTINES非標準組込みモジュール”参照)とOpenMP のモジュールOMP_LIB(“2.371 OMP_LIB非標準組込みモジュール”参照)があります。

1.14 有効範囲

プログラム単位、共通ブロック、外部手続は、大域要素です。大域名の有効範囲は、1つのプログラムです。大域名は同じプログラムの中で、他の大域名の識別に使用してはなりません。

次の名前は、その構文範囲を有効範囲とする構文内要素です。構文内要素の有効範囲内では、他の構文内要素はそれと同じ名前をもってはなりません。

- FORALL 構文およびFORALL 文の指標変数名
- DO CONCURRENT(“2.120 DO構文”参照)の指標変数名
- SELECT TYPE 構文の結合名
- ASSOCIATE 構文の結合名
- BLOCK 構文の宣言部でASYNCHRONOUS 文およびVOLATILE 文を除き、明示的に宣言された実体

FORALL 構文またはDO CONCURRENT(“2.120 DO構文”参照)の指標変数名は、その構文範囲を有効範囲とする構文内要素です。構文内要素の有効範囲内では、他の構文内要素はそれと同じ名前をもってはなりません。FORALL 文の指標変数名は、その文を有効範囲とする文内要素です。DATA 文または配列構成子内のDO 変数として指定した変数名は、そのDO 形反復を有効範囲とする文内要素です。文関数定義文の仮引数は、その文を有効範囲とする文内要素です。文内要素の有効範囲内では、他の文内要素はそれと同じ名前をもってはなりません。

他の要素は、1つの有効域を有効範囲とする、局所要素です。有効域は、派生型定義、手続引用仕様本体(その中に含まれる派生型定義および手続引用仕様本体を除く)、プログラム単位または副プログラム(その中に含まれる派生型定義、手続引用仕様本体、および副プログラムを除く)のいずれかです。

その束縛ラベルは、英大文字と英小文字との区別を無視したうえで、プログラムの他のどの言語要素の束縛ラベルとも同じであってはなりません。

型パラメタ名は、それを含む派生型定義と同じ有効範囲をもちます。型パラメタ名は、その派生型の定義の外では、その型に対する派生型指定子の中の型パラメタキーワードとしてまたは型パラメタ問合せの型パラメタ名としてだけ書くことができます。

型束縛手続の束縛名は、それを含む派生型定義と同じ有効範囲をもちます。束縛名は、その派生型の定義の外では、手続参照の中の束縛名としてだけ書くことができます。

成分名または束縛名は、参照可能な有効範囲でだけ書くことができ、総称指定が総称名ではない場合の総称束縛は、その型の要素と同じ有効範囲をもちます。

ASSOCIATE 構文およびSELECT TYPE 構文の結合名は、それぞれのブロックで別々の有効範囲をもちます。それぞれのブロックにおいて、結合名は、宣言時の型、実行時の型、次元数、および上下限をもちます。

SELECT TYPE 構文またはASSOCIATE 構文の有効域において参照可能な大域識別子または局所識別子がその結合名と同じ名前である場合、その名前は、SELECT TYPE 構文またはASSOCIATE構文のブロックでは、結合名として解釈されます。同じ有効域の他の場所では、その名前は大域識別子または局所識別子として解釈されます。

1.14.1 結合

ある実体をプログラム単位または副プログラムをまたがって参照する場合には、引数による結合(“1.12.6 手続引用”参照)、共通ブロック要素による結合(“2.82 COMMON文”参照)、親子結合、または参照結合(“1.11.2.2 モジュール引用”参照)による方法があります。

1.14.1.1 親子結合

親子結合とは、ある有効域に含まれる派生型定義および副プログラムの有効域において、その親有効域の局所要素を参照すること、引用仕様本体において、その引用仕様本体のIMPORT 文によって親有効域の名前付き要素を参照すること、および、サブモジュールの有効域において、その祖先モジュールの要素を参照することをいいます。この時、同じ名前の宣言がその有効域にあってはなりません。分離引用仕様本体は、IMPORT文を指定しなくても親有効域の要素を親子結合で引用することができます。

親有効域の要素がVOLATILE 属性またはASYNCHRONOUS 属性をもっていなくても、参照される要素はそれらをもつことができます。

親子結合の例:

```
subroutine external( )
  implicit none
  integer :: a,b
  ...
  contains
    subroutine internal( )
      implicit none
      integer :: a
      ...
      a = b                ! a はinternal の局所要素です
                           ! b は親子結合により、external の局所要素を参照します
      ...
    end subroutine internal
end subroutine external
```

この例の代入文a = b において、a はinternal 内で宣言された局所要素であり、external 内で宣言されているa とは異なる実体です。b は、external で宣言された実体を親子結合により参照します。

1.14.1.2 構文結合

SELECT TYPE 文(“2.431 SELECT TYPE構文”参照)の実行は、その構文の選択子と結合名との結合を確定します。ASSOCIATE 文の実行は、その構文のそれぞれの選択子とそれに対応する結合名との結合を確定します。

選択子が割付け要素ならば、割付け済でなければなりません。結合名はデータ実体と結合し、ALLOCATABLE 属性をもちません。

選択子がPOINTER 属性をもつならば、結合されていなければなりません。結合名はポインタの指示先と結合し、POINTER 属性をもちません。

選択子がベクトル添字をもつ部分配列以外の変数である場合、選択子で指定されたデータ実体と結合します。それ以外の場合、選択された式の値と結合します。その式は、ブロックの実行に先立って評価されます。それぞれの結合名は、ブロックの実行中、対応する選択子と結合します。
ブロックの中では、それぞれの選択子が、対応する結合名によって識別され、参照可能となります。

1.15 IEEE 例外およびIEEE 算術

組込みモジュールIEEE_EXCEPTIONS、IEEE_ARITHMETIC、およびIEEE_FEATURES は、IEEE 例外およびIEEE 算術を提供します。

モジュールIEEE_ARITHMETIC は、IEEE_EXCEPTIONS の参照結合を包含し、IEEE_EXCEPTIONS で公開のものはすべてIEEE_ARITHMETIC で公開されます。

これらのモジュールで参照結合する型および手続は、組込み型および組込み手続ではありません。

形式

```
USE,INTRINSIC :: IEEE_ARITHMETIC
USE,INTRINSIC :: IEEE_FEATURES
```

1.15.1 IEEE の派生型および定数

モジュールIEEE_EXCEPTIONS、IEEE_ARITHMETIC、およびIEEE_FEATURES では、5種類の成分非公開の派生型を定義します。

1.15.1.1 IEEE_EXCEPTIONS モジュール

IEEE_EXCEPTIONS モジュールでは、次のものを定義します。

IEEE_FLAG_TYPE

例外を表すフラグを識別します。

- IEEE_INVALID
- IEEE_OVERFLOW
- IEEE_DIVIDE_BY_ZERO
- IEEE_UNDERFLOW
- IEEE_INEXACT

このモジュールでは、更に名前付き配列定数として次のものが定義されます。

- IEEE_USUAL = (/ IEEE_OVERFLOW , IEEE_DIVIDE_BY_ZERO , IEEE_INVALID /)
- IEEE_ALL = (/ IEEE_USUAL , IEEE_UNDERFLOW , IEEE_INEXACT/)

IEEE_STATUS_TYPE

現在の浮動小数点数状態を退避します。

1.15.1.2 IEEE_ARITHMETIC モジュール

IEEE_ARITHMETIC モジュールでは、次のものを定義します。

IEEE_CLASS_TYPE

浮動小数点数の値の種類を識別します。

- IEEE_SIGNALING_NAN
- IEEE_QUIET_NAN
- IEEE_NEGATIVE_INF
- IEEE_NEGATIVE_NORMAL
- IEEE_NEGATIVE_DENORMAL
- IEEE_NEGATIVE_ZERO
- IEEE_POSITIVE_ZERO
- IEEE_POSITIVE_DENORMAL
- IEEE_POSITIVE_NORMAL
- IEEE_POSITIVE_INF
- IEEE_OTHER_VALUE

IEEE_ROUND_TYPE

丸めモードを識別します。

- IEEE_NEAREST
- IEEE_TO_ZERO
- IEEE_UP
- IEEE_DOWN
- IEEE_OTHER

要素別処理演算子 ==

2つの値が同じときに真、そうでないときに偽を返します。

要素別処理演算子 /=

2つの値が異なるときに真、そうでないときに偽を返します。

1.15.1.3 IEEE_FEATURES モジュール

IEEE_FEATURES モジュールでは、次のものを定義します。

IEEE_FEATURES_TYPE

IEEE の機能で必要とされるものを表します。

- IEEE_DATATYPE
- IEEE_DENORMAL
- IEEE_DIVIDE
- IEEE_HALTING
- IEEE_INEXACT_FLAG
- IEEE_INF
- IEEE_INVALID_FLAG
- IEEE_NAN
- IEEE_ROUNDING
- IEEE_SQRT
- IEEE_UNDERFLOW_FLAG

1.15.2 IEEE の例外

IEEE の例外は、次のとおりです。

- IEEE_OVERFLOW

  実数の組込み演算または組込み代入の結果が制限より大きな絶対値をもつ場合、または複素数の組込み演算または組込み代入の実部または虚部が制限より大きな絶対値をもつ場合。

- IEEE_DIVIDE_BY_ZERO

  実数または複素数の被除数がゼロ以外であり、かつ除数がゼロである場合。

- IEEE_INVALID

  実数または複素数の演算または代入が無効である場合。

- IEEE_UNDERFLOW

  実数の組込み演算もしくは組込み代入の結果が、制限より小さな絶対値をもち精度が失われることが検出された場合、または複素数の組込み演算または組込み代入の実部または虚部が制限より小さな絶対値をもち精度が失われることが検出された場合。

- IEEE_INEXACT

  実数型または複素数型の演算または代入が厳密でない場合。

それぞれの例外は対応するフラグをもち、それらのフラグは、値として例外通知または例外非通知を取ります。この値は、関数IEEE_GET_FLAG を呼び出して得ることができます。いずれの場合も初期値は例外非通知とし、対応する例外が発生したとき例外通知になります。フラグの状態は、サブルーチンIEEE_SET_FLAG またはIEEE_SET_STATUS で変更できます。手続中でいったん状態が例外通知になったとき、サブルーチンIEEE_SET_FLAG またはIEEE_SET_STATUS で例外非通知に設定されるまで状態は例外通知のままです。

宣言式の評価によって、例外が発生する可能性があります。

1.15.3 IEEE 丸めモード

丸めモードは、次の4つです。

- IEEE_NEAREST

  厳密な値を、表現できる最も近い値に丸めます（直近丸め）。

- IEEE_TO_ZERO

  厳密な値を、ゼロに向かう方向で表現できる最も近い値に丸めます（切捨て）。

- IEEE_UP

  厳密な値を、+ ∞に向かう方向で表現できる最も近い値に丸めます（切上げ）。

- IEEE_DOWN

  厳密な値を、- ∞に向かう方向で表現できる最も近い値に丸めます（切下げ）。

現在どのモードであるかは、関数IEEE_GET_ROUNDING_MODE を用いて問い合わせることができます。結果の値は、上の4つのうちのいずれかです。丸めモードが合致していないときはIEEE_OTHER です。

丸めモードの変更はサブルーチンIEEE_SET_ROUNDING_MODE を用います。

1つのプログラム内では、同じ形式をもつすべての定数は同じ値をもちます。したがって、定数値は、丸めモードによって影響されません。

1.15.4 IEEE 下位けたあふれモード

下位けたあふれモードの制御は、関数IEEE_SET_UNDERFLOW_MODE の呼出しによって行えます。どちらの下位けたあふれモードが有効になっているかは、関数IEEE_GET_UNDERFLOW_MODE を用いて問い合わせることができます。この機能が用意されているかどうかは、関数IEEE_SUPPORT_UNDERFLOW_MODE を用いて問い合わせることができます。

下位けたあふれモードは、関数IEEE_SUPPORT_UNDERFLOW_CONTROL が真を返す型の浮動小数点数演算にだけ影響します。

1.15.5 IEEE 停止モード

例外が発生した後のプログラムの実行を中断するか続行するかを実行時に制御できます。この制御は、サブルーチンIEEE_SET_HALTING_MODE の呼出しによって行えます。復帰の時点では停止モードが入った時点と同じであることを保証します。

1.15.6 IEEE 浮動小数点数状態

例外、丸めモード、下位けたあふれモードおよび停止のそれぞれに対応して用意されているフラグ全体の値を、浮動小数点数状態と呼びます。浮動小数点数状態は、派生型IEEE_STATUS_TYPE のスカラ変数に対し、サブルーチンIEEE_GET_STATUS を用いて退避し、サブルーチンIEEE_SET_STATUS を用いて設定することができます。その変数のうち、ある特定のフラグの値を見つける機能はありません。浮動小数点数状態の一部は、サブルーチンIEEE_GET_FLAG 、IEEE_GET_HALTING_MODE 、およびIEEE_GET_ROUNDING_MODE を用いて退避し、サブルーチンIEEE_SET_FLAG、IEEE_SET_HALTING_MODE、およびIEEE_SET_ROUNDING_MODE を用いて回復することができます。

1.15.7 IEEE 例外値

例外的な浮動小数点数の値は以下のとおりです。

- 準正規数は、絶対値がごく小さな数であって、精度は低いです。
- 無限大(+∞ および -∞)は、けたあふれまたはゼロでの除算によって発生します。
- 非数(NaN) は、未定義の値であるか、または無効な演算の結果生じた値です。

これらの例外値でない値は、正規の値といいます。

関数IEEE_IS_FINITE、IEEE_IS_NAN、IEEE_IS_NEGATIVE、およびIEEE_IS_NORMAL は、それぞれ値が有限か、非数か、負か、正規かを確かめるときに使用します。関数IEEE_VALUE は、それぞれの種類の数を生成します。これには、無限大および非数を含んでいます。
また、関数IEEE_SUPPORT_DENORMAL、IEEE_SUPPORT_INF、およびIEEE_SUPPORT_NAN を用いて、実数型についてそれぞれの機能が利用可能かどうかを問い合わせることができます。

1.15.8 IEEE 算術

- IEEE_SUPPORT_DATATYPE

  関数IEEE_SUPPORT_DATATYPE を用いて、各実数の種別についてIEEE 算術が用意されているかどうかを問い合わせることができます。

- IEEE_SUPPORT_DIVIDE

  問合せ関数IEEE_SUPPORT_DIVIDE は、除算精度を問い合わせることができます。

- IEEE_SUPPORT_NAN

  問合せ関数IEEE_SUPPORT_NAN は、IEEE 非数を用意しているかどうかを問い合わせることができます。

- IEEE_SUPPORT_INF

  問合せ関数IEEE_SUPPORT_INF は、IEEE 無限大を用意しているかどうかを問い合わせることができます。

- IEEE_SUPPORT_DENORMAL

  問合せ関数IEEE_SUPPORT_DENORMAL は、IEEE 準正規数を用意しているかどうかを問い合わせることができます。

- IEEE_SUPPORT_SQRT

  関数IEEE_SUPPORT_SQRT を用いて、特定の種別の実数に対してSQRT が規格に基づいて実装されているかどうかを問い合わせることができます。

- IEEE_SUPPORT_STANDARD

  問合せ関数IEEE_SUPPORT_STANDARD は、実数の種別それぞれに対して、IEEE の機能をすべて用意しているかどうかを問い合わせることができます。

1.15.9 IEEE 手続の概要

各モジュール内で定義されているすべての手続について、示されている仮引数名は、実引数のキーワード指定をするとき、引数キーワードとして使える名前です。

1.15.9.1 問合せ関数

IEEE_EXCEPTIONS

IEEE_EXCEPTIONS モジュールは、次の問合せ関数をもっています。

- IEEE_SUPPORT_FLAG(“2.264 IEEE_SUPPORT_FLAG組込みモジュール関数”参照)

  例外を用意しているか？

- IEEE_SUPPORT_HALTING(“2.265 IEEE_SUPPORT_HALTING組込みモジュール関数”参照)

  IEEE の停止制御を用意しているか？

IEEE_ARITHMETIC

IEEE_ARITHMETIC モジュールは、次の問合せ関数をもっています。

- IEEE_SUPPORT_DATATYPE（“2.261 IEEE_SUPPORT_DATATYPE組込みモジュール関数”参照）

  IEEE 算術を用意しているか？

- IEEE_SUPPORT_DENORMAL（“2.262 IEEE_SUPPORT_DENORMAL組込みモジュール関数”参照）

  IEEE 準正規数を用意しているか？

- IEEE_SUPPORT_DIVIDE（“2.263 IEEE_SUPPORT_DIVIDE組込みモジュール関数”参照）

  IEEE の除算を用意しているか？

- IEEE_SUPPORT_INF（“2.266 IEEE_SUPPORT_INF組込みモジュール関数”参照）

  IEEE 無限大を用意しているか？

- IEEE_SUPPORT_IO（“2.267 IEEE_SUPPORT_IO組込みモジュール関数”参照）

  IEEE の書式処理を用意しているか？

- IEEE_SUPPORT_NAN（“2.268 IEEE_SUPPORT_NAN組込みモジュール関数”参照）

  IEEE 非数を用意しているか？

- IEEE_SUPPORT_ROUNDING（“2.269 IEEE_SUPPORT_ROUNDING組込みモジュール関数”参照）

  IEEE 丸めモードを用意しているか？

- IEEE_SUPPORT_SQRT（“2.270 IEEE_SUPPORT_SQRT組込みモジュール関数”参照）

  IEEE の平方根を用意しているか？

- IEEE_SUPPORT_STANDARD（“2.271 IEEE_SUPPORT_STANDARD組込みモジュール関数”参照）

  IEEE の機能をすべて用意しているか？

- IEEE_SUPPORT_UNDERFLOW_CONTROL（“2.272 IEEE_SUPPORT_UNDERFLOW_CONTROL組込みモジュール関数”参照）

  IEEE の下位けたあふれ制御を用意しているか？

1.15.9.2 要素別処理関数

IEEE_ARITHMETIC モジュールは、次の要素別処理関数をもっています。ただし、これらはIEEE_SUPPORT_DATATYPE(X) およびIEEE_SUPPORT_DATATYPE(Y) が両方とも真である実数X およびY に対して適用されます。

- IEEE_CLASS（“2.239 IEEE_CLASS組込みモジュール関数”参照）

  IEEE 実数の種類を判定します。

- IEEE_COPY_SIGN（“2.240 IEEE_COPY_SIGN組込みモジュール関数”参照）

  IEEE の関数copysign です。

- IEEE_IS_FINITE（“2.246 IEEE_IS_FINITE組込みモジュール関数”参照）

  値が有限かどうかを判定します。

- IEEE_IS_NAN（“2.247 IEEE_IS_NAN組込みモジュール関数”参照）

  値がIEEE 非数かどうかを判定します。

- IEEE_IS_NORMAL（“2.249 IEEE_IS_NORMAL組込みモジュール関数”参照）

  値が正規かどうか、すなわち無限大、非数、または準正規数でないかどうかを判定します。

- IEEE_IS_NEGATIVE（“2.248 IEEE_IS_NEGATIVE組込みモジュール関数”参照）

  値が負かどうかを判定します。

- IEEE_LOGB（“2.250 IEEE_LOGB組込みモジュール関数”参照）

  IEEE 浮動小数点数のかさ上げ分をはずした指数部を返します。

- IEEE_NEXT_AFTER（“2.251 IEEE_NEXT_AFTER組込みモジュール関数”参照）

  Y 方向でX の隣にある表現可能な数を返します。

- IEEE_REM（“2.252 IEEE_REM組込みモジュール関数”参照）

  IEEE の関数rem です。すなわちX - Y × N を返します。ただし、N はX / Y の厳密な値に最も近い整数とします。

- IEEE_RINT（“2.253 IEEE_RINT組込みモジュール関数”参照）

  現在の丸めモードに従って整数に変換します。

- IEEE_SCALB（“2.254 IEEE_SCALB組込みモジュール関数”参照）

  X × 2I を返します。

- IEEE_UNORDERED（“2.273 IEEE_UNORDERED組込みモジュール関数”参照）

  IEEE の関数unordered です。X またはY が非数のときに真、それ以外のときは偽です。

- IEEE_VALUE（“2.274 IEEE_VALUE組込みモジュール関数”参照）

  IEEE 浮動小数点数を生成します。

1.15.9.3 変形関数

IEEE_ARITHMETIC モジュールは、次の変形関数をもっています。

- IEEE_SELECTED_REAL_KIND（“2.255 IEEE_SELECTED_REAL_KIND組込みモジュール関数”参照）

  精度と指数範囲が与えられたIEEE 実数の種別型パラメタ値です。

1.15.9.4 要素処理サブルーチン

IEEE_EXCEPTIONS モジュールは、次の要素処理サブルーチンをもっています。

- IEEE_GET_FLAG（“2.241 IEEE_GET_FLAG組込みモジュールサブルーチン”参照）

  現在の例外フラグを取得します。

- IEEE_GET_HALTING_MODE（“2.242 IEEE_GET_HALTING_MODE組込みモジュールサブルーチン”参照）

  例外に対する停止モードを取得します。

1.15.9.5 非要素処理サブルーチン

IEEE_EXCEPTIONS

IEEE_EXCEPTIONS モジュールは、次の非要素処理サブルーチンをもっています。

- IEEE_GET_STATUS（“2.244 IEEE_GET_STATUS組込みモジュールサブルーチン”参照）

  浮動小数点数環境の現在の状態を取得します。

- IEEE_SET_FLAG（“2.256 IEEE_SET_FLAG組込みモジュールサブルーチン”参照）

  例外フラグを設定します。

- IEEE_SET_HALTING_MODE（“2.257 IEEE_SET_HALTING_MODE組込みモジュールサブルーチン”参照）

  例外が発生したときに続行するか停止するかを制御します。

- IEEE_SET_STATUS（“2.259 IEEE_SET_STATUS組込みモジュールサブルーチン”参照）

  浮動小数点数環境の状態を回復させます。

要素別処理でないサブルーチンIEEE_SET_FLAG およびIEEE_SET_HALTING_MODE は純粋です。IEEE_EXCEPTIONS に含まれる要素別処理でないサブルーチンには、他に純粋のものはありません。

IEEE_ARITHMETIC

IEEE_ARITHMETIC モジュールは、次の要素別処理でないサブルーチンをもっています。

- IEEE_GET_ROUNDING_MODE（“2.243 IEEE_GET_ROUNDING_MODE組込みモジュールサブルーチン”参照）
  現在のIEEE 丸めモードを取得します。
- IEEE_GET_UNDERFLOW_MODE（“2.245 IEEE_GET_UNDERFLOW_MODE組込みモジュールサブルーチン”参照）
  現在の下位けたあふれモードを取得します。
- IEEE_SET_ROUNDING_MODE（“2.258 IEEE_SET_ROUNDING_MODE組込みモジュールサブルーチン”参照）
  IEEE 丸めモードを設定します。
- IEEE_SET_UNDERFLOW_MODE（“2.260 IEEE_SET_UNDERFLOW_MODE組込みモジュールサブルーチン”参照）
  下位けたあふれモードを設定します。

IEEE_ARITHMETIC に含まれる要素別処理でないサブルーチンには、純粋のものはありません。

1.16 FORALL制御

FORALL制御は、FORALL構文（“2.187 FORALL構文”参照）、単純FORALL文（“2.189 単純FORALL文”参照）、およびDO CONCURRENT（“2.120 DO構文”参照）で使用します。FORALL制御は、以下の形式です。

(*forall-triplet-spec-list* [, *scalar-mask-expr*])

forall-triplet-spec-list は、コンマで区切られた*forall-triplet-spec* の並びです。

forall-triplet-spec は、FORALL三つ組指定で以下の形式です。

[*integer-type-spec* ::] *index-name* = *subscript* : *subscript* [: *stride*]

integer-type-spec は、整数型指定子で以下の形式です。

INTEGER [*kind-selector*]

index-name は指標変数名であり、整数型の名前付きスカラ変数でなければなりません。

subscript は添字であり、スカラ整数式でなければなりません。

stride は刻み幅であり、スカラ整数式でなければなりません。

subscript および*stride* は、それが現れるFORALL三つ組指定並び中のどの指標変数名も引用してはなりません。

scalar-mask-expr はスカラ選別式であり、論理型のスカラ式でなければなりません。

スカラ選別式の中で引用する手続は、純粋手続でなければなりません。

1.16.1 指標変数名の値の決定

FORALL三つ組指定並び中の添字および刻み幅の式が評価されます。これらの式は、任意の順序で評価され、指標変数の値は、次のように決定されます。

1. 添字m_1、添字m_2、および刻み幅m_3は、指標変数名と同じ種別型パラメタをもつ整数型となります。これらの値は、それぞれ最初の添字の式、2番目の添字の式、および刻み幅の式を評価した結果です。刻み幅を省略した場合、m_3の値は1となります。m_3の値は、0であってはなりません。
2. $(m_2 - m_1 + m_3) / m_3$を*count*とするとき、*count*が0以下であればその構文の実行は終了します。それ以外のとき、指標変数名の値の集合は以下のとおりです。

   $m_1+(k-1) \times m_3$　　k は 1, 2, ..., *count*

1.16.2 FORALL制御選別式

スカラ選別式がある場合、その式は指標値のそれぞれの組合せに対して評価されます。スカラ選別式がない場合、真の値をもつ式があるものとして評価されます。

1.17 共配列

共配列はゼロでない共次元数を含むデータ要素であり、いかなる像からも、直接参照または定義できます。共配列は配列でも、スカラでもかまいません。共配列は、型宣言文(“2.3 型宣言文”)、ALLOCATABLE 文(“2.19 ALLOCATABLE文”)、CODIMENSION 文(“2.80 CODIMENSION文”)、またはTARGET文(“2.476 TARGET文”)の共配列指定(“1.17.1 共配列指定”)で指定します。共配列は、関数結果であってはなりません。BLOCK構文(“2.57 BLOCK構文”)の有効範囲で共配列指定は指定できません。

共配列には、共形状明示配列(“1.17.2 共形状明示配列”)と割付け共配列(“1.17.3 割付け共配列”)があります。

1つの像の各共配列は、他のすべての像で、同じ型、型パラメタ、および上下限である共配列でなければなりません。どの像の共配列でも、共添字を使って引用することができます。

共配列の実体は、仮引数であるか、ALLOCATABLE属性、またはSAVE属性をもたなければなりません。

共配列末端成分をもつ実体は、仮引数であるか、またはSAVE属性をもたなければなりません。

派生型定義(“1.5.11.1 派生型定義”)のデータ成分定義文に共配列指定(“1.17.1 共配列指定”)ができます。

共配列は、

- C_PTR またはC_FUNPTR型であってはなりません。
- PARAMETER属性、POINTER属性、BIND属性、またはVALUE属性を指定してはなりません。
- 参照結合、または親子結合によって参照する場合、VOLATILE 属性を指定してはなりません。
- 結合実体または共通ブロック実体に指定できません。
- 成分にALLOCATABLE属性またはPOINTER属性を指定できません。
- パラメタ付き派生型、またはパラメタ付き派生型を成分としてもつ型であってはなりません。

1.17.1 共配列指定

共配列指定には、共形状明示指定と共形状無指定指定があります。共形状明示指定については“1.17.2 共形状明示配列”、共形状無指定指定については“1.17.3 割付け共配列” を参照してください。

1.17.2 共形状明示配列

共形状明示配列は、共形状明示指定によって、共配列の共次元数および共上下限を宣言します。割付けでない共配列は、共形状明示指定で宣言しなければなりません。

共形状明示指定は、次の形式です。

[[*lower-cobound* :] *upper-cobound,*] ... [*lower-cobound* :] *

lower-cobound は、共下限であり、宣言式でなければなりません。

upper-cobound は、共上限であり、宣言式でなければなりません。

共次元数は、共上限の数に1を足した数と同じです。実体の次元数と共次元数の合計は30を超えてはなりません。

各々の共下限と共上限の値は、共配列の共上下限を決定します。

もし、共下限が省略されたら、省略値は1です。共上限は共下限より小さくてはなりません。

共形状明示配列は、仮引数でないならばSAVE属性が必要です。

共形状明示配列の例:

```
integer,save:: coarray[2:3,*]
```

1.17.3 割付け共配列

割付け共配列はALLOCATABLE 属性をもつ共配列です。共形状無指定指定並びで、共次元数を宣言します。共上下限は、割付けまたは引数結合により決定されます。

共形状無指定指定は、以下の形式(コロンのみ)です。

```
:
```

ALLOCATABLE 属性をもつ共配列は、共配列指定が共形状無指定指定並びでなければなりません。

割付け共配列の共次元数は、共形状無指定指定並びのコロンの数と同じです。実体の次元数と 共次元数の合計は30を超えてはなりません。

未割付けの割付け共配列の共上下限は不定です。そして、参照または定義できません。

割付け共配列の割付けについは、ALLOCATE 文(“2.20 ALLOCATE文”)を参照してください。

RETURN文またはEND文の実行による割付け共配列の解放で、すべての像の暗黙の同期があります。

割付け共配列の例:

```
integer,allocatable:: coarray(:) [:,:]
```

1.17.4 共配列引用

共配列引用は、次の形式です。

```
part [ %part ] ...
```

part は部分参照です。部分参照は、次の形式です。

```
part-name [ ( subscript -list ) ] [ image-selector ] [ ( substrings ) ]
```

part-name は部分名です。もっとも左側の部分名は、共配列でなければなりません。

‘%’演算子の使用は構造体成分の引用を意味します。詳細については、“1.5.11.6 構造体成分”を参照してください。

(*subscript-list*) の指定を含む場合は、*name* は配列でなければなりません。配列引用については、“1.5.8.1 配列引用”を参照してください。

image-selector は像選択子(“1.17.5 像選択子”)です。

image-selector をもつ実体は、共添字付き実体です。*image-selector* が指定された場合、

- 部分名が配列なら、(*subscript-list*) が現れなければなりません。
- C_PTRまたはC_FUNPTR 型であってはなりません。
- 部分名が選択子であってはなりません。
- ポインタ成分または割付け成分は、指定できません。

自身の像では、共配列は像選択子を省略して引用することができます。

共配列の部分実体は、共配列です。ただし、共添字付き実体は、共配列ではありません

(*substrings*)の指定を含む場合は、*name* は文字型でなければなりません。部分列引用については、“1.5.7 部分列”を参照してください。

1.17.5 像選択子

像選択子は、共添字付き実体の像番号を決定します。

像選択子は、次の形式です。

```
left-square-bracket cosubscript-list right-square-bracket
```

cosubscript は、共添字です。共添字は、スカラ整数式でなければなりません。

left-square-bracket は、左角括弧です。左角括弧は、[です。

right-square-bracket は、右角括弧です。右角括弧は、] です。

共添字の数は、実体の共次元数と同じでなければなりません。

像選択子内の共添字の値は、共次元の共上下限の範囲でなければなりません。

像選択子の共添字並びは、配列要素の添字式の並びが上下限を考慮して添字位置を決定するのと同じ方法で、共上下限を考慮して、像番号を決定します。

像選択子が現れるデータ参照は、C_PTRまたはC_FUNPTR型であってはなりません。

共添字付き実体は、多相的であってはなりません。

像選択子の例:

```
integer,save:: coarray_scalar [ * ]
integer,save:: coarray_array(2) [ 2 , * ]
coarray_scalar[ 1 ] = 1
coarray_array( 2 ) [ 1, 1 ] = 2
coarray_array( 1:2 ) [ 1, 1 ] = 3
coarray_array( : ) [ 1, 1 ] = 4

coarray_array [ 1, 1 ] = 5  !  誤りです。（ subscript-list ）が現れなければなりません。
```

1.18 像制御文

像制御文を実行すると、像における実行順序はいくつかのセグメントに分割されます。像制御文には以下があります。

- SYNC ALL 文（“2.467 SYNC ALL文”参照）
- SYNC IMAGES 文（“2.468 SYNC IMAGES文”参照）
- SYNC MEMORY 文（“2.469 SYNC MEMORY文”参照）
- 割付け実体に共配列が指定されたALLOCATE 文（“2.20 ALLOCATE文”参照）またはDEALLOCATE 文（“2.116 DEALLOCATE文”参照）
- CRITICAL 文（“2.102 CRITICAL構文”参照）またはEND CRITICAL 文（“2.136 END CRITICAL文”参照）
- LOCK 文（“2.327 LOCK文”参照）またはUNLOCK 文（“2.494 UNLOCK文”参照）
- ブロックまたは手続を完了させ、暗黙のうちに共配列を解放する文
- 共配列が指定されたMOVE_ALLOC組込みサブルーチン（“2.359 MOVE_ALLOC組込みサブルーチン”参照）
- STOP 文（“2.460 STOP文”参照）
- 主プログラムのEND 文（“2.132 END文”参照）

CRITICAL 文、END CRITICAL 文、LOCK 文、UNLOCK 文以外のすべての像制御文はSYNC MEMORY 文の効果を含んでいます。

ある文の実行により複数の手続が呼び出される場合、CRITICAL文およびEND CRITICAL文を除く像制御文を実行する手続呼出しが複数あってはなりません。

1.18.1 セグメント

各像において、像制御文によって区切られる文の集合をセグメントと呼びます。ある像制御文の直前のセグメントには、その像制御文中の式の評価も含みます。

像制御文の実行、または使用者定義の順番付けによって、ある像Pのi番目のセグメントP_iと、ある像Qのj番目のセグメントQ_jの実行される順番を保証することができます。使用者定義の順番付けについては“2.469 SYNC MEMORY文”を参照してください。順番を保証しない場合、像の実行速度によってはP_iとQ_jは同時に実行されることがあります。同じ像で実行されるセグメント同士は順番付けされています。

アトミックサブルーチンの実行によって共配列を参照または定義する場合、セグメント同士が順番付けされていなくてもかまいません。アトミックサブルーチンを使用しない場合、以下の条件を満たす必要があります。

- ある像のあるセグメントにおいて変数が定義される場合、そのセグメントと順番付けされたセグメントでなければ、参照、定義、または未定義になってはなりません。
- ある像のあるセグメントにおいて、共配列の割付け部分実体の割付け状態、または共配列のポインタ部分実体のポインタ結合が変更される場合、そのセグメントと順番付けされたセグメントでなければ、その部分実体は、参照または定義されてはなりません。
- 像Pにおいて、共配列でない仮引数を定義する手続の実行がセグメントP_i, P_i+1, … , P_kにまたがる場合、像Qのセグメント Q_jがP_i に先行するかP_k の後に続かない限り、実引数がセグメントQ_jにおいて参照、定義、または未定義になってはなりません。

1.18.2 同期状態指定子

同期状態指定子には、状態変数を指定するSTAT指定子と、誤り通報変数を指定するERRMSG指定子があります。状態変数に代入される値と名前付き定数については、“2.305 ISO_FORTRAN_ENV組込みモジュール”を参照してください。

STAT指定子が指定されたLOCK 文、SYNC ALL 文、SYNC IMAGES 文、SYNC MEMORY 文、またはUNLOCK 文が成功すると、状態変数に0が代入されます。

SYNC ALL 文またはSYNC IMAGES 文にSTAT指定子が現れ、かつ実行を終えた像との同期を含む場合、状態変数にはSTAT_STOPPED_IMAGEの値が代入され、SYNC ALL 文またはSYNC IMAGES 文の効果はSYNC MEMORY 文の効果と同じになります。

LOCK 文にSTAT 指定子が現れ、かつロック変数が実行中の像によりロックされている場合、状態変数にはSTAT_LOCKEDの値が代入されます。UNLOCK 文にSTAT 指定子が現れ、かつロック変数がロックされていない場合、状態変数にはSTAT_UNLOCKEDの値が代入され、ロック変数が他の像によってロックされている場合、状態変数にはSTAT_LOCKED_OTHER_IMAGEの値が代入されます。

STAT指定子が省略されたLOCK 文、SYNC ALL 文、SYNC IMAGES 文、SYNC MEMORY 文、またはUNLOCK 文の実行中に誤り条件が起きると、誤り終了処理が開始します。

LOCK 文、SYNC ALL 文、SYNC IMAGES 文、SYNC MEMORY 文、またはUNLOCK 文にERRMSG指定子が現れ、かつそれらの文の実行中に誤り条件が起きた場合、誤り通報変数に説明のメッセージが代入されます。誤り条件が起きなかった場合、変数の値は変更されません。

1.18.3 LOCK_TYPE 型

LOCK_TYPE は非公開の成分をもつ派生型です。すべての成分は割付けでなくポインタでもありません。型パラメタをもたない拡張可能な型です。BIND(C) 属性および型パラメタをもたず、連続型ではありません。すべての成分は暗黙の初期化をもちます。

LOCK_TYPE 型のスカラ変数は、ロック変数です。ロック変数は、ロック状態またはロック解除状態のどちらかの状態です。ロック解除状態はLOCK_TYPE 型の初期値です。この値は、構造体構成子LOCK_TYPE() の示す値です。他のすべての値は、ロック状態を現します。ロック変数の値は、LOCK 文およびUNLOCK 文により変更できます。

LOCK_TYPE型の名前付き変数は、共配列でなければなりません。共配列でないLOCK_TYPE 型の部分成分をもつ名前付き変数は、共配列でなければなりません。

ロック変数は、以下の場合を除き、変数定義文脈に現れてはなりません。

- LOCK 文またはUNLOCK 文のロック変数として現れる。
- ALLOCATE 文またはDEALLOCATE 文中の割付け実体として現れる。
- 明示的引用仕様をもつ手続への参照の実引数として現れ、対応する仮引数がINTENT(INOUT)属性をもつ。

LOCK_TYPE型の部分実体をもつ変数は、以下の場合を除き、変数定義文脈に現れてはなりません。

- ALLOCATE 文またはDEALLOCATE 文中の割付け実体として現れる。
- 明示的引用仕様をもつ手続への参照の実引数として現れ、対応する仮引数がINTENT(INOUT)属性をもつ。

派生型定義(“1.5.11.1 派生型定義”参照)にEXTENDS が現れて、定義される型が、LOCK_TYPE 型の末端成分をもっているなら、親型は、LOCK_TYPE 型の末端成分をもたなければなりません。

第2章 文および手続の詳細

この章では、Fortran の各文、構文、組込み手続、およびサービスルーチンの構文規則および使用方法について説明します。

2.1 代入文

代入文は、等号'='の右側の式の評価結果を、等号'='の左側の変数に代入します。

代入文は、以下の形式です。

```
variable = expr          または
fun_expr = expr
```

variable は、変数です。変数は、大きさ引継ぎ配列の全体配列引用であってはなりません。

fun_expr は、データポインタの結果をもつ関数引用でなければなりません。

expr は、式です。

代入文は組込み代入文または利用者定義代入文のいずれかです。

利用者定義代入文は、組込み代入文でない代入文であり、サブルーチン副プログラムおよびASSIGNMENT 指定を伴う引用仕様宣言によって定義されます。利用者定義代入については、"1.12.7.3.3 利用者定義代入"および"1.12.7.2 手続引用仕様宣言"を参照してください。

組込み代入は、変数が共配列でない割付け配列の場合を除き、変数と式とは、形状適合しなければなりません。

変数が整数型、実数型、または複素数型である場合、式は整数型、実数型、または複素数型のいずれかでなければなりません。変数が文字型である場合、式は変数と同じ種別型パラメタをもつ文字型でなければなりません。変数が論理型である場合、式は論理型でなければなりません。変数が派生型である場合、式は変数と同じ派生型でなければなりません。変数は多相的であってはなりません。

式が配列であるなら、変数も配列でなければなりません。変数が配列である場合、式はスカラであってもかまいません。この場合、変数のすべての要素がスカラ式の値で確定されます。

変数がポインタである場合、変数は指示先の型、型パラメタ、および形状が式のそれと適合するような、確定可能な指示先と結合していなければなりません。

変数がポインタである場合、式の値は、変数の指示先に代入されます。

変数および式の型が数値型であり、異なる数値型または異なる種別型パラメタをもつ場合、式の値は以下の表の規則に従って変数の型および型パラメタに変換されます。

変数の型	代入される値
整数型	INT (式 , KIND = KIND (変数))
実数型	REAL (式 , KIND = KIND (変数))
複素数型	CMPLX (式 , KIND = KIND (変数))

関数INT、REAL、CMPLX、およびKIND は、総称組込み関数です。

変数および式の型が論理型であり、異なる種別型パラメタをもつ場合、式の値は変数の種別型パラメタに変換されます。

変数および式の型が文字型であり、文字長パラメタが異なる場合、式の長さは変数の長さに変換されます。変数の長さが式の長さよりも小さい場合、式の値は、変数と同じ長さになるように右側を切り詰めます。変数の長さが式の長さよりも大きい場合、式の値は、変数と同じ長さになるように右側に空白を補い拡張します。

派生型の組込み代入においては、ポインタ成分についてはポインタ代入を用い、ポインタでも割付けでもない成分については、組込み代入を用います。割付け成分については、以下の順の操作を用います。

1. 変数の成分が割り付けられている場合、解放されます。
2. 式の成分が割り付けられている場合、変数の対応する成分は同じ形状で割り付けられ、式の成分の値は変数の対応する成分に組込み代入を用いて代入されます。

割付け成分の代入において式の成分が割り付けられていない場合、変数の成分は代入文の実行後、割り付けられていません。

共添字をもつ共配列を除く変数が未割付けの場合、式は同じ次元数でなければなりません。その変数が割り付けられている場合、式と変数が異なる形状、または変数が長さ無指定の文字型で式の対応する長さ型パラメタ値が異なるとき、割り付けられた変数は解放されます。

共添字をもつ共配列は、式と適合する形状と型パラメタで割り付いていなければなりません。

変数が割り付けられていない割付け変数であるかまたは割り付けられていない割付け変数になる場合、変数の無指定の型パラメタ、形状および各上下限は、式の対応する長さ型パラメタ、形状、およびLBOUND(式) と等しく割り付けられます。ただし、変数が配列で式がスカラの場合、変数は直前の上下限のままです。

変数が多相的ならば、共配列または関数引用でなくALLOCATABLE属性をもたなければなりません。

組込み代入文の例:

```
real :: a=1.5, b(10)=1.0
integer :: i=2, j(10)
character (len = 5) :: string5 = "abcde"
character (len = 7) :: string7 = "cdefghi"
type person
  integer :: age
  character (len = 25) :: name
end type person
type (person) :: person1, person2
i = a                           ! i にはint(a) が代入されます
i = j                           ! 形状適合しないのでエラーです
j = i                           ! j の各要素にはi の値が代入されます
j = b                           ! j の各要素にはb の各要素が整数型に
                                ! 変換されて代入されます
string5 = string7               ! string5にはstring7の右側2文字が
                                ! 切り詰められた値が代入されます
string7 = string5               ! string7にはstring5の右側に空白2文字が
                                ! 補われた値が代入されます
person1 % age = 5
person1 % name = "john"
person2 = person1               ! person2 の各成分にはperson1 の対応する
                                ! 各成分が代入されます
```

2.2 ポインタ代入文

ポインタ代入によって、ポインタは指示先と結合するか、空状態、または不定結合状態になります。ポインタと指示先との間の以前のどんな結合も解放されます。ポインタ代入文には、ポインタ代入と手続ポインタ代入があり、以下の形式です。

data-pointer-object [(*bounds-spec-list*)] => *data-target*　　または
data-pointer-object (*bounds-remapping-list*) => *data-target*　　または
proc-pointer-object => *proc-target*

data-pointer-object は、データポインタ実体であり、以下の形式です。

variable-name　　または
variable % *data-pointer-component-name*

データポインタ実体は、POINTER 属性をもたなければなりません。

bounds-spec-list は、上下限指定並びであり、上下限指定は次の形式です。

下限式 :

bounds-remapping-list は、上下限再配置並びであり、上下限再配置は次の形式です。

下限式 : **上限式**

data-target は、データ指示先指定であり、以下の形式です。

variable または
expr

variable は変数です。変数はTARGET 属性をもつか、TARGET 属性をもつ部分実体であるか、またはPOINTER 属性をもたなければなりません。変数は、共添字付き実体であってはなりません。

expr は式です。式はデータポインタとなる結果を返さなければなりません。

データ指示先指定は、ポインタと同じ型、種別型パラメタ、および次元数をもたなければなりません。

データ指示先指定は、ベクトル添字をもつ部分配列であってはなりません。

proc-pointer-object は、手続ポインタ実体であり、以下の形式です。

proc-pointer-name または
proc-component-ref

proc-component-ref は、手続ポインタ成分であり、以下の形式です。

variable % *procedure-component-name*

手続ポインタ実体は、手続ポインタでなければなりません。

proc-target は、手続指示先です。

手続指示先には、組込みでない要素別処理手続名を指定できません。

手続指示先は、以下の形式です。

expr または
procedure-name または
proc-component-ref

expr は式です。関数引用であり、手続ポインタとなる結果を返さなければなりません。

procedure-name は、手続名です。

手続名は、外部手続名、モジュール手続名、仮手続名、内部手続名、個別組込み関数名、または手続ポインタ名でなければなりません。ここで個別組込み関数名は、以下のいずれかでなければなりません。

ABS、ACOS、AIMAG、AINT、ALOG、ALOG10、AMOD、ANINT、ASIN、ATAN、ATAN2、CABS、CCOS、CEXP、CLOG、CONJG、COS、COSH、CSIN、CSQRT、DABS、DACOS、DASIN、DATAN、DATAN2、DCOS、DCOSH、DDIM、DEXP、DIM、DINT、DLOG、DLOG10、DMOD、DNINT、DPROD、DSIGN、DSIN、DSINH、DSQRT、DTAN、DTANH、EXP、IABS、IDIM、IDNINT、INDEX、ISIGN、LEN、MOD、NINT、SIGN、SIN、SINH、SQRT、TAN、またはTANHです。

指示先指定がポインタでないなら、ポインタ代入文は、ポインタ実体を指示先に結合します。指示先指定がすでに結合されているポインタである場合、ポインタ実体をその指示先と同じ実体に結合します。指示先が空状態のポインタであるか、または組込み関数NULL の引用である場合、ポインタ実体の結合状態も空状態になります。指示先が不定結合状態のポインタであるならば、ポインタ実体も不定結合状態になります。

ポインタ実体の直前までの指示先との結合は、すべて上書きされます。

構造体のポインタ成分へのポインタ代入は、派生型の組込み代入文または利用者定義代入文の実行によって行うこともできます。

データポインタ実体は、ALLOCATE 文による割付けによっても、ポインタは指示先に結合できます。

ポインタは、引用可能または確定可能な指示先と結合されない限り、引用または確定することはできません。

2.2.1 データポインタ代入

データポインタ実体が多相的でなく、データ指示先が多相的であり、実行時の型が宣言時の型と異なる場合、代入指示先はデータポインタ実体の型の成分になります。それ以外の場合、代入指示先はデータ指示先になります。

データ指示先がポインタでない場合、データポインタ実体は、代入指示先に結合されたポインタになります。ポインタの場合、データポインタ実体のポインタ結合状態はデータ指示先のものになり、データ指示先が実体と結合されるとき、データポインタ実体は代入指示先に結合されたものになります。データ指示先が割付け実体ならば、割付け済でなければなりません。

データポインタ実体が多相的な場合、データポインタ実体の実行時の型はデータ指示先の実行時の型になります。

データポインタ実体が連続派生型またはBIND 属性をもつ型の場合、データ指示先の実行時の型はその派生型でなければなりません。

データポインタ実体がCONTIGUOUS属性(“2.88 CONTIGUOUS文”参照)をもつ場合、データ指示先はCONTIGUOUS(“2.88 CONTIGUOUS文”参照)でなければなりません。

データ指示先が空状態ポインタの場合、データ指示先の無指定型パラメタでない型パラメタに対応するデータポインタ実体の宣言時の型の無指定型パラメタでないすべての型パラメタは、対応するデータ指示先の対応する型パラメタとして同じ値をもたなければなりません。

そうでない場合、データポインタ実体の宣言時の型の無指定型パラメタでないすべての型パラメタは、データ指示先の対応する型パラメタとして同じ値をもたなければなりません。

データポインタ実体がデータ指示先の無指定型パラメタに対応する無指定型パラメタでない型パラメタをもつ場合、データ指示先は結合状態が不定のポインタであってはなりません。

上下限再配置並びを指定した場合、データ指示先は空状態または不定ポインタであってはならず、データ指示先の大きさはデータポインタ実体の大きさより小さくてはなりません。また、データ指示先は、次元数が1または単純CONTIGUOUSでなければなりません。

上下限再配置並びを指定しない場合、データポインタ実体の次元の寸法はデータ指示先の対応する次元の寸法になります。また、各次元の上限は、下限と寸法の和より1だけ少ない値になります。

上下限再配置並びおよび上下限指定並びを指定しないとき、各次元の下限は、データ指示先を引数とする組込み関数LBOUNDの結果の値になります。

データポインタ代入文の例:

```
real, pointer :: a
real, target :: b = 5.0
a => b                  ! ポインタa は指示先b と結合します
end
```

2.2.2 手続ポインタ代入

手続指示先がポインタでない場合、手続ポインタ実体は、手続指示先と競合します。ポインタの場合、手続ポインタ実体のポインタ結合状態は手続指示先のものになり、手続指示先が手続に結合しているとき、手続ポインタ実体は同じ手続と結合します。

手続ポインタ実体が明示的引用仕様をもつ場合、その特性は手続指示先と同じでなければなりません。ただし、手続ポインタ実体が純粋でなくても手続指示先が純粋であるかまたは手続ポインタ実体が要素別処理でなくても手続指示先が要素別処理組込み手続であるときは、同じである必要はありません。

手続ポインタ実体または手続指示先の特性が明示的引用仕様を必要とする場合、手続ポインタ実体と手続指示先の両方は明示的引用仕様をもたなければなりません。

手続ポインタ実体が暗黙的引用仕様をもち、明示型または関数として引用される場合、手続指示先は関数でなければなりません。

手続ポインタ実体が暗黙的引用仕様をもち、サブルーチンとして引用されるとき、手続指示先はサブルーチンでなければなりません。

手続指示先および手続ポインタ実体が関数の場合、それらは同じ型でなければならず、対応する型パラメタは共に無指定であるかまたは同じ値をもたなければなりません。

手続名が個別手続名であり、かつ総称名であるとき、ポインタ実体と結合されるのは個別手続だけです。

手続ポインタ代入文の例:

```
interface
  function ifunc () result(irst)
    integer :: irst
  end function
end interface
procedure(ifunc), pointer :: pifunc
pifunc => ifunc    ! 手続ポインタpifunc は、手続ifunc と結合します。
print *,pifunc()
end
function ifunc() result(irst)
  integer :: irst
  irst=1
end function
```

2.3 型宣言文

型宣言文は、データ実体の型、型パラメタ、およびその他の属性を宣言します。

型宣言文は、以下の形式です。

type-spec [[, *attr-spec*]... ::] *entity-decl-list*

type-spec は宣言型指定子であり、以下の形式です。

組込み型指定子	または
TYPE (**組込み型指定子**)	または
TYPE (*type-name*)	または
CLASS(*type-name*)	または
CLASS(*)	または
TYPE(*)	

組込み型指定子は、以下の形式です。

INTEGER [*kind-selector*]	または
REAL [*kind-selector*]	または
DOUBLE PRECISION	または
COMPLEX [*kind-selector*]	または
CHARACTER [*char-selector*]	または
LOGICAL [*kind-selector*]	または
BYTE	

kind-selector は以下の形式です。

([KIND =] *kind*)	または
* *mem-length*	

char-selector は、以下の形式です。

(LEN = *char-length-parm* , KIND = *kind*)	または
(*char-length-parm* , [KIND =] *kind*)	または
(KIND = *kind* [, LEN = *char-length-parm*])	または
([LEN =]*char-length-parm*)	または
* *char-length* （**廃止予定事項**）	

char-length は、以下の形式です。

(*char-length-parm*)	または
scalar-int-literal-constant	

scalar-int-literal-constant はスカラ整定数表現です。

char-length-parm は、以下の形式です。

scalar-int-expr	または
*	または
:	

kind は種別型パラメタであり、整数型のスカラ定数式でなければなりません。

mem-length は、領域の長さを示し、整数型のスカラ定数式でなければなりません。型指定子がINTEGER、REAL、およびLOGICAL の場合、*mem-length* の値は*kind* の値と同じ意味をもちます。型指定子がCOMPLEX の場合、*mem-length* の値は、*kind* の値を2倍にしたものと同じ意味です。

scalar-int-expr は、宣言式です。

':'の文字長パラメタ値は、無指定型パラメタであることを示します。

型指定子と指定可能な*kind* および*mem-length* の値、および型との対応を、以下の表に示します。

型指定子	*kind*	*mem-length*	型
INTEGER	1 2 4または 指定なし 8	1 2 4または 指定なし 8	1バイトの整数型 2バイトの整数型 4バイトの整数型 （基本整数型） 8バイトの整数型
REAL	2 4または 指定なし 8 16	2 4または 指定なし 8 16	半精度実数型 単精度実数型 （基本実数型） 倍精度実数型 4倍精度実数型
DOUBLE PRECISION	-----	-----	倍精度実数型
COMPLEX	2 4または 指定なし 8 16	4 8または 指定なし 16 32	半精度複素数型 単精度複素数型 （基本複素数型） 倍精度複素数型 4倍精度複素数型
LOGICAL	1 2 4または 指定なし 8	1 2 4または 指定なし 8	1バイトの論理型 2バイトの論理型 4バイトの論理型 （基本論理型） 8バイトの論理型
CHARACTER	1または 指定なし	-----	1バイトの文字型 （ASCII 文字型、 基本文字型） （長さは*char-selector*により指定される）
BYTE	-----	-----	1バイトの整数型

*derived-type-spec*は、派生型指定子です。派生型指定子については、“1.5.11.8 派生型指定子”を参照してください。

attr-spec は属性指定子であり、以下の形式です。

```
ALLOCATABLE                                                    または
ASYNCHRONOUS                                                   または
AUTOMATIC                                                      または
CODIMENSION left-square-bracket coarray-spec right-square-bracket または
CONTIGUOUS                                                     または
DIMENSION ( array-spec )                                       または
EXTERNAL                                                       または
INTENT ( intent-spec )                                         または
INTRINSIC                                                      または
language-binding-spec                                          または
OPTIONAL                                                       または
PARAMETER                                                      または
POINTER                                                        または
PRIVATE                                                        または
PROTECTED                                                      または
PUBLIC                                                         または
SAVE                                                           または
STATIC                                                         または
TARGET                                                         または
```

```
VALUE                                                    または
VOLATILE
```

coarray-spec は、共配列指定です。詳細については“1.17.1 共配列指定”を参照してください。

left-square-bracket は、左角括弧です。左角括弧は、[です。

right-square-bracket は、右角括弧です。右角括弧は、] です。

array-spec は、配列形状指定です。

intent-spec は授受特性指定で、以下の形式です。

```
IN          または
OUT         または
INOUT
```

language-binding-spec は、言語束縛指定子で以下の形式です。

```
BIND ( C [, NAME = スカラ文字定数式] )
```

各属性指定子の詳細については、対応する属性宣言文(“2.19 ALLOCATABLE文”、“2.31 ASYNCHRONOUS文”、“2.41 AUTOMATIC文”、“2.52 BIND文”、“2.80 CODIMENSION文”、“2.88 CONTIGUOUS文”、“2.119 DIMENSION文”、“2.179 EXTERNAL文”、“2.290 INTENT文”、“2.292 INTRINSIC文”、“2.373 OPTIONAL文”、“2.375 PARAMETER文”、“2.379 POINTER文”、“2.387 PRIVATE文”、“2.394 PROTECTED文”、“2.395 PUBLIC文”、“2.422 SAVE文”、“2.459 STATIC文”、“2.476 TARGET文”、“2.498 VALUE文”、および“2.500 VOLATILE文”)を参照してください。

1つの型宣言文中で、同じ属性指定子を2回以上指定してはなりません。

entity-decl-list はコンマで区切られたデータ要素宣言の並びです。

entity-decl は、以下の形式です。

```
object-name [ ( array-spec ) ] [ * length ] [ initialization ]          または
object-name [ * length ] [ ( array-spec ) ] [ initialization ]          または
object-name [ ( array-spec ) ] [ left-square-bracket  coarray-spec  right-square-bracket  ] ■
■  [ * length ] [ initialization ]                                      または
object-name [ * length ] [ ( array-spec ) ] ■
■  [ left-square-bracket  coarray-spec  right-square-bracket  ] ■
■  [ initialization ]                                                   または
function-name [ * length ]
```

object-name は、実体名です。

length は長さ指定であり、以下の形式です。

```
char-length         または
mem-length
```

データ要素宣言中の**char-length* は、型指定がCHARACTER の場合のみ指定できます。データ要素宣言中の**mem-length* は、型指定がINTEGER、REAL、COMPLEX、およびLOGICAL の場合のみ指定できます。

char-length 型パラメタ値としての':' は、POINTER 属性またはALLOCATABLE 属性をもつ要素または成分の宣言内でだけ使用できます。

initialization は初期値指定であり、以下の形式です。

```
= initialization-expr           または
=> NULL( )                      または
=> initial-data-target          または
/ data-stmt-value-list /
```

initialization-expr は定数式です。

initial-data-target は初期指示先データです。特定子(“1.5.17 特定子”参照)を指定します。

data-stmt-value-list は、コンマで区切られた1つ以上の定数表現並びです。詳細については、“2.112 DATA文”を参照してください。

function-name は関数名であり、外部関数、組込み関数、仮手続関数、または文関数の名前でなければなりません。

1つの有効域内で、1つのデータ要素に同じ属性を2回以上指定してはなりません。

POINTER 属性またはALLOCATABLE 属性をもって宣言された配列は、形状無指定配列でなければなりません。

POINTER 属性およびALLOCATABLE 属性をもたない関数結果の配列は、形状明示配列でなければなりません。

POINTER 属性を指定したとき、TARGET 属性、およびINTRINSIC 属性は同時に指定できません。

TARGET 属性を指定したとき、POINTER 属性、EXTERNAL 属性、INTRINSIC 属性、およびPARAMETER 属性は同時に指定できません。

PARAMETER 属性は、仮引数、ポインタ、割付け、末端成分に割付けをもつ派生型の実体、関数、および共通ブロック中の実体には指定できません。

INTENT 属性、OPTIONAL 属性、およびVALUE 属性は仮引数にだけ指定することができます。

EXTERNAL 属性およびINTRINSIC 属性は、関数にだけ指定することができます。

1つの配列にALLOCATABLE 属性とPOINTER 属性との両方を指定してはなりません。

PARAMETER 属性があるときは、‘=*initialization-expr*’を指定しなければなりません。

初期値指定がある場合、データ要素宣言並びの前の区切り、2連コロン‘::’を省略してはなりません。

初期値指定は、実体名が仮引数、関数結果、初期値設定プログラム単位以外における名前付き共通ブロック中の実体、無名共通ブロック中の実体、割付け配列、末端成分に割付け配列を含む派生型の実体、自動割付け変数、および関数名のいずれかであるならば、指定してはなりません。

初期値指定に‘=>’を書いたとき、実体はPOINTER 属性をもたなければなりません。
初期値指定に‘=*initialization-expr*’を指定したとき、実体はPOINTER 属性をもってはなりません。

初期値指定に初期指示先データを指定したとき、その特定子(“1.5.17 特定子”参照)は、割付け可能でなく、TARGET属性とSAVE属性をもたなければなりません。その特定子はベクトル添字をもたず、すべての添字、三つ組み添字、文字部分列開始位置、文字部分列終了位置は、定数式でなければなりません。ポインタ実体とその特定子は、データポインタ初期化適合(“2.112 DATA文”参照)でなければなりません。

VOLATILE 属性を指定したとき、PARAMETER 属性、INTRINSIC 属性、およびEXTERNAL 属性は同時に指定できません。

VALUE 属性を指定したとき、PARAMETER 属性、EXTERNAL 属性、POINTER 属性、ALLOCATABLE 属性、INTENT(INOUT) 属性、およびINTENT(OUT) 属性は同時に指定できません。

VALUE 属性を指定したとき、大きさ引継ぎ配列、共配列、共配列末端成分、多相的、型引継ぎ、および次元引継ぎは同時に指定できません。

VALUE 属性は、純粋サブルーチン、および後始末サブルーチンの仮引数に指定してはなりません。

VALUE 属性は、仮手続および手続ポインタに対して指定してはなりません。

言語束縛指定子は、モジュールの宣言部にだけ書くことができます。

言語束縛指定子で宣言された要素は、相互利用可能変数でなければなりません。

言語束縛指定子がNAME 指定子とともに指定された場合、データ要素宣言並びはただ1つのデータ要素宣言でなければなりません。

PROTECTED 属性は、モジュールの宣言部にだけ指定できます。

PROTECTED 属性は、共通ブロック中にない名前付き変数に対してだけ指定できます。

PROTECTED 属性を指定したとき、EXTERNAL 属性、INTRINSIC 属性、およびPARAMETER 属性は、指定してはなりません。

PROTECTED 属性をもつポインタでない実体が参照結合で参照された場合、その実体は、変数を定義または未定義を行う文およびポインタ代入文中のデータ指示先に現れてはなりません。

PROTECTED 属性をもつポインタである実体が参照結合で参照された場合、その実体は、次のいずれに現れてはなりません。

- NULLIFY 文中のポインタ実体
- ポインタ代入文中のデータポインタ実体
- ALLOCATE 文中またはDEALLOCATE 文中の割付け実体

- 仮引数にINTENT(OUT) 属性またはINTENT(INOUT) 属性をもつポインタがある手続を引用するとき、その仮引数に結合する実引数

初期値指定をもつ実体は、共通ブロック中の実体およびPARAMETER 属性をもつ実体を除いて、暗黙的にSAVE 属性をもちます。

内部関数、モジュール関数、および引用仕様本体の関数は、長さが'*'の文字型であってはなりません。

EXTERNAL 属性とPOINTER 属性の両方をもつ手続は、手続ポインタです。

CLASS 指定子は、多相的実体を宣言するのに使われます。CLASS 指定子が型名を含むとき、それが多相的実体の宣言時の型になります。

CLASS(*) 指定子で宣言された実体は、無制限多相的です。無制限多相的データ要素は、宣言時の型をもちません。すべての他の要素と宣言時の型が異なるとみなされます。

CLASS によって宣言された要素は、仮引数であるか、ALLOCATABLE 属性またはPOINTER 属性をもっていなければならず、VALUE 属性をもってはなりません。

CONTIGUOUS属性は、ポインタ配列、形状引継ぎ配列、または次元引継ぎ仮引数だけに指定することができます。

型宣言文の例:

```
integer :: a, b, c              ! a、b、およびc は、整数型のスカラ実体です
real , dimension (2,4) :: d     ! d は、(2,4) の実数型配列です
integer :: e = 2                ! e は、初期値2をもつ変数です
```

2.4 文関数定義文(廃止予定事項)

文関数は、1つの文によって定義される関数です。

文関数定義文は以下の形式です。

function-name ([*dummy-arg-name-list*]) = *scalar-expr*

function-name は、文関数名です。

dummy-arg-name-list は、コンマで区切った仮引数名の並びです。

scalar-expr は、スカラ式です。

scalar-expr の一次子は、定数表現、名前付き定数、変数引用、関数引用、仮手続関数引用、および組込み演算だけで構成されなければなりません。スカラ式中に文関数引用を書く場合には、その文関数は、その有効域内で前もって定義されていなければならず、定義しようとする文関数の名前を引用することはできません。

スカラ式中のそれぞれの変数引用は、その文関数の仮引数の引用であってもかまいませんし、その文関数定義文を含む有効域から参照可能な変数の引用であってもかまいません。

仮引数の有効範囲は、文関数定義文内です。

文関数は、手続の引数として使用することはできません。

文関数引用の値は、仮引数に対応した実引数の値を用いて式を評価したものとし、必要であれば関数の宣言した型および型属性に合わせて結果が変換されます。

文関数定義文の例:

```
mean(i,j) = (i+j)/2
n = mean(2,4)          ! n には、3が代入されます
```

2.5 ABORTサービスサブルーチン

機能説明

標準エラー出力ファイルにメッセージを出力し、プログラムの実行を異常終了します。

形式

```
CALL ABORT
```

使用例

```
      use service_routines,only:abort
      open(10,file='x.dat',err=10)
      call sub
      stop 'program main end'
   10 call abort                          ! ABORT の呼出し
      end
      subroutine sub
      write(6,fmt="(1x,a)") "abort test"
      end
```

2.6 ABS組込み関数

ABS 関数は、絶対値を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
ABS	-----	1	1バイトの整数型	1バイトの整数型
	I2ABS		2バイトの整数型	2バイトの整数型
	IIABS		2バイトの整数型	2バイトの整数型
	IABS		4バイトの整数型	4バイトの整数型
	JIABS		4バイトの整数型	4バイトの整数型
	-----		8バイトの整数型	8バイトの整数型
	-----		実数型または複素数型	実数型または複素数型
	ABS		単精度実数型	単精度実数型
	DABS		倍精度実数型	倍精度実数型
	QABS		4倍精度実数型	4倍精度実数型
	CABS		単精度複素数型	単精度実数型
	CDABS		倍精度複素数型	倍精度実数型
	CQABS		4倍精度複素数型	4倍精度実数型

result = ABS (*A*)

A

整数型、実数型または複素数型でなければなりません。

result

A が整数型または実数型の場合は*A* と同じ型です。*A* が複素数型の場合は実数型です。

機能説明

ABS、I2ABS、IIABS、IABS、JIABS、DABS、QABS、CABS、CDABS、およびCQABSは、絶対値を求める際に使用します。

A が整数型および実数型の場合、結果の値は |*A*| となります。*A* が複素数型の場合、結果の値は SQRT(REAL(*A*)² + IMAG(*A*)²) となります。

総称名ABS は、すべての整数型、実数型および複素数型の引数に使用することができます。

それぞれの関数の結果の型は、*A* が整数型および実数型の場合、*A* と同じ型および種別型パラメタです。*A* が複素数型の場合、*A* と同じ種別型パラメタをもつ実数型です。

使用例

```
x = abs(-4.0)          ! x には4.0が代入されます
```

2.7 ACCESSサービス関数

機能説明

ファイルのアクセスモードと存在の有無を検査します。

形式

iy = ACCESS (*fname* , *mode*)

fname

基本文字型スカラ。検査するファイル名を指定します。

mode

基本文字型スカラ。検査する内容を示し、以下の値が指定できます。

```
' r '  :  Read 属性をテストします。
' w '  :  Write 属性をテストします。
' x '  :  eXecute 属性をテストします。
'   '  :  ファイルの存在をテストします。
```

関数結果

基本整数型スカラ。関数結果は、*mode* で指定された内容をすべて満たす場合、0を返却します。*mode* 内に上記 4文字以外が 1文字以上出現した場合、-1を返却します。それ以外の場合は、システムが返却するエラー値を返却します。

使用例

カレントディレクトリにtest.f90 が存在し、ファイル属性がユーザに対して Read 属性および Write 属性がある場合。

```
use service_routines,only:access
write(6,*) access("test.f90", "r")        ! 0
write(6,*) access("test.f90", "x")        ! システムのエラー値
write(6,*) access("test.f90", " ")        ! 0
write(6,*) access("test.f90", "rw ")      ! 0
end
```

2.8 ACHAR組込み関数

ACHAR 関数は、ASCII 大小順序における位置*I*の文字を返却します。

分類

要素別処理関数

形式

result = ACHAR (*I* [, *KIND*])

I

整数型でなければなりません。

KIND（省略可能）

スカラ整数定数式でなければなりません。

result

長さ1の文字型です。*KIND*が指定された場合、種別型パラメタは*KIND*の値に従います。*KIND*を省略したとき、種別型パラメタは基本文字型です。

機能説明

ACHAR 関数は、ASCII 大小順序における位置*I*の文字を返却します。

基本文字型で表現できる任意の文字C に対して、ACHAR(IACHAR(C)) は、値C をもちます。

0 <= *I* <= 255 でない場合、結果の値は不定となります。

関数の結果の型は、種別型パラメタ*KIND*が指定された場合、種別型パラメタ*KIND*の長さ1の文字型となります。*KIND*が省略された場合、関数の結果の型は、長さ1の基本文字型です。

使用例

```
c = achar(65)          ! c には'A' が代入されます
```

2.9 ACOS組込み関数

ACOS 関数は、ラジアン値を結果とする逆余弦を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
ACOS	-----	1	実数型 または複素数型	実数型 または複素数型
	ACOS		単精度実数型	単精度実数型
	DACOS		倍精度実数型	倍精度実数型
	QACOS		4倍精度実数型	4倍精度実数型

result = ACOS (*X*)

X

実数型、または複素数型でなければならず、実数型の場合、ABS(*X*) <= 1.0 でなければなりません。

result

*X*と同じ型です。

機能説明

ACOS は、実数型データまたは複素数型データの逆余弦を求めます。DACOS およびQACOS は、実数型データの逆余弦を求めます。

結果の値は、実数型の場合、ラジアン値で、その範囲は 0.0 <= ACOS(*X*) <= π となります。複素数型の場合、実数部はラジアンで、その範囲は 0 <= REAL(ACOS(*X*)) <= π となります。

総称名ACOS は、すべての実数型および複素数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
r = acos(.5)
```

2.10 ACOSD組込み関数

ACOSD 関数は、度数値を結果とする逆余弦を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
ACOSD	-----	1	実数型	実数型
	ACOSD		単精度実数型	単精度実数型
	DACOSD		倍精度実数型	倍精度実数型
	QACOSD		4倍精度実数型	4倍精度実数型

result = ACOSD (*X*)

X

実数型でなければならず、ABS(*X*) <= 1.0 でなければなりません。

result

*X*と同じ型です。

機能説明

ACOSD、DACOSD、およびQACOSD は、実数型データの逆余弦を求めます。

結果の値は度数値で ACOSD(*X*) = 180/π × ACOS(*X*) となり、その範囲は 0.0 <= ACOSD(*X*) <= 180.0 となります。

総称名ACOSD は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
r = acosd(.5)
```

2.11 ACOSH組込み関数

ACOSH 関数は、逆双曲線余弦を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
ACOSH	-----	1	実数型または 複素数型	実数型または 複素数型

result = ACOSH (*X*)

X

実数型または複素数型でなければならず、実数型の場合、*X*>= 1.0 でなければなりません。

result

*X*と同じ型です。

機能説明

ACOSH は、実数型データおよび複素数型データの逆双曲線余弦を求めます。

結果の値は、複素数の場合、虚部はラジアン値で、その範囲は 0 <= AIMAG(ACOSH(*X*)) <= π となります。

関数の結果の型は、*X*と同じです。

使用例

```
r = acosh(1.5)
```

2.12 ACOSQ組込み関数

ACOSQ 関数は、象限値を結果とする逆余弦を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
ACOSQ	-----	1	実数型	実数型
	ACOSQ		単精度実数型	単精度実数型
	DACOSQ		倍精度実数型	倍精度実数型
	QACOSQ		4倍精度実数型	4倍精度実数型

result = ACOSQ (*X*)

X

実数型でなければならず、ABS(*X*) <= 1.0 でなければなりません。

result

X と同じ型です。

機能説明

ACOSQ、DACOSQ、およびQACOSQ は、実数型データの逆余弦を求めます。

結果の値は象限値で ACOSQ(*X*) = 2/π × ACOS(*X*) となり、その範囲は 0.0 <= ACOSQ(*X*) <= 2.0 となります。

総称名ACOSQ は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*X* と同じです。

使用例

```
r = acosq(.5)
```

2.13 ADJUSTL組込み関数

ADJUSTL 関数は、文字列の先頭の空白を末尾に挿入する左詰めを行います。

分類

要素別処理関数

形式

result = ADJUSTL (*STRING*)

STRING

文字型でなければなりません。

result

STRING と同じ長さ、同じ種別型パラメタの文字型です。

機能説明

ADJUSTL は、文字列の先頭の空白を末尾に挿入する左詰めを行います。

STRING の先頭の空白を削除し、空白を末尾に挿入した文字列を返却します。

使用例

```
character(len=7)::adjusted
adjusted = adjustl(' string')      ! adjusted には'string ' が代入されます
```

2.14 ADJUSTR組込み関数

ADJUSTR 関数は、文字列の末尾の空白を先頭に挿入する右詰めを行います。

分類

要素別処理関数

形式

result = ADJUSTR (*STRING*)

STRING

文字型でなければなりません。

result

STRING と同じ長さ、同じ種別型パラメタの文字型です。

機能説明

ADJUSTR は、文字列の末尾の空白を先頭に挿入する右詰めを行います。

STRING の末尾の空白を削除し、空白を先頭に挿入した文字列を返却します。

使用例

```
character(len=7)::adjusted
adjusted = adjustr('string ')      ! adjusted には' string' が代入されます
```

2.15 AIMAG組込み関数

AIMAG 関数は、複素数の虚部を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
AIMAG、IMAG	-----	1	複素数型	実数型
	AIMAG		単精度複素数型	単精度実数型
	IMAG		単精度複素数型	単精度実数型
	DIMAG		倍精度複素数型	倍精度実数型
	QIMAG		4倍精度複素数型	4倍精度実数型

result = AIMAG (*Z*)
result = IMAG (*Z*)

Z

複素数型でなければなりません。

result

Z と同じ種別型パラメタをもつ実数型です。

機能説明

AIMAG、IMAG、DIMAG、およびQIMAG は、複素数の虚部を返却します。

総称名AIMAG またはIMAG は、すべての複素数型の引数に使用することができます。

それぞれの関数の結果の型は、*Z* と同じ種別型パラメタをもつ実数型です。

使用例

```
r = aimag((-4.0,5.0))       ! r には5.0が代入されます
```

2.16 AINT組込み関数

AINT 関数は、実数型データの小数点部分の切り捨てを行います。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
AINT	-----	1 または 2	実数型 [, 整数型]	実数型
	AINT	1	単精度実数型	単精度実数型
	DINT		倍精度実数型	倍精度実数型
	QINT		4倍精度実数型	4倍精度実数型

result = AINT (*A* [, *KIND*])

A

実数型でなければなりません。

KIND（省略可能）

スカラ整数定数式でなければなりません。

result

実数型です。*KIND* が指定された場合、種別型パラメタは*KIND* の指定に従います。

KIND が省略された場合、種別型パラメタは*A* と同じになります。

機能説明

AINT、DINT、およびQINT は、実数型データの小数点部分の切り捨てを行います。

ABS(*A*) < 1.0 の場合、結果の値は0.0となります。それ以外の場合は、*A* の絶対値以下で最大の整数値に*A* の符号を付けた値となります。

総称名AINT は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、種別型パラメタ*KIND* が指定された場合、種別型パラメタ*KIND* の実数型となります。*KIND* が省略された場合、関数の結果の型は*A* と同じです。

使用例

```
r = aint(-7.32,4)       ! r には種別4の値 -7.0が代入されます
```

2.17 ALARMサービス関数

機能説明

特定時間が経過した後に、サブルーチンを実行させます。

形式

iy = ALARM (*time* , *sub*)

time

基本整数型スカラ。経過させる時間を秒単位で指定します。

0を指定した場合、アラームはオフになり、サブルーチン*sub*は呼び出されません。

sub

*time*秒後に実行するサブルーチン名を指定します。

関数結果

基本整数型スカラ。アラームの残り時間の秒数が返却されます。

使用例

```
use service_routines,only:alarm
integer :: iy
external :: sub
iy = alarm(2,sub)              ! 2秒後にサブルーチンsub を実行します
  ...
end
```

2.18 ALL組込み関数

ALL 関数は、*MASK*の第*DIM*次元の要素の値がすべて真であるかどうかを判定します。

分類

変形関数

形式

result = ALL (*MASK* [,*DIM*])

MASK

論理型です。スカラであってはなりません。

DIM (省略可能)

整数型スカラであって、1 <= *DIM*<= n の範囲の値でなければなりません。ここで、n は*MASK*の次元数とします。対応する実引数は、省略可能な仮引数であってはなりません。

result

*MASK*と同じ型および同じ種別型パラメタです。*DIM*が省略されている、または*MASK*が1次元配列の場合は、スカラとなります。それ以外の場合は、(n-1)次元の配列となります。ここで、n は*MASK*の次元数とします。形状は、*MASK*の形状を(d_1, d_2 ..., d_n) としたとき、(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_n) とします。

機能説明

ALL は、*MASK*の第*DIM*次元がすべて真であるかどうかを判定します。

－ *DIM*を省略している場合

*MASK*の全要素が真である場合、または*MASK*の大きさが0であるとき結果の値は真とし、*MASK*の要素に1つ以上の偽がある場合は偽とします。

－ *DIM*を指定している場合

*MASK*が1次元の場合、結果の値はALL(*MASK*) とします。
*MASK*が2次元以上の場合、結果の要素 (S_1, S_2, ..., S_{DIM-1}, S_{DIM+1}, ..., S_n) は、ALL(*MASK*(S_1, S_2, ..., S_{DIM-1}, : , S_{DIM+1}, ..., S_n)) となります。

関数の結果の型は、*MASK*と同じ種別型パラメタをもつ論理型です。

使用例

```
integer, dimension (2,3) :: a, b
logical, dimension (2) :: c
logical, dimension (3) :: d
```

```
logical :: e
a = reshape((/1,2,3,4,5,6/), (/2,3/))          ! a は ┌       ┐
                                               !      | 1 3 5 |
                                               !      | 2 4 6 |
                                               !      └       ┘
b = reshape((/1,2,3,5,6,4/), (/2,3/))          ! b は ┌       ┐
                                               !      | 1 3 6 |
                                               !      | 2 5 4 |
                                               !      └       ┘
e = all(a==b)                 ! e には偽が代入されます
d = all(a==b, 1)              ! d には(/.true.,.false.,.false./) が代入されます
c = all(a==b, 2)              ! c には(/.false.,.false./) が代入されます
```

2.19 ALLOCATABLE文

ALLOCATABLE 文は、割付け変数を宣言します。配列を宣言する場合には、形状無指定配列でなければならず、ALLOCATE 文を実行して領域を割り付けた時に形状が決まります。

ALLOCATABLE 文は、以下の形式です。

```
ALLOCATABLE [ :: ] allocatable-decl-list
```

allocatable-decl-list は、割付け宣言並びです。*allocatable-decl* は、以下の形式です。

```
object-name [ ( deferred-shape-spec-list ) ]  [ left-square-bracket  coarray-spec  right-square-bracket ]
```

object-name は実体名です。

deferred-shape-spec-list は無指定上下限の並びです。

deferred-shape-spec は以下の形式(コロンのみ)です。

```
:
```

coarray-spec は、共配列指定です。詳細については"1.17.1 共配列指定"を参照してください。

left-square-bracket は、左角括弧です。左角括弧は、[です。

right-square-bracket は、右角括弧です。右角括弧は、] です。

配列名のDIMENSION 属性が有効域内の他の場所で指定されている場合、その配列形状指定は、無指定上下限並びでなければなりません。

ALLOCATABLE 文の例:

```
integer :: a,b,c(:,:,:)                    ! c の配列形状指定
dimension :: b(:,:)                        ! b の配列形状指定
allocatable a(:),b,c                       ! a、b、およびc は割付け配列
allocate (a(2),b(3,-1:1),c(10,10,10))      ! 形状を指定し領域を割り付けます
  ...
deallocate (a,b,c)                         ! 領域を解放します。
```

2.20 ALLOCATE文

ALLOCATE 文は、ポインタ指示先および割付け変数を動的に生成します。

ALLOCATE 文は、以下の形式です。

```
ALLOCATE ( [ type-spec:: ]allocation-list [ , alloc-opt-list ] )
```

type-spec は、型指定子です。型指定子には、組込み型指定子および派生型指定子があります。

組込み型指定子については、“2.3 型宣言文”を参照してください。また、派生型指定子については、“1.5.11.8 派生型指定子”を参照してください。

allocation-list は、コンマで区切られた割付け指定の並びです。*allocation* は、以下の形式です。

allocate-object [(*allocate-shape-spec-list*)] ■
■ [*left-square-bracket allocate-coarray-spec right-square-bracket*]

allocate-object は、割付け実体であり、以下の形式です。

variable-name または
structure-component

variable-name は、変数名です。

structure-component は、構造体成分です。

割付け実体は、ポインタまたは割付け変数でなければなりません。

割付け実体は、長さ0の文字型であってもかまいません。

割付け実体は、共添字付き実体であってはなりません。

allocate-shape-spec-list は、割付け上下限の並びです。割付け上下限の数は、割付け実体の次元数と同じでなければなりません。SOURCE 指定子またはMOLD 指定子に配列を指定した場合、省略することができます。

allocate-shape-spec は以下の形式です。

[*allocate-lower-bound* :] *allocate-upper-bound*

allocate-lower-bound は割付け下限であり、スカラ整数式でなければなりません。

allocate-upper-bound は割付け上限であり、スカラ整数式でなければなりません。

allocate-coarray-spec は、割付け共配列指定です。割付け共配列指定は、次の形式です。

[*allocate-coshape-spec-list* ,] [*allocate-lower-bound* :] *

allocate-coshape-spec は、割付け共形状指定です。割付け共形状指定は、次の形式です。

[*allocate-lower-bound* :] *allocate-upper-bound*

割付け共配列指定が現れた場合、割付け実体は共配列でなければなりません。

割付け共形状指定の数は、割付け実体の共次元数より、1少なくなければなりません。

割付け実体が共配列なら、型指定子にC_PTR または C_FUNPTR型を指定してはなりません。

割付け実体が共配列なら、SOURCE指定子の定数式の宣言時の型は、C_PTR、C_FUNPTR、LOCK_TYPE または成分がLOCK_TYPE型であってはなりません。

left-square-bracket は、左角括弧です。左角括弧は、[です。

right-square-bracket は、右角括弧です。右角括弧は、] です。

alloc-opt-list は割付け指定選択子並びです。*alloc-opt* は、以下の形式です。

STAT = *stat-variable* または
ERRMSG = *errmsg-variable* または
SOURCE = *source-expr* または
MOLD = *source-expr*

stat-variable は状態変数であり、スカラ整変数でなければなりません。

errmsg-variable は誤り通報変数であり、スカラ基本文字変数でなければなりません。

source-expr は初期値指定式であり、式です。

状態変数、初期値指定式、誤り通報変数はいずれもそれらが現れるALLOCATE 文中で割り付けてはなりません。更に、同じALLOCATE 文に現れる割付け実体中の値、上下限、長さ型パラメタ、割付け状態、結合状態のいずれにも依存してはなりません。

STAT 指定子のあるALLOCATE 文の実行が正常に終了した場合、状態変数には値0が設定されます。ALLOCATE 文の実行中に誤り条件が発生した場合、状態変数には実行時の診断メッセージの番号が設定されます。ラージページ機能を使用する場合の注意事項については、“Fortran使用手引書”の“ALLOCATE／DEALLOCATE文”を参照してください。割付けに成功した割付け実体は、“割り付けられている”という状態または“結合している”というポインタ結合状態になり、割付けに失敗した割付け実体は、直前の割付け状態またはポインタ結合状態を保ちます。

誤り条件がALLOCATE 文の実行中に発生したとき、誤り通報変数に状態が代入されます。誤り条件が発生しなかったときは、誤り通報変数の値は変更されません。

STAT 指定子のないALLOCATE 文の実行中に誤り条件が発生した場合、その実行可能プログラムの実行は終了します。

配列に対して割付け上下限の並びを指定したALLOCATE 文を実行すると、割付け上限および割付け下限の値がその配列の上下限になります。上下限の式中のどれかの変数が後に再確定されたり不定にされても、その配列の上下限には影響しません。割付け下限を省略したとき、暗黙値は1とします。上限が下限より小さい場合、その次元の寸法および配列の大きさは、0です。

配列に対して割付け上下限の並びを省略したALLOCATE 文を実行すると、初期値指定式の上下限がその配列の上下限になります。

共配列に対してALLOCATE 文を実行すると、共下限式と共上限式の値が、その共配列の共上下限になります。共上下限式中のどれかの変数が後に再確定されたり不定にされたりしても、共上下限表現には影響がありません。もし共下限を省略すると、省略値は1です。共上限は共下限より小さい値であってはなりません。

割付け実体の長さ型パラメタの値が指定され、それがSOURCE 指定子またはMOLD 指定子の初期値指定式に対応する型パラメタの値と異なる場合、誤り条件が発生します。SOURCE 指定子が指定され割付けが成功した場合、割付け実体の値は、初期値指定式のものとなります。

割り付けられている割付け変数を更に割り付けようとした場合、そのALLOCATE 文で誤り条件が発生します。

割付け変数が割り付けられているかどうかは、組込み関数ALLOCATED（“2.21 ALLOCATED組込み関数”参照）によって知ることができます。

指示先と結合しているポインタを割り付けたとき、ALLOCATE 文に指定した配列上下限によって要求されるポインタ指示先が新たに生成され、ポインタはその指示先と結合します。

ポインタの結合状態は、組込み関数ASSOCIATED（“2.30 ASSOCIATED組込み関数”参照）によって知ることができます。

関数結果がポインタである関数の実行の開始時には、そのポインタの結合状態は不定です。そのような関数の戻りの前には、そのポインタに指示先を結合するかまたはそのポインタの結合状態を空状態にしなければなりません。

それぞれの割付け実体は、データポインタまたは割付け変数でなければなりません。

文中で割付け実体が無指定型パラメタをもつ場合、型指定子、SOURCE 指定子、またはMOLD 指定子が現れなければなりません。

型指定子がある場合、その指定する型は、それぞれの割付け実体と型が適合していなければなりません。

割付け実体が無制限多相的または抽象型である場合、型指定子、SOURCE 指定子、またはMOLD 指定子が現れなければなりません。

それぞれの割付け実体の対応する型パラメタが引き継がれる仮引数である場合に限り、型指定子中の型パラメタ値は“*”でなければなりません。

型指定子がある場合、それぞれの割付け実体の種別型パラメタ値は、その型指定子で対応する型パラメタ値と同じでなければなりません。

割付け実体が配列で、配列の定数式の指定がない場合、割付け上下限並びを指定しなければなりません。割付け実体がスカラの場合、割付け上下限並びを指定してはなりません。

割付け実体が配列で割付け上下限並びを指定し、

- SOURCE 指定子が現れる場合、初期値指定式は割付け上下限並びと形状適合しなければなりません。
- MOLD 指定子が現れる場合、初期値指定式は割付け実体と次元数が異なってもかまいません。

割付け実体が配列で割付け上下限並びを省略した場合、SOURCE 指定子またはMOLD 指定子の初期値指定式は、割付け実体と同じ次元数でなければなりません。

割付け上下限並び中の割付け上下限の個数は、割付け実体の次元数と同じでなければなりません。

1つの割付け選択子並び中に、各割付け選択子を2回以上指定してはなりません。

SOURCE 指定子がある場合、型指定子またはMOLD 指定子があってはなりません。また、初期値指定式と型が適合していなければなりません。

MOLD 指定子がある場合、型指定子またはSOURCE 指定子があってはなりません。また、割付け実体は初期値指定式と型が適合していなければなりません。

MOLD 指定子の初期値指定式は、型パラメタをもつ派生型であってはなりません。

対応する割付け実体と初期値指定式の種別型パラメタは、同じ値でなければなりません。

割付け実体、その上下限パラメタおよび型パラメタは、状態変数の値にも誤り通報変数の値にも依存してはなりません。更に、同じALLOCATE 文に現れる割付け実体中の値、上下限、長さ型パラメタ、割付け状態、結合状態のいずれにも依存してはなりません。

型指定子がある場合、多相的実体の割付けをすると、指定された実行時の型をもつ実体を割り付けます。初期値指定式がある場合、実行時の型および型パラメタがその式と同じである実体を割り付けます。それ以外の場合、宣言時の型と同じ実行時の型をもつ実体を割り付けます。

型指定子を伴うALLOCATE 文が実行されると、型指定子中の型パラメタ値で、型パラメタが指定されます。型パラメタとして指定された値が、割付け実体の宣言中で指定された値と異なる場合、誤り条件が発生します。

ALLOCATE 文における型指定子中の型パラメタ値が“*”の場合、引継ぎ型パラメタの現在の値を指します。型パラメタ値が式の場合、その式中の変数が後に再確定されても不定になっても、型パラメタ値に影響しません。

初期値指定式がポインタの場合、結合していなければなりません。初期値指定式が割付け実体の場合、割り付けられていなければなりません。

MOLD 指定子の初期値指定式が変数の場合、この変数値は確定していなくてもかまいません。MOLD 指定子が指定されたとき割付けが成功しても、割付け実体の値は不定です。

割付け実体に共配列を指定する場合、動的な型と対応する型パラメタの値は、あらゆる像で同じでなければなりません。上下限および共上下限の値は、すべての像で同じでなければなりません。共配列が仮引数なら、すべての実引数は、同じ共配列でなければなりません。

割付け実体が共配列であるALLOCATE文を実行したとき、すべての像が暗黙のうちに同期します。各々の像上で、すべての像が文を実行するまで、この文に続くセグメントの実行は遅延します。

ALLOCATE 文の例:

```
integer, pointer :: n
integer, allocatable, dimension (:,:) :: k
integer, allocatable, dimension (:) , codimension [:,:] :: ca ! 共配列
allocate (k(4,-2:3))                  ! k の形状を確定し、領域を割り付けます
allocate (n)                          ! n は生成された実体と結合します
allocate (ca(3) [ 2,* ] )             ! 共配列ca の形状を確定し、領域を割り付けます
```

2.21 ALLOCATED組込み関数

ALLOCATED 関数は、割付け変数が割り付けられているかどうかを判定します。

分類

問合せ関数

形式

result = ALLOCATED (*ALLOCATABLE*)

ALLOCATABLE

割付け変数でなければなりません。

result

基本論理型スカラです。

機能説明

ALLOCATED は、割付け変数が割り付けられているかどうかを判定します。

割付け変数が割り付けられているとき、結果の値は真とし、割り付けられていないときは偽とします。

関数の結果の型は、基本論理型です。

使用例

```
integer, allocatable :: i(:,:)
logical :: l
allocate (i(2,3))
l = allocated (i)                ! l には真が代入されます
```

2.22 ANINT組込み関数

ANINT 関数は、実数型データの四捨五入を行います。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
ANINT	-----	1または2	実数型 [, 整数型]	実数型
	ANINT	1	単精度実数型	単精度実数型
	DNINT		倍精度実数型	倍精度実数型
	QNINT		4倍精度実数型	4倍精度実数型

result = ANINT (*A* [, *KIND*])

A

実数型でなければなりません。

KIND（省略可能）

スカラ整数定数式でなければなりません。

result

実数型です。*KIND* が指定された場合、種別型パラメタは*KIND* の指定に従います。

KIND が省略された場合、種別型パラメタは*A* と同じになります。

機能説明

ANINT、DNINT、およびQNINT は、実数型データの四捨五入を行います。

A が0.0または正の場合、結果の値はAINT(*A*+0.5) となります。*A* が負の場合は、AINT(*A*-0.5) となります。

総称名ANINT は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、種別型パラメタ*KIND* が指定された場合、種別型パラメタ*KIND* の実数型となります。*KIND* が省略された場合、関数の結果の型は*A* と同じです。

使用例

```
x = anint (7.73)                 ! x には8.0が代入されます
```

2.23 ANY組込み関数

ANY 関数は、*MASK* の第*DIM* 次元の要素に1つでも真の値があるかどうかを判定します。

分類

変形関数

形式

result = ANY (*MASK* [, *DIM*])

MASK

論理型です。スカラであってはなりません。

DIM (省略可能)

整数型スカラであって、1 <= *DIM*<= n の範囲の値でなければなりません。ここで、n は*MASK*の次元数とします。対応する実引数は、省略可能な仮引数であってはなりません。

result

*MASK*と同じ型および同じ種別型パラメタです。*DIM*が省略されている、または*MASK*が1次元配列の場合は、スカラとなります。それ以外の場合は、(n-1)次元の配列となります。ここで、n は*MASK*の次元数とします。形状は、*MASK*の形状を (d_1, d_2, ..., d_n) としたとき、(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_n) とします。

機能説明

ANY は、*MASK*の第*DIM*次元の要素の値に真があるかどうかを判定します。

－ *DIM*を省略している場合

*MASK*の1要素が真である場合、結果の値は真とし、*MASK*のどの要素も真でない場合、または*MASK*の大きさが0である場合は、偽とします。

－ *DIM*を指定している場合

*MASK*が1次元の場合、結果の値はANY(*MASK*) とします。
*MASK*が2次元以上の場合、結果の要素 (S_1, S_2, ..., S_{DIM-1} , S_{DIM+1} ..., S_n) は、ANY(*MASK*(S_1, S_2, ..., S_{DIM-1}, :, S_{DIM+1}, ..., S_n)) となります。

関数の結果の型は、*MASK*と同じ種別型パラメタをもつ論理型です。

使用例

```
integer, dimension (2,3) :: a, b
logical, dimension (2) :: c
logical, dimension (3) :: d
logical :: e
a = reshape((/1,2,3,4,5,6/), (/2,3/))                ! a は┌     ┐
                                                     !     | 1 3 5 |
                                                     !     | 2 4 6 |
                                                     !     └     ┘
b = reshape((/1,2,3,5,6,4/), (/2,3/))                ! b は┌     ┐
                                                     !     | 1 3 6 |
                                                     !     | 2 5 4 |
                                                     !     └     ┘
e = any (a==b)                  ! e には真が代入されます
d = any (a==b, 1)               ! d には(/.true.,.true.,.false./) が代入されます
c = any (a==b, 2)               ! c には(/.true.,.true./) が代入されます
```

2.24 ASIN組込み関数

ASIN 関数は、ラジアン値を結果とする逆正弦を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
ASIN	-----	1	実数型 または複素数型	実数型 または複素数型
	ASIN		単精度実数型	単精度実数型
	DASIN		倍精度実数型	倍精度実数型
	QASIN		4倍精度実数型	4倍精度実数型

result = ASIN (*X*)

X

実数型または複素数型でなければなりません。実数型の場合、ABS(*X*) <= 1.0 でなければなりません。

result

*X*と同じ型です。

機能説明

ASIN は、実数型データまたは複素数型データの逆正弦を求めます。DASIN およびQASIN は、実数型データの逆正弦を求めます。

結果の値は、実数型の場合、ラジアン値で、その範囲は ABS(ASIN(*X*)) <= π/2 となります。

複素数型の場合、実数部はラジアン値で、その範囲は ABS(REAL(ASIN(*X*))) <= π/2 となります。

総称名ASIN は、すべての実数型および複素数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
r = asin(.5)
```

2.25 ASIND組込み関数

ASIND 関数は、度数値を結果とする逆正弦を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
ASIND	-----	1	実数型	実数型
	ASIND		単精度実数型	単精度実数型
	DASIND		倍精度実数型	倍精度実数型
	QASIND		4倍精度実数型	4倍精度実数型

result = ASIND (*X*)

X

実数型でなければなりません。ABS(*X*) <= 1.0 でなければなりません。

result

*X*と同じ型です。

機能説明

ASIND、DASIND、およびQASIND は、実数型データの逆正弦を求めます。

結果の値は度数値で ASIND(*X*) = 180/π × ASIN(*X*) となり、その範囲は ABS(ASIND(*X*)) <= 90.0 となります。

総称名ASIND は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
r = asind(.5)
```

2.26 ASINH組込み関数

ASINH 関数は、逆双曲線正弦を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
ASINH	-----	1	実数型 または複素数型	実数型 または複素数型

result = ASINH (*X*)

X

実数型または複素数型でなければなりません。

result

X と同じ型です。

機能説明

ASINH は、実数型データおよび複素数型データの逆双曲線正弦を求めます。

結果の値は、複素数型の場合、虚部はラジアン値で、その範囲は ABS(AIMAG(ASINH(*X*))) <= π/2 となります。

関数の結果の型は、*X* と同じです。

使用例

```
r = asinh(0.1)
```

2.27 ASINQ組込み関数

ASINQ 関数は、象限値を結果とする逆正弦を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
ASINQ	-----	1	実数型	実数型
	ASINQ		単精度実数型	単精度実数型
	DASINQ		倍精度実数型	倍精度実数型
	QASINQ		4倍精度実数型	4倍精度実数型

result = ASINQ (*X*)

X

実数型でなければなりません。ABS(*X*) <= 1.0 でなければなりません。

result

X と同じ型です。

機能説明

ASINQ、DASINQ、およびQASINQ は、実数型データの逆正弦を求めます。

結果の値は象限値で ASINQ(*X*) = 2/π × ASIN(*X*) となり、その範囲は ABS(ASINQ(*X*)) <= 1.0 となります。

総称名ASINQ は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*X* と同じです。

使用例

```
r = asinq(.5)
```

2.28 ASSIGN文（廃止事項）

ASSIGN 文は、文番号をスカラ整変数に割り当てます。

ASSIGN 文は、以下の形式です。

```
ASSIGN label TO scalar-int-variable
```

label は文番号であり、そのASSIGN 文と同じ有効域内にある飛び先文またはFORMAT 文の文番号でなければなりません。

scalar-int-variable は、スカラ整変数であり、基本整数型の名前付き変数でなければなりません。

ASSIGN 文を実行すると、その文番号がその整変数に割り当てられます。文番号値で確定になっている間、その整変数は、割当て形GO TO 文中で、または入出力文の書式識別子としてだけ、引用することができます。

ASSIGN 文の例:

```
   assign 10 to i
   goto i
10 continue
```

2.29 ASSOCIATE構文

ASSOCIATE 構文は、ブロックの実行中に、名前付き要素と式または変数を結合します。これらの名前付き構文内要素は、結合要素といい、その名前を結合名といいます。

2.29.1 ASSOCIATE 構文の形

ASSOCIATE 構文は以下の形式です。

```
[ associate-construct-name: ] ASSOCIATE ( association-list )
    block
END ASSOCIATE [ associate-construct-name ]
```

associate-construct-name は、ASSOCIATE 構文名です。

association-list は結合並びです。以下の形式です。

```
association => selector
```

association は結合名です。

selector は選択子です。以下の形式です。

```
expr            または
variable
```

expr は、式です。*variable* は、変数です。変数は、共添字付き実体であってはなりません。

選択子が変数でないか、またはベクトル添字をもつ変数である場合、結合名は変数確定の文脈中に現れてはなりません。
結合名は、それが属するASSOCIATE 文中の他の結合名と同じであってはなりません。
ASSOCIATE 文にASSOCIATE 構文名を指定する場合は、対応するEND ASSOCIATE 文にも同じASSOCIATE 構文名を指定しなければなりません。

ASSOCIATE 文にASSOCIATE 構文名を指定しない場合は、対応するEND ASSOCIATE 文にASSOCIATE 構文名を指定してはなりません。

式の結合例:

```
real :: a,b
  ...
associate( c => sin(b) )
print *, a+c
end associate
```

2.29.1.1 ASSOCIATE構文の実行

ASSOCIATE 構文を実行すると、その中のASSOCIATE 文が実行されてからブロックが実行されます。ブロックの実行中は、各結合名は、対応する選択子と結合した実体を指します。結合要素は選択子に対して宣言時の型および型パラメタを引き継ぎます。選択子が多相的であるとき結合要素は多相的になります。
END ASSOCIATE 文への飛び越しができるのは、そのASSOCIATE 構文の内部からだけです。

2.29.1.2 結合名の属性

SELECT TYPE 構文とASSOCIATE 構文の各結合要素は、対応する選択子と同じ次元数と共次元数をもちます。各次元の下限は、組込み関数LBOUNDを選択子の対応する次元数に適用した結果です。各次元の上限は、その下限と寸法の和から1を引いた値です。共次元の共上下限は、対応する選択子と同じです。
選択子が変数であって、その変数がASYNCHRONOUS 属性、TARGET 属性またはVOLATILE 属性をもつときだけ、結合要素は同じ属性をもちます。
選択子が変数であって、その変数がPOINTER 属性をもつとき、結合要素はTARGET 属性をもちます。
選択子の変数がCONTIGUOUS（“2.88 CONTIGUOUS文”参照）であるときだけ、結合要素はCONTIGUOUS になります。
結合要素が多相的であるとき、その選択子の実行時の型および型パラメタ値を引き継ぎます。
選択子がOPTIONAL 属性をもつとき、その選択子は実在しなければなりません。
選択子が変数確定の文脈に現れることができないか、またはベクトル添字をもつ配列である場合、結合名は変数確定の文脈に現れてはなりません。

2.30 ASSOCIATED組込み関数

ASSOCIATED 関数は、ポインタが指示先と結合しているかどうかを判定します。

分類

問合せ関数

形式

result = ASSOCIATED (*POINTER* [, *TARGET*])

POINTER

ポインタでなければなりません。ポインタの結合状態が不定であってはなりません。

TARGET（省略可能）

ポインタまたは指示先でなければなりません。*POINTER*と同じ型、種別型パラメタ、および次元数でなければなりません。ポインタの場合は、ポインタの結合状態が不定であってはなりません。

result

基本論理型スカラです。

機能説明

ASSOCIATED は、ポインタの結合状態を調査します。

*POINTER*が指示先と結合していない、または*TARGET*に結合していないポインタが指定されている場合、結果の値は偽とします。それ以外の場合は、以下のとおりとします。

－ *TARGET* が省略されている場合

POINTER が指示先と結合している場合は結果の値は真とし、結合していない場合は偽とします。

－ *TARGET* が指定され、手続の場合

POINTER が *TARGET* と結合しているならば、結果の値は真とします。

－ *TARGET* が指定され、手続ポインタの場合

POINTER と結合している指示先が *TARGET* と同じ手続と結合しているならば、結果は真です。
POINTER または *TARGET* が空状態の場合、結果は偽とします。

－ *TARGET* が指定されていてスカラの指示先の場合

TARGET が大きさゼロでなく、*POINTER* と結合している指示先が *TARGET* と同じ領域を指示しているならば、結果は真とし、それ以外の場合は偽とします。
POINTER が空状態の場合、結果の値は偽とします。

－ *TARGET* が指定されていて配列の指示先の場合

POINTER と *TARGET* がすべての要素が同じ領域を指示していて、同じ形状で、大きさゼロでなく、配列要素順序で同じ記憶単位の場合、結果の値は真とし、それ以外の場合は偽とします。
POINTER が空状態の場合、結果の値は偽とします。

－ *TARGET* が指定されていてスカラのポインタの場合

POINTER と結合している指示先と *TARGET* と結合している指示先が、大きさゼロでなく、同じ記憶単位を指示しているとき、結果の値は真とし、それ以外の場合は偽とします。
POINTER または *TARGET* が空状態の場合、結果の値は偽とします。

－ *TARGET* が指定されていて配列のポインタの場合

POINTER と結合している指示先と *TARGET* と結合している指示先とが、同じ形状で、大きさゼロでなく、配列要素順序で同じ記憶単位の場合、結果の値は真とし、それ以外の場合は偽とします。
POINTER または *TARGET* が空状態の場合、結果の値は偽とします。

関数の結果の型は、基本論理型です。

使用例

```
real, pointer :: a, b, e
real, target :: c, f
logical :: l
a => c
b => c
e => f
l = associated (a)            ! l には真が代入されます
l = associated (a,c)          ! l には真が代入されます
l = associated (a,b)          ! l には真が代入されます
l = associated (a,f)          ! l には偽が代入されます
l = associated (a,e)          ! l には偽が代入されます
```

2.31 ASYNCHRONOUS文

ASYNCHRONOUS 文は以下の形式です。

```
ASYNCHRONOUS [ :: ] object-name-list
```

object-name-list は、実体名並びです。

この文は、並び中の実体に対して、ASYNCHRONOUS 属性を指定します。

ASYNCHRONOUS 属性は変数が、非同期入出力で扱うかどうかを指定します。

次の条件がすべて満たされる場合、変数は、有効域内でASYNCHRONOUS 属性をもたなければなりません。

・ 変数が、有効域内で、実行文または宣言式中に現れている場合。かつ、

- 変数が非同期入出力で関連づけられ、有効域内の任意の文が実行中である場合。

実体がASYNCHRONOUS 属性をもつ場合、その部分実体は、すべてASYNCHRONOUS 属性をもちます。

非同期入出力文で変数を次のいずれかに指定した場合、その実体には、そのデータ転送文の有効域内で暗黙的にASYNCHRONOUS 属性が与えられます。この属性は、明示的な宣言で確定することができます。

- 入出力項目並び
- 変数群要素
- SIZE 指定子

参照結合しているモジュール内の要素がASYNCHRONOUS 属性をもたないときでも、局所有効域ではASYNCHRONOUS 属性をもつことができます。

ASYNCHRONOUS 文の例:

```
asynchronous:: a
```

2.32 ATAN組込み関数

ATAN 関数は、ラジアン値を結果とする逆正接を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
ATAN	-----	1	実数型 または複素数型	実数型 または複素数型
	ATAN		単精度実数型	単精度実数型
	DATAN		倍精度実数型	倍精度実数型
	QATAN		4倍精度実数型	4倍精度実数型
	-----	2	実数型 , 実数型	実数型

result = ATAN (*X*)　　または
result = ATAN (*Y* , *X*)

X

Y が指定された場合は、*X* は *Y* と同じ種別型パラメタをもつ実数型でなければなりません。
Y が指定されなかった場合、*X* は実数型または複素数型でなければなりません。

Y

実数型でなければなりません。

result

X と同じ型です。

機能説明

ATAN は、実数型データまたは複素数型データの逆正弦を求めます。DATAN およびQATAN は、実数型データの逆正接を求めます。引数は *Y* = 0.0 かつ *X* = 0.0 であってはなりません。

結果の値は、*Y* が指定された場合は、ATAN2 (*Y*, *X*) と同じになります。*Y* が指定されない場合は、実数部は、ラジアン値で、その範囲は ABS(ATAN(*X*)) <= π /2 となります。

総称名ATAN は、すべての実数型および複素数型の引数に使用することができます。

それぞれの関数の結果の型は、*X* と同じです。

使用例

```
a = atan(.5)
```

2.33 ATAN2組込み関数

ATAN2 関数は、ラジアン値を結果とする2引数逆正接を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
ATAN2	-----	2	実数型 , 実数型	実数型
	ATAN2		単精度実数型 , 単精度実数型	単精度実数型
	DATAN2		倍精度実数型 , 倍精度実数型	倍精度実数型
	QATAN2		4倍精度実数型 , 4倍精度実数型	4倍精度実数型

result = ATAN2(*Y* , *X*)

Y

実数型でなければなりません。

X

*Y*と同じ型および同じ種別型パラメタをもつ実数型でなければなりません。

result

*Y*と同じ型です。

機能説明

ATAN2、DATAN2、およびQATAN2 は、実数型データの逆正接を求めます。

引数は*Y* = 0.0 かつ*X* = 0.0 であってはなりません。

結果の値はラジアン値で *X* ≠ 0.0 のとき ATAN2(*Y*,*X*) = ATAN(*Y*/*X*)となり、*X* = 0.0 のときは*Y*の符号をもつ π /2 となります。結果の範囲は ABS(ATAN2(*Y*,*X*)) <= π となります。

総称名ATAN2 は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*Y*と同じです。

使用例

```
x = atan2 (1., 1.)
```

2.34 ATAN2D組込み関数

ATAN2D 関数は、度数値を結果とする 2引数逆正接を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
ATAN2D	-----	2	実数型 , 実数型	実数型

総称名	個別名	引数の数	引数の型	結果の型
	ATAN2D		単精度実数型 , 単精度実数型	単精度実数型
	DATAN2D		倍精度実数型 , 倍精度実数型	倍精度実数型
	QATAN2D		4倍精度実数型 , 4倍精度実数型	4倍精度実数型

result = ATAN2D (*Y* , *X*)

Y

実数型でなければなりません。

X

*Y*と同じ型および同じ型パラメタをもつ実数型でなければなりません。

result

*Y*と同じ型です。

機能説明

ATAN2D、DATAN2D、およびQATAN2D は、実数型データの逆正接を求めます。

引数は*Y* = 0.0 かつ*X* = 0.0 であってはなりません。

結果の値は度数値で *X* ≠ 0.0 のとき ATAN2D(*Y*,*X*) = ATAND (*Y*/*X*) = 180/ π ×ATAN(Y/X) となり、*X* = 0.0のときは*Y*の符号をもつ90.0となります。結果の範囲は ABS(ATAN2D(*Y*,*X*)) <= 180.0 となります。

総称名ATAN2D は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*Y*と同じです。

使用例

```
x = atan2d (1.,1.)
```

2.35 ATAN2Q組込み関数

ATAN2Q 関数は、象限値を結果とする2引数逆正接を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
ATAN2Q	-----	2	実数型 , 実数型	実数型
	ATAN2Q		単精度実数型 , 単精度実数型	単精度実数型
	DATAN2Q		倍精度実数型 , 倍精度実数型	倍精度実数型
	QATAN2Q		4倍精度実数型 , 4倍精度実数型	4倍精度実数型

result = ATAN2Q (*Y* , *X*)

Y

実数型でなければなりません。

X

*Y*と同じ型および同じ型パラメタをもつ実数型でなければなりません。

result

*Y*と同じ型です。

機能説明

ATAN2Q、DATAN2Q、およびQATAN2Q は、実数型データの逆正接を求めます。

引数は*Y* = 0.0 かつ*X* = 0.0 であってはなりません。

結果の値は度数値で *X* ≠ 0.0 のとき ATAN2Q(*Y*,*X*) = ATANQ(*Y*/*X*) = 2/ π × ATAN(*Y*/*X*) となり、*X* = 0.0のときは*Y*の符号をもつ1.0となります。結果の範囲は ABS(ATAN2Q(*Y*,*X*)) <= 2.0 となります。

総称名ATAN2Q は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*Y*と同じです。

使用例

```
x = atan2q (1.,1.)
```

2.36 ATAND組込み関数

ATAND 関数は、度数値を結果とする逆正接を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
ATAND	-----	1	実数型	実数型
	ATAND		単精度実数型	単精度実数型
	DATAND		倍精度実数型	倍精度実数型
	QATAND		4倍精度実数型	4倍精度実数型

result = ATAND (*X*)

X

実数型でなければなりません。

result

*X*と同じ型です。

機能説明

ATAND、DATAND、およびQATAND は、実数型データの逆正接を求めます。

結果の値は度数値で ATAND(*X*) = 180/ π × ATAN(*X*) となり、その範囲は ABS(ATAND(*X*)) <= 90.0 となります。

総称名ATAND は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
a = atand(.5)
```

2.37 ATANH組込み関数

ATANH 関数は、逆双曲線正接を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
ATANH	-----	1	実数型 または複素数型	実数型 または複素数型

result = ATANH (*X*)

X

実数型または複素数型でなければならず、実数型の場合、ABS(*X*) < 1.0 でなければなりません。

result

*X*と同じ型です。

機能説明

ATANH は、実数型データおよび複素数型データの逆双曲線正接を求めます。

結果の値は、複素数型の場合、虚部はラジアン値で、その範囲は ABS(AIMAG(ATANH(*X*))) <= π/2 となります。

関数の結果の型は、*X*と同じです。

使用例

```
r = atanh(0.76)
```

2.38 ATANQ組込み関数

ATANQ 関数は、象限値を結果とする逆正接を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
ATANQ	-----	1	実数型	実数型
	ATANQ		単精度実数型	単精度実数型
	DATANQ		倍精度実数型	倍精度実数型
	QATANQ		4倍精度実数型	4倍精度実数型

result = ATANQ (*X*)

X

実数型でなければなりません。

result

*X*と同じ型です。

機能説明

ATANQ、DATANQ、およびQATANQ は、実数型データの逆正接を求めます。

結果の値は象限値で ATANQ(*X*) = 2/π × ATAN(*X*) となり、その範囲は ABS(ATANQ(*X*)) <= 1.0 となります。

総称名ATANQ は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
a = atanq(.5)
```

2.39 ATOMIC_DEFINE組込みサブルーチン

ATOMIC_DEFINE サブルーチンは、変数をアトミックに定義します。

分類

アトミックサブルーチン

形式

CALL ATOMIC_DEFINE (*ATOM*, *VALUE* [, *STAT*])

ATOM

スカラの共配列または共添字付き実体でなければなりません。また、種別ATOMIC_INT_KIND をもつ整数型または種別ATOMIC_LOGICAL_KIND をもつ論理型でなければなりません。INTENT(OUT) 引数です。*VALUE* が論理型であるか種別ATOMIC_INT_KIND をもつ整数型である場合、*ATOM* の値は*VALUE* の値となります。そうでなければ、*ATOM* の値はINT(*VALUE*, ATOMIC_INT_KIND) の値となります。

VALUE

ATOM が整数型なら整数型スカラ、*ATOM* が論理型なら論理型スカラでなければなりません。INTENT(IN) の引数です。

STAT (省略可能)

共添字付き実体でないスカラの整数型変数でなければなりません。INTENT(OUT) 引数です。実行が正常に終了した場合、変数には値0が設定されます。実行中に誤り条件が発生した場合、実行時の診断メッセージの番号が設定されます。

使用例

```
use iso_fortran_env, only: atomic_int_kind
integer(atomic_int_kind), save:: k[*]
call atomic_define(k[1],1) ! 像番号1の変数kに1が代入されます
```

2.40 ATOMIC_REF組込みサブルーチン

ATOMIC_REF サブルーチンは、変数をアトミックに参照します。

分類

アトミックサブルーチン

形式

CALL ATOMIC_REF (*VALUE*, *ATOM* [, *STAT*])

VALUE

*ATOM*が整数型なら整数型スカラ、*ATOM*が論理型なら論理型スカラでなければなりません。INTENT(OUT) 引数です。論理型であるか種別ATOMIC_INT_KINDをもつ整数型である場合、*VALUE*の値は*ATOM*の値となります。そうでなければ、*VALUE*の値はINT(*ATOM*, KIND(*VALUE*)) の値となります。

ATOM

スカラの共配列または共添字付き実体でなければなりません。また、種別ATOMIC_INT_KIND をもつ整数型または種別ATOMIC_LOGICAL_KIND をもつ論理型でなければなりません。INTENT(IN) 引数です。

STAT (省略可能)

共添字付きでないスカラの整数型変数でなければなりません。INTENT(OUT)引数です。実行が正常に終了した場合、変数には値0が設定されます。実行中に誤り条件が発生した場合、実行時の診断メッセージの番号が設定されます。

使用例

```
use iso_fortran_env, only: atomic_int_kind
integer(atomic_int_kind) :: k[*], v
save:: k
k=0
sync all
call atomic_ref(v,k[1]) ! 変数vに0が代入されます
```

2.41 AUTOMATIC文

AUTOMATIC 文は、実体をスタックに割り付けることを指定します。AUTOMATIC 属性をもつ実体は、RETURN 文またはEND 文の実行後に不定となります。

AUTOMATIC 文は、以下の形式です。

```
AUTOMATIC [ [ :: ] object-name-list ]
```

object-name-list はコンマで区切られた実体名の並びです。

object-name は、仮引数名、手続名、関数結果の名前、自動割付け変数の名前、結合実体の名前、共配列、および共通ブロック実体の名前であってはなりません。

object-name-list のないAUTOMATIC 文は、SAVE 属性をもつ実体を除いて、その有効域内のすべての可能な項目を指定したかのように扱われます。

AUTOMATIC 文はモジュールの宣言部および初期値設定プログラム単位には指定できません。また、主プログラムでは効果がありません。

AUTOMATIC 文の例:

```
subroutine sub
automatic :: i,j        ! i および j は、スタックに割り付けられます
```

2.42 BACKSPACE文

BACKSPACE 文は、指定された装置に接続されているファイルの、ファイル位置を移動します。

BACKSPACE 文の実行は、指定した装置における非同期データ転送操作のすべてに対して、待機操作を行います。

BACKSPACE 文は、以下の形式です。

```
BACKSPACE external-file-unit            または
BACKSPACE ( position-spec-list )
```

external-file-unit は外部ファイル装置であり、スカラ整数式でなければなりません。

position-spec-list は、コンマで区切られた位置付け指定子の並びです。

position-spec は、以下の形式です。

```
[ UNIT= ] external-file-unit            または
IOMSG = iomsg                           または
IOSTAT= io-stat                         または
ERR= err-label
```

位置付け指定子並びにおいて、UNIT 指定子は必ず1つ指定しなければならず、他の指定子はそれぞれ1つ指定することができます。

文字列‘UNIT=’をUNIT 指定子から省略する場合、UNIT 指定子は、位置付け指定子並びの最初の項目でなければなりません。

io-stat は、スカラ基本整数変数です。

IOSTAT 指定子が指定された場合、IOSTAT 指定子に指定された変数には、以下の値が設定されます。

- 誤り条件、ファイル終了条件、および記録終了条件が検出されない場合は、0

・ 誤り条件が検出された場合は、1 または実行時の診断メッセージの番号

err-label は文番号であり、このBACKSPACE 文と同じ有効域内の飛び先文の文番号でなければなりません。

ERR 指定子が指定され、BACKSPACE 文の実行中に誤り条件が検出された場合、ERR 指定子に指定された文番号をもつ文が次に実行されます。

BACKSPACE 文は、ファイルに現在記録が存在するならば、現在記録の前に位置付けます。現在記録がなければ、直前記録の前に位置付けられます。現在記録も直前記録もなければ、ファイル位置は変わりません。

直前記録がファイル終了記録の時、ファイルは、ファイル終了記録の前に位置付けられます。

BACKSPACE 文が暗黙的にファイル終了記録を書いた場合には、ファイルは、ファイル終了記録の直前記録の前に位置付けられます。

接続されているが存在しないファイルに対して、BACKSPACE 文を実行してはなりません。

並び書式または変数群書式を使って書かれた記録を超えるBACKSPACE 文を実行してはなりません。

iomsg はスカラ基本文字変数でなければなりません。入出力文の実行中に誤り条件が発生した場合、*iomsg* には説明のためのメッセージが代入されます。

誤り条件が発生しない場合、*iomsg* の値は変更されません。

直接探査として接続されているファイルを、BACKSPACE 文で参照してはなりません。書式なし流れ探査として接続されているファイルを、BACKSPACE 文で参照してはなりません。

BACKSPACE 文の例:

```
backspace 10                ! 装置番号10に接続されているファイルのファイル位置を移動します
backspace (10,err=100)      ! 装置番号10に接続されているファイルのファイル位置を移動し、
                            ! 誤り条件が発生した場合には、文番号100の文が次に実行されます
```

2.43 BESSEL_J0組込み関数

BESSEL_J0関数は、第1種ベッセル関数です。次数はゼロです。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
BESSEL_J0	-----	1	実数型	実数型

result = BESSEL_J0 (*X*)

X

実数型でなければなりません。

result

X と同じ型です。

機能説明

第1種ベッセル関数です。次数はゼロです。

単精度実数型の引数の場合、引数は ABS(*X*) < 8.23E+05 でなければなりません。

倍精度実数型の引数の場合、引数は DABS(*X*) < 3.53D+15 でなければなりません。

4倍精度実数型の引数の場合、引数は QABS(*X*) < $2.0^{62} \times \pi$ でなければなりません。

使用例

```
a = bessel_j0( 1.0 )
```

2.44 BESSEL_J1組込み関数

BESSEL_J1関数は、第1種ベッセル関数です。次数は1です。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
BESSEL_J1	-----	1	実数型	実数型

result = BESSEL_J1 (*X*)

X

実数型でなければなりません。

result

*X*と同じ型です。

機能説明

第1種ベッセル関数です。次数は1です。

単精度実数型の引数の場合、引数は ABS(*X*) < 8.23E+05 でなければなりません。

倍精度実数型の引数の場合、引数は DABS(*X*) < 3.53D+15 でなければなりません。

4倍精度実数型の引数の場合、引数は $QABS(X) < 2.0^{62} \times \pi$ でなければなりません。

使用例

```
a = bessel_j1( 1.0 )
```

2.45 BESSEL_JN組込み関数

BESSEL_JN関数は、第1種ベッセル関数です。

分類

BESSEL_JN (*N*, *X*) : 要素別処理関数

BESSEL_JN (*N1*, *N2*, *X*) : 変形関数

形式

総称名	個別名	引数の数	引数の型	結果の型
BESSEL_JN	-----	2	整数型 , 実数型	実数型
		3	整数型 , 整数型 , 実数型	実数型

result = BESSEL_JN (*N* , *X*)　　または
result = BESSEL_JN (*N1* , *N2* , *X*)

N

整数型でなければなりません。負の値であってはなりません。

N1

整数型のスカラでなければなりません。負の値であってはなりません。

N2

整数型のスカラでなければなりません。負の値であってはなりません。

X

実数型でなければなりません。
変形関数では、スカラでなければなりません。

result

1. BESSEL_JN (*N*, *X*)

   *X*と同じ型です。

2. BESSEL_JN (*N1* , *N2* , *X*)

   *X*と同じ型です。形状がMAX(*N2* - *N1* + 1 , 0)の1次元配列です。

機能説明

1. BESSEL_JN (*N*, *X*)

   第1種ベッセル関数です。次数は*N*です。

2. BESSEL_JN (*N1* , *N2* , *X*)

   第1種ベッセル関数です。結果の*i*番目要素に対する次数は、*N1* + *i* - 1です。

単精度実数型の引数の場合、引数は ABS(*X*) < 8.23E+05 でなければなりません。

倍精度実数型の引数の場合、引数は DABS(*X*) < 3.53D+15 でなければなりません。

4倍精度実数型の引数の場合、引数は QABS(*X*) < $2.0^{62} \times \pi$ でなければなりません。

使用例

```
a = bessel_jn( 1 , 1.0 )
```

2.46 BESSEL_Y0組込み関数

BESSEL_Y0関数は、第2種ベッセル関数です。次数はゼロです。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
BESSEL_Y0	-----	1	実数型	実数型

result = BESSEL_Y0 (*X*)

X

実数型でなければなりません。ゼロより大きい値でなければなりません。

result

*X*と同じ型です。

機能説明

第2種ベッセル関数です。次数はゼロです。

単精度実数型の引数の場合、引数は 0 < *X* < 8.23E+05 でなければなりません。

倍精度実数型の引数の場合、引数は 0 < *X* < 3.53D+15 でなければなりません。

4倍精度実数型の引数の場合、引数は 0 < *X* < $2.0^{62} \times \pi$ でなければなりません。

使用例

```
a = bessel_y0( 1.0 )
```

2.47 BESSEL_Y1組込み関数

BESSEL_Y1関数は、第2種ベッセル関数です。次数は1です。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
BESSEL_Y1	-----	1	実数型	実数型

result = BESSEL_Y1 (*X*)

X

実数型でなければなりません。ゼロより大きい値でなければなりません。

result

*X*と同じ型です。

機能説明

第2種ベッセル関数です。次数は1です。

単精度実数型の引数の場合、引数は 0 < *X* < 8.23E+05 でなければなりません。

倍精度実数型の引数の場合、引数は 0 < *X* < 3.53D+15 でなければなりません。

4倍精度実数型の引数の場合、引数は $0 < X < 2.0^{62} \times \pi$ でなければなりません。

使用例

```
a = bessel_y1( 1.0 )
```

2.48 BESSEL_YN組込み関数

BESSEL_YN関数は、第2種ベッセル関数です。

分類

BESSEL_YN (*N*, *X*) : 要素別処理関数

BESSEL_YN (*N1*, *N2*, *X*) : 変形関数

形式

総称名	個別名	引数の数	引数の型	結果の型
BESSEL_YN	-----	2	整数型 , 実数型	実数型
		3	整数型 , 整数型 , 実数型	実数型

result = BESSEL_YN (*N* , *X*)　　または
result = BESSEL_YN (*N1* , *N2* , *X*)

N

整数型でなければなりません。負の値であってはなりません。

N1

整数型のスカラでなければなりません。負の値であってはなりません。

N2

整数型のスカラでなければなりません。負の値であってはなりません。

X

実数型でなければなりません。ゼロより大きい値でなければなりません。
変形関数では、スカラでなければなりません。

result

1. BESSEL_YN (*N*, *X*)

   *X*と同じ型です。

2. BESSEL_YN (*N1* , *N2* , *X*)

   *X*と同じ型です。形状がMAX(*N2* - *N1* + 1 , 0)の1次元配列です。

機能説明

1. BESSEL_YN (*N*, *X*)

   第2種ベッセル関数です。次数は*N*です。

2. BESSEL_YN (*N1* , *N2* , *X*)

   第2種ベッセル関数です。結果の*i*番目要素に対する次数は、*N1* + *i* -1です。

単精度実数型の引数の場合、引数は 0 < *X* < 8.23E+05 でなければなりません。

倍精度実数型の引数の場合、引数は 0 < *X* < 3.53D+15 でなければなりません。

4倍精度実数型の引数の場合、引数は $0 < X < 2.0^{62} \times \pi$ でなければなりません。

使用例

```
a = bessel_yn( 1 , 1.0 )
```

2.49 BGE組込み関数

BGE 関数は、ビットごとに大小比較します。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
BGE	-----	2	整数型または 非10進定数表現 , 整数型または 非10進定数表現	基本論理型

result = BGE (*I* , *J*)

I

整数型または非10進定数表現でなければなりません。

J

整数型または非10進定数表現でなければなりません。

result

基本論理型です。

機能説明

BGE は、*I*と*J*をビットごとに大小比較し、*I*が*J*以上の場合、真を返却します。それ以外は、偽を返却します。

どちらかに非10進定数表現を指定した場合、非10進定数表現は、指定した整数型に変換されます。両方に非10進定数表現を指定した場合、非10進定数表現は基本整数型に変換されます。

関数の結果の型は、基本論理型です。

使用例

```
logical::l
l = bge(-1,1)              ! l には真が代入されます
```

2.50 BGT組込み関数

BGT 関数は、ビットごとに大小比較します。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
BGT	-----	2	整数型または 非10進定数表現 , 整数型または 非10進定数表現	基本論理型

result = BGT (*I* , *J*)

I

整数型または非10進定数表現でなければなりません。

J

整数型または非10進定数表現でなければなりません。

result

基本論理型です。

機能説明

BGT は、*I*と*J*をビットごとに大小比較し、*I*が*J*より大きい場合、真を返却します。それ以外は、偽を返却します。

どちらかに非10進定数表現を指定した場合、非10進定数表現は、指定した整数型に変換されます。両方に非10進定数表現を指定した場合、非10進定数表現は基本整数型に変換されます。

関数の結果の型は、基本論理型です。

使用例

```
logical::l
l = bgt(-1,1)                  ! l には真が代入されます
```

2.51 BICサービスサブルーチン

機能説明

第2引数*i*の第1引数*pos*ビット目に0を設定します。

形式

CALL BIC (*pos* , *i*)

pos

基本整数型の整数式。
pos が負またはBIT_SIZE(*i*) 以上の場合は、値は設定されません。

i

基本整数型のスカラ変数。

使用例

```
use service_routines,only:bic
integer :: i
i = -1
call bic(31,i)
write(6,fmt="(1x,z8.8)") i          ! 7FFFFFFFが出力されます
end
```

2.52 BIND文

BIND 文は以下の形式です。

language-binding-spec [::] *bind-entity-list*

language-binding-spec は言語束縛指定子です。*language-binding-spec* は以下の形式です。

BIND (C [, NAME = *scalar-char-initialization-expr*])

scalar-char-initialization-expr はスカラ文字定数式です。

bind-entity-list は束縛要素並びです。*bind-entity-list* は以下の形式です。

entity-name または
/ *common-block-name* /

entity-name は要素名です。

common-block-name は共通ブロック名です。

スカラ文字定数式は、基本文字型の種別をもたなければなりません。

スカラ文字定数式の最初と最後の連続する空白を無視した値が、1以上の長さであり有効な識別子でなければなりません。

BIND 属性をもつ実体は、暗黙的にSAVE 属性をもちます。また、SAVE 文を指定して、明示的にSAVE 属性を指定することもできます。

BIND 文中の束縛要素に要素名を指定する場合、そのBIND 文は、モジュールの宣言部に現れなければならず、その要素は相互利用可能な変数でなければなりません。POINTER属性、ALLOCATABLE属性をもつ変数、および共配列であってはなりません。

言語束縛指定子にNAME 指定子を指定する場合、束縛要素並び中の要素は、1つでなければなりません。

束縛要素が共通ブロックの場合、その共通ブロックのすべての変数は、相互利用可能でなければなりません。

この文は、並び中の変数および共通ブロックに対して、BIND 属性を指定します。

2.53 BISサービスサブルーチン

機能説明

第2引数*i*の第1引数*pos*ビット目に1を設定します。

形式

CALL BIS (*pos* , *i*)

pos

基本整数型スカラ。
pos が負またはBIT_SIZE(*i*) 以上の場合は、値は設定されません。

i

基本整数型スカラ。

使用例

```
use service_routines,only:bis
integer :: i
i = 1
call bis(31,i)
write(6,fmt="(1x,z8.8)") i          ! 80000001が出力されます
end
```

2.54 BITサービス関数

機能説明

第2引数*i*の第1引数*pos*ビット目が真か偽かを検査します。

形式

ly = BIT (*pos* , *i*)

pos

基本整数型スカラ。検査するビットを指定します。

i

基本整数型スカラ。検査する対象を指定します。

関数結果

基本論理型スカラ。*i*の第*pos*ビットが1の場合は真、*i*の第*pos*ビットが0の場合は偽を返却します。*pos*が0未満または*i*のビット数以上の場合は偽を返却します。

使用例

```
use service_routines,only:bit
logical :: l(2)
i = 10
l = (/bit(0,i),bit(1,i)/)       ! l には(/.false.,.true./) が代入されます
end
```

2.55 BIT_SIZE組込み関数

BIT_SIZE 関数は、整数*I*のビット数を返却します。

分類

問合せ関数

形式

result = BIT_SIZE (*I*)

I

整数型でなければなりません。スカラまたは配列です。

result

*I*と同じ種別型パラメタをもつ整数型スカラです。

機能説明

BIT_SIZE は、整数*I*のビット数を返却します。

関数の結果の型は、*I*と同じ種別型パラメタをもつ整数型スカラです。

以下の値の固定値となります。

引数の型	結果の値
1バイトの整数型	8_1
2バイトの整数型	16_2
4バイトの整数型	32_4
8バイトの整数型	64_8

使用例

```
integer :: i, m, n
integer,dimension(2) :: j
m = bit_size (i)            ! m には32_4が代入されます
n = bit_size (j)            ! n には32_4が代入されます
```

2.56 BLE組込み関数

BLE 関数は、ビットごとに大小比較します。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
BLE	-----	2	整数型または 非10進定数表現 , 整数型または 非10進定数表現	基本論理型

result = BLE (*I* , *J*)

I

整数型または非10進定数表現でなければなりません。

J

整数型または非10進定数表現でなければなりません。

result

基本論理型です。

機能説明

BLE は、*I*と*J*をビットごとに大小比較し、*I*が*J*以下の場合、真を返却します。それ以外は、偽を返却します。

どちらかに非10進定数表現を指定した場合、非10進定数表現は、指定した整数型に変換されます。両方に非10進定数表現を指定した場合、非10進定数表現は基本整数型に変換されます。

関数の結果の型は、基本論理型です。

使用例

```
logical::l
l = ble(-1,1)                ! l には偽が代入されます
```

2.57 BLOCK構文

BLOCK 構文は、構文内に宣言文を含むことができる実行構文です。

BLOCK 構文の形式は以下です。

```
[ block-construct-name : ] BLOCK
  [ specification-part ]
  block
END BLOCK [ block-construct-name ]
```

*block-construct-name*はBLOCK 構文名です。BLOCK 文にBLOCK 構文名を指定する場合、対応するEND BLOCK 文にもBLOCK 構文名を指定しなければなりません。BLOCK 文にBLOCK 構文名を指定しない場合、END BLOCK 文にも指定してはなりません。

specification-part は宣言部です。

block はブロックです。ブロックは0個以上の実行文または実行構文の並びです。

BLOCK 構文には、以下を含めてはなりません。

- 長さ型パラメタをもつ派生型(“1.5.11.2 派生型パラメタ”参照)の定義、実体宣言、または成分宣言
- 共配列の宣言(“1.17 共配列”参照)
- 文関数定義文(“2.4 文関数定義文(廃止予定事項)”参照)
- ASSIGN 文(“2.28 ASSIGN文(廃止事項)”参照)
- COMMON 文(“2.82 COMMON文”参照)
- EQUIVALENCE 文(“2.160 EQUIVALENCE文”参照)
- 割当て形GO TO 文(“2.223 割当て形GO TO文(廃止事項)”参照)
- IMPLICIT 文(“2.282 IMPLICIT文”参照)
- INTENT 文(“2.290 INTENT文”参照)
- NAMELIST 文(“2.361 NAMELIST文”参照)
- OPTIONAL 文(“2.373 OPTIONAL文”参照)
- USE 文(“2.496 USE文”参照)
- VALUE 文(“2.498 VALUE文”参照)

BLOCK 構文内に現れるSAVE 文(“2.422 SAVE文”参照)は並びをもたなければならず、共通ブロック名を指定してはなりません。

ASYNCHRONOUS文(“2.31 ASYNCHRONOUS文”参照)およびVOLATILE文(“2.500 VOLATILE文”参照)を除く宣言文は、BLOCK 構文内を有効域とする要素を宣言します。

BLOCK 構文の外側からBLOCK 構文内に飛び込んではなりません。

BLOCK 構文の例:

```
k = 5
block
  integer k ! kはBLOCK構文内を有効域とします
  k = 10
  print *,k ! 10が出力されます
end block
print *,k   ! 5が出力されます
```

2.58 BLOCK DATA文

BLOCK DATA 文は、初期値設定プログラム単位を開始します。初期値設定プログラム単位については、“1.11.4 初期値設定プログラム単位”を参照してください。

BLOCK DATA 文は以下の形式です。

BLOCK DATA [*block-data-name*]

block-data-name は、初期値設定プログラム単位名です。初期値設定プログラム単位名は、そのプログラム中の大域要素であり、他のプログラム単位、外部手続、または共通ブロックの名前と同じであってはなりません。また、その初期値設定プログラム単位のどの局所名とも、同じであってはなりません。

初期値設定プログラム単位名は、指定しなくてもかまいません。1つのプログラム中に、2つ以上の名前のない初期値設定プログラム単位があってはなりません。

BLOCK DATA 文の例:

```
block data
  common /com/ a,b,c
  integer :: a=1, b=2, c=3
end block data
```

2.59 BLT組込み関数

BLT 関数は、ビットごとに大小比較します。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
BLT	-----	2	整数型または 非10進定数表現 , 整数型または 非10進定数表現	基本論理型

result = BLT (*I* , *J*)

I

整数型または非10進定数表現でなければなりません。

J

整数型または非10進定数表現でなければなりません。

result

基本論理型です。

機能説明

BLT は、*I*と*J*をビットごとに大小比較し、*I*が*J*より小さい場合、真を返却します。それ以外は、偽を返却します。

どちらかに非10進定数表現を指定した場合、非10進定数表現は、指定した整数型に変換されます。両方に非10進定数表現を指定した場合、非10進定数表現は基本整数型に変換されます。

関数の結果の型は、基本論理型です。

使用例

```
logical::l
l = blt(-1,1) ! l には偽が代入されます
```

2.60 BTEST組込み関数

BTEST 関数は、整数値のビットがオン(1)かオフ(0)かを判定します。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
BTEST	-----	2	1バイトの整数型, 整数型	基本論理型
	BITEST		2バイトの整数型, 整数型	基本論理型
	BTEST		4バイトの整数型, 整数型	基本論理型
	BJTEST		4バイトの整数型, 整数型	基本論理型
	-----		8バイトの整数型, 整数型	基本論理型

result = BTEST (*I* , *POS*)

I

整数型でなければなりません。

POS

整数型でなければなりません。0 <= *POS* < BIT_SIZE(*I*) でなければなりません。

result

基本論理型です。

機能説明

BTEST、BITEST、およびBJTEST は、整数値のビットがオン(1)かオフ(0)かを判定します。

I の第*POS* ビットが1の場合、真を返却します。*I* の第*POS* ビットが0の場合、偽を返却します。

総称名BTEST は、すべての整数型の引数に使用することができます。

それぞれの関数の結果の型は、基本論理型です。

使用例

```
logical :: l
l = btest(1,0)          ! l には真が代入されます
l = btest(4,1)          ! l には偽が代入されます
l = btest(32,5)         ! l には真が代入されます
```

2.61 BYTE型宣言文

BYTE 型宣言文は、1バイトの整数型のデータ実体を宣言します。

BYTE 型宣言文は、以下の形式です。

```
BYTE [ [ , attr-spec ]... :: ] entity-decl-list
```

型宣言文の詳細については、“2.3 型宣言文”を参照してください。

2.62 CALL文

CALL 文は、サブルーチンを引用し、指定された実引数を渡します。

CALL 文は以下の形式です。

CALL *procedure-designator* [([*actual-arg-spec-list*])]

procedure-designator は手続特定子です。以下の形式です。

procedure-name または
proc-component-ref または
data-ref % *binding-name*

procedure-name は、手続名です。

proc-component-ref は、手続成分引用です。

data-ref %*binding-name* は、データ参照%束縛名です。

手続名は、手続または手続ポインタの名前でなければなりません。

束縛名は、データ参照の宣言時の型の束縛名でなければなりません。

データ参照が配列である場合、引用する型束縛手続は、PASS 属性をもたなければなりません。

actual-arg-spec-list は、コンマで区切られた実引数指定子の並びです。

actual-arg-spec は、以下の形式です。

[*keyword* =] *actual-arg*

keyword は引数キーワードであり、仮引数名でなければなりません。

手続の引用仕様が暗黙的である場合には、引数キーワードを指定することはできません。

引数キーワードは、先行するすべての実引数指定子の引数キーワードを省略した場合にだけ、省略することができます。

それぞれの引数キーワードは、その手続の明示的引用仕様中の仮引数の名前でなければなりません。

actual-arg は実引数であり、以下の形式です。

expr または
variable または
procedure-name または
/label または
%VAL(*expr*)

expr は式です。

variable は変数です。

procedure-name は、手続名です。

組込み手続でない要素別処理手続は、実引数に指定することはできません。

実引数の手続名は、文関数の名前であってはならず、その名前が個別名でもある場合を除き、手続の総称名であってはなりません。

label は文番号であり、CALL 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。

%VAL は、実引数を値渡しにすることを指定します。

実引数の型、型パラメタ、および形状は、その手続の仮引数の特性に適合しなければなりません。引数の結合については“1.12.6 手続引用”を参照してください。

CALL 文の例:

```
x = 3.0
call alpha(x,y)
end program
subroutine alpha(a,b)
  implicit none
  real, intent(in) :: a
  real, intent(out) :: b
```

```
  ...
end subroutine alpha
```

2.63 CASE構文

CASE 構文は、それを構成するブロックのうち、多くても1つの実行を選択します。

CASE 構文は、以下の形式です。

```
[ case-construct-name : ] SELECT CASE ( case-expr )
    [ CASE case-selector [ case-construct-name ]
        block ] ...
END SELECT [ case-construct-name ]
```

case-construct-name は、CASE 構文名です。

SELECT CASE 文にCASE 構文名を指定する場合、対応するEND SELECT 文にも同じCASE 構文名を指定しなければなりません。SELECT CASE 文にCASE 構文名を指定しない場合、対応するEND SELECT 文にCASE 構文名を指定してはなりません。CASE 文にCASE 構文名を指定する場合、対応するSELECT CASE 文にも同じCASE 構文名を指定しなければなりません。

case-expr は場合式であり、スカラ整数式、スカラ文字式、またはスカラ論理式でなければなりません。

case-selector は場合選択子であり、以下の形式です。

```
( case-value-range-list )        または
DEFAULT
```

DEFAULT 場合選択子は、1つのCASE 構文内に1つだけ指定することができます。

case-value-range-list は、コンマで区切られた場合値範囲の並びです。

case-value-range は、以下の形式です。

```
case-value                       または
case-value :                     または
: case-value                     または
case-value : case-value
```

case-value は場合値であり、スカラ整数定数式、スカラ文字定数式、またはスカラ論理定数式でなければなりません。

1つのCASE 構文中の場合値は、すべて場合式と同じ型でなければなりません。文字型については、長さは異なってもかまいませんが、種別型パラメタは同じでなければなりません。場合式が論理型である場合、コロン‘:’を使った場合値範囲は指定できません。

1つのCASE 構文において、場合値範囲に重なりがあってはなりません。すなわち、2つ以上の場合値範囲と一致する場合式の値が存在してはなりません。

block は、ブロックです。ブロックは、0個以上の実行文または実行構文の並びです。ブロックに実行文または実行構文が1つも含まれない場合、そのようなブロックの実行は、効果をもちません。

SELECT CASE 文を実行すると、その中の場合式が評価され、その結果を場合指標と呼びます。場合指標がいずれかの場合選択子と一致したとき、そのCASE 文に続くブロックが実行されます。

場合指標の値を*c* とした場合、場合値範囲との一致は以下のように評価されます。

- 場合値範囲がコロンなしの単一値*v* である場合、論理型であれば式*c*.EQV.*v* が真となるときに、整数型または文字型の場合は、*c* == *v* となるときに一致します。
- 場合値範囲が‘*low*:*high*’の形の場合、式 *low* <= *c* .AND. *c* <= *high* が真となるときに一致します。
- 場合値範囲が‘*low*:’の形である場合、式 *low* <= *c* が真となるときに一致します。
- 場合値範囲が‘:*high*’の形である場合、式 *c* <= *high* が真となるときに一致します。

場合指標がほかのどの選択子とも一致せず、DEFAULT 選択子がある場合、場合指標はDEFAULT 選択子と一致します。

場合指標がほかのどの選択子とも一致せず、DEFAULT 選択子がない場合、場合指標はどの選択子とも一致しません。

場合指標の値と一致する場合選択子がない場合、CASE 構文の実行は、そのCASE 構文中のどのブロックも実行せずに完了します。

CASE 構文の例:

```
select case (i)
  case (:-2)
    print*, "i is less than or equal to -2"
  case (0)
    print*, "i is equal to 0"
  case (1:97)
    print*, "i is in the range 1 to 97, inclusive"
  case default
    print*, "i is either -1 or greater than 97"
end select
```

2.64 CASE文

CASE 文は、CASE 構文中に指定し、直後にあるブロックの実行される条件を指定します。

CASE 文は、以下の形式です。

```
CASE case-selector [ case-construct-name ]
```

case-selector は、場合選択子であり、以下の形式です。

```
( case-value-range-list )                    または
DEFAULT
```

DEFAULT 場合選択子は、1つのCASE 構文内に1つだけ指定することができます。

case-value-range-list は、コンマで区切られた場合値範囲の並びです。

case-value-range は、以下の形式です。

```
case-value                                   または
case-value :                                 または
: case-value                                 または
case-value : case-value
```

case-value は場合値であり、スカラ整数定数式、スカラ文字定数式、またはスカラ論理定数式でなければなりません。

case-construct-name は、CASE 構文名です。

CASE 構文の詳細については、“2.63 CASE構文”を参照してください。

2.65 CBRT組込み関数

CBRT 関数は、実数型データの立方根を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
CBRT	-----	1	実数型	実数型
	CBRT		単精度実数型	単精度実数型
	DCBRT		倍精度実数型	倍精度実数型
	QCBRT		4倍精度実数型	4倍精度実数型

result = CBRT (*X*)

X

実数型でなければなりません。

result

*X*と同じ型です。

機能説明

CBRT、DCBRT、およびQCBRT は、実数型データの立方根を求めます。

総称名CBRT は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
real :: r
r = cbrt(27.)              ! r には3.0が代入されます
```

2.66 CEILING組込み関数

CEILING 関数は、引数の値以上で最小の整数値を返却します。

分類

要素別処理関数

形式

result = CEILING (*A* [, *KIND*])

A

実数型でなければなりません。

KIND（省略可能）

スカラ整数定数式でなければなりません。

result

整数型です。*KIND* が指定された場合、種別型パラメタは*KIND* の指定に従います。
KIND が省略された場合、種別型パラメタは基本整数型になります。

機能説明

CEILING は、引数の値以上で最小の整数値を返却します。

A 以上で最小の整数を返却します。

関数の結果が指定された整数型で表現できない場合、結果の値は不定となります。

関数の結果の型は、種別型パラメタ*KIND* が指定された場合、種別型パラメタ*KIND* の整数型となります。*KIND* が省略された場合、関数の結果の型は基本整数型です。

使用例

```
i = ceiling (-4.7)         ! i には-4が代入されます
i = ceiling (4.7)          ! i には5が代入されます
```

2.67 CHANGEENTRY文

CHANGEENTRY文は、外部手続名の加工方法を変更します。

CHANGEENTRY文は、以下の形式です。

```
CHANGEENTRY [ :: ] external-proc-name-list
```

external-proc-name-list はコンマで区切られた外部手続名の並びです。

external-proc-name は、そのCHANGEENTRY文が指定された手続で定義される外部手続の名前であってもかまいません。

external-proc-name は、そのCHANGEENTRY文が指定された手続で定義される外部手続の名前である場合を除いて、暗黙的にEXTERNAL 属性をもちます。

BIND 属性と同時に指定できません。

外部手続名の加工方法については、“Fortran使用手引書”を参照してください。

CHANGEENTRY文の例:

```
changeentry :: csub
call csub( )
end
```

2.68 CHAR組込み関数

CHAR 関数は、整数型データを処理系大小順序における文字コードと解釈して、長さ1の文字型データに変換します。

分類

要素別処理関数

形式

result = CHAR (*I* [, *KIND*])

I

整数型でなければなりません。

KIND（省略可能）

スカラ整数定数式でなければなりません。

result

長さ1の文字型です。*KIND*が指定された場合、種別型パラメタは*KIND*の指定に従います。*KIND* が省略された場合、種別型パラメタは基本文字型になります。

機能説明

CHAR は、整数型データを処理系大小順序における文字コードと解釈して長さ1の文字型データに変換します。

関数の結果の型は、種別型パラメタ*KIND*が指定された場合、種別型パラメタ*KIND*の長さ1の文字型となります。*KIND*が省略された場合、関数の結果の型は、長さ1の基本文字型です。

使用例

```
c = char(65)                    ! c には'A' が代入されます
```

2.69 CHARACTER型宣言文

CHARACTER 型宣言文は、文字型のデータ実体を宣言します。

CHARACTER 型宣言文は、以下の形式です。

CHARACTER [*char-selector*] [[, *attr-spec*]... ::] *entity-decl-list*

型宣言文の詳細については、“2.3 型宣言文”を参照してください。

2.70 CHDIRサービス関数

機能説明

現在のデフォルトディレクトリを変更します。

形式

iy = CHDIR (*dirname*)

dirname

基本文字型スカラ。変更するディレクトリ名を指定します。

関数結果

基本整数型スカラ。デフォルトディレクトリが変更できたときは0、それ以外のときは0以外を返却します。

使用例

```
use service_routines,only:chdir
integer :: iy
iy = chdir("/tmp")
end
```

2.71 CHMODサービス関数

機能説明

ファイルのパーミッションモードを変更します。

形式

iy = CHMOD (*fname* , *mode*)

fname

基本文字型スカラ。パーミッションモードを変更するファイル名を指定します。

mode

基本文字型スカラ。パーミッションモードを指定します。

関数結果

基本整数型スカラ。パーミッションモードが変更できた場合は0、それ以外のときは、システムが返却するエラー値を返却します。

使用例

```
use service_routines,only:chmod
character(len=8) :: fname="test.f90"
write(6,*) chmod(fname, "600")
end
```

2.72 CLASS型宣言文

CLASS 型宣言文は、多相的実体を宣言します。

CLASS 型宣言文は、以下の形式です。

```
CLASS ( type-name ) [ [ , attr-spec ]... :: ] entity-decl-list          または
CLASS ( * ) [ [ , attr-spec ]... :: ] entity-decl-list
```

型宣言文の詳細については、“2.3 型宣言文”を参照してください。

2.73 CLASS DEFAULT文

CLASS DEFAULT 文は、SELECT TYPE 構文中に指定します。

CLASS DEFAULT 文は、以下の形式です。

```
CLASS DEFAULT [ select-construct-name ]
```

select-construct-name は、SELECT 構文名です。

SELECT TYPE 構文の詳細については、“2.431 SELECT TYPE構文”を参照してください。

2.74 CLASS IS文

CLASS IS 文は、SELECT TYPE 構文中に指定します。

CLASS IS 文は、以下の形式です。

```
CLASS IS( derived-type-spec ) [ select-construct-name ]
```

derived-type-spec は、派生型指定子です。派生型指定子については、“1.5.11.8 派生型指定子”を参照してください。

derived-type-spec は、各長さ型パラメタが引き継がれることを指定しなければなりません。

derived-type-spec は、連続派生型またはBIND 属性をもつ型を指定できません。

select-construct-name は、SELECT 構文名です。

SELECT TYPE 構文の詳細については、“2.431 SELECT TYPE構文”を参照してください。

2.75 CLOCKサービスサブルーチン

機能説明

実行可能プログラムの実行開始からのCPU 時間を返却します。CPU 時間はプログラムが実行されているプロセスとそのプロセス内の全スレッドで利用されたCPU 時間です。

形式

```
CALL CLOCK ( g , i1 , i2 )
```

g

4バイトの整数型、8バイトの整数型、単精度実数型、倍精度実数型、4倍精度実数型のスカラ変数。
i1 で指定した単位のCPU 時間が返却されます。

i1

基本整数型スカラ。返却単位を示します。
指定する値は、以下の値です。以下の値以外の値を指定したときは、秒単位で返却されます。

=0：秒単位
=1：ミリ秒単位
=2：マイクロ秒単位

i2

基本整数型スカラ。*g* に指定した変数の型を指定します。
指定する値は、以下の値です。以下の値以外の値を指定したときは、4バイトの整数型で返却されます。

=0：4バイトの整数型
=1：単精度実数型
=2：倍精度実数型
=3：4倍精度実数型
=4：8バイトの整数型

使用例

```
use service_routines,only:clock
real :: g,g1
call clock(g,2,1)
do i=1,30
  write(10,*)i,i*i
end do
call clock(g1,2,1)
print *,'CPU TIME (in micro second) = ', g1-g
end
```

2.76 CLOCKMサービスサブルーチン

機能説明

実行可能プログラムの実行開始からのCPU 時間を返却します。CPU 時間はプログラムが実行されているプロセスとそのプロセス内の全スレッドで利用されたCPU 時間です。

形式

```
CALL CLOCKM ( i )
```

i

4バイトの整数型のスカラ変数。CPU 時間が返却されます。

使用例

```
use service_routines,only:clockm
integer :: t,t1
call clockm(t)
do i=1,30
  write(10,*) i,i*i
end do
call clockm(t1)
print *,'CPU TIME (in milli second) = ', t1-t
end
```

2.77 CLOCKVサービスサブルーチン

機能説明

実行可能プログラムの実行開始からのCPU 時間を返却します。CPU 時間はプログラムが実行されているプロセスとそのプロセス内の全スレッドで利用されたCPU時間です。

形式

```
CALL CLOCKV ( g1 , g2 , i1 , i2 )
```

g1

4バイトの整数型、8バイトの整数型、単精度実数型、倍精度実数型、4倍精度実数型のスカラ変数。
常に0 が返却されます。

g2

4バイトの整数型、8バイトの整数型、単精度実数型、倍精度実数型、4倍精度実数型のスカラ変数。
i1 で指定した単位のCPU 時間が返却されます。

i1

基本整数型スカラ。返却される値の単位を指定します。
指定する値は、以下の値です。以下の値以外の値を指定したときは、秒単位で返却されます。

=0：秒単位
=1：ミリ秒単位
=2：マイクロ秒単位

i2

基本整数型スカラ。*g1* および*g2* に指定した変数の型を指定します。
指定する値は、以下の値です。以下の値以外の値を指定したときは、4バイトの整数型で返却されます。

=0：4バイトの整数型
=1：単精度実数型
=2：倍精度実数型
=3：4倍精度実数型
=4：8バイトの整数型

使用例

```
use service_routines,only:clockv
integer :: i
real :: g1,g2,g3,g4
call clockv(g1,g2,2,1)
do i=1,30
  write(10,*)i,i*i
end do
call clockv(g3,g4,2,1)
print *,'CPU TIME (in micro second) = ', g4-g2
end
```

2.78 CLOSE文

CLOSE 文は、外部ファイルと装置との接続を解除します。

CLOSE 文は、以下の形式です。

```
CLOSE ( close-spec-list )
```

close-spec-list は、コンマで区切られた解除指定子の並びです。

close-spec は、以下の形式です。

```
[ UNIT = ] external-file-unit          または
IOMSG = iomsg                          または
IOSTAT = io-stat                       または
ERR = err-label                        または
STATUS = status
```

解除指定子並びにおいて、UNIT 指定子は必ず指定しなければならず、他の指定子はそれぞれ1つ指定することができます。

文字列‘UNIT=’をUNIT 指定子から省略する場合、UNIT 指定子は、解除指定子並びの最初の項目でなければなりません。

external-file-unit は、外部ファイル装置であり、スカラ整数式でなければなりません。

*iomsg*はスカラ基本文字変数でなければなりません。入出力文の実行中に誤り条件が発生した場合、*iomsg*には説明のためのメッセージが代入されます。

誤り条件が発生しない場合、*iomsg* の値は変更されません。

io-stat は、スカラ整変数でなければなりません。

IOSTAT 指定子が指定された場合、IOSTAT 指定子に指定された変数には、以下の値が設定されます。

- 誤り条件、ファイル終了条件、および記録終了条件が検出されない場合は、0
- 誤り条件が検出された場合は、1または実行時の診断メッセージの番号

err-label は文番号であり、このCLOSE 文と同じ有効域内の飛び先文の文番号でなければなりません。

ERR 指定子が指定され、CLOSE 文の実行中に誤り条件が検出された場合、ERR 指定子に指定された文番号をもつ文が次に実行されます。

status は、スカラ基本文字式であり、その値は'KEEP'、'DELETE'、または 'FSYNC' でなければなりません。

STATUS 指定子は、指定された装置に接続しているファイルの後処理を決定します。'KEEP' または 'FSYNC' は、CLOSE 文の実行直前の状態が 'SCRATCH' であるファイルに指定してはなりません。存在するファイルに対して'KEEP' または 'FSYNC'を指定すると、そのファイルは、CLOSE 文の実行後も存在します。存在しないファイルに対して'KEEP' または 'FSYNC' を指定すると、そのファイルは、CLOSE 文の実行後も存在しません。'DELETE' を指定すると、CLOSE 文の実行後はファイルが存在しなくなります。'FSYNC'を指定すると、ファイルを同期します。STATUS 指定子を省略すると'KEEP' が想定されますが、CLOSE 文の実行直前の状態が'SCRATCH' の場合は、'DELETE' となります。

CLOSE 文の例:

```
close (8,status='keep')          ! 装置番号8と接続されているファイルの接続を解除し、
                                 ! CLOSE 文実行後もファイルは存在させます
close (err=200,unit=9)           ! 装置番号9と接続されているファイルの接続を解除し、
                                 ! 誤り条件が発生した場合には、文番号200の文が
                                 ! 次に実行されます
```

2.79 CMPLX組込み関数

CMPLX 関数は、数値型データを複素数型へ変換します。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
CMPLX	-----	2 または 3	整数型、実数型または 非10進定数表現 [, 整数型、実数型または非10 進定数表現] , 整数型	複素数型
	-----	2	複素数型 , 整数型	複素数型
	-----	1 または 2	整数型、実数型または 非10進定数表現 [, 整数型、実数型または非10 進定数表現]	単精度複素数型
	-----	1	複素数型	単精度複素数型
DCMPLX	-----	1 または 2	整数型または実数型 [, 整数型または実数型]	倍精度複素数型
	-----	1	複素数型	倍精度複素数型
QCMPLX	-----	1 または 2	整数型または実数型 [, 整数型または実数型]	4倍精度複素数型
	-----	1	複素数型	4倍精度複素数型

result = CMPLX (*X* [, *Y*] [, *KIND*])
result = DCMPLX (*X* [, *Y*])
result = QCMPLX (*X* [, *Y*])

X

整数型、実数型、複素数型または非10進定数表現でなければなりません。

Y (省略可能)

整数型、実数型、または非10進定数表現でなければなりません。*X* が複素数型の場合は指定できません。

KIND（省略可能）

スカラ整数定数式でなければなりません。

result

CMPLX

複素数型です。*KIND*が省略された場合は、基本複素数型となります。*KIND*が指定された場合は、結果の型は*KIND*に従います。

DCMPLX

倍精度複素数型です。

QCMPLX

4倍精度複素数型です。

機能説明

CMPLX、DCMPLX、およびQCMPLX は、数値型データを複素数型へ変換します。

－ CMPLX の場合

*Y*が省略されていて、*X*が複素数型でない場合、*Y*=0.0E0 が指定されたものとして扱います。

*Y*が省略されていて、*X*が複素数型の場合、*Y*=AIMAG(*X*) が指定されたものとして扱います。

結果の値は、実部がREAL(*X*[,*KIND*])、虚部がREAL(*Y*[,*KIND*]) としたものを返却します。

関数の結果の型は複素数型です。*KIND*が指定された場合、種別型パラメタは*KIND*の指定に従います。*KIND*が省略された場合、種別型パラメタは単精度実数型と同じになります。

－ DCMPLX の場合

*Y*が省略されていて、*X*が複素数型でない場合、*Y*=0.0D0 が指定されたものとして扱います。

*Y*が省略されていて、*X*が複素数型の場合、*Y*=DIMAG(*X*) が指定されたものとして扱います。

結果の値は、実部がDBLE(*X*)、虚部がDBLE(*Y*) としたものを返却します。

関数の結果の型は、倍精度複素数型です。

－ QCMPLX の場合

*Y*が省略されていて、*X*が複素数型でない場合、*Y*=0.0Q0 が指定されたものとして扱います。

*Y*が省略されていて、*X*が複素数型の場合、*Y*=QIMAG(*X*) が指定されたものとして扱います。

結果の値は、実部がQEXT(*X*)、虚部がQEXT(*Y*) としたものを返却します。

関数の結果の型は、4倍精度複素数型です。

使用例

```
complex :: y,z
y = cmplx(3.2,4.7)                ! y には(3.2,4.7) が代入されます
z = cmplx(3.2)                    ! z には(3.2,0.0) が代入されます
```

2.80 CODIMENSION文

CODIMENSION 文は、共配列を宣言します。

CODIMENSION 文は、以下の形式です。

```
CODIMENSION [ :: ] coarray-decl-list
```

coarray-decl は、次の形式です。

```
coarray-name left-square-bracket  coarray-spec  right-square-bracket
```

coarray-name は、共配列の名前です。

coarray-spec は、共配列指定です。詳細については“1.17.1 共配列指定”を参照してください。

*left-square-bracket*は、左角括弧です。左角括弧は、[です。

*right-square-bracket*は、右角括弧です。右角括弧は、] です。

CODIMENSION 文の例:

```
codimension :: a[*] , b[2,*]
```

2.81 COMMAND_ARGUMENT_COUNT組込み関数

COMMAND_ARGUMENT_COUNT 関数は、コマンド引数の個数を取得します。

分類

問合せ関数

形式

result = COMMAND_ARGUMENT_COUNT ( )

result

基本整数型スカラです。

機能説明

結果の値は指定したコマンド引数の個数です。指定したコマンド引数がない場合、結果の値はゼロです。コマンド名はコマンド引数としては数えません。

使用例

```
i = command_argument_count ()                ! % a.out arg1 arg2 arg3
                                             ! のとき、i にはコマンド引数の個数である
                                             ! 3が代入されます
```

2.82 COMMON文

COMMON 文は、共通ブロックを宣言します。共通ブロックは、プログラム中のどの有効域からも参照可能な物理的な記憶場所のブロックです。

COMMON 文は、以下の形式です。

```
COMMON [ / [ common-block-name ] / ] common-block-object-list ■
■ [ [ , ] / [ common-block-name ] / common-block-object-list ]...
```

common-block-name は、共通ブロック名です。

common-block-object-list は、コンマで区切られた共通ブロック実体の並びです。

common-block-object は、以下の形式です。

```
variable-name [ ( explicit-shape-spec-list ) ]          または
proc-pointer-name
```

variable-name は、変数名です。

proc-pointer-name は、手続ポインタ名です。

共通ブロック実体は、仮引数、割付け変数、末端成分に割付け変数をもつ派生型の実体、自動割付け変数、関数名、入口名、BIND 属性をもつ変数、結果名、共配列、または参照結合によって参照可能になった名前を指定してはなりません。

explicit-shape-spec-list は、明示上下限並びです。*explicit-shape-spec* は、以下の形式です。

[*lower-bound* :] *upper-bound*

lower-bound は、下限であり、定数宣言式でなければなりません。*lower-bound* を省略したとき、その暗黙値は、1です。

upper-bound は、上限であり、定数宣言式でなければなりません。

explicit-shape-spec-list が指定されている場合、DIMENSION 属性をもつと宣言され、配列の性質を指定されます。そのような変数は、POINTER 属性をもってはなりません。

共通ブロック実体が派生型実体であるとき、その派生型は連続型またはBIND 属性をもつ型でなければならず、暗黙的初期値指定をもってはなりません。

各COMMON 文において、共通ブロック名に続く共通ブロック実体並び中に名前を書いたデータ実体は、その共通ブロック中にあると宣言されます。先頭の共通ブロック名を省略した場合、最初の共通ブロック実体並び中に名前を書いたデータ実体は、無名共通ブロック中にあると宣言されます。共通ブロック名を省略した2つの斜線に続く共通ブロック実体並び中に名前を書いたデータ実体もまた、無名共通ブロック中にあると宣言されます。

1つの有効域内において、1つ以上のCOMMON 文に、同じ名前をもつ共通ブロックまたは無名共通ブロックを2回以上書いてもかまいません。それらに続く共通ブロック実体並びは、その順に全部1つにつないだ並びとして扱われます。無名共通ブロックの共通ブロック実体並びも、その順に1つにつないだ並びとして扱われます。

各共通ブロックの記憶列は、共通ブロック実体並び中の、すべてのデータ実体の記憶列から形成されます。記憶列の順序は、その有効域内での共通ブロック実体並びの出現順と同じです。この記憶列は、EQUIVALENCE 文による記憶列共有結合で記憶列結合したすべての記憶単位を含むように拡張されます。この拡張は、最後の記憶単位の後ろに記憶単位を追加する方向にだけ許されます。

プログラム中で、同じ名前をもつ共通ブロックの、記憶列の先頭の記憶単位は、すべて同じ記憶単位となります。無名共通ブロックの、記憶列の先頭の記憶単位は、すべて同じ記憶単位となります。これにより、異なる有効域間での実体の結合を可能にします。

基本整数型、基本実数型、倍精度実数型、基本複素数型、基本論理型、または数値連続型であってポインタでない実体は、これらの型のポインタでない実体とだけ結合できます。

基本文字型または文字連続型であってポインタでない実体は、これらの型のポインタでない実体とだけ結合できます。

数値連続型および文字連続型以外の派生型であってポインタでない実体は、同じ型のポインタでない実体とだけ結合できます。

基本整数型、基本実数型、倍精度実数型、基本複素数型、基本論理型、および基本文字型以外の組込み型であってポインタでない実体は、同じ型および同じ型パラメタをもったポインタでない実体とだけ結合できます。

ポインタは、同じ型、型パラメタ、および次元数をもつポインタとだけ結合できます。

無名共通ブロックは、以下の事項を除いて、名前付き共通ブロックと同じ性質をもちます。

- RETURN 文またはEND 文の実行によって、名前付き共通ブロック中のデータ実体は、その共通ブロック名をSAVE 文で宣言した場合を除いて、不定になる場合があります。
- 同じ名前の名前付き共通ブロックは、プログラム中のすべての有効域で同じ大きさでなければなりません。
- 名前付き共通ブロック中のデータ実体は、初期値設定プログラム単位中でDATA 文または型宣言文によって初期確定にすることができます。

EQUIVALENCE 文によって、2つの異なる共通ブロックの記憶列を結合してはなりません。

EQUIVALENCE 文による記憶列共有結合によって、COMMON 文中で指定された先頭の実体の記憶単位よりも前に、記憶単位を付け加える拡張を行ってはなりません。

COMMON 文の例:

```
common /first/ a,b,c              ! a、b、およびc は共通ブロックfirst の中の実体です
common d,e,f, /second/ g          ! d、e、およびf は無名共通ブロック中の実体です
                                  ! g は共通ブロックsecond 中の実体です
common /first/ h                  ! h は共通ブロックfirst のc の後ろに追加されます
```

2.83 COMPILER_OPTIONS組込みモジュール関数

機能説明

翻訳時に有効なコンパイラオプションの情報を返却します。

返却するオプション情報については“Fortran使用手引書”を参照してください。

分類

問合せ関数

形式

result = COMPILER_OPTIONS ( )

result

基本文字型スカラです。

使用例

```
use,intrinsic::iso_fortran_env,only:compiler_options
print *,compiler_options()
```

2.84 COMPILER_VERSION組込みモジュール関数

機能説明

翻訳に使用したコンパイラのバージョン情報を返却します。

返却するバージョン情報については“Fortran使用手引書”を参照してください。

分類

問合せ関数

形式

result = COMPILER_VERSION ( )

result

基本文字型スカラです。

使用例

```
use,intrinsic::iso_fortran_env,only:compiler_version
print *,compiler_version()
```

2.85 COMPLEX型宣言文

COMPLEX 型宣言文は、複素数型のデータ実体を宣言します。

COMPLEX 型宣言文は、以下の形式です。

COMPLEX [*kind-selector*] [[, *attr-apec*]... ::] *entity-decl-list*

型宣言文の詳細については、“2.3 型宣言文”を参照してください。

2.86 CONJG組込み関数

CONJG 関数は、与えられた複素数データの共役複素数を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
CONJG	-----	1	複素数型	複素数型

総称名	個別名	引数の数	引数の型	結果の型
	CONJG		単精度複素数型	単精度複素数型
	DCONJG		倍精度複素数型	倍精度複素数型
	QCONJG		4倍精度複素数型	4倍精度複素数型

result = CONJG(*Z*)

Z

複素数型でなければなりません。

result

*Z*と同じ型です。

機能説明

CONJG、DCONJG、およびQCONJG は、与えられた複素数データと共役なものを求めます。

結果の値は、CMPLX(REAL(*Z*), -IMAG(*Z*))となります。

総称名CONJG は、すべての複素数型の引数に使用することができます。

それぞれの関数の結果の型は、*Z*と同じです。

使用例

```
complex :: x
x = conjg ((2.1, -3.2))              ! x には(2.1,3.2) が代入されます
```

2.87 CONTAINS文

CONTAINS 文は、主プログラム、モジュール、または副プログラムの本体を、それらが含む内部副プログラムまたはモジュール副プログラムから分離します。

CONTAINS 文は、以下の形式です。

```
CONTAINS
```

CONTAINS 文は、非実行文です。

CONTAINS 文は、初期値設定プログラム単位および内部副プログラム内に指定することはできません。

CONTAINS 文の例:

```
subroutine outside (a)
  implicit none
  real , intent(in) :: a
  integer :: i,j
  real :: x
    ...
  call inside (i)
  x = sin(3.89)                ! 内部手続sin の引用。組込み手続sin の引用ではありません。
    ...
  contains
  subroutine inside(k)         ! 内部サブルーチンinside は、外部サブルーチン
                               ! outside( ) の中だけで有効です。
    implicit none
    integer, intent(in) :: k
      ...
  end subroutine inside
  function sin (m)             ! 内部関数sin は、外部サブルーチンoutside( ) の
                               ! 中だけで有効です。
    implicit none
    real :: sin
```

```
    real , intent(in) :: m
      ...
  end function sin
end subroutine outside
```

2.88 CONTIGUOUS文

CONTIGUOUS 属性は、形状引継ぎ配列、ポインタ配列、または次元引継ぎに指定することができ、

- 形状引継ぎ配列に指定した場合、CONTIGUOUS であることを宣言します。
- ポインタ配列に指定した場合、CONTIGUOUS である指示先とだけポインタ結合することを宣言します。
- 次元引継ぎに指定した場合、CONTIGUOUS であることを宣言します。

CONTIGUOUS 文は、以下の形式です。

```
CONTIGUOUS [ :: ] object-name-list
```

object-name-list はコンマで区切られた実体名の並びです。

この文は、並び中の実体に対して、CONTIGUOUS 属性を指定します。*object-name* は、形状引継ぎ配列、ポインタ配列、または次元引継ぎでなければなりません。

大きさがゼロでない、かつ長さがゼロでない実体は、次の条件を満たす場合、CONTIGUOUS です。

- 実体がCONTIGUOUS 属性をもっている。または、
- ポインタ配列でもなく形状引継ぎ配列でもない、全配列である。または、
- 形状引継ぎ配列がCONTIGUOUS な実引数と結合している。または、
- 配列がALLOCATE 文で割り付けられている。または、
- ポインタがCONTIGUOUS な指示先と結合している。または、
- 次元引継ぎ仮引数の有効な実引数がCONTIGUOUS である。または、
- 部分配列であり、次の条件を満たす。
  - 実体がCONTIGUOUS である。かつ、
  - ベクトル添字が現れない。かつ、
  - 部分配列は、連続した部分集合をもっている。かつ、
  - 部分列が現れる場合は、文字型要素全体である。かつ、
  - 構造体成分の場合、右端が配列である。

CONTIGUOUS **文の例**:

```
integer,pointer,dimension(:,:)::c
contiguous::c
```

2.88.1 単純CONTIGUOUS

次の条件を満たすとき、配列は単純CONTIGUOUS です。

- 部分配列の場合、ベクトル添字が現れない。かつ、
- 部分配列の場合、添字三つ組は、最終の添字三つ組を除きコロン(:)である。かつ、
- 部分配列の場合、最後の添字三つ組に、刻み幅が現れない。かつ、
- 部分配列の場合、スカラ整数式の添字が添字三つ組より前に現れない。かつ、
- 文字部分列が現れない。かつ、

- 構造体成分の場合、右端が配列である。かつ、
- 以下の条件を満たす。
  - 実体はCONTIGUOUS 属性をもつ。または、
  - 配列実体はポインタ配列または形状引継ぎ配列ではない。

単純CONTIGUOUS の例:

```
integer,dimension(5,3):: array
 ...
array(2:4,3)              ! 単純CONTIGUOUS
array(:,2)                ! 単純CONTIGUOUS
```

2.89 CONTINUE文

CONTINUE 文の実行は、効果をもちません。

CONTINUE 文は、以下の形式です。

```
CONTINUE
```

CONTINUE 文の例:

```
    do 10 i=1,100
      a(i) = b(i,i)
10 continue
```

2.90 COS組込み関数

COS 関数は、ラジアン値を引数とする余弦を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
COS	-----	1	実数型または 複素数型	実数型または 複素数型
	COS		単精度実数型	単精度実数型
	DCOS		倍精度実数型	倍精度実数型
	QCOS		4倍精度実数型	4倍精度実数型
	CCOS		単精度複素数型	単精度複素数型
	CDCOS		倍精度複素数型	倍精度複素数型
	CQCOS		4倍精度複素数型	4倍精度複素数型

result = COS (*X*)

X

実数型または複素数型でなければなりません。

result

*X*と同じ型です。

機能説明

COS、DCOS、QCOS、CCOS、CDCOS、およびCQCOSは、実数型データまたは複素数型データの余弦を求めます。

引数はラジアン値を指定します。

単精度実数型の引数の場合、引数は ABS(*X*) < 8.23E+05 でなければなりません。

倍精度実数型の引数の場合、引数は DABS(*X*) < 3.53D+15 でなければなりません。

4倍精度実数型の引数の場合、引数は QABS(*X*) < 2.0Q0^{62} × π でなければなりません。

単精度複素数型の引数の場合、ABS(REAL(*X*)) < 8.23E+05 かつ ABS(IMAG(*X*)) < 89.415E0 でなければなりません。

倍精度複素数型の引数の場合、DABS(DREAL(*X*)) < 3.53D+15 かつ DABS(DIMAG(*X*)) < 710.475D0 でなければなりません。

4倍精度複素数型の引数の場合、QABS(QREAL(*X*)) < 2.0Q0^{62} × πかつ QABS(QIMAG(*X*)) < 11357.125Q0 でなければなりません。

総称名COS は、すべての実数型または複素数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
r = cos(.5)
```

2.91 COSD組込み関数

COSD 関数は、度数値を引数とする余弦を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
COSD	-----	1	実数型	実数型
	COSD		単精度実数型	単精度実数型
	DCOSD		倍精度実数型	倍精度実数型
	QCOSD		4倍精度実数型	4倍精度実数型

result = COSD (*X*)

X

実数型でなければなりません。

result

*X*と同じ型です。

機能説明

COSD、DCOSD、およびQCOSD は、実数型データの余弦を求めます。

引数は度数値を指定します。

結果の値は、COSD(*X*) = COS(π/180 × *X*) となります。

単精度実数型の引数の場合、引数は ABS(*X*) < 4.72E+07 でなければなりません。

倍精度実数型の引数の場合、引数は DABS(*X*) < 2.03D+17 でなければなりません。

4倍精度実数型の引数の場合、引数は QABS(*X*) < 2.0Q0^{62} × 180 でなければなりません。

総称名COSD は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
r = cosd(.5)
```

2.92 COSH組込み関数

COSH 関数は、実数型データの双曲線余弦を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
COSH	-----	1	実数型 または複素数型	実数型 または複素数型
	COSH		単精度実数型	単精度実数型
	DCOSH		倍精度実数型	倍精度実数型
	QCOSH		4倍精度実数型	4倍精度実数型

result = COSH (*X*)

X

実数型または複素数型でなければなりません。

result

*X*と同じ型です。

機能説明

COSH は、実数型データまたは複素数型データの双曲線余弦を求めます。DCOSH およびQCOSHは、実数型データの双曲線余弦を求めます。

単精度実数型の引数の場合、引数は ABS(*X*) < 89.415E0 でなければなりません。

倍精度実数型の引数の場合、引数は DABS(*X*) < 710.475D0 でなければなりません。

4倍精度実数型の引数の場合、引数は QABS(*X*) < 11357.125Q0 でなければなりません。

単精度複素数型の引数の場合、引数は ABS(REAL(*X*)) < 89.415E0 かつ ABS(AIMAG(*X*)) < 8.23E+05 でなければなりません。

倍精度複素数型の引数の場合、引数は DABS(DREAL(*X*)) < 710.475D0 かつ DABS(DIMAG(*X*)) < 3.53D+15 でなければなりません。

4倍精度複素数型の引数の場合、引数は QABS(QREAL(*X*)) < 11357.125Q0 かつ QABS(QIMAG(*X*)) < $2.0Q0^{62} \times \pi$ でなければなりません。

総称名COSH は、すべての実数型および複素数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
r = cosh(.5)
```

2.93 COSQ組込み関数

COSQ 関数は、象限値を引数とする余弦を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
COSQ	-----	1	実数型または 複素数型	実数型または 複素数型

総称名	個別名	引数の数	引数の型	結果の型
	COSQ		単精度実数型	単精度実数型
	DCOSQ		倍精度実数型	倍精度実数型
	QCOSQ		4倍精度実数型	4倍精度実数型
	CCOSQ		単精度複素数型	単精度複素数型
	CDCOSQ		倍精度複素数型	倍精度複素数型
	CQCOSQ		4倍精度複素数型	4倍精度複素数型

result = COSQ (*X*)

X

実数型または複素数型でなければなりません。

result

*X*と同じ型です。

機能説明

COSQ、DCOSQ、QCOSQ、CCOSQ、CDCOSQ、およびCQCOSQ は、実数型データまたは複素数型データの余弦を求めます。

引数は象限値を指定します。

結果の値は、COSQ(*X*) = COS(π /2 × *X*) となります。

単精度複素数型の引数の場合、ABS(REAL(*X*)) < 5.24E+05 かつ ABS(IMAG(*X*)) < 56.92E0 でなければなりません。

倍精度複素数型の引数の場合、DABS(DREAL(*X*)) < 2.25D+15 かつ DABS(DIMAG(*X*)) < 452.305D0 でなければなりません。

4倍精度複素数型の引数の場合、QABS(QREAL(*X*)) < 2.0Q0^{63}かつ QABS(QIMAG(*X*)) < 7230.125Q0 でなければなりません。

総称名COSQ は、すべての実数型および複素数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
r = cosq(.5)
```

2.94 COTAN組込み関数

COTAN 関数は、ラジアン値を引数とする余接を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
COTAN	-----	1	実数型	実数型
	COTAN		単精度実数型	単精度実数型
	DCOTAN		倍精度実数型	倍精度実数型
	QCOTAN		4倍精度実数型	4倍精度実数型

result = COTAN (*X*)

X

実数型でなければなりません。

result

*X*と同じ型です。

機能説明

COTAN、DCOTAN、およびQCOTAN は、実数型データの余接を求めます。

引数はラジアン値を指定します。

単精度実数型の引数の場合、引数は ABS(*X*) < 8.23E+05 でなければなりません。

倍精度実数型の引数の場合、引数は DABS(*X*) < 3.53D+15 でなければなりません。

4倍精度実数型の引数の場合、引数は QABS(*X*) < 2.0Q0^{62} × π でなければなりません。

総称名COTAN は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
r = cotan(.5)
```

2.95 COTAND組込み関数

COTAND 関数は、度数値を引数とする余接を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
COTAND	-----	1	実数型	実数型
	COTAND		単精度実数型	単精度実数型
	DCOTAND		倍精度実数型	倍精度実数型
	QCOTAND		4倍精度実数型	4倍精度実数型

result = COTAND (*X*)

X

実数型でなければなりません。

result

*X*と同じ型です。

機能説明

COTAND、DCOTAND、およびQCOTAND は、実数型データの余接を求めます。

引数は度数値を指定します。

結果の値は、COTAND(*X*) = COTAN(π /180 × *X*) となります。

単精度実数型の引数の場合、引数は ABS(*X*) < 4.72E+07 でなければなりません。

倍精度実数型の引数の場合、引数は DABS(*X*) < 2.03D+17 でなければなりません。

4倍精度実数型の引数の場合、引数は QABS(*X*) < 2.0Q0^{62} × 180 でなければなりません。

総称名COTAND は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
r = cotand(.5)
```

2.96 COTANQ組込み関数

COTANQ 関数は、象限値を引数とする余接を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
COTANQ	-----	1	実数型	実数型
	COTANQ		単精度実数型	単精度実数型
	DCOTANQ		倍精度実数型	倍精度実数型
	QCOTANQ		4倍精度実数型	4倍精度実数型

result = COTANQ(*X*)

X

実数型でなければなりません。

result

*X*と同じ型です。

機能説明

COTANQ、DCOTANQ、およびQCOTANQ は、実数型データの余接を求めます。

引数は象限値を指定します。

結果の値は、COTANQ(*X*) = COTAN($\pi/2 \times X$) となります。

総称名COTANQ は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
r = cotanq(.5)
```

2.97 COUNT組込み関数

COUNT 関数は、*MASK* の第*DIM* 次元にある真の要素の個数を数えます。

分類

変形関数

形式

result = COUNT (*MASK* [, *DIM* , *KIND*])

MASK

論理型です。スカラであってはなりません。

DIM（省略可能）

整数型スカラであって、1 <= *DIM* <= n の範囲の値でなければなりません。ここで、n は*MASK*の次元数とします。対応する実引数は、省略可能な仮引数であってはなりません。

KIND（省略可能）

スカラ整数型定数式でなければなりません。

result

整数型です。*KIND*が指定された場合、種別型パラメタは*KIND*の指定に従います。*KIND*が省略された場合、種別型パラメタは基本整数型になります。
*DIM*が省略されている、または*MASK*が1次元配列の場合は、スカラとなります。それ以外の場合は、(n-1)次元の配列となります。ここで、n は*MASK*の次元数とします。
形状は、*MASK*の形状を (d_1, d_2, ..., d_n) としたとき、(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_n) とします。

機能説明

COUNT は、*MASK*の第*DIM*次元に存在する真の要素の個数を返却します。

－ *DIM*を省略している場合

*MASK*中の真の要素の個数を返却します。

－ *DIM*を指定している場合

*MASK*が1次元の場合、結果の値はCOUNT(*MASK*) とします。
*MASK*が2次元以上の場合、結果の要素 (S_1, S_2, ..., S_{DIM-1}, S_{DIM+1}, ..., S_n) は、COUNT(*MASK*(S_1, S_2, ..., S_{DIM-1}, : , S_{DIM+1}, ..., S_n))となります。

関数の結果の型は、種別型パラメタ*KIND*が指定された場合、種別型パラメタ*KIND*の整数型となります。*KIND*が省略された場合、関数の結果の型は基本整数型です。

使用例

```
integer, dimension (2,3) :: a, b
integer, dimension (2) :: c
integer, dimension (3) :: d
integer :: e
a = reshape((/1,2,3,4,5,6/), (/2,3/))            ! a は  ┌     ┐
                                                 !       | 1 3 5 |
                                                 !       | 2 4 6 |
                                                 !       └     ┘
b = reshape((/1,2,3,5,6,4/), (/2,3/))            ! b は  ┌     ┐
                                                 !       | 1 3 6 |
                                                 !       | 2 5 4 |
                                                 !       └     ┘
e = count(a==b)                  ! e には3が代入されます
d = count(a==b, 1)               ! d には(/2,1,0/) が代入されます
c = count(a==b, 2)               ! c には(/2,1/) が代入されます
```

2.98 CO_MAX組込みサブルーチン

CO_MAX サブルーチンは各像の最大値を要素別に計算します。

すべての像の同じ文から呼び出さなければなりません。像制御文が許される文脈から呼び出さなければなりません。

分類

集団サブルーチン

形式

CALL CO_MAX (*A* [, *RESULT_IMAGE*, *STAT*, *ERRMSG*])

A

整数型、実数型、または文字型でなければなりません。すべての像において同じ型および型パラメタをもたなければなりません。INTENT(INOUT) 引数です。*A*がスカラの場合、*A*の値はすべての像の中で最大の*A*の値となります。*A*が配列の場合、*A*の各要素には、その要素のすべての像の中で最大の値がそれぞれ代入されます。*A*が配列の場合はすべての像において同じ形状でなければなりません。*RESULT_IMAGE*が指定された場合、*RESULT_IMAGE*を像番号とする像の*A*にのみ計算結果が代入され、他の像における*A*は不定になります。*RESULT_IMAGE*が指定されていない場合、すべての像の*A*に計算結果が代入されます。

RESULT_IMAGE **(省略可能)**

スカラの整数型でなければなりません。INTENT(IN) 引数です。すべての像において同じ値をもたなければなりません。

STAT (省略可能)

共添字付き実体でないスカラの整数型変数でなければなりません。INTENT(OUT) 引数です。実行が正常に終了した場合、変数には値0が設定されます。実行中に誤り条件が発生した場合、変数には以下のいずれかが設定されます。

- ISO_FORTRAN_ENV 組込みモジュールで定義されているSTAT_STOPPED_IMAGE の値
- 実行時の診断メッセージの番号

ERRMSG (省略可能)

共添字付き実体でない基本文字型のスカラ変数でなければなりません。INTENT(INOUT) 引数です。

使用例

```
integer, save :: k[*]
k = this_image()
sync all
call co_max(k) ! すべての像のkに、最大の像番号が代入されます
```

2.99 CO_MIN組込みサブルーチン

CO_MIN サブルーチンは各像の最小値を要素別に計算します。

すべての像の同じ文から呼び出さなければなりません。像制御文が許される文脈から呼び出さなければなりません。

分類

集団サブルーチン

形式

CALL CO_MIN (*A* [, *RESULT_IMAGE*, *STAT*, *ERRMSG*])

A

整数型、実数型、または文字型でなければなりません。すべての像において同じ型および型パラメタをもたなければなりません。INTENT(INOUT) 引数です。*A* がスカラの場合、*A* の値はすべての像の中で最小の*A* の値となります。*A* が配列の場合、*A* の各要素には、その要素のすべての像の中で最小の値がそれぞれ代入されます。*A* が配列の場合は、すべての像において同じ形状でなければなりません。*RESULT_IMAGE* が指定された場合、*RESULT_IMAGE* を像番号とする像の*A* にのみ計算結果が代入され、他の像における*A* は不定になります。*RESULT_IMAGE* が指定されていない場合、すべての像の*A* に計算結果が代入されます。

RESULT_IMAGE (省略可能)

スカラの整数型でなければなりません。INTENT(IN) 引数です。すべての像において同じ値をもたなければなりません。

STAT (省略可能)

共添字付き実体でないスカラの整数型変数でなければなりません。INTENT(OUT) 引数です。実行が正常に終了した場合、変数には値0が設定されます。実行中に誤り条件が発生した場合、変数には以下のいずれかが設定されます。

- ISO_FORTRAN_ENV 組込みモジュールで定義されているSTAT_STOPPED_IMAGE の値
- 実行時の診断メッセージの番号

ERRMSG (省略可能)

共添字付き実体でない基本文字型のスカラ変数でなければなりません。INTENT(INOUT) 引数です。

使用例

```
integer, save :: k[*]
k = this_image()
sync all
call co_min(k) ! すべての像のkに、最小の像番号が代入されます
```

2.100 CO_SUM組込みサブルーチン

CO_SUM サブルーチンはすべての像の要素を合計します。

すべての像の同じ文から呼び出さなければなりません。像制御文が許される文脈から呼び出さなければなりません。

分類

集団サブルーチン

形式

CALL CO_SUM (*A* [, *RESULT_IMAGE*, *STAT*, *ERRMSG*])

A

数値型でなければなりません。すべての像において同じ型および型パラメタをもたなければなりません。INTENT(INOUT) 引数です。*A* がスカラの場合、*A* の値はすべての像における*A* の値の合計となります。*A* が配列の場合、*A* の各要素には、すべての像の要素の値の合計がそれぞれ代入されます。*A* が配列の場合は、すべての像において同じ形状でなければなりません。*RESULT_IMAGE* が指定された場合、*RESULT_IMAGE* を像番号とする像の*A* にのみ計算結果が代入され、他の像における*A* は不定になります。*RESULT_IMAGE* が指定されていない場合、すべての像の*A* に計算結果が代入されます。

RESULT_IMAGE **(省略可能)**

スカラの整数型でなければなりません。INTENT(IN) 引数です。すべての像において同じ値をもたなければなりません。

STAT **(省略可能)**

共添字付き実体でないスカラの整数型変数でなければなりません。INTENT(OUT) 引数です。実行が正常に終了した場合、変数には値0が設定されます。実行中に誤り条件が発生した場合、変数には以下のいずれかが設定されます。

- ISO_FORTRAN_ENV 組込みモジュールで定義されているSTAT_STOPPED_IMAGE の値
- 実行時の診断メッセージの番号

ERRMSG **(省略可能)**

共添字付き実体でない基本文字型のスカラ変数でなければなりません。INTENT(INOUT) 引数です。

使用例

```
integer, save :: k[*]
k = this_image()
sync all
call co_sum(k) ! すべての像のkに、「1+…+最大の像番号」が代入されます
```

2.101 CPU_TIME組込みサブルーチン

CPU_TIME サブルーチンは、処理時間を返却します。処理時間はプログラムが実行されているプロセスとそのプロセス内の全スレッドで利用されたCPU 時間です。

分類

サブルーチン

形式

CALL CPU_TIME (*TIME*)

TIME

実数型スカラでなければなりません。INTENT(OUT) の引数です。プログラムの開始から処理系が要した処理時間が秒単位で設定されます。処理系が処理時間を取得できない場合、-1.0が設定されます。

使用例

```
call cpu_time(start_time)
x = cos(2.0)
call cpu_time(end_time)
cos_time = end_time - start_time          ! x=cos(2.0) の実行にかかった時間を計算します
```

2.102 CRITICAL構文

CRITICAL 構文は、複数の像が同時にブロックを実行することを制限します。

CRITICAL 構文は、以下の形式です。

```
[ critical-construct-name : ] CRITICAL
 block
END CRITICAL [ critical-construct-name ]
```

*critical-construct-name*はCRITICAL 構文名です。CRITICAL 文にCRITICAL 構文名を指定する場合、対応するEND CRITICAL 文にもCRITICAL 構文名を指定しなければなりません。CRITICAL 文にCRITICAL 構文名を指定しない場合、END CRITICAL 文にも指定してはなりません。

*block*はブロックです。ブロックは、0個以上の実行文または実行構文の並びです。ブロック中にRETURN文(“2.415 RETURN文”参照)、または像制御文(“1.18 像制御文”参照)を含んではなりません。CRITICAL 構文内からCRITICAL 構文の外へ飛び越したり、CRITICAL 構文の外からCRITICAL 構文内に飛び込んだりしてはなりません。

CRITICAL構文の実行中に手続呼び出しによって像制御文を実行してはなりません。

CRITICAL構文の例:

```
INTEGER, SAVE :: A[*]
A[1] = 0
K = THIS_IMAGE()
SYNC ALL
CRITICAL
  A[1] = A[1] + K  ! ある像がこの文の実行中に他の像がこの文を実行することはありません
END CRITICAL
```

2.103 CSHIFT組込み関数

CSHIFT 関数は、配列の要素に対して循環シフトを行います。

分類

変形関数

形式

result = CSHIFT (*ARRAY* , *SHIFT* [, *DIM*])

ARRAY

どの型でもかまいません。スカラであってはなりません。

SHIFT

整数型です。*ARRAY*が1次元のときはスカラでなければなりません。*ARRAY*が2次元以上のときは、スカラまたは次元の配列で、その形状は*ARRAY*の形状を($d_1, d_2, ..., d_n$)としたとき、($d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_n$)でなければなりません。ここで、nは*ARRAY*の次元数とします。

DIM(省略可能)

整数型スカラであって、1 <= *DIM*<= nの範囲の値でなければなりません。ここで、nは*ARRAY*の次元数とします。省略された場合は、1が指定されたものとみなします。

result

*ARRAY*と同じ型、同じ種別型パラメタおよび同じ形状です。

機能説明

CSHIFT は、配列の要素に対して循環シフトを行います。

－ *ARRAY*が1次元の場合

結果の要素I は、*ARRAY*(1+ MODULO(I + *SHIFT* -1, SIZE(*ARRAY*))) と等しいものとします。

－ *ARRAY*が2次元以上の場合

結果の部分配列 (S_1, S_2, ..., S_{DIM-1} , S_{DIM+1}, ..., S_n) の値は、CSHIFT(*ARRAY*(S_1, S_2, ..., S_{DIM-1}, : , S_{DIM+1}, ..., S_n), SHIFT = SH, DIM = 1) となります。ここでSH は、*SHIFT* がスカラの場合は*SHIFT*、配列の場合は *SHIFT*(S_1, S_2, ..., S_{DIM-1}, S_{DIM+1}, ..., S_n) とします。

使用例

```
integer, dimension (2,3) :: a, b
integer, dimension (3) :: c, d
a = reshape((/1,2,3,4,5,6/), (/2,3/))   ! a は     ┌       ┐
                                        !          | 1 3 5 |
                                        !          | 2 4 6 |
                                        !          └       ┘
c = (/1,2,3/)
b = cshift(a,1)                         ! b には   ┌       ┐
                                        !          | 2 4 6 |
                                        !          | 1 3 5 |
                                        !          └       ┘ が代入されます
b = cshift(a,-1,2)                      ! b には   ┌       ┐
                                        !          | 5 1 3 |
                                        !          | 6 2 4 |
                                        !          └       ┘ が代入されます
b = cshift(a,c,1)                       ! b には   ┌       ┐
                                        !          | 2 3 6 |
                                        !          | 1 4 5 |
                                        !          └       ┘ が代入されます
d = cshift(c,2)                         ! d には(/3,1,2/) が代入されます
```

2.104 CTIMEサービス関数

機能説明

システム時間をASCII コードの文字列に変換して、返却します。

形式

ch = CTIME (*time*)

time

基本整数型スカラ。システム時間を指定します。

関数結果

基本文字型スカラ。長さ24の文字列が返却されます。

使用例

```
use service_routines,only:ctime,time
character(len=24) :: ch
ch = ctime(time( ))
end
```

2.105 CYCLE文

CYCLE 文は、DO ループ中に指定し、残りのループ範囲の実行を1回飛び越します。

CYCLE 文は、以下の形式です。

```
CYCLE [ do-construct-name ]
```

do-construct-name は、DO 構文名です。

CYCLE 文はDO 構文の範囲内に指定しなければなりません。

DO 構文名は、そのCYCLE 文が含まれるDO 構文のDO 構文名でなければなりません。

CYCLE 文を実行すると、DO 構文名で識別されるDO 構文の、残りのループ範囲の実行を1回飛び越します。DO 構文名の指定がない場合、そのCYCLE 文を含むDO 構文のうち、もっとも内側のDO 構文の、残りのループ範囲の実行を1回飛び越します。

CYCLE 文の例:

```
outer: do i=1, 10
  inner: do j=1, 10
    if (i>ii) cycle outer
    if (j>jj) cycle          ! 内側のループの、残りのループ範囲の実行を飛び越します
    ...
  end do inner
end do outer
```

2.106 C_ASSOCIATED組込みモジュール関数

機能説明

C_ASSOCIATED 組込みモジュール関数は、C_PTR 型またはC_FUNPTR 型ポインタの結合状態を判定します。

C_PTR_1 の結合状態を示すか、または*C_PTR_1* と *C_PTR_2* が同じ要素と結合しているかどうかを示します。

分類

問合せ関数

形式

result = C_ASSOCIATED (*C_PTR_1* [, *C_PTR_2*])

C_PTR_1

C_PTR 型またはC_FUNPTR 型のスカラでなければなりません。

C_PTR_2(**省略可能**)

C_PTR_1 と同じ型のスカラでなければなりません。

result

基本論理型スカラです。

関数結果

－ *C_PTR_2* が省略されている場合

C_PTR_1 がC 言語の空ポインタであるときは偽です。それ以外のときは真です。

－ *C_PTR_2* が存在する場合

C_PTR_1 がC 言語の空ポインタであるときは偽です。それ以外のとき、*C_PTR_1* が*C_PTR_2* と等しければ真、そうでなければ偽です。

使用例

```
use , intrinsic :: iso_c_binding,only:c_funptr,c_funloc,c_associated
interface
function ifun() bind(c)
end function
end interface
type (c_funptr) :: cfunptr
cfunptr = c_funloc(ifun)
print *, c_associated( cfunptr )
```

2.107 C_FUNLOC組込みモジュール関数

機能説明

引数のC 言語アドレスを返却します。

分類

問合せ関数

形式

result = C_FUNLOC (*X*)

X

相互利用可能な手続または相互利用可能な手続と結合した手続ポインタでなければなりません。

result

C_FUNPTR 型のスカラです。

関数結果

結果は、暗黙的引用仕様の手続ポインタ成分をもつ派生型です。

この結果は、C_F_PROCPOINTER 組込みモジュールサブルーチンの呼出しの際、実引数CPTRとして使うことができます。

使用例

```
use , intrinsic :: iso_c_binding,only:c_funptr,c_funloc
interface
  function ifun () bind(c)
  end function
end interface
type (c_funptr) :: cfunptr
cfunptr = c_funloc(ifun)
```

2.108 C_F_POINTER組込みモジュールサブルーチン

機能説明

C 言語のポインタからFortran のデータポインタを生成します。

分類

サブルーチン

形式

CALL C_F_POINTER (*CPTR* , *FPTR* [, *SHAPE*])

CPTR

C_PTR 型のスカラでなければなりません。INTENT(IN) の引数です。その値は、次のいずれかでなければなりません。

- 相互利用可能なデータ要素のC 言語のアドレス
- 相互利用可能ではない引数をもったC_LOC 組込みモジュール関数への引用の結果

CPTR は、TARGET 属性をもたないFortran 変数のアドレスと同じであってはなりません。

FPTR

ポインタでなければなりません。INTENT(OUT) の引数です。

- *CPTR* の値が相互利用可能なデータ要素のC 言語のアドレスである場合、*FPTR* はその要素の型および相互利用可能な型および型パラメタをもつデータポインタでなければなりません。この場合、*FPTR* は*CPTR* の指示先とポインタ結合します。*FPTR* が配列のとき、その形状は*SHAPE* によって指定され、その下限は1です。

- *CPTR*の値が、相互利用可能ではない引数Xによる C_LOC(X) 組込みモジュール関数の引用の結果である場合、*FPTR*は、X と同じ型および型パラメタをもった非多相的なスカラポインタでなければなりません。この場合、X または X がポインタであるときその指示先は、解放されていてはなりません。*FPTR* は、X またはその指示先とポインタ結合します。

SHAPE（省略可能）

整数型の1次元配列です。INTENT(IN) の引数です。*FPTR*が配列のときに限り、*SHAPE*を指定しなければなりません。その要素数は、*FPTR* の次元と等しくなければなりません。

使用例

```
use , intrinsic :: iso_c_binding,only:c_loc,c_ptr,c_f_pointer
type(c_ptr) :: cptr
integer,pointer :: x,y
integer,target :: t
t=1
x=>t
cptr=c_loc(x)
call c_f_pointer ( cptr,y)
```

2.109 C_F_PROCPOINTER組込みモジュールサブルーチン

機能説明

C 言語の関数ポインタの指示先と手続ポインタを結合します。

分類

サブルーチン

形式

CALL C_F_PROCPOINTER (*CPTR* , *FPTR*)

CPTR

C_FUNPTR 型のスカラでなければなりません。INTENT(IN) の引数です。その値は、相互利用可能である手続のC 言語のアドレスでなければなりません。

FPTR

手続ポインタでなければなりません。INTENT(OUT) の引数です。*FPTR* に対する引用仕様は、*CPTR* の指示先と相互利用可能でなければなりません。*FPTR* は、*CPTR* の指示先とポインタ結合します。

使用例

```
use , intrinsic :: iso_c_binding,only:c_funptr,c_funloc,c_f_procpointer
interface
  function ifun() bind(c)
  end function
end interface
type (c_funptr) :: cfunptr
procedure(ifun), pointer :: pifun
cfunptr = c_funloc(ifun)
call c_f_procpointer(cfunptr, pifun)
```

2.110 C_LOC組込みモジュール関数

機能説明

引数のC 言語のアドレスを返却します。

分類

問合せ関数

形式

result = C_LOC (*X*)

X

TARGETまたはPOINTER属性をもたなければなりません。

以下のいずれかでなければなりません。

- 相互利用可能な型および型パラメタをもつ
- 長さ型パラメタをもたない多相的でないスカラ

ポインタは、結合していなければなりません。割付けは、割付けていなければなりません。

長さゼロであってはなりません。配列の場合、大きさゼロでないCONTIGUOUS(“2.88 CONTIGUOUS文”参照)でなければなりません。

共添字付き実体であってはなりません。

result

C_PTR 型のスカラです。

関数結果

*X*がスカラ型データ要素の場合、その結果は、C_PTR 型が*X*の型および型パラメタと一致するスカラポインタ要素をもつ派生型です。*X*が配列データ要素の場合、その結果は、C_PTR 型が*X*の型および型パラメタと一致するスカラポインタ要素をもつ派生型です。*X*が相互利用可能、または相互利用可能な型および型パラメタをもつデータ要素である場合、結果は、そのC 言語アドレスになります。

使用例

```
use , intrinsic :: iso_c_binding,only:c_ptr,c_loc
integer(kind=4), target :: t
type(c_ptr) :: cptr
cptr = c_loc(t)
```

2.111 C_SIZEOF組込みモジュール関数

機能説明

引数のサイズをバイト単位で返却します。

分類

問合せ関数

形式

result = C_SIZEOF (*X*)

X

相互利用可能なデータで、大きさ引継ぎ配列であってはなりません。

result

8バイトの整数型のスカラです。

関数結果

引数がスカラの場合、実体のサイズを返却します。引数が配列の場合、1要素のサイズと配列の要素数の乗数を返却します。

使用例

```
use,intrinsic::iso_c_binding,only:c_sizeof,c_int,c_size_t
integer(c_int), dimension(10) :: i
integer(c_size_t)             :: k
k = c_sizeof(i)    ! kには40が代入されます
print *,k
end
```

2.112 DATA文

DATA 文は、変数に初期値を与えます。

DATA 文は、以下の形式です。

```
DATA data-stmt-set [ [ , ] data-stmt-set ]...
```

data-stmt-set は、以下の形式です。

```
data-stmt-object-list / data-stmt-value-list /
```

data-stmt-object-list は、コンマで区切られた初期化項目の並びです。*data-stmt-object* は、以下の形式です。

```
variable                    または
data-implied-do
```

variable は変数です。

data-implied-do は初期化DO 形反復で、以下の形式です。

```
( data-i-do-object-list , [ integer-type-spec :: ] data-i-do-variable = scalar-int-expr , scalar-int-expr ■
■ [ , scalar-int-expr ] )
```

data-i-do-object-list は、コンマで区切られた初期化DO 形項目の並びです。*data-i-do-object* は、以下の形式です。

```
array-element                   または
scalar-structure-component      または
data-implied-do
```

array-element は、配列要素です。

scalar-structure-component は、スカラ構造体成分です。

integer-type-spec は、整数型指定子で以下の形式です。

```
INTEGER [ kind-selector ]
```

data-i-do-variable は、名前付きスカラ整変数でなければなりません。

scalar-int-expr は、定数、部分定数、および外側を囲む初期化DO 形反復のDO 変数だけを一次子とする、スカラ整数式でなければなりません。

1つの変数またはその一部分に対して、プログラム中で明示的初期値指定を2回以上指定してはなりません。ポインタでない実体または部分実体に対する型定義に暗黙的初期値指定があるとき、その実体または部分実体を初期化項目として書いてはなりません。

DATA 文中の前もって宣言されていない変数は、その後、暗黙の型規則を確認する型宣言に書いてもかまいません。DATA 文に書いた配列名、部分配列または配列要素は、その配列の性質が前もって宣言されていなければなりません。

名前付き共通ブロック中の変数を除いて、名前付き変数は、その一部分でもDATA 文で初期化されていれば暗黙的にSAVE 属性をもちます。

初期化項目並び中、または初期化DO 形反復並び中に書いた変数名は、仮引数、参照結合または親子結合によって参照可能な変数、初期値設定プログラム単位以外の名前付き共通ブロックの変数、無名共通ブロックの変数、関数名、関数結果名、自動割付け変数、割付け変数、または末端成分に割付け変数をもつ派生型の実体であってはなりません。

初期化項目である変数において、添字、部分配列添字、部分列開始位置、および部分列終了位置は、定数式でなければなりません。

初期化DO 形項目である配列要素またはスカラ構造体成分において、各添字は、定数、部分定数、およびその初期化DO 形反復または外側を囲むDO 形反復のDO 変数を一次子とする式でなければなりません。

data-stmt-value-list は、コンマで区切られた初期値表現の並びです。*data-stmt-value* は以下の形式です。

[*data-stmt-repeat* *] *data-stmt-constant*

data-stmt-repeat は初期化反復数であり、以下の形式です。

scalar-int-constant または
scalar-int-constant-subobject

scalar-int-constant は、スカラ整定数であり、値は正または0でなければなりません。

scalar-int-constant-subobject は、整数型のスカラ部分定数であり、値は正または0でなければなりません。

data-stmt-constant は、初期値定数であり、以下の形式です。

scalar-constant または
scalar-constant-subobject または
signed-int-literal-constant または
signed-real-literal-constant または
NULL( ) または
initial-data-target または
structure-constructor

scalar-constant は、スカラ定数です。

scalar-constant-subobject は、スカラ部分定数です。

signed-int-literal-constant は、任意符号付き整定数表現です。

signed-real-literal-constant は、任意符号付き実定数表現です。

initial-data-target は、初期指示先データです。特定子(“1.5.17 特定子”参照)を指定します。

structure-constructor は、構造体構成子です。

初期化項目並びは、展開されてポインタおよびスカラ変数の列になります。初期化項目並び中に書いた添字なしのポインタでない配列名は、配列要素順序に従った配列要素全体の列と等価です。部分配列も、配列要素順序に従ったその配列要素の列と等価です。初期化DO形反復は、DO 構文と同じ手順で、DO 変数の制御に従って展開され、配列要素または構造体成分の列になります。

初期値表現並びは、展開されて初期値定数の列になります。初期値反復数は、続く初期値定数を列の中に指定回数だけ繰り返します。省略したときは、反復数を1とします。

大きさ0の配列および繰返し回数が0であるDO形反復は、変数の展開列中に変数を与えませんが、長さ0のスカラ文字変数は、列中に変数を与えます。反復数が0である初期値定数は、スカラ初期値定数の展開列中に初期値定数を与えません。

変数の展開列と初期値定数の展開列とは、1対1に対応します。各初期値定数は、対応する変数の初期値、NULL( )、または初期指示先データを指定します。2つの展開列の長さは同じでなければなりません。

NULL( )、および初期指示先データでない初期値定数は、組込み代入の規則に従って、対応する変数に適合可能でなければなりません。その変数は、組込み代入の規則に従って初期値定数の値で初期確定になります。

変数がPOINTER 属性をもつとき、対応する初期値定数はNULL( )、または初期指示先データでなければなりません。

NULL( ) を指定した場合、そのポインタの初期の結合状態は空状態になります。

初期値指定に初期指示先データを指定したとき、その特定子は、割付け可能でなく、TARGET属性とSAVE属性をもたなければなりません。その特定子はベクトル添字をもたず、すべての添字、三つ組み添字、文字部分列開始位置、文字部分列終了位置は、定数式でなければなりません。ポインタ実体とその特定子は、データポインタ初期化適合でなければなりません。

ポインタ変数または成分が、ポインタと結合先で同じ型、同じ次元数、ポインタの無指定でない型パラメタが対応する結合先と同じ値、およびポインタがCONTIGUOUS属性をもつなら結合先がCONTIGUOUSなら、データポインタ初期化適合します。

DATA 文の例:

```
real :: a
integer ,dimension(-3:3) :: ary1
integer ,dimension(100) :: ary2
data a/3.78/ ,ary1/7*1/              ! a には3.78が、ary1 の各要素には1が、
                                     ! 初期値として与えられます
```

```
data (ary2(i),i=1,100)/100*6/          ! ary2 の各要素には6が、初期値として
                                       ! 与えられます
```

2.113 DATEサービスサブルーチン

機能説明

プログラムの実行年月日を次の形式で第1引数に指定した変数に返却します。

yy-mm-dd

yy ：西暦年の下位2けた
mm ：月
dd ：日

形式

CALL DATE (*ch*)

ch

長さ8の基本文字型スカラ。
引数*ch*の長さが8より大きい場合、空白が補われます。

使用例

```
use service_routines,only:date
character(len=10) :: dt
call date(dt)
write (6,fmt="(1x,a)") dt
end
```

2.114 DATE_AND_TIME組込みサブルーチン

DATE_AND_TIME サブルーチンは、日付と時間を返却します。

分類

サブルーチン

形式

CALL DATE_AND_TIME ([*DATE* , *TIME* , *ZONE* , *VALUES*])

DATE（省略可能）

基本文字型スカラでなければなりません。INTENT(OUT) の引数です。*DATE* の先頭から現在の日付がCCYYMMDD の形式で格納されます。CC は世紀、YY は西暦年、MMは月、DD は日です。

TIME（省略可能）

基本文字型スカラでなければなりません。INTENT(OUT) の引数です。*TIME* の先頭から現在の時間がhhmmss.sss の形式で格納されます。hh は時、mm は分、ss.sss は秒です。

ZONE（省略可能）

基本文字型スカラでなければなりません。INTENT(OUT) の引数です。*ZONE* の先頭から、UTC からの時差がshhmm の形式で格納されます。s は符号、hh は時、mm は分です。

VALUES（省略可能）

基本整数型の1 次元配列でなければなりません。INTENT(OUT) の引数です。各要素には以下の内容に示す値が格納されます。

VALUES(1)	年
VALUES(2)	月
VALUES(3)	日

VALUES(4)	分で表したUTC からの時差
VALUES(5)	時
VALUES(6)	分
VALUES(7)	秒
VALUES(8)	ミリ秒

使用例

```
! 2013年6月20日22時20分02.5秒に日本で使用した場合
integer :: dt(8)
character (len=10) :: time, date, zone
call date_and_time (date, time, zone, dt)
! date には"20130620" が代入されます
! time には"222002.500" が代入されます
! zone には"+0900" が代入されます
! dt には(/2013,6,20,540,22,20,2,500/) が代入されます
```

2.115 DBLE組込み関数

DBLE 関数は、数値型データを倍精度実数型に変換します。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
DBLE	-----	1	1バイトの整数型	倍精度実数型
	DFLOTI		2バイトの整数型	倍精度実数型
	DFLOAT		4バイトの整数型	倍精度実数型
	DFLOTJ		4バイトの整数型	倍精度実数型
	-----		8バイトの整数型	倍精度実数型
	DBLE		単精度実数型	倍精度実数型
	-----		実数型、複素数型または非10進定数表現	倍精度実数型
	-----		倍精度実数型	倍精度実数型
	DBLEQ		4倍精度実数型	倍精度実数型
	-----		単精度複素数型	倍精度実数型
	DREAL		倍精度複素数型	倍精度実数型
	-----		4倍精度複素数型	倍精度実数型

result = DBLE (*A*)

A

整数型、実数型、複素数型、または非10進定数表現でなければなりません。

result

倍精度実数型です。

機能説明

DBLE、DFLOTI、DFLOAT、DFLOTJ、DBLEQ、およびDREAL は、数値型データを倍精度実数型に変換します。

*A*が整数型または実数型の場合、*A*を倍精度実数型に変換した近似値を返却します。*A*が複素数型の場合、*A*の実部を倍精度実数型に変換した値の近似値を返却します。*A*が非10進定数表現の場合、倍精度実数型の変数が非10進定数表現によって指定されたビット列をもったときと同じ値を返却します。

関数の結果の絶対値がHUGE(1.0_8) を超えた場合、結果の値は不定となります。

総称名DBLE は、すべての整数型、実数型、複素数型、および非10進定数表現の引数に使用することができます。

それぞれの関数の結果の型は、倍精度実数型です。

使用例

```
double precision d
d = dble(1)                    ! d には1.0d0 が代入されます
```

2.116 DEALLOCATE文

DEALLOCATE 文は割付け変数またはポインタ指示先を解放し、ポインタを空状態にします。

DEALLOCATE 文は、以下の形式です。

```
DEALLOCATE ( allocate-object-list [ , dealloc-opt-list ] )
```

allocate-object-list は、コンマで区切られた割付け実体の並びです。

allocate-object は、以下の形式です。

```
variable-name                  または
structure-component
```

variable-name は、変数名です。

structure-component は、構造体成分です。

割付け実体は、ポインタまたは割付け変数でなければなりません。

dealloc-opt-list は、解放指定選択子並びです。解放指定選択子は、以下の形式です。

```
STAT = stat-variable           または
ERRMSG = errmsg-variable
```

stat-variable は状態変数であり、スカラ整変数でなければなりません。

errmsg-variable は誤り通報変数であり、スカラ基本文字変数でなければなりません。

STAT 指定子のあるDEALLOCATE 文の実行が正常に終了した場合、状態変数には値0が設定されます。DEALLOCATE 文の実行中に誤り条件が発生した場合、状態変数には実行時の診断メッセージの番号が設定されます。

誤り条件がALLOCATE 文またはDEALLOCATE 文の実行中に発生したとき、誤り通報変数に状態が代入されます。誤り条件が発生しなかったときは、誤り通報変数の値は変更されません。

STAT 指定子のないDEALLOCATE 文の実行中に誤り条件が発生した場合、その実行可能プログラムの実行は終了します。

割り付けられていない割付け変数を解放しようとすると、DEALLOCATE 文で誤り条件が発生します。TARGET 属性をもつ割付け変数は、結合されたポインタを通して解放してはなりません。TARGET 属性をもつ割付け変数を解放すると、その割付け変数と結合しているすべてのポインタの結合状態は、不定になります。

DEALLOCATE 文中にポインタがある場合、そのポインタの結合状態は確定でなければなりません。空状態のポインタまたはALLOCATE 文により生成されたのではない指示先と結合しているポインタを解放しようとすると、DEALLOCATE 文で誤り条件が発生します。

ポインタがTARGET 属性をもつ割付け変数と結合している場合、そのポインタを解放してはなりません。

ポインタ指示先を解放すると、その指示先またはその指示先の一部分と結合している他のポインタのポインタ結合状態は、不定になります。

割付け実体が共配列であるDEALLOCATE 文を実行したとき、すべての像が暗黙のうちに同期します。各々の像上で、すべての像が文を実行するまで、この文に続くセグメントの実行は遅延します。

DEALLOCATE 文の例:

```
deallocate (a,b,stat=s)      ! a およびb を解放します
                             ! 誤り条件が発生しなければ、
                             ! s に0が設定されます
```

2.117 DIGITS組込み関数

DIGITS 関数は、*X*を数体系の数として表現したときの有効けた数の値を返却します。

分類

問合せ関数

形式

result = DIGITS (*X*)

X

整数型または実数型でなければなりません。

result

基本整数型スカラです。

機能説明

DIGITS は、*X*を数体系の数として表現したときの有効けた数の値を返却します。

－ *X*が整数型の場合

全ビット数から符号ビットを引いた値(BIT_SIZE(*X*)-1)を返却します。

－ *X*が実数型の場合

小数部のビット数を返却します。

関数の結果の型は、基本整数型です。

以下の値の固定値となります。

引数の型	結果の値
1バイトの整数型	7
2バイトの整数型	15
4バイトの整数型	31
8バイトの整数型	63
半精度実数型	11
単精度実数型	24
倍精度実数型	53
4倍精度実数型	113

使用例

```
real :: r
integer :: i
i = digits(r)                ! i には24が代入されます
```

2.118 DIM組込み関数

DIM 関数は、2つの引数の差分を返却します。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
DIM	-----	2	1バイトの整数型 , 1バイトの整数型	1バイトの整数型
	I2DIM		2バイトの整数型 , 2バイトの整数型	2バイトの整数型
	IIDIM		2バイトの整数型 , 2バイトの整数型	2バイトの整数型
	IDIM		4バイトの整数型 , 4バイトの整数型	4バイトの整数型
	JIDIM		4バイトの整数型 , 4バイトの整数型	4バイトの整数型
	-----		8バイトの整数型 , 8バイトの整数型	8バイトの整数型
	-----		実数型	実数型
	DIM		単精度実数型 , 単精度実数型	単精度実数型
	DDIM		倍精度実数型 , 倍精度実数型	倍精度実数型
	QDIM		4倍精度実数型 , 4倍精度実数型	4倍精度実数型

result = DIM (*X* , *Y*)

X

整数型または実数型でなければなりません。

Y

*X*と同じ型および種別型パラメタでなければなりません。

result

*X*と同じ型です。

機能説明

DIM、I2DIM、IIDIM、IDIM、JIDIM、DDIM、およびQDIM は、2つの引数の差分を返却します。

*Y*が*X*よりも大きい場合は0を返却します。それ以外の場合は、*X* - *Y*を返却します。

総称名DIM は、すべての整数型または実数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
z = dim(1.1,0.8)                ! z には0.3が代入されます
z = dim(0.8,1.1)                ! z には0.0が代入されます
```

2.119 DIMENSION文

DIMENSION 文は、配列の形状を宣言します。

DIMENSION 文は、以下の形式です。

DIMENSION [::] *array-name* (*array-spec*) [, *array-name* (*array-spec*)]...

array-name は配列名です。

array-spec は、配列形状指定であり、以下の形式です。

explicit-shape-spec-list	または
assumed-shape-spec-list	または
deferred-shape-spec-list	または
assumed-size-spec	または
implied-shape-spec-list	または
assumed-rank-spec	

explicit-shape-spec-list は、明示上下限並びです。*explicit-shape-spec* は、以下の形式です。

[*lower-bound* :] *upper-bound*

lower-bound は、下限であり、宣言式でなければなりません。*lower-bound* を省略したとき、その暗黙値は1です。

upper-bound は、上限であり、宣言式でなければなりません。

upper-bound が *lower-bound* より小さいとき、添字範囲は空、その次元の寸法は0、配列の大きさは0とします。

explicit-shape-spec-list は、その配列の各次元の上下限を明示的な値で指定します。*explicit-shape-spec-list* で宣言される配列は、形状明示配列です。上下限が非定数式の値に依存する形状明示配列は、仮引数、関数結果、または自動割付け配列でなければなりません。

assumed-shape-spec-list は、引継ぎ上下限並びです。*assumed-shape-spec* は、以下の形式です。

[*lower-bound*] :

assumed-shape-spec-list で宣言される配列は、形状引継ぎ配列であり、結合される実引数配列から形状を引き継ぐ、ポインタでない仮配列です。

deferred-shape-spec-list は、無指定上下限並びです。*deferred-shape-spec* は、以下の形式(コロンのみ)です。

:

deferred-shape-spec-list で宣言される配列は、形状無指定配列であり、割付け配列または配列ポインタです。

assumed-size-spec は、大きさ引継ぎ配列形状指定であり、以下の形式です。

[*explicit-shape-spec-list* ,] [*lower-bound* :] *

assumed-size-spec で宣言される配列は、大きさ引継ぎ配列であり、結合される実引数から、その大きさを引き継ぐ仮配列です。

implied-shape-spec-list で宣言される配列は、暗黙形状配列です。暗黙形状配列については、"1.5.15 暗黙形状配列"を参照してください。

assumed-rank-spec で宣言される実体は、次元引継ぎ実体であり、*assumed-rank-spec* は以下の形式(2つのピリオド)です。"1.5.16 次元引継ぎ"を参照してください。

..

DIMENSION 文の例:

```
subroutine sub(e,f,i)
dimension a(3,2,1)              ! a は3 × 2 × 1 の大きさの形状明示配列です
dimension b(-3:3)               ! b は下限が-3、要素数7の形状明示配列です
dimension c(1:i)                ! c は下限が1、上限がi の自動割付け配列です
dimension d(:,:,:)              ! d は次元数3の形状無指定配列です
dimension e(:,:)                ! e は次元数2の形状引継ぎ配列です
dimension f(*)                  ! f は大きさ引継ぎ配列です
allocatable :: d
```

2.120 DO構文

DO 構文は、実行構文の列の繰返し実行を指定します。

DO 構文は、以下の形式です。

```
[ do-construct-name : ] DO [ label ] [ loop-control ]
    block
do-term-stmt
```

do-construct-name は、DO 構文名です。

label は文番号です。

loop-control は、DO 制御であり、以下の形式です。

```
[ , ] do-variable = scalar-expr , scalar-expr [ , scalar-expr ]      または
[ , ] WHILE ( scalar-logical-expr )                                  または
[ , ] UNTIL ( scalar-logical-expr )                                  または
[ , ] CONCURRENT forall-header
```

do-variable は、DO 変数です。DO 変数は整数型、基本実数型、または倍精度実数型の名前付きスカラ変数でなければなりません。基本実数型および倍精度実数型のDO 変数は廃止事項です。

*scalar-expr*は整数型、基本実数型、または倍精度実数型のスカラ式でなければなりません。基本実数型および倍精度実数型の*scalar-expr* は廃止事項です。

scalar-logical-expr は、スカラ論理式です。

forall-header はFORALL制御(“1.16 FORALL制御”参照)です。

*block*は、ブロックです。ブロックは、0個以上の実行文または実行構文の並びです。ブロックに実行文または実行構文が1つも含まれない場合、そのようなブロックの実行は、効果をもちません。

do-term-stmt はDO 端末文であり、以下の形式です。

```
END DO [ do-construct-name ]          または
CONTINUE                              または
action-stmt
```

action-stmt は、CONTINUE 文、GO TO 文、RETURN 文、STOP 文、EXIT 文、CYCLE 文、END 文、または算術IF 文であってはなりません。

DO文にDO 構文名を指定し、対応するDO 端末文がEND DO 文であればEND DO 文に同じDO 構文名を指定しなければなりません。DO 文にDO 構文名を指定しない場合、対応するDO 端末文にDO 構文名を指定してはなりません。

DO 文に文番号を指定する場合、対応するDO 端末文に同じ文番号を付けなければなりません。

DO 文に文番号を指定しない場合、対応するDO 端末文はEND DO 文でなければなりません。

DO端末文が、他のDO構文と共有していないEND DO 文またはCONTINUE 文であるDO 構文を、整構造DO 構文といいます。DO端末文が*action-stmt* であるか、またはDO 端末文を他のDO 構文と共有しているDO 構文を、不整構造DO 構文といいます。

DO 文を実行すると、そのDO 構文は活動状態になります。

DO 制御がDO 変数をもつ場合、DO 制御中のスカラ式がそれぞれDO 変数の型で評価され、始値m_1、限界値m_2、および増分値m_3 が定められます。3番目のスカラ式が省略されている場合、増分値m_3は1となります。DO 変数には始値m_1 が設定され、DO 変数の型が整数型の場合、MAX((m_2-m_1+m_3)/m_3,0)、実数型または倍精度実数型の場合、MAX(INT((m_2-m_1+m_3)/m_3),0)、で定められた繰返し数の回数だけDO ループの範囲が実行されます。繰返し数は0であってもかまいません。DO 変数は、ループ範囲が実行される度に増分値m_3 だけ増やされます。繰返し数分だけDO ループの範囲が実行された後、DO 構文は休止状態になります。

DO 制御を省略した場合、繰返し数が無限大の値の場合と同じです。

DO 制御が、[,] WHILE (*scalar-logical-expr*) の形である場合、DO 制御を省略して、ブロックの先頭に次の文を挿入したのと同じ効果をもちます。

IF (.NOT. (*scalar-logical-expr*)) EXIT

DO 制御が、[,] UNTIL (*scalar-logical-expr*) の形である場合、DO 制御を省略して、ブロックの最後に次の文を挿入したのと同じ効果をもちます。

IF (*scalar-logical-expr*) EXIT

CYCLE 文を使用して、残りのループ範囲の実行を1回飛び越すことができます(“2.105 CYCLE文”参照)。

EXIT 文を使用して、DO 構文の実行を終了させることができます(“2.172 EXIT文”参照)。

DO制御が、[,] CONCURRENT *forall-header* の形である場合、次の仕様が適用されます。

- 繰返しの実行順序は規定されません。
- 外側の構文に属するCYCLE文(“2.105 CYCLE文”参照)を指定することはできません。
- 外側の構文に分岐することはできません。
- EXIT文(“2.172 EXIT文”参照)、RETURN文(“2.415 RETURN文”参照)、またはSTOP文(“2.460 STOP文”参照)を指定することはできません。
- 組込みモジュールIEEE_EXCEPTIONS(“1.15.1.1 IEEE_EXCEPTIONS モジュール”参照)の手続IEEE_GET_FLAG(“2.241 IEEE_GET_FLAG組込みモジュールサブルーチン”参照)、IEEE_SET_HALTING_MODE(“2.257 IEEE_SET_HALTING_MODE組込みモジュールサブルーチン”参照)、またはIEEE_GET_HALTING_MODE(“2.242 IEEE_GET_HALTING_MODE組込みモジュールサブルーチン”参照)の引用は現れてはなりません。
- EXITサービスサブルーチン(“2.173 EXITサービスサブルーチン”参照)の引用は現れてはなりません。
- 引用する手続は、純粋でなければなりません。
- 参照される実体は、それ以前に値を定義しなければならず、他のいかなる繰返しも、定義してはなりません。
- 2つ以上の繰返しで定義される実体の値は、ループの終了で不定になります。
- ポインタが参照される場合、それ以前に結合しなければならず、いかなる繰返しの間も、結合状態を変更してはなりません。
- 2つ以上の繰返しでポインタの結合状態が変更される場合、結合状態は、ループの終了で不定になります。
- 2つ以上の繰返しで割り付けた割付け実体は、それと同じ繰返しで解放しなければなりません。1つの繰返しだけで割付け実体を割付けまたは解放した場合、異なる繰返しで実体を割付け、解放、定義、または参照してはなりません。
- 入出力文は、1つの繰返しでデータを出力し、異なった繰返しで同じ記録を入力してはなりません。

DO 構文の例:

```
do i=1,100 ! 100回繰り返します
  do while (a>b) ! a>b の間、繰り返します
    do 10 j=1,100,3 ! 34回繰り返します
      ...
  10 continue
  end do
end do
```

DO構文(DO CONCURRENT)の例:

```
real,dimension(5):: a,b,c
...
do concurrent ( k=1:5 )
  c(k) = a(k)+sqrt(b(k))
end do
```

2.121 DO文

DO 文は、DO 構文の開始を示します。

DO 文は、以下の形式です。

```
[ do-construct-name : ] DO [ label ] [ loop-control ]
```

do-construct-name は、DO 構文名です。

label は文番号です。

loop-control は、DO 制御であり、以下の形式です。

```
[ , ] do-variable = scalar-expr , scalar-expr [ , scalar-expr ]   または
[ , ] WHILE ( scalar-logical-expr )                               または
[ , ] UNTIL ( scalar-logical-expr )                               または
[ , ] CONCURRENT forall-header
```

do-variable は、DO 変数です。DO 変数は整数型、基本実数型、または倍精度実数型の名前付きスカラ変数でなければなりません。基本実数型および倍精度実数型のDO 変数は廃止事項です。

scalar-expr は整数型、基本実数型、または倍精度実数型のスカラ式でなければなりません。
基本実数型および倍精度実数型の*scalar-expr* は廃止事項です。

scalar-logical-expr は、スカラ論理式です。

forall-header はFORALL制御(“1.16 FORALL制御”参照)です。

DO 構文の詳細については、“2.120 DO構文”を参照してください。

2.122 DOT_PRODUCT組込み関数

DOT_PRODUCT 関数は、ベクトルの内積を行います。

分類

変形関数

形式

```
result = DOT_PRODUCT ( VECTOR_A , VECTOR_B )
```

VECTOR_A

数値型または論理型です。1次元配列でなければなりません。

VECTOR_B

VECTOR_A が数値型のときは数値型、論理型のときは論理型でなければなりません。1次元配列で、*VECTOR_A* と同じ大きさでなければなりません。

result

引数が数値型の場合、*VECTOR_A* * *VECTOR_B* の型および種別型パラメタとします。引数が論理型の場合、*VECTOR_A* .and. *VECTOR_B* の型および種別型パラメタとします。結果はスカラです。

機能説明

DOT_PRODUCT は、内積を行います。

－ *VECTOR_A* が整数型または実数型の場合

SUM(*VECTOR_A* * *VECTOR_B*) を返却します。大きさ0の配列の場合は、0を返却します。

－ *VECTOR_A* が複素数型の場合

SUM(CONJG(*VECTOR_A*) * *VECTOR_B*) を返却します。大きさ0の配列の場合は、(0.0, 0.0) を返却します。

－ *VECTOR_A* が論理型の場合

ANY(*VECTOR_A* .and. *VECTOR_B*) を返却します。大きさ0の配列の場合は、偽を返却します。

使用例

```
i = dot_product((/3,4,5/),(/6,7,8/)) ! i には86が代入されます
```

2.123 DOUBLE PRECISION型宣言文

DOUBLE PRECISION 型宣言文は、倍精度実数型のデータ実体を宣言します。

DOUBLE PRECISION 型宣言文は、以下の形式です。

```
DOUBLE PRECISION [ [ , attr-spec ]... :: ] entity-decl-list
```

型宣言文の詳細については、“2.3 型宣言文”を参照してください。

2.124 DPROD組込み関数

DPROD 関数は、2つの単精度実数型データを倍精度実数型に変換した上で乗算を行います。

分類

要素別処理関数

形式

result = DPROD (*X* , *Y*)

X

単精度実数型でなければなりません。

Y

単精度実数型でなければなりません。

result

倍精度実数型です。

機能説明

DPROD は、2つの単精度実数型データを倍精度実数型に変換した上で乗算を行います。

関数の結果は、DBLE(*X*) × DBLE(*Y*) となります。

関数の結果の型は、倍精度実数型です。

使用例

```
double precision :: dub
dub = dprod(3.e2,4.4e4) ! dub には13.2d6 が代入されます
```

2.125 DRANDサービス関数

機能説明

0.0から1.0までの乱数を発生させます。

形式

dy = DRAND (*i*)

i

基本整数型スカラ。

関数結果

倍精度実数型スカラ。*i* の値に従って、関数結果値が異なります。

=0　　：乱数列内の次の値が関数値として返却されます。
=1　　：乱数列内の最初の値が関数値として返却されます。
上記以外 ：乱数列発生のためのシード値として、引数の値が使われます。この新しく発生させられた乱数列の最初の値が関数値として返却されます。

使用例

```
use service_routines,only:drand
do i=1,10
  print *,drand(0)                ! 10個の乱数値を獲得
end do
end
```

2.126 DSHIFTL組込み関数

DSHIFTL 関数は、結合した左シフトをします。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
DSHIFTL	-----	3	整数型 または 非10進定数表現, 整数型 または 非10進定数表現, 整数型	整数型

result = DSHIFTL （ *I* , *J* , *SHIFT* ）

I

整数型または非10進定数表現でなければなりません。

J

整数型または非10進定数表現でなければなりません。

SHIFT

整数型でなければなりません。*SHIFT* <= BIT_SIZE(*I*) かつ *SHIFT* <= BIT_SIZE(*J*) でなければなりません。また、負の値であってはなりません。

result

I または *J* と同じ整数型です。

機能説明

DSHIFTL は、*I* と *J* を結合した左シフトをします。
IOR(SHIFTL(*I*, *SHIFT*) , SHIFTR (*J*, BIT_SIZE (*J*) - *SHIFT*)) と同じです。

I および *J* が整数型の場合、*I* と *J* は同じ型でなければなりません。

I または *J* のどちらかに非10進定数表現を指定した場合、非10進定数表現は、指定した整数型に変換されます。両方に非10進定数表現を指定してはなりません。

関数の結果の型は、*I* または *J* と同じ整数型です。

使用例

```
k = dshiftl(1,X'40000000',2)          ! k には5が代入されます
```

2.127 DSHIFTR組込み関数

DSHIFTR 関数は、結合した右シフトをします。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
DSHIFTR	-----	3	整数型 または 非10進定数表現, 整数型 または 非10進定数表現, 整数型	整数型

result = DSHIFTR (*I* , *J* , *SHIFT*)

I

整数型または非10進定数表現でなければなりません。

J

整数型または非10進定数表現でなければなりません。

SHIFT

整数型でなければなりません。*SHIFT* <= BIT_SIZE(*I*) かつ *SHIFT* <= BIT_SIZE(*J*) でなければなりません。また、負の値であってはなりません。

result

I または *J* と同じ整数型です。

機能説明

DSHIFTR は、*I* と *J* を結合した右シフトをします。
IOR(SHIFTL (*I*, BIT_SIZE (*I*) - *SHIFT*), SHIFTR (*J*, *SHIFT*)) と同じです。

I および *J* が整数型の場合、*I* と *J* は同じ型でなければなりません。

I または *J* のどちらかに非10進定数表現を指定した場合、非10進定数表現は、指定した整数型に変換されます。両方に非10進定数表現を指定してはなりません。

関数の結果の型は、*I* または *J* と同じ整数型です。

使用例

```
k = dshiftr( 1,16,3 )          ! k には536870914が代入されます
```

2.128 DTIMEサービス関数

機能説明

直前に呼ばれたDTIME サービス関数からのCPU 時間を返却します。CPU 時間はプログラムが実行されているプロセスとそのプロセス内の全スレッドで利用されたCPU 時間です。

形式

y = DTIME (*tm*)

tm

基本実数型配列。*tm*(1) にはユーザCPU 時間、*tm*(2) にはシステムCPU 時間が設定されます。
tm の要素数が2より少ない場合、実行動作は保証されません。*tm* の要素数が2より大きい場合、3要素目以降の値は変更されません。

関数結果

基本実数型スカラ。*tm*(1) と*tm*(2) の合計値が秒単位で返却されます。

エラーが発生した場合、*tm*(1)、*tm*(2) は不定となり、-1.0が返却されます。

返却される実行時間は、以下のようになります。

- 最初にDTIME サービス関数を実行した場合、実行開始時からのCPU 時間が返却されます。
- 2回目以降は、直前に実行されたDTIME サービス関数からのCPU 時間が返却されます。

使用例

```
use service_routines,only:dtime
real :: tm(2),y
tm = 1.0
y = dtime(tm)              ! 1回目の呼出し
do i=1,5000
  write(*,*) i,i*i
end do
y = dtime(tm)              ! 1回目の呼出しからのCPU 時間が返却されます
end
```

2.129 ELSE文

ELSE 文はIF 構文中に指定し、それ以前にあるすべてのブロックが実行されていない場合に、直後にあるブロックを実行することを指定します。

ELSE 文は、以下の形式です。

```
ELSE [ if-construct-name ]
```

if-construct-name は、IF 構文名です。

IF 構文の詳細については、“2.277 IF構文”を参照してください。

2.130 ELSE IF文

ELSE IF 文は、IF 構文中に指定し、直後にあるブロックの実行される条件を指定します。

ELSE IF 文は、以下の形式です。

```
ELSE IF ( scalar-logical-expr ) THEN [ if-construct-name ]
```

scalar-logical-expr は、スカラ論理式です。

if-construct-name は、IF 構文名です。

IF 構文の詳細については、“2.277 IF構文”を参照してください。

2.131 ELSEWHERE文

ELSEWHERE文はWHERE構文中に指定し、それ以前にある構造WHERE文およびELSEWHERE文の選別式の値が偽である要素の振る舞いを指定します。

ELSEWHERE 文は、以下の形式です。

```
ELSEWHERE ( mask-expr ) [ where-construct-name ]        または
ELSEWHERE [ where-construct-name ]
```

*mask-expr*は選別式であり、論理型の配列式でなければなりません。選別式の指定をもつELSEWHERE 文は、選別ELSEWHERE 文です。

where-construct-name は、WHERE 構文名です。

WHERE 構文の詳細については、“2.503 WHERE構文”を参照してください。

2.132 END文

END PROGRAM 文、END FUNCTION 文、END SUBROUTINE 文、END MODULE 文、およびEND BLOCK DATA 文をEND 文といいます。

END PROGRAM 文、END SUBROUTINE 文、END FUNCTION 文は実行文です。これらの文は、飛び先文としても指定することができます。END PROGRAM 文を実行すると、プログラムの実行が終了します。END SUBROUTINE 文またはEND FUNCTION 文の実行は、副プログラム中のRETURN 文の実行と同じ意味をもちます。

END MODULE 文およびEND BLOCK DATA 文は、非実行文です。

各文の詳細については、“2.147 END PROGRAM文”、“2.141 END FUNCTION文”、“2.151 END SUBROUTINE文”、“2.145 END MODULE文”、および“2.135 END BLOCK DATA文”を参照してください。

2.133 END ASSOCIATE文

END ASSOCIATE 文は、ASSOCIATE 構文の終了を指定します(“2.29 ASSOCIATE構文”参照)。

END ASSOCIATE 文は、以下の形式です。

```
END ASSOCIATE [ associate-construct-name ]
```

associate-construct-name は、ASSOCIATE 構文名です。

2.134 END BLOCK文

END BLOCK 文は、BLOCK 構文の終了を指定します(“2.57 BLOCK構文”参照)。

END BLOCK 文は、以下の形式です。

```
END BLOCK [ block-construct-name ]
```

block-construct-name は、BLOCK 構文名です。

2.135 END BLOCK DATA文

END BLOCK DATA 文は、初期値設定プログラム単位の終了を指定します。END BLOCK DATA 文は、非実行文です。初期値設定プログラム単位については、“1.11.4 初期値設定プログラム単位”を参照してください。

END BLOCK DATA 文は、以下の形式です。

```
END [ BLOCK DATA [ block-data-name ] ]
```

block-data-name は、初期値設定プログラム単位名です。

END BLOCK DATA 文に初期値設定プログラム単位名を指定する場合、対応するBLOCK DATA 文に指定された初期値設定プログラム単位名と同じでなければなりません。

END BLOCK DATA 文の例:

```
block data blkd
  common /com/ a,b,c
    integer :: a=1, b=2, c=3
end block data blkd
```

2.136 END CRITICAL文

END CRITICAL 文は、CRITICAL 構文の終了を指定します(“2.102 CRITICAL構文”参照)。

END CRITICAL 文は、以下の形式です。

```
END CRITICAL [ critical-construct-name ]
```

critical-construct-name は、CRITICAL 構文名です。

2.137 END DO文

END DO 文は、DO 構文の終了を指定します。

END DO 文は、以下の形式です。

```
END DO [ do-construct-name ]
```

do-construct-name は、DO 構文名です。

DO 構文の詳細については、“2.120 DO構文”を参照してください。

2.138 END ENUM文

END ENUM 文は、列挙体定義の終了を指定します。列挙体定義の詳細は“1.5.11.7 列挙体および列挙子”を参照してください。

END ENUM 文は、以下の形式です。

```
END ENUM
```

2.139 ENDFILE文

ENDFILE 文は、そのファイルの直後記録として、ファイル終了記録を書き、ファイルをファイル終了記録の後ろに位置付けます。

ENDFILE 文の実行は、指定した装置における非同期データ転送操作のすべてに対して、待機操作を行います。

ENDFILE 文は、以下の形式です。

```
ENDFILE external-file-unit                    または
ENDFILE ( position-spec-list )
```

external-file-unit は外部ファイル装置であり、スカラ整数式でなければなりません。

position-spec-list は、コンマで区切られた位置付け指定子の並びです。

position-spec は、以下の形式です。

```
[ UNIT= ] external-file-unit          または
IOMSG = iomsg                         または
IOSTAT= io-stat                       または
ERR= err-label
```

位置付け指定子並びにおいて、UNIT 指定子は必ず1つ指定しなければならず、他の指定子はそれぞれ1つ指定することができます。

文字列‘UNIT=’をUNIT 指定子から省略する場合、UNIT 指定子は、位置付け指定子並びの最初の項目でなければなりません。

*iomsg*はスカラ基本文字変数でなければなりません。入出力文の実行中に誤り条件が発生した場合、*iomsg*には説明のためのメッセージが代入されます。

誤り条件が発生しない場合、*iomsg* の値は変更されません。

io-stat は、スカラ整変数でなければなりません。

IOSTAT 指定子が指定された場合、IOSTAT 指定子に指定された変数には、以下の値が設定されます。

- 誤り条件、ファイル終了条件、および記録終了条件が検出されない場合は、0

・ 誤り条件が検出された場合は、1または実行時の診断メッセージの番号

err-label は文番号であり、このENDFILE 文と同じ有効域内の飛び先文の文番号でなければなりません。

ERR 指定子が指定され、ENDFILE 文の実行中に誤り条件が検出された場合、ERR 指定子に指定された文番号をもつ文が次に実行されます。

ENDFILE 文を実行した後でデータ転送入出力文またはENDFILE 文を実行するには、まずBACKSPACE 文(“2.42 BACKSPACE文”参照)またはREWIND 文(“2.416 REWIND文”参照)を使ってファイルを再位置付けしなければなりません。

流れ探査として接続されているファイルにENDFILE 文を実行すると、ファイルの終点は現在のファイル位置のままになります。現在の位置より前のファイル記憶単位だけが、すでに書かれたものとみなされ、以後はこれらのファイル記憶単位だけを読むことができます。以後の流れ出力文で、このファイルに更にデータを書くこともできます。

接続されていて存在しないファイルに対してENDFILE 文を実行すると、ファイル終了記録を書く前にファイルが生成されます。

直接探査として接続されているファイルを、ENDFILE 文で参照してはなりません。ACTION 指定子の値がREAD であるファイルを、ENDFILE 文で参照してはなりません。

ENDFILE 文の例:

```
endfile 10        ! 装置番号10に接続されているファイルにファイル終了記録を書きます
```

2.140 END FORALL文

END FORALL 文は、FORALL 構文の終了を指定します。

END FORALL 文は、以下の形式です。

```
END FORALL [ forall-construct-name ]
```

forall-construct-name は、FORALL 構文名です。

FORALL 構文の詳細については、“2.187 FORALL構文”を参照してください。

2.141 END FUNCTION文

END FUNCTION 文は、関数副プログラムの終了を指定します。END FUNCTION 文は実行文であり、飛び先文としても指定することができます。END FUNCTION 文の実行は、関数副プログラム中のRETURN 文の実行と同じ意味をもちます。

END FUNCTION 文は以下の形式です。

```
END [ FUNCTION [ function-name ] ]
```

モジュール関数および内部関数のEND FUNCTION 文は、キーワード‘FUNCTION’を指定しなければなりません。

function-name は、関数名です。

END FUNCTION 文に関数名を指定する場合、対応するFUNCTION 文に指定された関数名と同じでなければなりません。

関数副プログラムの詳細については、“1.12.1 関数副プログラム”を参照してください。

END FUNCTION 文の例:

```
function foo(i,j)
  integer :: foo,i,j
  foo =i*(j+2)
end function foo
```

2.142 END IF文

END IF 文は、IF 構文の終了を指定します。

END IF 文は、以下の形式です。

```
END IF [ if-construct-name ]
```

if-construct-name は、IF 構文名です。

IF 構文の詳細については、“2.277 IF構文”を参照してください。

2.143 END INTERFACE文

END INTERFACE 文は、引用仕様宣言の終了を指定します。

END INTERFACE 文は、以下の形式です。

```
END INTERFACE [ generic-spec ]
```

generic-spec は、総称指定であり、以下の形式です。

```
generic-name                          または
OPERATOR ( defined-operator )         または
ASSIGNMENT ( = )                      または
dtio-generic-spec
```

generic-spec を指定する場合、対応するINTERFACE文で指定された総称指定と同じでなければなりません。

generic-name は、総称名です。

defined-operator は利用者定義演算子であり、以下の形式です。

```
intrinsic-operator                    または
. operator-name .
```

intrinsic-operator は組込演算子です。

operator-name は、利用者が定義した240文字までの利用者定義演算子の名前です。

ASSIGNMENT(=) は、利用者定義代入です。

dtio-generic-spec は派生型入出力手続であり、以下の形式です。

```
READ ( FORMATTED )                    または
READ ( UNFORMATTED )                  または
WRITE ( FORMATTED )                   または
WRITE ( UNFORMATTED )
```

引用仕様宣言の詳細については、“1.12.7.2 手続引用仕様宣言”を参照してください。

END INTERFACE 文の例:

```
interface                             ! 総称指定のないINTERFACE 文
  subroutine ex(a,b,c)
    implicit none
    real, dimension (2,8) :: a,b,c
    intent(in) :: a
    intent(out) :: b
  end subroutine ex
  function why (t,f)
    implicit none
    logical, intent(in) :: t,f
    logical :: why
  end function why
end interface

interface swap                        ! 総称名swap
  subroutine real_swap(x,y)
```

```
    implicit none
    real , intent(inout) :: x,y
  end subroutine real_swap
  subroutine int_swap(x,y)
    implicit none
    integer, intent(inout) :: x,y
  end subroutine int_swap

end interface swap

interface operator (*)                    ! * 演算子の拡張
  function set_intersection(a,b)
    use set_module                        ! 派生型set が定義されているモジュール
    implicit none
    type(set) , intent(in) :: a,b
    type(set) :: set_intersection
  end function set_intersection
end interface operator(*)

interface assignment(=)                   ! 代入の拡張
  subroutine integer_to_bit (n,b)
    implicit none
    logical, intent(out) :: n
    integer, intent(in) :: b(:)
  end subroutine integer_to_bit
end interface assignment(=)
```

2.144 END MAP文

END MAP 文は、STRUCTURE 文による派生型定義内において、共用体宣言のブロックを終了します。派生型定義の詳細については、“1.5.11.1 派生型定義”を参照してください。

END MAP 文は、以下の形式です。

```
END MAP
```

END MAP 文の例:

```
structure /complex_element/
  union
    map
      real :: real,imag
    end map
    map
      complex :: complex
    end map
  end union
end structure
record /complex_element/ x
x%real = 2.0
x%imag = 3.0
print *,x%complex          ! 複素数型の要素complex は、(2.0,3.0) で確定しています
```

2.145 END MODULE文

END MODULE 文は、モジュールの終了を指定します。END MODULE 文は、非実行文です。

END MODULE 文は、以下の形式です。

```
END [ MODULE [ module-name ] ]
```

module-name は、モジュール名です。

END MODULE 文にモジュール名を指定する場合、MODULE 文に指定されたモジュール名と同じでなければなりません。

モジュールの詳細については、“1.11.2 モジュール”を参照してください。

END MODULE 文の例:

```
module mod
  implicit none
    type mytype
      real :: a,b(2,4)
      integer :: n,o,p
    end type mytype
end module mod

subroutine zee()
  use mod
  implicit none
  type(mytype) :: bee, dee
    ...
end subroutine zee
```

2.146 END MODULE PROCEDURE副プログラム文

END MODULE PROCEDURE 副プログラム文は、MODULE PROCEDURE 副プログラムの終了を指定します。END MODULE PROCEDURE 副プログラム文は、非実行文です。

END MODULE PROCEDURE 副プログラム文は、以下の形式です。

END [PROCEDURE [*procedure-name*]]

procedure-name は、手続名です。

END MODULE PROCEDURE 副プログラム文に手続名を指定する場合、MODULE PROCEDURE 副プログラム文に指定された手続名と同じでなければなりません。

MODULE PROCEDURE 副プログラムの詳細については、“1.11.5 MODULE PROCEDURE副プログラム”を参照してください。

END MODULE PROCEDURE 副プログラム文の例:

```
module m
  interface
    module subroutine sub(a)
    end subroutine
  end interface
end
submodule(m)submod
  contains
    module procedure sub
      print *,a
    end procedure sub
end submodule submod
```

2.147 END PROGRAM文

END PROGRAM 文は、主プログラムの終了を指定します。END PROGRAM 文は実行文であり、飛び先文としても指定することができます。END PROGRAM 文を実行すると、プログラムの実行が終了します。

END PROGRAM 文は、以下の形式です。

```
END [ PROGRAM [ program-name ] ]
```

program-name は、プログラム名です。

END PROGRAM 文にプログラム名を指定する場合、対応するPROGRAM 文に指定されたプログラム名と同じでなければなりません。

主プログラムの詳細については、“1.11.1 主プログラム”を参照してください。

END PROGRAM 文の例:

```
program main
  ...
end program main
```

2.148 END SELECT文

END SELECT 文は、CASE 構文またはSELECT TYPE 構文の終了を指定します(“2.63 CASE構文”、および“2.431 SELECT TYPE構文”参照)。

END SELECT 文は、以下の形式です。

```
END SELECT [ case-construct-name ]
```

case-construct-name は、CASE 構文名またはSELECT TYPE 構文名です。

2.149 END STRUCTURE文

END STRUCTURE 文は、STRUCTURE 文による派生型定義の終了を指定します。派生型定義の詳細については、“1.5.11.1 派生型定義”を参照してください。

END STRUCTURE 文は、以下の形式です。

```
END STRUCTURE
```

END STRUCTURE 文の例:

```
structure /complex_element/
  union
    map
      real :: real,imag
    end map
    map
      complex :: complex
    end map
  end union
end structure
record /complex_element/ x
x%real = 2.0
x%imag = 3.0
print *,x%complex              ! 複素数型の要素complex は、(2.0,3.0) で確定しています
```

2.150 END SUBMODULE文

END SUBMODULE 文は、サブモジュールの終了を指定します。END SUBMODULE 文は、非実行文です。

END SUBMODULE 文は、以下の形式です。

```
END [ SUBMODULE [ submodule-name ] ]
```

submodule-name は、サブモジュール名です。

END SUBMODULE 文にサブモジュール名を指定する場合、SUBMODULE 文に指定されたサブモジュール名と同じでなければなりません。

サブモジュールの詳細については、“1.11.3 サブモジュール”を参照してください。

END SUBMODULE 文の例:

```
submodule(mod)submod
  contains
    module subroutine sub1(val)
      integer, intent(in) :: val
    end subroutine sub1
end submodule submod
```

2.151 END SUBROUTINE文

END SUBROUTINE 文は、サブルーチン副プログラムの終了を指定します。END SUBROUTINE文は実行文であり、飛び先文としても指定することができます。END SUBROUTINE 文の実行は、副プログラム中のRETURN 文の実行と同じ意味をもちます。

END SUBROUTINE 文は、以下の形式です。

```
END [ SUBROUTINE [ subroutine-name ] ]
```

モジュールサブルーチンおよび内部サブルーチンのEND SUBROUTINE 文は、キーワード‘SUBROUTINE’を指定しなければなりません。

subroutine-name は、サブルーチン名です。

END SUBROUTINE 文にサブルーチン名を指定する場合、対応するSUBROUTINE 文に指定されたサブルーチン名と同じでなければなりません。

サブルーチン副プログラムの詳細については、“1.12.2 サブルーチン副プログラム”を参照してください。

END SUBROUTINE 文の例:

```
pure subroutine zee ( var1 , var2 )
  ...
end subroutine zee
```

2.152 END TYPE文

END TYPE 文は、TYPE 文による派生型定義の終了を指定します。派生型定義の詳細については、“1.5.11.1 派生型定義”を参照してください。

END TYPE 文は、以下の形式です。

```
END TYPE [ type-name ]
```

type-name は、派生型名です。

END TYPE 文に派生型名を指定する場合、対応するTYPE 文に指定された派生型名と同じでなければなりません。

END TYPE 文の例:

```
type coordinates
  real :: x,y = 40.0          ! y には、暗黙的初期値指定として40.0を設定します
end type coordinates
```

2.153 END UNION文

END UNION 文は、STRUCTURE 文による派生型定義内において、共用体宣言を終了します。派生型定義の詳細については、“1.5.11.1 派生型定義”を参照してください。

END UNION 文は、以下の形式です。

```
END UNION
```

END UNION 文の例:

```
structure /complex_element/
  union
    map
      real :: real,imag
    end map
    map
      complex :: complex
    end map
  end union
end structure
record /complex_element/ x
x%real = 2.0
x%imag = 3.0
print *,x%complex          ! 複素数型の要素complex は、(2.0,3.0) で確定しています
```

2.154 END WHERE文

END WHERE 文は、WHERE 構文の終了を指定します。

END WHERE 文は、以下の形式です。

```
END WHERE [ where-construct-name ]
```

where-construct-name は、WHERE 構文名です。

WHERE 構文の詳細については、“2.503 WHERE構文”を参照してください。

2.155 ENTRY文(廃止予定事項)

ENTRY 文は、そのENTRY 文を含む関数副プログラム内またはサブルーチン副プログラム内の特定の実行文から実行を開始するような手続引用を可能にします。ENTRY 文は外部副プログラムまたはモジュール副プログラムの中にだけ書くことができます。

ENTRY 文は、以下の形式です。

```
ENTRY entry-name [ ( [ dummy-arg-list ] ) [ suffix ] ]
```

entry-name は、入口名です。

dummy-arg-list は、コンマで区切られた仮引数の並びです。

dummy-arg は、以下の形式です。

```
dummy-arg-name                    または
*
```

dummy-arg-name は、仮引数名です。

* の指定は、ENTRY 文がサブルーチン副プログラムに含まれている場合にだけ指定することができます。

suffix は、次のいずれかです。

```
手続言語束縛指定子 [ RESULT ( result-name ) ]          または
RESULT ( result-name ) [ 手続言語束縛指定子]
```

result-name は、結果名です。

ENTRY 文は、実行構文の中に書いてはなりません。

RESULT 句は、ENTRY 文が関数副プログラムに含まれている場合にだけ指定することができます。RESULT 句を指定する場合には、その関数副プログラムの有効域内のいかなる単純宣言文にも、型宣言文にも、入口名を書いてはなりません。結果名は入口名と同じであってはなりません。

ENTRY 文が関数副プログラムに含まれている場合には、その副プログラムに追加の関数が定義されます。その関数の名前は指定した入口名とし、その結果変数はRESULT 句の指定がない場合にはその入口名となり、RESULT 句の指定がある場合には結果名となります。関数結果の特性は、結果変数の指定によって指定されます。ENTRY 文に書いた関数結果の特性が、FUNCTION 文に書いた関数結果の特性と同じである場合には、それらの結果変数は同じ変数を識別します。

それ以外の場合には、それらの結果変数は記憶域結合され、すべてがPOINTER 属性をもたないスカラでなければなりません。

ENTRY 文がサブルーチン副プログラムに含まれている場合には、その副プログラムに追加のサブルーチンが定義されます。そのサブルーチンの名前は、指定した入口名です。

ENTRY 文の例:

```
program main
  i=2
  call square(i)
  j=2
  call quad(j)
  print *, i,j            ! 4 16 が出力されます
end program main

subroutine quad(k)
  k=k*k
entry square(k)
  k=k*k
  return
end subroutine quad
```

2.156 ENUM文

ENUM 文は、列挙体定義の開始を指定します。列挙体定義の詳細は“1.5.11.7 列挙体および列挙子”を参照してください。

ENUM 文は、以下の形式です。

```
ENUM , BIND ( C )
```

2.157 ENUMERATOR文

ENUMERATOR 文は、列挙体定義の要素であり、列挙子を定義します。列挙体定義の詳細は“1.5.11.7 列挙体および列挙子”を参照してください。

ENUMERATOR 文は、以下の形式です。

```
ENUMERATOR [::] named-constant [ = scalar-int-initialization-expr ]
```

named-constant は列挙子並びであり、基本整数型の名前付き定数です。

scalar-int-initialization-expr はスカラ整数定数式です。

列挙子に= を書いたとき、列挙子並びの前に、区切りの2連コロンを書かなければなりません。

2.158 EOSHIFT組込み関数

EOSHIFT 関数は、配列の要素に対して切捨てシフトを行います。

分類

変形関数

形式

```
result = EOSHIFT ( ARRAY , SHIFT [ , BOUNDARY , DIM ] )
```

ARRAY

どの型でもかまいません。スカラであってはなりません。

SHIFT

整数型です。*ARRAY*が1次元のときはスカラでなければなりません。*ARRAY*が2次元以上のときは、スカラまたは(n-1)次元の配列で、その形状は*ARRAY*の形状を (d_1, d_2, ..., d_n) としたとき、(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1} , ..., d_n) でなければなりません。

BOUNDARY（省略可能）

*ARRAY*と同じ型でなければなりません。*ARRAY*が1次元のときはスカラでなければなりません。*ARRAY*が2次元以上のときは、スカラまたは(n-1)次元の配列で、その形状は*ARRAY*の形状を(d_1, d_2, ..., d_n) としたとき、(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1} , ..., d_n) でなければなりません。

省略された場合は、以下の値が設定されたものとみなします。*ARRAY*が派生型の場合、*BOUNDARY*は省略できません。

ARRAY の型	*BOUNDARY* の値
整数型	0
実数型	0.0
複素数型	(0.0,0.0)
論理型	.false.
文字型	LEN(*ARRAY*)個の空白

DIM（省略可能）

整数型スカラであって、1 <= *DIM*<= n の範囲の値でなければなりません。ここで、n は*ARRAY*の次元数とします。省略された場合は、1が指定されたものとみなします。

result

*ARRAY*と同じ型、同じ種別型パラメタ、および同じ形状です。

機能説明

EOSHIFT は、配列の要素に対して切り捨てシフトを行います。

結果の要素 (S_1, S_2, ..., S_n)は、*ARRAY*(S_1, S_2, ..., S_{DIM-1}, S_{DIM+SH}, S_{DIM+1}, ..., S_n) とします。
SH は、LBOUND(*ARRAY, DIM*) <= S_{DIM+SH} <= UBOUND(*ARRAY, DIM*) をみたすとき、
SHIFT または *SHIFT*(S_1,S_2, ..., S_{DIM-1}, $S_{DIM+1,}$ ..., S_n) とし、それ以外のときは、
BOUNDARY または *BOUNDARY*(S_1,S_2, ..., S_{DIM-1}, $S_{DIM+1,}$ ..., S_n) とします。

使用例

```
integer, dimension (2,3) :: a, b
integer, dimension (3) :: c, d
a = reshape((/1,2,3,4,5,6/), (/2,3/))       ! a は ┌       ┐
                                            !      | 1 3 5 |
                                            !      | 2 4 6 |
                                            !      └       ┘
c = (/1,2,3/)
b = eoshift(a,1)                            ! b には  ┌       ┐
                                            !         | 2 4 6 |
                                            !         | 0 0 0 |
                                            !         └       ┘ が代入されます
b = eoshift(a,-1,0,2)                       ! b には  ┌       ┐
                                            !         | 0 1 3 |
                                            !         | 0 2 4 |
                                            !         └       ┘ が代入されます
b = eoshift(a,-c,1)                         ! b には  ┌       ┐
                                            !         | 1 1 1 |
                                            !         | 1 1 1 |
                                            !         └       ┘ が代入されます
d = eoshift(c,2)                            ! d には(/3, 0, 0/) が代入されます
```

2.159 EPSILON組込み関数

EPSILON 関数は、*X*を数体系の数として表現したときに、1に対して無視できる値を返却します。

分類

問合せ関数

形式

result = EPSILON (*X*)

X

実数型でなければなりません。スカラまたは配列です。

result

*X*と同じ型および同じ種別型パラメタをもつスカラです。

機能説明

EPSILON は*X*を数体系の数として表現したときに、1に対して無視できる値を返却します。

関数の結果は$\mathrm{RADIX}(X)^{(1-\mathrm{DIGITS}(X))}$となり、その型は*X*と同じです。

以下の値の固定値となります。

引数の型	結果の値
半精度実数型	9.7656E-04
単精度実数型	1.19209290E-07
倍精度実数型	2.220446049250313D-16
4倍精度実数型	1.92592994438723585305597794258492 73Q-0034

使用例

```
! 精度誤差を考慮した実数値の比較
function equals (a, b)
  implicit none
  logical :: equals
  real, intent(in) :: a, b
  real :: eps
  eps = abs(a) * epsilon(a)     ! 許される範囲の差分を定義します
  if (eps == 0) then
    eps = tiny (a)              ! 0より小さいなら最小の正の値を許される範囲の差分とします
  end if
  if (abs(a-b) > eps) then
    equals = .false.            ! 差がeps より大きいなら等価とはしません
  else
    equals = .true.             ! 差がeps より小さいなら等価とします
  end if
  return
end function equals
```

2.160 EQUIVALENCE文

EQUIVALENCE 文は、有効域内で2つ以上の実体が同じ記憶単位を共有することを指定します。これを記憶列結合といいます。記憶列結合している実体が異なる型または型パラメタをもつとき、EQUIVALENCE 文は、型変換または数学的な等価化を行いません。スカラ変数と配列が記憶列結合しているとき、スカラ変数は配列の性質をもちませんし、配列はスカラの性質をもちません。

EQUIVALENCE 文は、以下の形式です。

EQUIVALENCE *equivalence-set-list*

equivalence-set-list は、コンマで区切られた結合対応の並びです。

equivalence-set は以下の形式です。

(*equivalence-object* , *equivalence-object-list*)

equivalence-object は結合実体であり、以下の形式です。

variable-name または
array-element または
substring

variable-name は変数名です。

array-element は配列要素であり、添字式は整数定数式でなければなりません。

substring は文字部分列であり、文字部分列式は整数定数式でなければなりません。部分列の長さは0であってはなりません。

結合実体は、仮引数、ポインタ、割付け変数、末端成分に割付け変数をもつ派生型の実体、連続型でない派生型の実体、成分にポインタをもつ連続型の派生型の実体、自動割付け変数、関数名、入口名、結果名、BIND 属性をもつ変数、共通ブロック内にあってBIND属性をもつ変数、名前付き定数、構造体成分、またはこれらの一部であってはなりません。結合実体はTARGET属性をもってはなりません。

結合実体は、参照結合によって参照可能になった名前であってはなりません。

結合実体の1つが基本整数型、基本実数型、倍精度複素数型、基本複素数型、基本論理型、または数値連続型であるとき、結合対応中の実体はすべてこれらの型でなければなりません。

結合実体の1つが基本文字型または文字連続型であるとき、結合対応中の実体はすべてこれらの型でなければなりません。記憶列共有するデータ実体の長さは異なっていてもかまいません。

結合実体の1つが数値連続型でも文字連続型でもない派生型であるとき、結合対応中の実体はすべてこれらと同じ型でなければなりません。

結合実体の1つが基本整数型、基本実数型、倍精度複素数型、基本複素数型、基本論理型、および基本文字型を除いた組込み型であるとき、結合対応中の実体はすべてこれらと同じ型パラメタをもった同じ型でなければなりません。

EQUIVALENCE 文によって、同じ記憶単位が1つの記憶列中に2回以上現れるような指定をしてはなりません。同様に、連続する記憶単位を不連続とするような指定をしてはなりません。

EQUIVALENCE 文の例:

```
equivalence (a,b,c(2))          ! a、b、およびc(2) は、同じ記憶列を共有します
```

2.161 ERF組込み関数

ERF 関数は、実数データ*X*に対する誤差関数の値を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
ERF	-----	1	実数型	実数型
	ERF		単精度実数型	単精度実数型
	DERF		倍精度実数型	倍精度実数型
	QERF		4倍精度実数型	4倍精度実数型

result = ERF (*X*)

X

実数型でなければなりません。

result

*X*と同じ型です。

機能説明

ERF、DERF、およびQERF は、実数データ*X*に対する誤差関数の値を求めます。

結果の範囲は、-1.0 <= ERF(*X*) <= 1.0 です。

総称名ERF は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
r = erf(.5)
```

2.162 ERFC組込み関数

ERFC 関数は、実数データ*X*に対する相補誤差関数の値を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
ERFC	-----	1	実数型	実数型
	ERFC		単精度実数型	単精度実数型
	DERCF		倍精度実数型	倍精度実数型
	QERFC		4倍精度実数型	4倍精度実数型

result = ERFC (*X*)

X

実数型でなければなりません。

result

*X*と同じ型です。

機能説明

ERFC、DERFC、およびQERFC は、実数データ*X*に対する相補誤差関数の値を求めます。

結果の範囲は、0.0 <= ERFC(*X*) <= 2.0 です。

総称名ERFC は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
r = erfc(1.0)
```

2.163 ERFC_SCALED組込み関数

ERFC_SCALED 関数は、実数データ*X*に対するスケーリング相補誤差関数の値を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
ERFC_SCALED	-----	1	実数型	実数型

result = ERFC_SCALED (*X*)

X

実数型でなければなりません。

result

*X*と同じ型です。

機能説明

ERFC_SCALED は、実数データ*X*に対するスケーリング相補誤差関数の値を求めます。

総称名ERFC_SCALED は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

単精度実数型の引数の場合、引数は *X* > -9.3824 でなければなりません。

倍精度実数型の引数の場合、引数は *X* > -26.6287 でなければなりません。

4倍精度実数型の引数の場合、引数は *X* > -106.56 でなければなりません。

使用例

```
r = erfc_scaled(10.0)
```

2.164 ERROR STOP文

ERROR STOP 文は、プログラムの実行を誤り終了します。

ERROR STOP 文は、以下の形式です。

```
ERROR STOP [ stop-code ]
```

stop-code は終了符号であり、以下の形式です。

```
scalar-default-char-initialization-expr    または
scalar-int-initialization-expr
```

scalar-default-char-initialization-expr は、スカラ基本文字型の定数式です。

scalar-int-initialization-expr は、スカラ整数型の定数式です。

ERROR STOP 文を実行すると、標準エラー出力ファイルに対して診断メッセージを出力し、プログラムの実行を誤り終了します。

ERROR STOP 文の例:

```
if (a > b) then
  error stop ! プログラムの実行を誤り終了します
end if
```

2.165 ERRORサービスサブルーチン

機能説明

指定した文字列を標準出力ファイルに出力します。

形式

CALL ERROR (*string*)

string

基本文字型スカラ。

使用例

```
      use service_routines,only:error
      open(10,file='x.dat',err=10)
      write(10,*)' Fortran program'
      stop
   10 call error('Fortran I/O error')
      stop 200
      end
```

2.166 ERRSAVサービスサブルーチン

機能説明

エラー識別番号のエラー項目の先頭領域を、指定した領域に退避します。

形式

CALL ERRSAV (*errno* , *darea*)

errno

基本整数型スカラ。エラー識別番号を示します。

darea

基本文字型スカラ。長さは16です。
darea の型が文字型でない場合、結果は不定となります。

使用例

```
      use service_routines,only:errsav,errstr
      character(len=16) :: err113
   ! (1)estop,(2)mprint,(3)ecount,(4)inf
      call errsav(113,err113)
      write(err113(1:2),'(2a1)') 0,0
      call errstr(113,err113)
      open(10,file='x.dat',form='formatted')
      do i=1,15
        write(10)i
      end do
      close(10,status='delete')
      end
```

2.167 ERRSETサービスサブルーチン

機能説明

エラー識別番号に対応するエラー制御表内の情報を変更します。

形式

CALL ERRSET (*errno* , *estop* , *mprint* , *trace* , *uexit* , *r*)

errno

基本整数型スカラ。エラー識別番号を指定します。

estop

基本整数型スカラ。エラー打切り回数を指定します。

<=0　：変更されません。
>=256 ：エラー打切り回数は無制限となります。

mprint

基本整数型スカラ。メッセージの最大印刷回数を指定します。

=0　　：変更されません。
<0　　：最大印刷回数は0となります。
>=256　：最大印刷回数は無制限となります。

trace

基本整数型スカラ。トレースバックマップを印刷するかどうかを指定します。

=0　：変更されません。
=1　：印刷しないように変更されます。
=2　：印刷するように変更されます。

uexit

基本整数型スカラ。エラー修正について指定します。

=0　　：変更されません。
=1　　：システムの標準修正が行われます。
上記以外：利用者定義の修正サブルーチンが実行されます。

r

基本整数型スカラ。
errno が132以外の場合、変更する最大のエラー識別番号を指定します。
errno が132の場合、制御文字を用意するかどうかを指定します。

=1　　：制御文字を用意します。
上記以外：制御文字を用意しません。

使用例

```
use service_routines,only:errset
call errset(113,256,-1,1,0,0)
open(10,file='x.dat',form='formatted')
do i=1,15
  write(10) i
end do
close(10,status='delete')
end
```

2.168 ERRSTRサービスサブルーチン

機能説明

指定した領域の内容をエラー識別番号のエラー項目の先頭領域に設定します。

形式

CALL ERRSTR (*errno* , *darea*)

errno

基本整数型スカラ。エラー識別番号を示します。

darea

基本文字型スカラ。長さは16です。
darea の型が文字型でない場合、結果は不定となります。

使用例

```
  use service_routines,only:errstr,errsav
  character(len=16) :: err113
! (1)estop, (2)mprint, (3)ecount, (4)inf
  call errsav(113,err113)
```

```
write(err113(1:2),'(2a1)')0,0
call errstr(113,err113)
open(10,file='x.dat',form='formatted')
do i=1,15
  write(10) i
end do
close(10,status='delete')
end
```

2.169 ERRTRAサービスサブルーチン

機能説明

現在実行中のプログラム単位までのトレースバックマップを出力します。

形式

```
CALL ERRTRA
```

使用例

```
open(10,file='x.dat')
call sub1
end
subroutine sub1
  use service_routines,only:errtra
  call errtra
  do i=1,15
    write(10,*) i
  end do
end
```

2.170 ETIMEサービス関数

機能説明

実行可能プログラムの実行開始時からのCPU 時間を返却します。CPU 時間はプログラムが実行されているプロセスとそのプロセス内の全スレッドで利用されたCPU 時間です。

形式

y = ETIME (*tm*)

tm

基本実数型配列。*tm*(1) にはユーザCPU 時間、*tm*(2) にはシステムCPU 時間が設定されます。
tm の要素数が2より少ない場合、実行動作は保証されません。*tm* の要素数が2より大きい場合、3要素目以降の値は変更されません。

関数結果

基本実数型スカラ。*tm*(1) と*tm*(2) の合計値が秒単位で返却されます。

エラーが発生した場合、*tm*(1)、*tm*(2) は不定となり、-1.0が返却されます。

使用例

```
  use service_routines,only:etime
  real :: tm1(2),tm2(2),y1,y2
  y1 = etime(tm1)
  do i=1,5000
    write(*,*) i,i*i
  end do
  y2 = etime(tm2)
  print *,'REAL TIME=',y2-y1,'USER TIME=',tm2(1)-tm1(1), &
& 'SYSTEM TIME=',tm2(2)-tm1(2)
  end
```

2.171 EXECUTE_COMMAND_LINE組込みサブルーチン

EXECUTE_COMMAND_LINE組込みサブルーチンは、コマンドラインを実行します。本処理系では、コマンドラインの実行は同期します。

分類

サブルーチン

形式

CALL EXECUTE_COMMAND_LINE (*COMMAND* [, *WAIT*, *EXITSTAT*, *CMDSTAT*, *CMDMSG*])

COMMAND

基本文字型スカラでなければなりません。INTENT(IN)の引数です。この値がコマンドラインとして実行されます。

WAIT（省略可能）

基本論理型スカラでなければなりません。INTENT(IN)の引数です。本処理系では偽を指定しても、実行は同期します。

EXITSTAT（省略可能）

基本整数型スカラでなければなりません。INTENT(INOUT)の引数です。終了状態が設定されます。

CMDSTAT（省略可能）

基本整数型スカラでなければなりません。INTENT(OUT)の引数です。

- エラーが発生した場合、正の値が設定されます。
- *WAIT*に偽を指定した場合、-2が設定されます。
- 他の場合、ゼロが設定されます。

CMDMSG（省略可能）

基本文字型スカラでなければなりません。INTENT(INOUT)の引数です。エラーが発生した場合、説明のためのメッセージが代入されます。それ以外の場合、値は変更されません。

機能説明

本処理系では、現在のプロセスは、コマンドの実行が完了するまで待ち状態となります。
*CMDSTAT*にゼロ以外の値が設定される状態が発生し、*CMDSTAT*が省略された場合、エラー終了します。

使用例

```
call EXECUTE_COMMAND_LINE( 'x.sh' )
```

2.172 EXIT文

EXIT 文は、ループの終了、または他の構文の実行完了方法の1つです。

EXIT 文は、以下の形式です。

```
EXIT [ construct-name ]
```

construct-name は、構文名です。

*construct-name*が現れる場合、EXIT文は構文の範囲内に指定しなければなりません。*construct-name*を省略する場合、EXIT文はDO構文の範囲内に指定しなければなりません。

構文名は、そのEXIT文が含まれる構文の構文名でなければなりません。

EXIT文を実行すると、構文名で識別される構文の実行を終了します。構文名の指定がない場合、そのEXIT文を含むDO構文のうち、もっとも内側のDO構文の実行を終了します。

EXIT文は、CRITICAL構文またはDO CONCURRENT構文に現れることはできません。

EXIT 文の例:

```
outer: do i=1, 10
  inner: do j=1, 10
    if (i>a) exit outer
    if (j>b) exit              ! 内側のDO 構文を終了します
      ...
  enddo inner
enddo outer
```

2.173 EXITサービスサブルーチン

機能説明

実行可能プログラムの実行を中止します。

形式

```
CALL EXIT
```

使用例

```
      use service_routines,only:exit,setrcd
      open(10,file='x.dat',err=10)
      do i=1,15
        write(10,*,err=10) i
      end do
      stop 'PROGRAM END'
10    call setrcd(130)
      call exit
      end
```

2.174 EXP組込み関数

EXP 関数は、e を底とする実数または複素数の指数関数を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
EXP	-----	1	実数型または 複素数型	実数型または 複素数型
	EXP		単精度実数型	単精度実数型
	DEXP		倍精度実数型	倍精度実数型
	QEXP		4倍精度実数型	4倍精度実数型
	CEXP		単精度複素数型	単精度複素数型
	CDEXP		倍精度複素数型	倍精度複素数型
	CQEXP		4倍精度複素数型	4倍精度複素数型

result = EXP (*X*)

X

実数型または複素数型でなければなりません。

result

X と同じ型です。

機能説明

EXP、DEXP、QEXP、CEXP、CDEXP、およびCQEXP は、e を底とする実数または複素数の指数関数を求めます(e は 2.71828182845... です)。

単精度実数型の引数の場合、引数は *X* < 88.722E0 でなければなりません。

倍精度実数型の引数の場合、引数は *X* < 709.782D0 でなければなりません。

4倍精度実数型の引数の場合、引数は *X* < 11356.5Q0 でなければなりません。

単精度複素数型の引数の場合、REAL(*X*) < 88.722E0 かつ ABS(IMAG(*X*)) < 8.23E+05 でなければなりません。

倍精度複素数型の引数の場合、DREAL(*X*) < 709.782D0 かつ DABS(DIMAG(*X*)) < 3.53D+15 でなければなりません。

4倍精度複素数型の引数の場合、QREAL(*X*) < 11356.5Q0 かつ QABS(QIMAG(*X*)) < $2.0Q0^{62} \times \pi$ でなければなりません。

総称名EXP は、すべての実数型または複素数型の引数に使用することができます。

それぞれの関数の結果の型は、*X* と同じです。

使用例

```
a = exp(2.0)
```

2.175 EXP10組込み関数

EXP10 関数は、実数型データの10.0を底とする指数関数を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
EXP10	-----	1	実数型	実数型
	EXP10		単精度実数型	単精度実数型
	DEXP10		倍精度実数型	倍精度実数型
	QEXP10		4倍精度実数型	4倍精度実数型

result = EXP10 (*X*)

X

実数型でなければなりません。

result

X と同じ型です。

機能説明

EXP10、DEXP10、およびQEXP10 は、実数型データの10.0を底とする指数関数を求めます。

単精度実数型の引数の場合、引数は *X* < 38.531E0 でなければなりません。

倍精度実数型の引数の場合、引数は *X* < 308.254D0 でなければなりません。

4倍精度実数型の引数の場合、引数は *X* < 4932.0625Q0 でなければなりません。

総称名EXP10 は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*X* と同じです。

使用例

```
a = exp10(2.0)                    ! a には100.0が代入されます
```

2.176 EXP2組込み関数

EXP2 関数は、実数型データの2.0を底とする指数関数を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
EXP2	-----	1	実数型	実数型
	EXP2		単精度実数型	単精度実数型
	DEXP2		倍精度実数型	倍精度実数型
	QEXP2		4倍精度実数型	4倍精度実数型

result = EXP2 (*X*)

X

実数型でなければなりません。

result

*X*と同じ型です。

機能説明

EXP2、DEXP2、およびQEXP2 は、実数型データの2.0を底とする指数関数を求めます。

単精度実数型の引数の場合、引数は *X* < 128.0E0 でなければなりません。

倍精度実数型の引数の場合、引数は *X* < 1024.0D0 でなければなりません。

4倍精度実数型の引数の場合、引数は *X* < 16384.0Q0 でなければなりません。

総称名EXP2 は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
a = exp2(2.0)                    ! a には4.0が代入されます
```

2.177 EXPONENT組込み関数

EXPONENT 関数は、*X*を数体系の数として表現したときの指数部の値を返却します。

分類

要素別処理関数

形式

result = EXPONENT (*X*)

X

実数型でなければなりません。

result

基本整数型です。

機能説明

EXPONENT は、*X*を数体系の数として表現したときの指数部の値を返却します。

*X*が0.0のときは0を返却します。

X がIEEE 無限大または非数の場合、結果の値はHUGE(0) とします。

関数の結果の型は、基本整数型です。

使用例

```
i = exponent(3.8)              ! i には2が代入されます
i = exponent(-4.3)             ! i には3が代入されます
```

2.178 EXTENDS_TYPE_OF組込み関数

EXTENDS_TYPE_OF 組込み関数は、動的な型がその拡張型かどうかを問い合せます。

分類

問合せ関数

形式

result = EXTENDS_TYPE_OF (*A* , *MOLD*)

A

拡張可能型の実体でなければなりません。ポインタの場合、結合状態が不定であってはなりません。

MOLD

拡張可能型の実体でなければなりません。ポインタの場合、結合状態が不定であってはなりません。

result

基本論理型スカラです。

機能説明

MOLD が無制限多相的で、空状態のポインタまたは割り付けられていない割付けのいずれかの場合、結果は真です。この条件には該当せず、*A* が無制限多相的で、空状態のポインタまたは割付け状態が未割付けのいずれかのとき、結果は偽です。

A の動的な型が *MOLD* の動的な型の拡張型のとき、結果は真です。

空状態のポインタまたは割付け状態が未割付けな場合の動的な型は、その実体の宣言時の型です。

使用例

```
type ty1
  integer :: i
end type
type, extends(ty1) :: ty2
end type
class (ty1),pointer :: p1,p2
  ...
print *,extends_type_of(p1, p2)
end
```

2.179 EXTERNAL文

EXTERNAL 文は、外部手続名、仮手続名、および初期値設定プログラム単位名であることを宣言します。EXTERNAL 文に指定された名前が手続名である場合、その名前を実引数として指定できることを指定します。

EXTERNAL 文は、以下の形式です。

EXTERNAL [::] *external-name-list*

external-name-list は、コンマで区切られた外部名の並びです。*external-name* は、外部手続名、仮手続名、または初期値設定プログラム単位名でなければなりません。

組込み手続名がEXTERNAL 文に指定された場合、その名前は、その有効域内において組込み手続名ではなく、外部名となります。

EXTERNAL 文の例:

```
subroutine fred (a,b,sin)
external sin                ! sin は組込み関数ではなく、仮手続名です
call bill(a,sin)            ! sin はEXTERNAL 文に指定されているので、実引数として指定できます
```

2.180 FDATEサービスサブルーチン

機能説明

現在の日付と時刻をASCII コードに変換して、通知します。

形式

CALL FDATE (*string*)

string

基本文字型スカラ。

引数*string* は長さ24以上の文字型でなければなりません。

引数*string* の長さが返却値より長い場合、空白が補われます。

使用例

```
use service_routines,only:fdate
character(len=24) :: ch
call fdate(ch)
print *,ch
end
```

2.181 FGETCサービス関数

機能説明

書式付き順番探査入出力ファイルから1文字を読み込みます。

形式

iy = FGETC (*unit* , *ch*)

unit

基本整数型スカラ。データを読み込む書式付き順番探査入出力ファイルと結合している装置番号を指定します。

ch

基本文字型スカラ。読み込んだデータ1文字が設定されます。

関数結果

基本整数型スカラ。正常に読み込んだときは0、EOF を検出したときは-1、エラーが発生したときは0、-1以外の値が返却されます。

使用例

```
  use service_routines,only:fgetc
  character(len=100) :: ch1
  character(len=10),dimension(10) :: ch
  data ch/"a123456789","b123456789","c123456789","d123456789", &
&         "e123456789","f123456789","g123456789","h123456789", &
&         "i123456789","j123456789"/
  open(10,file='x.dat',form='formatted')
  do i=1,10
    write(10,fmt='(a10)') ch(i)
  end do
  rewind(10)
  do
```

```
  if (fgetc(10,ch1(i:i)) .eq. -1) exit
  if (ch1(i:i) .eq. '¥n') i=i-1
end do
end
```

2.182 FINAL文

FINAL 文は、派生型に対して後始末束縛を指定します。

FINAL 文は、以下の形式です。

```
FINAL [::] final-subroutine-name-list
```

final-subroutine-name-list は、後始末サブルーチン名並びです。

後始末サブルーチン名は、ひとつの仮引数をもつモジュール手続の名前でなければなりません。

そのモジュール手続の仮引数は、省略不可であり、後始末を指定する派生型の、ポインタでもなく、割付け変数でもなく、多相的でもない変数でなければなりません。また、INTENT(OUT) であってはなりません。

同じ型に対して、後始末サブルーチン名並びに同じ名を2回以上指定してはなりません。

同じ型に対して、同じ種別型パラメタおよび同じ次元数をもった仮引数をもつ後始末サブルーチン名を2回以上指定してはなりません。

2.183 FINDLOC組込み関数

FINDLOC関数は、指定した値の要素の位置を調べます。

分類

変形関数

形式

result = FINDLOC (*ARRAY* , *VALUE* , *DIM* [, *MASK* , *KIND* , *BACK*])　または
result = FINDLOC (*ARRAY* , *VALUE* [, *MASK* , *KIND* , *BACK*])

ARRAY

組込み型の配列でなければなりません。

VALUE

スカラでなければなりません。*ARRAY* と型が適合しなければなりません。

DIM

整数型スカラであって、1 <= *DIM* <= n の範囲の値でなければなりません。ここで、n は*ARRAY* の次元数とします。対応する実引数は、省略可能な仮引数であってはなりません。

MASK (**省略可能**)

論理型でなければならず、*ARRAY* と形状適合しなければなりません。

KIND (**省略可能**)

スカラ整数定数式でなければなりません。

BACK (**省略可能**)

論理型スカラでなければなりません。

result

KIND を指定した時、種別型パラメタ *KIND* の整数型です。*KIND* を省略したとき、基本整数型になります。

DIM が省略された場合は、大きさ n の1次元配列となります。n は*ARRAY* の次元数です。*DIM* が指定された場合は、*ARRAY* の形状を (d_1, d_2, ..., d_n) としたとき、形状は、(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_n) となります。

機能説明

FINDLOC は、*ARRAY* の中から*VALUE*と一致する値をもつ要素の添字を返します。

－ *DIM*を省略している場合

MASK を省略すると、*MASK* の全要素が真と等価です。

MASK 中の真の要素に対応する*VALUE*値と一致する各次元の添字値を要素とする1次元配列を返却します。*VALUE*と一致する要素がない、*MASK* の全要素が偽である、または、*ARRAY* の大きさがゼロの場合は、すべてゼロの1次元配列を返却します。

－ *DIM*を指定している場合

ARRAY が1次元の場合は、FINDLOC (*ARRAY*, *VALUE* [, *MASK*]) の要素と等しく、結果はスカラとなります。

他の場合、結果の要素(S_1, S_2, ..., S_{DIM-1}, S_{DIM+1}, ..., S_n) は、FINDLOC(*ARRAY*(S_1, S_2, ..., S_{DIM-1}, : , S_{DIM+1}, ..., S_n), *VALUE* , DIM = 1[,MASK = *MASK*])となります。ここで、n は*ARRAY* の次元数、*MASK*は*MASK*(S_1, S_2, ..., S_{DIM-1}, : , S_{DIM+1}, ..., S_n) とします。

－ *BACK*に偽を指定している、または、省略している場合

*VALUE*値と最初に一致する各次元の添字値が要素となります。

－ *BACK*に真を指定している場合

*VALUE*値と最後に一致する各次元の添字値が要素となります。

－ *KIND*を省略している場合

関数の結果の型は、基本整数型です。

－ *KIND*を指定している場合

関数の結果の型は、種別型パラメタ *KIND* の整数型です。

使用例

```
integer :: array(2,3)=0
integer :: result( 2 )
array(1,3)=1
result = findloc( array , 1)  ! resultには [1,3] が代入されます。
```

2.184 FLOOR組込み関数

FLOOR 関数は、引数の値以下で最大の整数値を返却します。

分類

要素別処理関数

形式

result = FLOOR (*A* [, *KIND*])

A

実数型でなければなりません。

KIND（**省略可能**）

スカラ整数定数式でなければなりません。

result

整数型です。*KIND* が指定された場合、種別型パラメタは*KIND* の指定に従います。
KIND が省略された場合、種別型パラメタは基本整数型になります。

機能説明

FLOOR は、引数の値以下で最大の整数値を返却します。

A 以下で最大の整数を返却します。

関数の結果が指定された整数型で表現できない場合、結果の値は不定となります。

関数の結果の型は、種別型パラメタ*KIND*が指定された場合、種別型パラメタ*KIND*の整数型となります。*KIND*が省略された場合、関数の結果の型は基本整数型です。

使用例

```
i = floor(-2.1)                ! i には-3が代入されます
j = floor(2.1)                 ! j には2が代入されます
```

2.185 FLUSHサービスサブルーチン

機能説明

指定した装置番号と接続しているファイルにバッファ上のデータを出力します。

形式

```
CALL FLUSH ( unit )
```

unit

基本整数型スカラ。装置番号を指定します。

使用例

```
use service_routines,only:flush,system
open(10,file='x.dat')
do i=1,15
  write(10,*)i
  call flush(10)
  ix = system('cat x.dat')
end do
end
```

2.186 FLUSH文

FLUSH 文は、実行すると、外部ファイルに書き込まれたデータを他の手続で引用したり、Fortran以外の手段で外部ファイルに置かれたデータをREAD 文で利用したりすることができます。

FLUSH 文は、ファイル位置については効果がありません。

接続されているが存在していないファイルに対してFLUSH 文を実行してもかまいませんが、いずれのファイルに対しても効果がありません。

FLUSH 文は、以下の形式です。

```
FLUSH external-file-unit                または
FLUSH ( flush-spec-list )
```

external-file-unit は外部ファイル装置であり、スカラ整数式でなければなりません。

flush-spec-list は、コンマで区切られた一掃指定子の並びです。

flush-spec は、以下の形式です。

```
[ UNIT = ] external-file-unit       または
IOSTAT = io-stat                    または
IOMSG = iomsg                       または
ERR = err-label
```

一掃指定子並びにおいて、UNIT 指定子は必ず1つ指定しなければならず、他の指定子はそれぞれ1つ指定することができます。

external-file-unit は、外部ファイル装置であり、スカラ整数式でなければなりません。

文字列'UNIT = 'をUNIT 指定子から省略する場合、UNIT 指定子は、一掃指定子並びの最初の項目でなければなりません。

io-stat は、スカラ整変数でなければなりません。

IOSTAT 指定子が指定された場合、IOSTAT 指定子に指定された変数には、以下の値が設定されます。

- 誤り条件、ファイル終了条件、および記録終了条件が検出されない場合は、0
- 誤り条件が検出された場合は、1または実行時の診断メッセージの番号

*iomsg*はスカラ基本文字変数でなければなりません。入出力文の実行中に誤り条件が発生した場合、*iomsg*には説明のためのメッセージが代入されます。

誤り条件が発生しない場合、*iomsg* の値は変更されません。

err-label は文番号であり、このFLUSH 文と同じ有効域内の飛び先文の文番号でなければなりません。

ERR 指定子が指定され、FLUSH 文の実行中に誤り条件が検出された場合、ERR 指定子に指定された文番号をもつ文が次に実行されます。

FLUSH 文の例:

```
flush 10           ! 装置番号10に接続されているファイルのバッファ上のデータを出力します
flush (10,err=100) ! 装置番号10に接続されているファイルのバッファ上のデータを出力し、
                   ! 誤り条件が発生した場合には、文番号100の文が次に実行されます
```

2.187 FORALL構文

FORALL 構文は、複数の代入、配列選別代入(WHERE)および入れ子になったFORALL 構文または単純FORALL 文を制御します。

FORALL 構文は、以下の形式です。

```
[ forall-construct-name : ] FORALL forall-header
    [ forall-body-construct ]...
END FORALL [ forall-construct-name ]
```

forall-construct-name は、FORALL 構文名です。

構造FORALL 文にFORALL 構文名を指定する場合は、対応するEND FORALL 文にも、同じFORALL 構文名を指定しなければなりません。構造FORALL 文にFORALL 構文名を指定しない場合は、対応するEND FORALL 文にFORALL 構文名を指定してはなりません。

forall-header はFORALL 制御(“1.16 FORALL制御”参照)です。

forall-body-construct は、FORALL 本体構文であり、以下の形式です。

```
代入文                    または
ポインタ代入文            または
単純WHERE 文              または
WHERE 構文                または
FORALL 構文               または
単純FORALL 文
```

FORALL 本体構文中の文は、FORALL 構文の指標変数名を確定してはなりません。

FORALL 本体構文中で引用する手続(利用者定義演算、代入または後始末によって引用する手続を含みます) は、純粋手続でなければなりません。

FORALL 構文が実行されると、FORALL 三つ組指定並び中の添字および刻み幅が評価されます。1つの指標変数がとりうる値の集合は、式$m_1 + (k\text{-}1) \times m_3$で定められます。ここで、$k$は、$k = 1, 2, ..., max$とし、$max$は、$(m_2 - m_1 + m_3) / m_3$です。$m_1$、$m_2$、および$m_3$は、それぞれFORALL 三つ組指定並び中の1番目の添字、2番目の添字、および刻み幅の評価結果です。刻み幅の指定がない場合、m_3の値は1となります。いずれかの指標変数に対して $max <= 0$ であるとき、そのFORALL 構文は実行されません。

スカラ選別式の指定があるとき、その式は指標値のそれぞれの組合せに対して評価されます。スカラ選別式の指定がないとき、真の値をもつ式があるものとして評価されます。指標値の有効な組合せは、すべての可能な組合せのうち、選別式の値が真である部分集合です。

FORALL 本体構文は、指標値の有効な組合せのすべてに対して、出現順に実行されます。

FORALL 構文中の代入文は、指標値の有効な組合せごとに、右辺の式、および左辺の変数中のすべての式が評価され、その後、右辺のそれぞれの式の値が、対応する左辺の変数に代入されます。

FORALL 構文中のポインタ代入文は、指標値の有効な組合せごとに、指示先およびポインタ実体中のすべての式が評価され、その後、それぞれのポインタ実体は、対応する指示先に結合されます。

FORALL 構文中のWHERE 構文中の文は、1つ1つ順番に実行されます。単純WHERE 文、構造WHERE 文、または選別ELSEWHERE 文が実行されるとき、それらの文の選別式は、外側のFORALL 構文によって決められる指標値の有効な組合せごとに評価され、外側のWHERE 構文に対応する制御配列によって選別されます。WHERE 構文中の代入文は、指標値の有効な組合せごとに実行され、その代入文に対する制御配列によって選別されます。WHERE 構文および単純WHERE 文の詳細については、“2.503 WHERE構文”および“2.505 単純WHERE文”を参照してください。

FORALL 構文中の単純FORALL 文またはFORALL 構文は、外側のFORALL 構文の指標値の有効な組合せごとに、FORALL 三つ組指定子並び中の添字および刻み幅の式が評価されます。内側のFORALL に対する指標値の可能な組合せの集合は、外側の指標値を含みます。次に、内側のFORALL の指標値のすべての組合せに対してスカラ選別式が評価され、内側のFORALL の有効な組合せ集合が得られます。その次に、内側のFORALL 中のそれぞれの文が、指標値の有効な組合せのすべてに対して実行されます。

FORALL 構文の実行は、DO 構文の実行とは異なり、FORALL 構文内の文を出現順に指標値のすべての組合せに対して実行します。したがってFORALL 構文内に複数の文がある場合、先の文で値が確定される変数を参照している場合には、その変数が指標値のすべての組合せに対して評価され、値が確定していることに注意してください。

FORALL 構文の例:

```
integer :: a(3,3)
integer :: n=3
a = reshape((/0,1,2,3,4,5,6,7,8/),(/3,3/))
```

```
! a は┌       ┐
!    | 0 3 6 |
!    | 1 4 7 |
!    | 2 5 8 |
!    └       ┘
```

```
forall ( i = 1 : n-1 )
  forall ( j = i+1 : n )
    a(i,j) = a(j,i)
  end forall
end forall
```

```
! a は┌       ┐
!    | 0 1 2 |
!    | 1 4 5 |
!    | 2 5 8 |
!    └       ┘
```

2.188 構造FORALL文

構造FORALL 文は、FORALL 構文の開始を指定します。

構造FORALL 文は、以下の形式です。

```
[ forall-construct-name : ] FORALL forall-header
```

forall-construct-name は、FORALL 構文名です。

forall-header はFORALL 制御(“1.16 FORALL制御”参照)です。

FORALL 構文の詳細については、“2.187 FORALL構文”を参照してください。

2.189 単純FORALL文

単純FORALL 文は、代入文およびポインタ代入文の実行を制御します。

単純FORALL 文は、以下の形式です。

```
FORALL forall-header forall-assignment-stmt
```

forall-header はFORALL 制御(“1.16 FORALL制御”参照)です。

forall-assignment-stmt は、以下の形式です。

assignment-stmt または
pointer-assignment-stmt

assignment-stmt は、代入文です。

pointer-assignment-stmt は、ポインタ代入文です。

単純FORALL 文は、FORALL 本体構文として1つの代入文またはポインタ代入文だけを含むFORALL 構文と同じです。FORALL 構文の詳細については、“2.187 FORALL構文”を参照してください。

2.190 FORKサービス関数

機能説明

呼出しプロセスと全く同じものを、子プロセスとして作成します。

形式

iy = FORK ( )

関数結果

基本整数型スカラ。子プロセスが作成できたときは、子プロセスID が返却されます。エラーが発生したときは、負の値が返却されます。

使用例

```
   use service_routines,only:fork
   integer :: pid
   pid = fork( )
   if (pid) 10,20,30
10 write(6,fmt="(5x,a)") "FORK FAILED"
   stop 240
20 write(6,fmt="(3x,a)") "CHILD PROCESS RUNNING // FORK SUCCEEDED //"
30 stop
   end
```

2.191 FORMAT文

FORMAT 文は、データの内部表現と書式付き記録列としての文字列との間の明示的な編集情報を用意します。

FORMAT 文は、以下の形式です。

```
FORMAT format-specification
```

format-specification は書式仕様です。書式仕様の詳細については、“1.8.1 書式仕様”を参照してください。

FORMAT 文には、文番号を付けなければなりません。

FORMAT 文の例:

```
10 format (e11.5)
20 format (3i8,e12.5)
```

2.192 FPUTCサービス関数

機能説明

書式付き順番探査入出力ファイルに1文字を書き出します。

形式

iy = FPUTC (*unit* , *ch*)

unit

基本整数型スカラ。データを書き込む書式付き順番探査入出力ファイルと結合している装置番号を指定します。

ch

長さ1の基本文字型スカラ。書き込むデータ1文字を設定します。

関数結果

基本整数型スカラ。正常に書き込んだときは0、エラーが発生したときは0以外の値が返却されます。

使用例

```
use service_routines,only:fputc
character(len=6) :: pr='INPUT>'
do i=1,6
  if (fputc(6,pr(i:i)) .ne. 0) stop 10
end do
read (*,*)x
end
```

2.193 FRACTION組込み関数

FRACTION 関数は、*X*を数体系の数として表現したときの小数部を返却します。

分類

要素別処理関数

形式

result = FRACTION (*X*)

X

実数型でなければなりません。

result

*X*と同じ型です。

機能説明

FRACTION は、*X*を数体系の数として表現したときの小数部を返却します。

関数の結果は $X \times \mathrm{RADIX}(X)^{(-\mathrm{EXPONENT}(X))}$ となり、その型は*X*と同じです。

*X*が0.0であるとき、結果の値は0.0とします。*X*がIEEE 無限大である場合、結果は非数になります。*X*がIEEE 非数である場合、結果は非数になります。

使用例

```
a = fraction(3.8)
```

2.194 FREEサービスサブルーチン

機能説明

MALLOC サービス関数によって獲得した領域を解放します。

形式

CALL FREE (*addr*)

addr

8バイトの整数型スカラです。
解放する領域の先頭番地を指定します。

引数には、必ずMALLOC サービス関数の領域獲得成功時の復帰値を指定しなければなりません。
上記以外の値を指定した場合の動作は保証されません。

使用例

```
use service_routines,only:free,malloc
integer(8) :: i
i = malloc(20_8)
  ...
call free(i)
end
```

2.195 FSEEKサービス関数

機能説明

順番探査入出力ファイルのオフセットの再配置を行います。

形式

iy = FSEEK (*unit* , *offset* , *from*)

unit

基本整数型スカラ。装置番号を指定します。
装置番号とファイルが接続されていない場合、書式付き順番探査入出力文として接続されます。

offset

基本整数型スカラ。*from* からのオフセットをバイト単位で指定します。

from

基本整数型スカラ。指定可能な値は、以下の値です。

=0：ファイルの先頭
=1：現在の位置
=2：ファイルの終端

関数結果

基本整数型スカラ。正常に再配置できたときは0、エラーが発生したときは0以外の値が返却されます。

使用例

```
      use service_routines,only:fseek
      open(10,file='x.dat',status='replace')
      do i=1,50
        write(10,*) i
      end do
      close(10)
      open(10,file='x.dat')
      i = fseek(10,0,2)
      write(10,*) 51
      i = fseek(10,0,0)
      do
        read(10,*,end=30) i
      end do
      goto 40
   30 print *,'EOF',i
   40 close(10)
      end
```

2.196 FSEEKO64サービス関数

機能説明

順番探査入出力ファイルのオフセットの再配置を行います。

形式

iy = FSEEK064 (*unit* , *offset* , *from*)

unit

基本整数型スカラ。装置番号を指定します。
装置番号とファイルが接続されていない場合、書式付き順番探査入出力文として接続されます。

offset

8バイトの整数型スカラ。*from* からのオフセットをバイト単位で指定します。

from

基本整数型スカラ。指定可能な値は、以下の値です。

=0：ファイルの先頭
=1：現在の位置
=2：ファイルの終端

関数結果

基本整数型スカラ。正常に再配置できたときは0、エラーが発生したときは0以外の値が返却されます。

使用例

```
      use service_routines,only:fseeko64
      open(10,file='x.dat',status='replace')
      do i=1,50
        write(10,*) i
      end do
      close(10)
      open(10,file='x.dat')
      i = fseeko64(10,0_8,2)
      write(10,*) 51
      i = fseeko64(10,0_8,0)
      do
        read(10,*,end=30) i
      end do
      goto 40
   30 print *,'EOF',i
   40 close(10)
      end
```

2.197 FSTATサービス関数

機能説明

指定した装置番号と接続したファイルの状態に関する情報を返却します。

形式

iy = FSTAT (*ix* , *status*)

ix

基本整数型スカラ。装置番号を指定します。

status

基本整数型配列。ファイルの状態に関する情報が設定されます。
指定する配列は要素数が13以上の配列変数でなければなりません。配列の要素数が12以下の場合の動作は保証されません。
設定される値は以下の値です。

status (1)　：ファイル・モード
status (2)　：ファイルのiノード番号
status (3)　：デバイスのiノード番号
status (4)　：デバイス識別子（特殊ファイルだけ設定されます）
status (5)　：ファイルに対するハード・リンク数
status (6)　：オーナーのユーザID
status (7)　：オーナーのグループID
status (8)　：ファイルの合計サイズ（バイト単位）
status (9)　：ファイルの最後のアクセス時間
status (10)　：ファイルの最後の変更時間
status (11)　：ファイルの最後のステータス変更時間
status (12)　：ファイル・システムの最適なブロック・サイズ
status (13)　：実際に割り当てられたブロック数

関数結果

基本整数型スカラ。

関数値には、以下の値が返却されます。

= 0　　：正常終了しました。
= 114　　：FSTAT サービス関数で指定した装置参照番号が、オープンされていません。
= 上記以外：ファイルの状態に関する情報を返却できませんでした。

使用例

```
use service_routines,only:fstat
integer :: st(13)
print *,fstat(10,st)
end
```

2.198 FSTAT64サービス関数

機能説明

指定した装置番号と接続したファイルの状態に関する情報を返却します。

形式

iy = FSTAT64 (*ix* , *status*)

ix

基本整数型スカラ。装置番号を指定します。

status

8バイトの整数型配列。ファイルの状態に関する情報が設定されます。
指定する配列は要素数が13以上の配列変数でなければなりません。配列の要素数が12以下の場合の動作は保証されません。
設定される値は以下の値です。

status (1)　：ファイル・モード
status (2)　：ファイルのiノード番号
status (3)　：デバイスのiノード番号
status (4)　：デバイス識別子（特殊ファイルだけ設定されます）
status (5)　：ファイルに対するハード・リンク数
status (6)　：オーナーのユーザID
status (7)　：オーナーのグループID
status (8)　：ファイルの合計サイズ（バイト単位）
status (9)　：ファイルの最後のアクセス時間

status (10) ：ファイルの最後の変更時間
status (11) ：ファイルの最後のステータス変更時間
status (12) ：ファイル・システムの最適なブロック・サイズ
status (13) ：実際に割り当てられたブロック数

関数結果

基本整数型スカラ。関数値には、以下の値が返却されます。

= 0　　：正常終了しました。
= 114　　：FSTAT64 サービス関数で指定した装置参照番号が、オープンされていません。
= 上記以外：ファイルの状態に関する情報を返却できませんでした。

使用例

```
use service_routines,only:fstat64
integer(kind=8) :: st(13)
print *,fstat64(10,st)
end
```

2.199 FTELLサービス関数

機能説明

書式付き順番探査入出力ファイルの現在のオフセットを返却します。

形式

iy = FTELL (*unit*)

unit

基本整数型スカラ。装置番号を指定します。
装置番号と接続している書式付き順番探査入出力ファイルはオープンされていなければなりません。

関数結果

基本整数型スカラ。書式付き順番探査入出力ファイルの現在のオフセットが返却されます。エラーが発生した場合、負の値が返却されます。

使用例

```
use service_routines,only:ftell,fseek
integer :: ix(10)
open(10,file='x.dat',status='replace')
do i=1,10
  write(10,*) i
  ix(i) = ftell(10)
  if (ix(i) < 0) exit
end do
close(10)
open(10,file='x.dat')
i = fseek(10,ix(3),0)
if (i < 0) stop
read(10,*) i
if (i /= 4) stop 'error'
close(10)
end
```

2.200 FTELLO64サービス関数

機能説明

書式付き順番探査入出力ファイルの現在のオフセットを返却します。

形式

iy = FTELL064 (*unit*)

unit

基本整数型スカラ。装置番号を指定します。
装置番号と接続している書式付き順番探査入出力ファイルはオープンされていなければなりません。

関数結果

8バイトの整数型スカラ。書式付き順番探査入出力ファイルの現在のオフセットが返却されます。エラーが発生した場合、負の値が返却されます。

使用例

```
use service_routines,only:ftell064,fseek064
integer(kind=8) :: ix(10),i
open(10,file='x.dat',status='replace')
do i=1,10
  write(10,*) i
  ix(i) = ftell064(10)
  if (ix(i) < 0) exit
end do
close(10)
open(10,file='x.dat')
i = fseek064(10,ix(3),0)
if (i < 0) stop
read(10,*) i
if (i /= 4) stop 'error'
close(10)
end
```

2.201 FUNCTION文

FUNCTION 文は、関数副プログラムを開始し、関数および関数結果の特性を宣言します。関数副プログラムについては、“1.12.1 関数副プログラム”を参照してください。

FUNCTION 文は、以下の形式です。

[*prefix-spec*]... FUNCTION *function-name* ([*dummy-arg-list*]) [*suffix*]

prefix-spec は、以下の形式です。

type-spec	または
ELEMENTAL	または
IMPURE	または
MODULE	または
PURE	または
RECURSIVE	

type-spec は宣言型指定子であり、以下の形式です。

組込み型指定子	または
TYPE (**組込み型指定子**)	または
TYPE (*type-name*)	または
CLASS(*type-name*)	または
CLASS(*)	

組込み型指定子は、以下の形式です。

INTEGER [*kind-selector*]	または
REAL [*kind-selector*]	または

```
DOUBLE PRECISION            または
COMPLEX [ kind-selector ]   または
CHARACTER [ char-selector ] または
LOGICAL [ kind-selector ]   または
BYTE
```

kind-selector は以下の形式です。

```
( [ KIND = ] kind )         または
* mem-length
```

char-selector は、以下の形式です。

```
( LEN = char-length-parm , KIND = kind )        または
( char-length-parm , [ KIND = ] kind )          または
( KIND = kind [ , LEN = char-length-parm ] )    または
( [LEN = ] char-length-parm )                   または
* char-length （廃止予定事項）
```

char-length は、以下の形式です。

```
( char-length-parm )                            または
scalar-int-literal-constant
```

char-length-parm は、以下の形式です。

```
scalar-int-expr             または
*                           または
:
```

scalar-int-literal-constant はスカラ整定数表現です。

kind は種別型パラメタであり、整数型のスカラ定数式でなければなりません。

mem-length は、領域の長さを示し、整数型のスカラ定数式でなければなりません。型指定子がINTEGER、REAL、およびLOGICAL の場合、*mem-length* の値は*kind* の値と同じ意味をもちます。型指定子がCOMPLEX の場合、*mem-length* の値は、*kind* の値を2倍にしたものと同じ意味です。

kind および *mem-length* の指定可能な値については、“2.3 型宣言文”を参照してください。

scalar-int-expr は、宣言式です。

‘**:**’の文字長パラメタ値は、無指定型パラメタであることを示します。

内部関数、モジュール関数、および引用仕様本体の関数名は、長さが‘*’であってはなりません。

*type-name*は、派生型の型名であり、その型名はその関数副プログラム内で定義されるか、参照結合または親子結合により参照可能になる型名でなければなりません。

関数結果の型および型パラメタは、FUNCTION 文の*type-spec* によって指定するか、または結果変数の名前を関数副プログラムの宣言部の型宣言文に書くことによって指定します。両方に指定することはできません。

‘ELEMENTAL’は、その関数が要素別処理関数であることを宣言します。

‘IMPURE’は、その関数が純粋関数でないことを宣言します。

‘MODULE’を指定した関数副プログラムは、分離モジュール手続です(“1.11.3 サブモジュール”参照)。

‘PURE’は、その関数が純粋関数であることを宣言します。

‘RECURSIVE’は、その関数が再帰的関数であることを宣言します。

‘ELEMENTAL’と‘RECURSIVE’の両方を指定することはできません。

‘IMPURE’と‘PURE’の両方を指定することはできません。

詳細については、“1.12.3 再帰的引用”、“1.12.4 純粋手続”、または“1.12.5 要素別処理手続”を参照してください。

function-name は、関数名です。RESULT 句の指定がない場合、結果変数の名前は関数名となります。

dummy-arg-list は、コンマで区切られた仮引数の並びです。*dummy-arg* は、以下の形式です。

dummy-arg-name

dummy-arg-name は、仮引数名です。

suffix は、次のいずれかです。

手続言語束縛指定子 [RESULT (*result-name*)] または
RESULT (*result-name*) [**手続言語束縛指定子**]

result-name は結果名であり、関数の結果変数を指定します。RESULT 句を指定する場合、結果名は関数名と同じであってはなりません。
RESULT 句を指定した場合、関数の結果変数の名前は結果名とし、その有効域内の実行文に書いたその関数名は、すべて再帰関数引用となります。
RESULT 句を指定しない場合、結果変数は関数名とし、その有効域内の実行文に書いたその関数名は、すべて結果変数の引用となります。

手続言語束縛指定子は、以下の形式です。

BIND (C [, NAME = **スカラ文字定数式**])

CLASS 指定子は、多相的実体を宣言します。CLASS 指定子が型名を含むとき、その型が多相的実体の宣言時の型になります。

CLASS(*) 指定子で宣言された実体は、無制限多相的です。無制限多相的データ要素は、宣言時の型をもちません。すべての他の要素と宣言時の型が異なるとみなされます。

FUNCTION **文の例**:

```
function sum(i,j) result(k)
  integer :: i,j,k
  ...
end function
```

2.202 GAMMA組込み関数

GAMMA 関数は、実数型データに対応するガンマ関数の値を計算します。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
GAMMA	-----	1	実数型	実数型
	GAMMA		単精度実数型	単精度実数型
	DGAMMA		倍精度実数型	倍精度実数型
	QGAMMA		4倍精度実数型	4倍精度実数型

result = GAMMA(*X*)

X

実数型でなければなりません。

result

X と同じ型です。

機能説明

GAMMA、DGAMMA、およびQGAMMA は、実数型データに対応するガンマ関数の値を計算します。

単精度実数型の引数の場合、引数は *X* < 35.039860E0 でなければなりません。

倍精度実数型の引数の場合、引数は *X* < 171.6243D0 でなければなりません。

4倍精度実数型の引数の場合、引数は *X* < 1.755Q+03 でなければなりません。

X の値は、負の整数、またはゼロであってはなりません。

総称名GAMMA は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*X* と同じです。

使用例

```
a = gamma(.5)
```

2.203 GETARGサービスサブルーチン

機能説明

第1引数*argno* 番目に指定した実行時オプションまたは利用者定義オプションの文字列を第2引数に設定します。

形式

```
CALL GETARG ( argno , argst )
```

argno

基本整数型スカラ。

argst

基本文字型スカラ。実行コマンド名または引数が設定されます。

注意事項

実行コマンド名の文字列の長さが、第2引数*argst*の長さより短い場合、指定した文字列の右側に空白が補われます。また、実行コマンド名の文字列の長さが、第2引数*argst*の長さより長い場合、指定した文字列の右側の余分な文字が無視され、第2引数*argst*の長さ分の文字列が設定されます。

使用例

```
use service_routines,only:getarg
integer :: i=1
character(len=10) :: ch
call getarg(i,ch)
print *,'ch=',ch
end
```

2.204 GETCサービス関数

機能説明

標準入力ファイルから1文字を読み込みます。

形式

```
iy = GETC ( ch )
```

ch

基本文字型スカラ。読み込んだデータ1文字が設定されます。

関数結果

基本整数型スカラ。正常に読み込んだときは0、EOF を検出したときは-1、エラーが発生したときは0、-1以外の値が返却されます。

使用例

```
use service_routines,only:getc
character(len=10) :: ch1
```

```
ch1 = ' '
do i=1,10
  if (getc(ch1(i:i)) == -1 ) exit
end do
call prompt(ch1)
read (*,*) i
end
```

2.205 GETCLサービスサブルーチン

機能説明

実行コマンド列に指定した引数の文字列を返却します。

形式

```
CALL GETCL ( string )
```

string

基本文字型スカラ。実行コマンド列に指定した引数の文字列が設定されます。*string*の長さが、GETCL の引数の長さより短い場合、指定した文字列の右側に空白が補われます。また、GETCL の引数の長さより長い場合、指定した文字列の右側の余分な文字が無視され、GETCL の引数の長さ分の文字列が設定されます。

使用例

```
use service_routines,only:getcl
character(len=15) :: ch
call getcl(ch)
print *,'argument line=',ch
end
```

2.206 GETCWDサービス関数

機能説明

現在の作業ディレクトリ名を返却します。

形式

```
iy = GETCWD ( ch )
```

ch

基本文字型スカラ。ディレクトリ名が設定されます。

引数*ch*の大きさが設定されるディレクトリ名の長さより長い場合、空白が補われます。

関数結果

基本整数型スカラ。
関数値には、以下の値が返却されます。

```
=0       : 正常終了しました。
=1       : 返却したディレクトリ名の長さが、指定された引数の長さを超えています。
上記以外 : ディレクトリ名を返却できませんでした。
```

使用例

```
use service_routines,only:getcwd
integer :: iy
character(len=40) ch
iy = getcwd(ch)
end
```

2.207 GETDATサービスサブルーチン

機能説明

現在の日付を取得します。

形式

```
CALL GETDAT ( year , month , day )
```

year

2バイトの整数型スカラ。現在の年が返却されます。

month

2バイトの整数型スカラ。現在の月が返却されます。

day

2バイトの整数型スカラ。現在の日が返却されます。

使用例

```
use service_routines,only:getdat
integer(2) :: year,month,date
call getdat(year,month,date)
write (6,fmt="(1x,i4,1x,i2,1x,i2)") year,month,date
end
```

2.208 GETENVサービスサブルーチン

機能説明

環境変数に定義した値を取得します。

形式

```
CALL GETENV ( env , string )
```

env

基本文字型スカラ。環境変数名を指定します。

string

基本文字型スカラ。*env* で指定した環境変数に定義した値が設定されます。

注意事項

第1引数に指定した環境変数の値の長さが、第2引数の長さより短い場合、空白が環境変数の値の右側に補われます。また、第2引数の長さより長い場合、環境変数の値の右側の余分な文字が無視され、第2引数の長さ分の文字列が設定されます。

第1引数で指定した環境変数が未定義の場合、第2引数には指定したスカラ変数の長さ分だけ空白が設定されます。

使用例

```
use service_routines,only:getenv
character(len=10) :: ch
call getenv('FORT90L',ch)
print *,'FORT90L=',ch
end
```

2.209 GETFDサービス関数

機能説明

装置番号に対するファイル記述子を返却します。

形式

iy = GETFD (*unit*)

unit

基本整数型スカラ。装置番号を指定します。

関数結果

基本整数型スカラ。装置番号と接続したファイルがオープンされている場合は装置番号に対するファイル記述子、オープンされていない場合は-1を返却します。

使用例

```
use service_routines,only:getfd
if (getfd(10) == getfd(11)) stop 'error'
end
```

2.210 GETGIDサービス関数

機能説明

実グループID を返却します。

形式

iy = GETGID ( )

関数結果

基本整数型スカラ。実グループID が返却されます。

使用例

```
use service_routines,only:getgid
print *,getgid( )
end
```

2.211 GETLOGサービスサブルーチン

機能説明

ログイン名を取得します。ジョブでプログラムを実行した場合、ログイン名は取得できません。

形式

CALL GETLOG (*uname*)

uname

基本文字型スカラ。ログイン名が設定されます。*uname* の文字長がログイン名よりも短い場合、ログイン名は切り捨てられます。ログイン名よりも長い場合、空白が補われます。

注意事項

ジョブでプログラムを実行する場合は、GET_ENVIRONMENT_VARIABLE 組込みサブルーチン(“2.219 GET_ENVIRONMENT_VARIABLE組込みサブルーチン”参照)またはGETENVサービスサブルーチン(“2.208 GETENVサービスサブルーチン”参照)を使用して、環境変数LOGNAMEの値を取得してください。環境変数LOGNAMEには、ジョブの実行環境のログイン名が設定されます。この環境変数は、システムによって設定されます。

使用例

```
use service_routines,only:getlog
character(len=10) :: ch
call getlog(ch)
print *,'login name=',ch
end
```

2.212 GETPARMサービスサブルーチン

機能説明

実行時オプションおよび利用者定義オプションに指定した文字長と文字列を取得します。

形式

CALL GETPARM (*len* , *parm*)

len

基本整数型スカラ。実行時オプションおよび利用者定義オプションの文字列の合計の長さが設定されます。

parm

基本文字型スカラ。実行時オプションおよび利用者定義オプションの文字列が設定されます。

注意事項

実行時オプションと利用者定義オプションで指定した文字列の合計の長さが、第2引数の長さより短い場合、空白が指定した文字列の右側に補われます。また、*parm* の長さより長い場合、指定した文字列の右側の余分な文字が無視され、第2引数の長さ分の文字列が設定されます。

使用例

```
use service_routines,only:getparm
integer :: leng
character(len=10) :: parm
call getparm(leng,parm)
write(6,*) 'length=',leng,',argument=',parm
end
```

2.213 GETPIDサービス関数

機能説明

現在のプロセスID を返却します。

形式

iy = GETPID( )

関数結果

基本整数型スカラ。現在のプロセスID が返却されます。

使用例

```
use service_routines,only:getpid
print *,getpid( )
end
```

2.214 GETTIMサービスサブルーチン

機能説明

現在の時間を取得します。

形式

CALL GETTIM (*hour* , *minute* , *second* , *second1_100*)

hour

2バイトの整数型スカラ。現在の時が設定されます。

minute

2バイトの整数型スカラ。現在の分が設定されます。

second

2バイトの整数型スカラ。現在の秒が設定されます。

second1_100

2バイトの整数型スカラ。現在の100分の1秒が設定されます。

使用例

```
use service_routines,only:gettim
integer(kind=2) :: h,m,s,s1_100
call gettim(h,m,s,s1_100)
write (6,fmt="(1x,i2,':',i2,':',i2,'.',i2)") h,m,s,s1_100
end
```

2.215 GETTODサービスサブルーチン

機能説明

GETTOD サービスサブルーチンは、現在の高分解能な実時間を返します。実時間は過去のある任意の時間からのマイクロ秒単位の経過時間です。通常、システムブートからの時間で表されます。

形式

```
CALL GETTOD ( g )
```

g

倍精度実数型スカラ。マイクロ秒単位の経過時間が設定されます。

使用例

```
use service_routines,only:gettod
real(kind = 8) :: g0, g1
call gettod(g0)
call sub
call gettod(g1)
write(6, *) g1 - g0, '(microsec)'
end
```

2.216 GETUIDサービス関数

機能説明

実ユーザID を返却します。

形式

```
iy = GETUID ( )
```

関数結果

基本整数型スカラ。実ユーザID が返却されます。

使用例

```
use service_routines,only:getuid
print *,getuid( )
end
```

2.217 GET_COMMAND組込みサブルーチン

GET_COMMAND 組込みサブルーチンは、プログラムが呼び出されたときのコマンド全体を取得します。

分類

サブルーチン

形式

CALL GET_COMMAND ([*COMMAND* , *LENGTH* , *STATUS*])

COMMAND（省略可能）

基本文字型スカラでなければなりません。INTENT(OUT) の引数です。プログラムが呼び出されたときのコマンド全体が設定されます。コマンドが存在しない場合は、すべて空白が代入されます。

LENGTH（省略可能）

基本整数型スカラでなければなりません。INTENT(OUT) の引数です。プログラムが呼び出されたときの有効コマンド長が設定されます。有効長には後続空白を含みません。コマンド長は引数*COMMAND*に設定されるときの長さに関係なく有効コマンド長が設定されます。また、引数*COMMAND*を省略した場合も有効コマンド長は設定されます。コマンドが存在しない場合は、長さとして0が設定されます。

STATUS（省略可能）

基本整数型スカラでなければなりません。INTENT(OUT) の引数です。*COMMAND*引数を指定して、長さが有効コマンド長より小さい場合、-1が設定されます。コマンドが存在しない場合は、1が設定されます。それ以外の場合、0が設定されます。

使用例

```
character(len=30) :: string
integer :: len
call get_command(string, len)   ! % a.out arg1 arg2 arg3
                                 ! のとき、string にはプログラムが呼び出されたとき
                                 ! のコマンド全体である"a.out arg1 arg2 arg3" が
                                 ! 設定されます
                                 ! len にはコマンドの長さである20が設定されます
```

2.218 GET_COMMAND_ARGUMENT組込みサブルーチン

GET_COMMAND_ARGUMENT 組込みサブルーチンは、プログラムが呼び出されたときのコマンド引数を取得します。

分類

サブルーチン

形式

CALL GET_COMMAND_ARGUMENT (*NUMBER* [, *VALUE* , *LENGTH* , *STATUS*])

NUMBER

基本整数型スカラでなければなりません。INTENT(IN) の引数です。コマンド引数の番号を指定します。指定したコマンド引数の番号の情報が、他の引数に設定されます。*NUMBER* で利用可能な値は、ゼロから組込み関数COMMAND_ARGUMENT_COUNT が返す引数の個数の間の値です。それ以外の値を指定した場合は、指定している他の引数にそれぞれのエラー状態が設定されます。
コマンド引数0は、起動されたプログラムのコマンド名です。
残りの引数は指定した順に、1から引数の個数までの連続的な番号が付けられます。

VALUE（省略可能）

基本文字型スカラでなければなりません。INTENT(OUT) の引数です。*NUMBER* によって決められるコマンド引数の値が設定されます。*NUMBER* によって決められるコマンド引数が存在しない場合は、*VALUE* にはすべて空白が設定されます。

LENGTH（省略可能）

基本整数型スカラでなければなりません。INTENT(OUT) の引数です。*NUMBER*によって決められるコマンド引数の有効長が設定されます。有効長には後続空白を含みません。コマンド引数長は引数*VALUE*に設定されるときの長さに関係なく有効コマンド長が

設定されます。また、引数*VALUE*を省略した場合も有効コマンド長は設定されます。*NUMBER*によって決められるコマンド引数が存在しない場合は、長さとして0が設定されます。

STATUS（省略可能）

基本整数型スカラでなければなりません。INTENT(OUT) の引数です。*VALUE* 引数を指定して、*NUMBER* によって決められるコマンド引数の有効長より小さい場合、-1が設定されます。*NUMBER*によって決められるコマンド引数が存在しない場合、コマンド名を含むコマンド引数の個数が設定されます。それ以外の場合、0が設定されます。

使用例

```
character(len=30) :: string
integer :: pos, len
pos = 2
call get_command_argument(pos, string, len)     ! % a.out arg1 arg2 arg3
                                                 ! のとき、string には2番目の
                                                 ! コマンド引数である"arg2" が
                                                 ! 設定されます
                                                 ! len には2番目のコマンド引数の
                                                 ! 長さである4が設定されます
```

2.219 GET_ENVIRONMENT_VARIABLE組込みサブルーチン

GET_ENVIRONMENT_VARIABLE 組込みサブルーチンは、環境変数の値を取得します。

分類

サブルーチン

形式

CALL GET_ENVIRONMENT_VARIABLE (*NAME* [, *VALUE* , *LENGTH* , *STATUS* , *TRIM_NAME*])

NAME

基本文字型スカラでなければなりません。INTENT(IN) の引数です。環境変数名を指定します。

VALUE（省略可能）

基本文字型スカラでなければなりません。INTENT(OUT) の引数です。*NAME*によって指定した環境変数の値が設定されます。環境変数が存在しないか、または値をもたない場合、*VALUE* にはすべて空白が設定されます。

LENGTH（省略可能）

基本整数型スカラでなければなりません。INTENT(OUT) の引数です。*NAME*によって指定した環境変数が存在し値をもつ場合、環境変数の値の長さが設定されます。それ以外の場合、長さとして0が設定されます。

STATUS（省略可能）

基本整数型スカラでなければなりません。INTENT(OUT) の引数です。*NAME*によって指定した環境変数が存在し、値をもたないか、または値が*VALUE*に正しく設定された場合、0が設定されます。引数*VALUE*を指定して長さが環境変数の有効長より小さい場合、-1が設定されます。環境変数が存在しない場合、1が設定されます。

TRIM_NAME（省略可能）

論理型スカラでなければなりません。INTENT(IN) 引数です。*TRIM_NAME*に偽を指定した場合は、指定した*NAME*の後続空白を有効な文字として扱います。*TRIM_NAME* を省略する、または*TRIM_NAME* に真を指定した場合は、指定した*NAME* の後続空白を無視します。

使用例

```
character(len=30) :: env = 'SHELL'
character(len=30) :: string
call get_environment_variable(env, string)      ! string にはSHELL 環境変数の値が
                                                 ! 設定されます
```

2.220 GMTIMEサービスサブルーチン

機能説明

システム時間をグリニッジ標準時間に従って、秒、分、時間、日、月、年、曜日、1月1日からの通算日付、夏時間かどうかを表す情報を取得します。

形式

```
CALL GMTIME ( time , t )
```

time

基本整数型スカラ。システム時間を指定します。

t

基本整数型配列。*time*で指定したシステム時間をグリニッジ標準時間に従って、以下の配列に設定します。要素数が9より小さい場合、サービスサブルーチンの動作は、保証されません。また、引数*t*の要素数が9より大きい場合、10要素目以降の値は変更されません。

配列	値
t(1)	秒 (0-59)
t(2)	分 (0-59)
t(3)	時 (0-23)
t(4)	日 (1-31)
t(5)	月 (0-11)
t(6)	1900からの通算年
t(7)	日曜日からの通算曜日 (0-6)
t(8)	1月1日からの通算日 (0-365)
t(9)	夏時間を示すフラグ(標準時間は0、夏時間は1)

使用例

```
use service_routines,only:gmtime,time
integer :: t(9)
call gmtime(time(),t)
write(6,fmt="(1x,9i4)") t
end
```

2.221 GO TO文

GO TO 文は、指定された文番号をもつ飛び先文に制御移行します。

GO TO 文は、以下の形式です。

```
GO TO label
```

label は、文番号であり、その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。

GO TO 文の例:

```
a = b
go to 10                ! 文番号10の飛び先文へ制御移行するため、
b = c                   ! この文は実行されません。
10 c = d
```

2.222 計算形GO TO文(廃止予定事項)

計算形GO TO 文は、指定された文番号並びのいずれか1つの飛び先文または、直後の文に制御移行します。

計算形GO TO 文は、以下の形式です。

```
GO TO ( label-list ) [ , ] scalar-int-expr
```

label-list は、コンマで区切られた文番号の並びです。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。

1つの*label-list* 中に、同じ文番号を2回以上書いてもかまいません。

scalar-int-expr は、スカラ整数式です。

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*である場合には、制御移行が起こり、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より小さい場合および*n* より大きい場合には、実行系列は、CONTINUE 文が実行されたかのように続けられます。

計算形GO TO 文の例:

```
      go to (10,20,30) i
      stop
10 a = a+1 ! i の値が1の時、この文が実行されます
      stop 1
20 a = a+2 ! i の値が2の時、この文が実行されます
      stop 2
30 a = a+3 ! i の値が3の時、この文が実行されます
      stop 3
```

2.223 割当て形GO TO文(廃止事項)

割当て形GO TO 文は、ASSIGN 文で変数に割り当てられた文番号をもつ飛び先文に制御移行します。

割当て形GO TO 文は、以下の形式です。

```
GO TO scalar-int-variable [ [ , ] ( label-list ) ]
```

scalar-int-variable は、スカラ整変数であり、基本整数型の名前付き変数でなければなりません。

label-list は、コンマで区切られた文番号の並びです。文番号は、その割当て形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。

割当て形GO TO 文が実行されたとき、その整変数は、同じ有効域内にある飛び先文の文番号の値で確定になっていなければなりません。その整変数は、その割当て形GO TO 文と同じ有効域内にあるASSIGN 文でだけ、文番号の値で確定にすることができます。

括弧でくくられた文番号並びを書いた場合、整変数に割り当てる文番号は、その並び中の文番号のいずれかでなければなりません。

割当て形GO TO 文を実行すると、制御移行が起こり、整変数に割り当てられている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。

割当て形GO TO 文の例:

```
      assign 10 to i
      goto i
10 continue
```

2.224 HOSTNMサービス関数

機能説明

現在のホスト名を返却します。

形式

iy = HOSTNM (*name*)

name

基本文字型スカラ。現在のホスト名が設定されます。

関数結果

基本整数型スカラ。ホスト名が返却できたときは0、エラーが発生したときは0以外の値を返却します。

使用例

```
use service_routines,only:hostnm
integer :: iy
character(len=100) :: name
iy = hostnm(name)
print *,'hostname = ',name
end
```

2.225 HUGE組込み関数

HUGE 関数は、*X*と同じ型および同じ種別型パラメタが取りうる最大値を返却します。

分類

問合せ関数

形式

result = HUGE (*X*)

X

整数型または実数型でなければなりません。スカラまたは配列です。

result

*X*と同じ型および種別型パラメタをもつスカラです。

機能説明

HUGE は、*X*と同じ型および種別型パラメタが取りうる最大値を返却します。

－ *X*が整数型の場合

関数の結果は、$\mathrm{RADIX}(X)^{\mathrm{DIGITS}(X)} - 1$となります。

－ *X*が実数型の場合

関数の結果は、$(1 - \mathrm{RADIX}(X)^{\mathrm{DIGITS}(X)}) \times \mathrm{RADIX}(X)^{\mathrm{MAXEXPONENT}(X)}$ となります。

関数の結果の型は、*X*と同じです。

以下の値の固定値となります。

引数の型	結果の値
1バイトの整数型	127_1
2バイトの整数型	32767_2
4バイトの整数型	2147483647_4
8バイトの整数型	9223372036854775807_8
半精度実数型	6.5504E+04
単精度実数型	3.40282347E+38
倍精度実数型	1.797693134862316D+308

引数の型	結果の値
4倍精度実数型	1.189731495357231765085759326628007 0Q+4932

使用例

```
a = huge(4.1)                ! a には3.40282347E+38が代入されます
```

2.226 HYPOT組込み関数

HYPOT 関数は、ユークリッド距離 $\sqrt{x^2+y^2}$ を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
HYPOT	-----	2	実数型 , 実数型	実数型

result = HYPOT (*X* , *Y*)

X

実数型でなければなりません。

Y

*X*と同じ型および同じ種別型パラメタをもつ実数型でなければなりません。

result

*X*と同じ型です。

機能説明

HYPOT は、ユークリッド距離 $\sqrt{x^2+y^2}$ を求めます。

関数の結果の型は、*X*と同じです。

使用例

```
r = hypot( 3.0,4.0)
```

2.227 IACHAR組込み関数

IACHAR 関数は、ASCII 大小順序における*C*の位置を返却します。

分類

要素別処理関数

形式

result = IACHAR (*C* [, *KIND*])

C

長さ1の文字型でなければなりません。

KIND(**省略可能**)

スカラ整数定数式でなければなりません。

result

整数型です。*KIND* が指定された場合、種別型パラメタは*KIND* の指定に従います。
KIND が省略された場合、種別型パラメタは基本整数型になります。

機能説明

IACHAR は、ASCII 大小順序における*C* の位置を返却します。

関数の結果の型は、種別型パラメタ*KIND* が指定された場合、種別型パラメタ*KIND* の整数型となります。*KIND* が省略された場合、関数の結果の型は基本整数型です。

使用例

```
i = iachar('c')                    ! i には99が代入されます
```

2.228 IALL組込み関数

IALL関数は、配列内の論理積を求めます。

分類

変形関数

形式

result = IALL (*ARRAY* , *DIM* [, *MASK*])　または
result = IALL (*ARRAY* [, *MASK*])

ARRAY

整数型の配列でなければなりません。

DIM

整数型スカラであって、1 <= *DIM* <= n の範囲の値でなければなりません。ここで、n は*ARRAY* の次元数とします。対応する実引数は、省略可能な仮引数であってはなりません。

MASK（**省略可能**）

論理型でなければならず、*ARRAY* と形状適合しなければなりません。

result

ARRAY と同じ型および同じ種別型パラメタです。*DIM* が省略された、または*ARRAY* が1次元配列の場合は、スカラとなります。それ以外の場合、*ARRAY* の形状を (d_1, d_2, ..., d_n) としたとき、形状は、(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_n) となります。

機能説明

IALL関数は、*ARRAY* 内の論理積を求めます。

－ *DIM*を省略している場合

MASK を省略すると、*MASK* の全要素が真と等価です。

MASK 中の真の要素に対応する*ARRAY* 内の論理積を返却します。*ARRAY* が大きさゼロであるとき、結果は -1 になります。

－ *DIM*を指定している場合

ARRAY が1次元の場合は、IALL (*ARRAY* [, *MASK*]) と等しく、結果はスカラとなります。

他の場合、結果の要素(S_1, S_2, ..., S_{DIM-1}, S_{DIM+1}, ..., S_n) は、IALL (*ARRAY*(S_1, S_2, ..., S_{DIM-1}, : , S_{DIM+1}, ..., S_n) [,MASK = *MASK*])となります。ここで、n は*ARRAY* の次元数、*MASK*は*MASK*(S_1, S_2, ..., S_{DIM-1}, : , S_{DIM+1}, ..., S_n) とします。

使用例

```
integer:: result
result = iall( [ 3, 7, 11 ] ) ! result には 3 が代入されます
```

2.229 IAND組込み関数

IAND 関数は、引数同士の論理積を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
IAND	-----	2	整数型、または 非10進定数表現 , 整数型、または 非10進定数表現	整数型
IAND , AND	-----	2	1バイトの整数型 , 1バイトの整数型	1バイトの整数型
	IIAND		2バイトの整数型 , 2バイトの整数型	2バイトの整数型
	IAND		4バイトの整数型 , 4バイトの整数型	4バイトの整数型
	AND		4バイトの整数型 , 4バイトの整数型	4バイトの整数型
	JIAND		4バイトの整数型 , 4バイトの整数型	4バイトの整数型
	-----		8バイトの整数型 , 8バイトの整数型	8バイトの整数型

result = IAND (*I* , *J*)
result = AND (*I* , *J*)

I

整数型でなければなりません。IAND は非10進定数表現を指定できます。

J

*I*と同じ種別型パラメタをもつ整数型でなければなりません。IAND は非10進定数表現を指定できます。

result

*I*または*J*と同じ型です。

機能説明

IAND、AND、IIAND、およびJIAND は、引数同士の論理積を返す関数です。

総称名IAND およびAND は、すべての整数型の引数に使用することができます。

総称名IANDの引数*I*または*J*のどちらかに非10進定数表現を指定した場合、非10進定数表現は、指定した整数型に変換されます。両方に非10進定数表現を指定してはなりません。

それぞれの関数の結果の型は、*I*または*J*と同じです。

使用例

```
i=53
j=45
k=iand(i,j) ! k には37が代入されます
```

2.230 IANY組込み関数

IANY関数は、配列内の論理和を求めます。

分類

変形関数

形式

result = IANY (*ARRAY* , *DIM* [, *MASK*])　または
result = IANY (*ARRAY* [, *MASK*])

ARRAY

整数型の配列でなければなりません。

DIM

整数型スカラであって、1 <= *DIM* <= n の範囲の値でなければなりません。ここで、n は*ARRAY*の次元数とします。対応する実引数は、省略可能な仮引数であってはなりません。

MASK（省略可能）

論理型でなければならず、*ARRAY*と形状適合しなければなりません。

result

*ARRAY*と同じ型および同じ種別型パラメタです。*DIM*が省略された、または*ARRAY*が1次元配列の場合は、スカラとなります。それ以外の場合、*ARRAY*の形状を (d_1, d_2, ..., d_n) としたとき、形状は、(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_n) となります。

機能説明

IANY関数は、*ARRAY*内の論理和を求めます。

－ *DIM*を省略している場合

*MASK*を省略すると、*MASK*の全要素が真と等価です。

*MASK*中の真の要素に対応する*ARRAY*内の論理和を返却します。*ARRAY*が大きさゼロであるとき、結果は ゼロ になります。

－ *DIM*を指定している場合

*ARRAY*が1次元の場合は、IANY (*ARRAY*[, *MASK*]) と等しく、結果はスカラとなります。

他の場合、結果の要素(S_1, S_2, ..., S_{DIM-1}, S_{DIM+1}, ..., S_n) は、IANY (*ARRAY*(S_1, S_2, ..., S_{DIM-1}, : , S_{DIM+1}, ..., S_n) [,MASK = *MASK*])となります。ここで、n は*ARRAY*の次元数、*MASK*は*MASK*(S_1, S_2, ..., S_{DIM-1}, : , S_{DIM+1}, ..., S_n) とします。

使用例

```
integer:: result
result = iany ( [ 3, 7, 11 ] ) ! result には 15 が代入されます
```

2.231 IARGCサービス関数

機能説明

実行時オプションまたは利用者定義オプションに指定した文字列の個数を返却します。

形式

iy = IARGC( )

関数結果

基本整数型スカラ。

使用例

```
use service_routines,only:iargc
print *,iargc()
end
```

2.232 IBCHNG組込み関数

IBCHNG 関数は、*I*の第*POS*ビット目を反転します。

分類

要素別処理関数

形式

result = IBCHNG (*I* , *POS*)

I

整数型でなければなりません。

POS

整数型でなければなりません。0 <= *POS* < BIT_SIZE(*I*) でなければなりません。

result

I と同じ型です。

機能説明

IBCHNG は、*I* の第 *POS* ビット目がオフ(0)の場合オン(1)にし、オンの場合オフにします。

関数の結果の型は、*I* と同じです。

使用例

```
i = ibchng(7,1)          ! i には5が代入されます
i = ibchng(8,2)          ! i には12が代入されます
```

2.233 IBCLR組込み関数

IBCLR 関数は、*I* の第 *POS* ビット目をオフ(0)にします。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
IBCLR	-----	2	1バイトの整数型 , 整数型	1バイトの整数型
	IIBCLR		2バイトの整数型 , 整数型	2バイトの整数型
	IBCLR		4バイトの整数型 , 整数型	4バイトの整数型
	JIBCLR		4バイトの整数型 , 整数型	4バイトの整数型
	-----		8バイトの整数型 , 整数型	8バイトの整数型

result = IBCLR (*I* , *POS*)

I

整数型でなければなりません。

POS

整数型でなければなりません。0 <= *POS* < BIT_SIZE(*I*) でなければなりません。

result

I と同じ型です。

機能説明

IBCLR、IIBCLR、およびJIBCLR は、*I* の第 *POS* ビット目をオフ(0)にします。

総称名IBCLR は、すべての整数型の引数に使用することができます。

それぞれの関数の結果の型は、*I* と同じです。

使用例

```
i = ibclr(37,2)                ! i には33が代入されます
```

2.234 IBITS組込み関数

IBITS 関数は、指定したビット列を取り出します。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
IBITS	-----	3	1バイトの整数型 , 整数型 , 整数型	1バイトの整数型
	IIBITS		2バイトの整数型 , 整数型 , 整数型	2バイトの整数型
	IBITS		4バイトの整数型 , 整数型 , 整数型	4バイトの整数型
	JIBITS		4バイトの整数型 , 整数型 , 整数型	4バイトの整数型
	-----		8バイトの整数型 , 整数型 , 整数型	8バイトの整数型

result = IBITS (*I* , *POS* , *LEN*)

I

整数型でなければなりません。

POS

整数型でなければなりません。0 <= *POS* + *LEN* <= BIT_SIZE(*I*) でなければなりません。

LEN

整数型でなければなりません。*LEN* >= 0 でなければなりません。

result

I と同じ型です。

機能説明

IBITS、IIBITS、およびJIBITS は、指定したビット列を取り出します。

I の *POS* ビット目から *LEN* ビット分を右詰めし、残りのビットを0にした値を返却します。

総称名IBITS は、すべての整数型の引数に使用することができます。

それぞれの関数の結果の型は、*I* と同じです。

使用例

```
i = ibits (37,2,2)             ! i には1が代入されます
```

2.235 IBSET組込み関数

IBSET 関数は、*I*の第*POS*ビット目をオン(1)にします。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
IBSET	-----	2	1バイトの整数型，整数型	1バイトの整数型
	IIBSET		2バイトの整数型，整数型	2バイトの整数型
	IBSET		4バイトの整数型，整数型	4バイトの整数型
	JIBSET		4バイトの整数型，整数型	4バイトの整数型
	-----		8バイトの整数型，整数型	8バイトの整数型

result = IBSET(*I* , *POS*)

I

整数型でなければなりません。

POS

整数型でなければなりません。0 <= *POS* < BIT_SIZE(*I*) でなければなりません。

result

*I*と同じ型です。

機能説明

IBSET、IIBSET、およびJIBSET は、*I*の第*POS*ビット目をオン(1)にします。

総称名IBSET は、すべての整数型の引数に使用することができます。

それぞれの関数の結果の型は、*I*と同じです。

使用例

```
i = ibset (37,1)                ! i には39が代入されます
```

2.236 IBTODサービスサブルーチン

機能説明

第2引数で指定した値の絶対値を文字に変換し、その下位4文字を第1引数に代入します。上位の余分な文字0は、空白に変換して代入します。

形式

CALL IBTOD (*i* , *j*)

i

基本整数型スカラ。変換した絶対値が設定されます。

j

基本整数型スカラ。絶対値に変換する値を指定します。

使用例

```
use service_routines,only:ibtod
integer :: i
call ibtod(i,123456)
write(6,fmt="(1x,a8)") i
end
```

2.237 ICHAR組込み関数

ICHAR 関数は、文字型データを処理系大小順序における位置を指す整数型データに変換します。

分類

要素別処理関数

形式

result = ICHAR (*C* [, *KIND*])

C

長さ1の文字型です。

KIND（省略可能）

スカラ整数定数式でなければなりません。

result

整数型です。*KIND*が指定された場合、種別型パラメタは*KIND*の指定に従います。*KIND*が省略された場合、種別型パラメタは基本整数型になります。

機能説明

ICHAR は、文字型データを処理系大小順序における位置を指す整数型データに変換します。

関数の結果の範囲は 0 <= ICHAR(*C*) <= 255 です。

関数の結果の型は、種別型パラメタ*KIND*が指定された場合、種別型パラメタ*KIND*の整数型となります。*KIND*が省略された場合、関数の結果の型は基本整数型です。

使用例

```
i = ichar('c')                  ! i にはASCII 大小順序により99が代入されます
```

2.238 IDATEサービスサブルーチン

機能説明

現在の日、月、年を取得します。

形式

CALL IDATE (*ia*)

ia

基本整数型配列。現在の日、月、年が設定されます。
IDATEサービスサブルーチンの引数*ia*の要素数が3より小さい場合、サービスサブルーチンの動作は、保証されません。また、引数*ia*の要素数が3より大きい場合、4要素目以降の値は変更されません。

使用例

```
use service_routines,only:idate
integer :: t(3)
call idate(t)
write(6,fmt="(1x,i2,'/',i2,'/',i4)") t
end
```

2.239 IEEE_CLASS組込みモジュール関数

IEEE 浮動小数点数の種類

分類

要素別処理関数

形式

result = IEEE_CLASS(*X*)

X

実数型でなければなりません。

result

派生型IEEE_CLASS_TYPE です。

機能説明

IEEE_CLASS は、IEEE 浮動小数点数の種類を返却します。

- IEEE_SUPPORT_NAN(*X*) が真であって、*X* の値が例外通知非数の場合、結果の値はIEEE_SIGNALING_NAN となります。
- IEEE_SUPPORT_NAN(*X*) が真であって、*X* の値が例外非通知非数の場合、結果の値はIEEE_QUIET_NAN となります。
- IEEE_SUPPORT_INF(*X*) が真であって、*X* の値が負の無限大の場合、結果の値はIEEE_NEGATIVE_INF となります。
- IEEE_SUPPORT_INF(*X*) が真であって、*X* の値が正の無限大の場合、結果の値はIEEE_POSITIVE_INF となります。
- IEEE_SUPPORT_DENORMAL(*X*) が真であって、*X* の値が負の準正規数の場合、
  結果の値はIEEE_NEGATIVE_DENORMAL となります。
- IEEE_SUPPORT_DENORMAL(*X*) が真であって、*X* の値が正の準正規数の場合、
  結果の値はIEEE_POSITIVE_DENORMAL となります。

X の値が負の正規数の場合、結果の値はIEEE_NEGATIVE_NORMAL となります。

X の値が負のゼロの場合、結果の値はIEEE_NEGATIVE_ZERO となります。

X の値が正のゼロの場合、結果の値はIEEE_POSITIVE_ZERO となります。

X の値が正の正規数の場合、結果の値はIEEE_POSITIVE_NORMAL となります。

それ以外の場合、結果の値はIEEE_OTHER_VALUE となります。

関数の結果の型は、派生型IEEE_CLASS_TYPE です。

IEEE_SUPPORT_DATATYPE(*X*) が偽になるとき、この手続を呼び出してはなりません。

使用例

```
use, intrinsic :: ieee_arithmetic
type(ieee_class_type) ::result
real :: x
x = -1.0
if (ieee_support_datatype(x)) then
  result = ieee_class(x) ! ieee_negative_normal
end if
if (result .eq. ieee_negative_normal) then
  print *, "negative_normal"
end if
```

2.240 IEEE_COPY_SIGN組込みモジュール関数

IEEE の符号の付け替え

分類

要素別処理関数

形式

result = IEEE_COPY_SIGN(*X*,*Y*)

X

実数型でなければなりません。

Y

実数型でなければなりません。

result

*X*と同じ型です。

機能説明

IEEE_COPY_SIGN は、*X*の絶対値で、*Y*の符号をもつ値を返却します。

関数の結果の型は、*X*と同じ型です。

IEEE_SUPPORT_DATATYPE(*X*) またはIEEE_SUPPORT_DATATYPE(*Y*) が偽になるとき、この手続を呼び出してはなりません。

使用例

```
use, intrinsic :: ieee_arithmetic
real ::result,x,y
x = 2.0
y = -1.0
if (ieee_support_datatype(x) .and. ieee_support_datatype(y)) then
  result = ieee_copy_sign(x,y) ! -2.0
end if
```

2.241 IEEE_GET_FLAG組込みモジュールサブルーチン

IEEE 例外フラグの取得

分類

要素別処理サブルーチン

形式

CALL IEEE_GET_FLAG(*FLAG*, *FLAG_VALUE*)

FLAG

派生型IEEE_FLAG_TYPE でなければなりません。値は、IEEE_INVALID、IEEE_OVERFLOW、IEEE_DIVIDE_BY_ZERO、IEEE_UNDERFLOW、またはIEEE_INEXACT のいずれかでなければなりません。

FALG_VALUE

基本論理型でなければなりません。INTENT(OUT) 引数です。結果の値は、*FLAG*で指定した例外フラグが例外通知のときに真が返却され、それ以外のときは偽が返却されます。

機能説明

例外フラグが例外通知かどうかを返却します。

使用例

```
use, intrinsic :: ieee_arithmetic
logical ::flag_value
call ieee_get_flag(ieee_overflow, flag_value)
```

2.242 IEEE_GET_HALTING_MODE組込みモジュールサブルーチン

IEEE 例外に対する停止モードの取得

分類

要素別処理サブルーチン

形式

CALL IEEE_GET_HALTING_MODE(*FLAG*,*HALTING*)

FLAG

派生型IEEE_FLAG_TYPE でなければなりません。値は、IEEE_INVALID、IEEE_OVERFLOW、IEEE_DIVIDE_BY_ZERO、IEEE_UNDERFLOW、またはIEEE_INEXACT のいずれかでなければなりません。

HALTING

基本論理型でなければなりません。INTENT(OUT) 引数です。*FLAG* で指定した例外が停止を引き起こすときは真が返却され、それ以外のときは偽が返却されます。

機能説明

例外に対する停止モードを返却します。

使用例

IEEE_OVERFLOW に対する停止モードを退避し、停止モードを実行を継続する状態で計算を行い、その後で停止モードを回復する例

```
use, intrinsic :: ieee_arithmetic
logical ::halting
type(ieee_flag_type) ::flag
flag = ieee_overflow
...
call ieee_get_halting_mode(flag, halting)        ! 停止モードを退避
if (ieee_support_halting(flag)) then
  call ieee_set_halting_mode(flag, .false.)      ! 停止モードを継続にする
  ...                                            ! 停止せずに計算を実行する
end if
call ieee_set_halting_mode(flag, halting)        ! 停止モードを回復
```

2.243 IEEE_GET_ROUNDING_MODE組込みモジュールサブルーチン

IEEE 丸めモードの取得

分類

サブルーチン

形式

CALL IEEE_GET_ROUNDING_MODE(*ROUND_VALUE*)

ROUND_VALUE

派生型IEEE_ROUND_TYPE のスカラでなければなりません。INTENT(OUT) 引数です。浮動小数点数の丸めモードが返却されます。返される値は、丸めモードが直近丸めであればIEEE_NEAREST、切捨てであればIEEE_TO_ZERO、切り上げであればIEEE_UP、切下げであればIEEE_DOWN になります。それ以外の場合はIEEE_OTHER となります。

機能説明

現在のIEEE 丸めモードを返却します。

使用例

丸めモードを退避し、丸めモードを直近丸め(IEEE_NEAREST) に設定して計算を行い、その後、丸めモードを回復する例

```
use, intrinsic :: ieee_arithmetic
type(ieee_round_type) ::round_value
real ::x
round_value = ieee_nearest
  ...
call ieee_get_rounding_mode(round_value)  ! 丸めモードを退避
if (ieee_support_datatype(x)) then
  if (ieee_support_rounding(round_value, x)) then
    call ieee_set_rounding_mode(ieee_nearest)
    ...  ! 丸めモードをnearest にして計算を実行する
  end if
end if
call ieee_set_rounding_mode(round_value)  ! 丸めモードを回復
```

2.244 IEEE_GET_STATUS組込みモジュールサブルーチン

IEEE 浮動小数点状態の取得

分類

サブルーチン

形式

CALL IEEE_GET_STATUS(*STATUS_VALUE*)

STATUS_VALUE

派生型IEEE_STATUS_TYPE のスカラでなければなりません。INTENT(OUT) 引数です。浮動小数点数の状態が返却されます。

機能説明

浮動小数点状態の現在の値を返却します。

浮動小数点状態とは、例外、丸めモード、下位けたあふれモードおよび停止に対して用意されているフラグ全体の値です。

使用例

浮動小数点状態をすべて退避し、すべての例外フラグを例外非通知に設定し、例外処理を含む計算を行い、その後、浮動小数点状態を回復する例

```
use, intrinsic :: ieee_arithmetic
type(ieee_status_type) ::status_value
...
call ieee_get_status(status_value)  ! 浮動小数点状態を退避
call ieee_set_flag(ieee_all, (/.false.,.false.,.false.,.false.,.false./))
! フラグを例外非通知に設定
...  ! 例外処理を含む計算を実行する
call ieee_set_status(status_value)  ! 浮動小数点状態を回復
```

2.245 IEEE_GET_UNDERFLOW_MODE組込みモジュールサブルーチン

IEEE 下位けたあふれモードの取得

分類

サブルーチン

形式

CALL IEEE_GET_UNDERFLOW_MODE(*GRADUAL*)

GRADUAL

基本論理型スカラでなければなりません。INTENT(OUT) 引数です。現在の下位けたあふれモードが漸次下位けたあふれである場合は、真が返却されます。現在の下位けたあふれモードが急衰下位けたあふれの場合は、偽が返却されます。

機能説明

現在の下位けたあふれモードを返却します。

漸次下位けたあふれとは、下位けたあふれが準正規数となることです。急衰下位けたあふれとは、下位けたあふれがゼロになることです。

IEEE_SUPPORT_UNDERFLOW_CONTROL(X) があるX に対して真でないとき、この手続を呼び出してはなりません。

使用例

急衰下位けたあふれで計算を実行し、以前のモードに回復する例

```
use, intrinsic :: ieee_arithmetic
logical ::save_underflow_mode
real ::x
...
if (ieee_support_underflow_control(x)) then
  call ieee_get_underflow_mode(save_underflow_mode)
  ! 下位けたあふれモードを退避
  call ieee_set_underflow_mode(gradual=.false.)
  ... ! 急衰下位けたあふれで計算を実行する
  call ieee_set_underflow_mode(save_underflow_mode)
  ! 下位けたあふれモードを回復
end if
```

2.246 IEEE_IS_FINITE組込みモジュール関数

IEEE 有限数の判定

分類

要素別処理関数

形式

result = IEEE_IS_FINITE(*X*)

X

実数型でなければなりません。

result

基本論理型です。

機能説明

IEEE_IS_FINITE は、*X*が有限のとき真を返却します。すなわち、IEEE_CLASS(*X*) が IEEE_NEGATIVE_NORMAL(負の正規数)、IEEE_NEGATIVE_DENORMAL(負の準正規数)、IEEE_NEGATIVE_ZERO(負のゼロ)、IEEE_POSITIVE_ZERO(正のゼロ)、IEEE_POSITIVE_DENORMAL(正の準正規数)、またはIEEE_POSITIVE_NORMAL(正の正規数)のいずれかのときは真を返却します。それ以外のときは偽を返却します。

関数の結果の型は、基本論理型です。

IEEE_SUPPORT_DATATYPE(*X*) が偽になるとき、この手続を呼び出してはなりません。

使用例

有限と判定される例

```
use, intrinsic :: ieee_arithmetic
logical ::result
real ::x
x = 1.0
if (ieee_support_datatype(x)) then
  result = ieee_is_finite(x) ! 真
end if
```

2.247 IEEE_IS_NAN組込みモジュール関数

IEEE 非数の判定

分類

要素別処理関数

形式

result = IEEE_IS_NAN(*X*)

X

実数型でなければなりません。

result

基本論理型です。

機能説明

IEEE_IS_NAN は、*X* がIEEE 非数のとき真を返却します。それ以外のとき偽を返却します。

関数の結果の型は、基本論理型です。

IEEE_SUPPORT_NAN(*X*) が偽になるとき、この手続を呼び出してはなりません。

使用例

```
use, intrinsic :: ieee_arithmetic
real :: x,y
logical ::result
y = 0.0
x = 0.0/y
if (ieee_support_nan(x)) then
  result = ieee_is_nan(x)     ! 真
end if
```

2.248 IEEE_IS_NEGATIVE組込みモジュール関数

IEEE 負数の判定

分類

要素別処理関数

形式

result = IEEE_IS_NEGATIVE(*X*)

X

実数型でなければなりません。

result

基本論理型です。

機能説明

IEEE_IS_NEGATIVE は、*X* の値が負のとき真を返却します。すなわち、IEEE_CLASS(*X*) がIEEE_NEGATIVE_NORMAL(負の正規数)、IEEE_NEGATIVE_DENORMAL(負の準正規数)、IEEE_NEGATIVE_ZERO(負のゼロ)、
またはIEEE_NEGATIVE_INF(負の無限大) のいずれかのときは真を返却します。それ以外のときは偽を返却します。

関数の結果の型は、基本論理型です。

IEEE_SUPPORT_DATATYPE(*X*) が偽になるとき、この手続を呼び出してはなりません。

使用例

```
use, intrinsic :: ieee_arithmetic
real ::x
logical ::result
x = 0.0
if (ieee_support_datatype(x)) then
  result = ieee_is_negative(x) ! 偽
end if
```

2.249 IEEE_IS_NORMAL組込みモジュール関数

IEEE 正規数の判定

分類

要素別処理関数

形式

result = IEEE_IS_NORMAL(*X*)

X

実数型でなければなりません。

result

基本論理型です。

機能説明

IEEE_IS_NORMAL は、*X*の値が正規のとき真を返却します。すなわち、IEEE_CLASS(*X*) が IEEE_NEGATIVE_NORMAL(負の正規数)、IEEE_NEGATIVE_ZERO(負のゼロ)、IEEE_POSITIVE_ZERO(正のゼロ) またはIEEE_POSITIVE_NORMAL(正の正規数) のいずれかのときは真を返却します。それ以外のときは偽を返却します。

関数の結果の型は、基本論理型です。

IEEE_SUPPORT_DATATYPE(*X*) が偽になるとき、この手続を呼び出してはなりません。

使用例

```
use, intrinsic :: ieee_arithmetic
real ::x
logical ::result
x = 0.0
if (ieee_support_datatype(x)) then
  result = ieee_is_normal(x)   ! 真
end if
```

2.250 IEEE_LOGB組込みモジュール関数

IEEE 浮動小数点数のかさ上げ分をはずした指数部

分類

要素別処理関数

形式

result = IEEE_LOGB(*X*)

X

実数型でなければなりません。

result

*X*と同じ型です。

機能説明

IEEE_LOGB は、以下の結果を返却します。

- *X* の値が0、無限大、または非数のいずれでもない場合、*X* のかさ上げ分をはずした指数部を返却します。この値は、EXPONENT(*X*)-1 に等しくなります。
- *X*の値が0の場合、IEEE_SUPPORT_INF(*X*) が真のとき -∞ を返却します。
  それ以外のときは -HUGE(*X*) を返却し、IEEE_DIVIDE_BY_ZERO が発生します。
- *X*の値が無限大の場合、無限大となります。
- *X*の値が非数の場合、非数となります。

関数の結果の型は、*X*と同じ型です。

IEEE_SUPPORT_DATATYPE(*X*) が偽になるとき、この手続を呼び出してはなりません。

使用例

```
use, intrinsic :: ieee_arithmetic
real :: result,x
x = -1.1
if (ieee_support_datatype(x)) then
  result = ieee_logb(x)
end if
```

result は、0.0になります。

2.251 IEEE_NEXT_AFTER組込みモジュール関数

与えられた値に対して、指定された方向にある最も近い実数値

分類

要素別処理関数

形式

result = IEEE_NEXT_AFTER(*X,Y*)

X

実数型でなければなりません。

Y

実数型でなければなりません。

result

*X*と同じ型です。

機能説明

IEEE_NEXT_AFTER は、以下の結果を返却します。

- *X*== *Y*の場合、*X*となります。この場合に、例外は発生しません。
- *X*≠ *Y*の場合、結果は*Y*方向に*X*の隣にある表現可能な値となります。0の直近の値は、符号にかかわらず0以外の値となります。*X*が有限でIEEE_NEXT_AFTER(*X, Y*) が無限大のとき、IEEE_OVERFLOW が発生します。IEEE_NEXT_AFTER(*X, Y*) が準正規数のとき、IEEE_UNDERFLOW が発生します。どちらのときもIEEE_INEXACT が発生します。

  関数の結果の型は、*X*と同じ型です。

  IEEE_SUPPORT_DATATYPE(*X*) またはIEEE_SUPPORT_DATATYPE(*Y*) が偽になるとき、この手続を呼び出してはなりません。

使用例

```
use, intrinsic :: ieee_arithmetic
real :: result,x,y
x=1.0
```

```
y=2.0
if (ieee_support_datatype(x) .and. ieee_support_datatype(y)) then
  result = ieee_next_after(x,y)
end if
```

result は、1.0+epsilon(x) になります。

2.252 IEEE_REM組込みモジュール関数

IEEE の剰余

分類

要素別処理関数

形式

result = IEEE_REM(*X*,*Y*)

X

実数型でなければなりません。

Y

実数型でなければなりません。

result

2つの引数のうち精度の高いほうの種別型パラメタをもつ実数型です。

機能説明

IEEE_REM は、丸めモードに関係なく厳密に*X* - *Y* × N となります。ただし、N は*X* / *Y* の厳密な値に最も近い整数です。
| N - *X* / *Y* |=1/2 の場合、N は偶数でなければなりません。結果がゼロになる場合そのゼロの符号は、*X* と同じになります。

関数の結果の型は、2つの引数のうち精度の高いほうの種別型パラメタをもつ実数型です。

IEEE_SUPPORT_DATATYPE(*X*) またはIEEE_SUPPORT_DATATYPE(*Y*) が偽になるとき、この手続を呼び出してはなりません。

使用例

```
use, intrinsic :: ieee_arithmetic
real :: result,x,y
x = 4.0
y = 3.0
if (ieee_support_datatype(x) .and. ieee_support_datatype(y)) then
  result = ieee_rem(x,y)
end if
```

result は、1.0になります。

2.253 IEEE_RINT組込みモジュール関数

IEEE 丸めモードでの整数値への変換

分類

要素別処理関数

形式

result = IEEE_RINT(*X*)

X

実数型でなければなりません。

result

X と同じ型です。

機能説明

IEEE_RINT は、現在の丸めモードに従って*X*を整数に丸めた値を返却します。結果がゼロになる場合そのゼロの符号は、*X*と同じになります。

関数の結果の型は、*X*と同じ型です。

IEEE_SUPPORT_DATATYPE(*X*) が偽になるとき、この手続を呼び出してはなりません。

使用例

```
use, intrinsic :: ieee_arithmetic
real :: result,x
x = 1.1
if (ieee_support_datatype(x)) then
  call ieee_set_rounding_mode(ieee_nearest) ! 丸めモードを直近丸めに設定
  result = ieee_rint(x)
end if
```

result は、1.0になります。

2.254 IEEE_SCALB組込みモジュール関数

IEEE 浮動小数点数と2の整数べき乗との積

分類

要素別処理関数

形式

result = IEEE_SCALB(*X*,*I*)

X

実数型でなければなりません。

I

整数型でなければなりません。

result

*X*と同じ型です。

機能説明

IEEE_SCALB は、以下の結果を返却します。

- *X*に2の*I*乗を掛けた値が正規数として表現できる場合は、結果はその値となります。
- *X*が有限で、*X*に2の*I*乗を掛けた値が大き過ぎる場合は、IEEE_OVERFLOW が発生します。IEEE_SUPPORT_INF(*X*)が真を返すとき、結果の値は無限大となります。ただし、その符号は*X*と同じです。それ以外のときは、SIGN(HUGE(*X*),*X*) となります。
- *X*に2の*I*乗を掛けた値が小さ過ぎ、精度が落ちる場合は、IEEE_UNDERFLOW が発生します。結果の値は、絶対値が2の*I*乗に最も近く、符号が*X*と同一の表現可能な数となります。
- *X*が無限大のときは、結果は*X*と同じになります。例外は発生しません。

関数の結果の型は、*X*と同じ型です。

IEEE_SUPPORT_DATATYPE(*X*) が偽になるとき、この手続を呼び出してはなりません。

使用例

```
use, intrinsic :: ieee_arithmetic
real :: result,x
integer ::i
x = 1.0
i = 2
if (ieee_support_datatype(x)) then
```

```
  result = ieee_scalb(x,i)
end if
```

result は、4.0になります。

2.255 IEEE_SELECTED_REAL_KIND組込みモジュール関数

指定された精度と指数範囲に対応する種別型パラメタ値

分類

変形関数

形式

result = IEEE_SELECTED_REAL_KIND([*P*, *R*, *RADIX*])

P **(省略可能)**

整数型スカラでなければいけません。省略した場合、値0を指定したとみなします。

R **(省略可能)**

整数型スカラでなければいけません。省略した場合、値0を指定したとみなします。

RADIX **(省略可能)**

整数型スカラでなければいけません。省略した場合、値2を指定したとみなします。

result

基本整数型スカラです。

機能説明

IEEE_SELECTED_REAL_KIND は、組込み関数PRECISIONによって返されるような*P*けた以上の10進精度、組込み関数RANGEによって返されるような*R*以上の10進指数範囲、および組込み関数RADIXによって返されるような*RADIX*の基数をもつIEEE 実数の種別型パラメタ値を返却します。複数の種別型パラメタが条件を満たす場合、最小の10進精度をもつ種別型パラメタ値を返却します。それが複数ある場合は、その中で最小の種別型パラメタ値を返却します。この型をもつデータ実体X に対しては、IEEE_SUPPORT_DATATYPE(X) の結果は真になります。

少なくとも1つの引数を指定しなければなりません。

*RADIX*は、値2だけが有効です。

種別型パラメタが利用できない場合の結果の値は、精度が利用できない場合は-1、指数範囲が利用できない場合は-2、両方とも利用できない場合は-3を返却します。*RADIX*が値2でない場合は、-5を返却します。

関数の結果の型は、基本整数型です。

使用例

```
use, intrinsic :: ieee_arithmetic
integer ::result
result = ieee_selected_real_kind(6,30)
```

result は、4になります。

2.256 IEEE_SET_FLAG組込みモジュールサブルーチン

IEEE 例外フラグの設定

分類

純粋サブルーチン

形式

CALL IEEE_SET_FLAG(*FLAG*,*FLAG_VALUE*)

FLAG

派生型IEEE_FLAG_TYPE でなければなりません。*FLAG* の値がIEEE_INVALID、IEEE_OVERFLOW、IEEE_DIVIDE_BY_ZERO、IEEE_UNDERFLOW 、または IEEE_INEXACT の場合、対応する例外フラグに値を設定します。*FLAG* が配列の場合、その要素には同じ値が設定されてはなりません。

FALG_VALUE

基本論理型でなければなりません。*FLAG*と形状適合していなければなりません。要素の値が真のとき、対応するフラグを例外通知に設定し、それ以外のときは例外非通知に設定します。

機能説明

FLAG で指定された例外フラグに、*FLAG_VALUE* で指定された値により、例外通知または例外非通知を設定します。

使用例

例外フラグIEEE_OVERFLOW に例外通知を設定する例

```
use, intrinsic :: ieee_arithmetic
call ieee_set_flag(ieee_overflow, .true.)
```

2.257 IEEE_SET_HALTING_MODE組込みモジュールサブルーチン

IEEE 例外に対する停止モードの設定

分類

純粋サブルーチン

形式

CALL IEEE_SET_HALTING_MODE(*FLAG*,*HALTING*)

FLAG

派生型IEEE_FLAG_TYPE でなければなりません。値は、IEEE_INVALID、IEEE_OVERFLOW、IEEE_DIVIDE_BY_ZERO、IEEE_UNDERFLOW、またはIEEE_INEXACT のいずれかでなければなりません。*FLAG*が配列の場合、その要素には同じ値が設定されてはなりません。

HALTING

基本論理型でなければなりません。*FLAG*と形状適合していなければなりません。要素の値が真のとき、対応する*FLAG*の要素で指定した例外が発生すると実行は停止し、それ以外のときは実行を続行します。

機能説明

例外が発生した後、続行するか停止するかを制御します。

IEEE_SUPPORT_HALTING(*FLAG*) が偽になるとき、この手続を呼び出してはなりません。

使用例

IEEE_OVERFLOW に対する停止モードを退避し、停止モードを実行を継続する状態で計算を行い、その後で停止モードを回復する例

```
use, intrinsic :: ieee_arithmetic
logical ::halting
type(ieee_flag_type) ::flag
flag = ieee_overflow
...
call ieee_get_halting_mode(flag, halting)       ! 停止モードを退避
if (ieee_support_halting(flag)) then
  call ieee_set_halting_mode(flag, .false.)     ! 停止モードを継続にする
  ... ! 停止せずに計算を実行する
end if
call ieee_set_halting_mode(flag, halting)       ! 停止モードを回復
```

2.258 IEEE_SET_ROUNDING_MODE組込みモジュールサブルーチン

IEEE 丸めモードの設定

分類

サブルーチン

形式

CALL IEEE_SET_ROUNDING_MODE(*ROUND_VALUE*)

ROUND_VALUE

派生型IEEE_ROUND_TYPE のスカラでなければなりません。浮動小数点数の丸めモードを指定します。

機能説明

現在のIEEE 丸めモードを、*ROUND_VALUE* で指定されたモードに変更します。

IEEE_SUPPORT_DATATYPE(X) が真になるX に対して、IEEE_SUPPORT_ROUNDING(*ROUND_VALUE*, X) が真にならなければ、この手続を呼び出してはなりません。

使用例

丸めモードを退避し、丸めモードを直近丸め(IEEE_NEAREST) に設定して計算を行い、その後、丸めモードを回復する例

```
use, intrinsic :: ieee_arithmetic
type(ieee_round_type) ::round_value, round_back
real ::x
round_value = ieee_nearest
...
call ieee_get_rounding_mode(round_back)   ! 丸めモードを退避
if (ieee_support_datatype(x)) then
  if (ieee_support_rounding(round_value, x)) then
    call ieee_set_rounding_mode(round_value)
    ...   ! 丸めモードをnearest にして計算を実行する
  end if
end if
call ieee_set_rounding_mode(round_back)   ! 丸めモードを回復
```

2.259 IEEE_SET_STATUS組込みモジュールサブルーチン

IEEE 浮動小数点状態の設定

分類

サブルーチン

形式

CALL IEEE_SET_STATUS(*STATUS_VALUE*)

STATUS_VALUE

派生型IEEE_STATUS_TYPE のスカラでなければなりません。浮動小数点状態の元の値を指定します。値は、この手続を呼び出す前にIEEE_GET_STATUS を呼び出して得た値でなければなりません。

機能説明

浮動小数点状態の値を元に戻します。

使用例

浮動小数点状態をすべて退避し、すべての例外フラグを例外非通知に設定し、例外処理を含む計算を行い、その後、浮動小数点状態を回復する例

```
use, intrinsic :: ieee_arithmetic
type(ieee_status_type) :: status_value
...
```

```
call ieee_get_status(status_value)   ! 浮動小数点状態を退避
call ieee_set_flag(ieee_all, (/.false.,.false.,.false.,.false.,.false./))
! フラグを例外非通知に設定
...   ! 例外処理を含む計算を実行する
call ieee_set_status(status_value)   ! 浮動小数点状態を回復
```

2.260 IEEE_SET_UNDERFLOW_MODE組込みモジュールサブルーチン

IEEE 下位けたあふれモードの設定

分類

サブルーチン

形式

CALL IEEE_SET_UNDERFLOW_MODE(*GRADUAL*)

GRADUAL

基本論理型スカラでなければなりません。この値が真のとき、現在の下位けたあふれモードを漸次下位けたあふれに設定します。この値が偽のとき、現在の下位けたあふれモードを急衰下位けたあふれに設定します。

機能説明

現在の下位けたあふれモードを設定します。

漸次下位けたあふれとは、下位けたあふれが準正規数となることです。急衰下位けたあふれとは、下位けたあふれがゼロになることです。

IEEE_SUPPORT_UNDERFLOW_CONTROL(X) が真でないとき、この手続を呼び出してはなりません。

使用例

急衰下位けたあふれで計算を実行し、以前のモードに回復する例

```
use, intrinsic :: ieee_arithmetic
logical ::save_underflow_mode
real ::x
...
if (ieee_support_underflow_control(x)) then
  call ieee_get_underflow_mode(save_underflow_mode)
  ! 下位けたあふれモードを退避
  call ieee_set_underflow_mode(gradual=.false.)
  ... ! 急衰下位けたあふれで計算を実行する
  call ieee_set_underflow_mode(save_underflow_mode)
  ! 下位けたあふれモードを回復
end if
```

2.261 IEEE_SUPPORT_DATATYPE組込みモジュール関数

IEEE 算術演算のサポート判定

分類

問合せ関数

形式

result = IEEE_SUPPORT_DATATYPE()　　または
result = IEEE_SUPPORT_DATATYPE(*X*)

X

実数型でなければなりません。

result

基本論理型スカラです。

機能説明

IEEE_SUPPORT_DATATYPE は、引数なしで呼び出された場合はすべての実数型に対して、引数が与えられた場合はそれと同じ種別型パラメタの実数型の変数に対して、処理系がIEEE 算術を用意しているとき、真を返却します。それ以外のときは偽を返却します。

関数の結果の型は、基本論理型スカラです。

使用例

```
use, intrinsic :: ieee_arithmetic
logical ::result
result = ieee_support_datatype()
```

2.262 IEEE_SUPPORT_DENORMAL組込みモジュール関数

IEEE 準正規数のサポート判定

分類

問合せ関数

形式

result = IEEE_SUPPORT_DENORMAL()　　または
result = IEEE_SUPPORT_DENORMAL(*X*)

X

実数型でなければなりません。

result

基本論理型スカラです。

機能説明

IEEE_SUPPORT_DENORMAL は、以下の結果を返却します。

- IEEE_SUPPORT_DENORMAL(*X*) は、IEEE_SUPPORT_DATATYPE(*X*) の値が真で、*X*と同じ種別型パラメタの実数型に対して処理系が準正規数の算術演算と代入を用意している場合に、真を返却します。それ以外の場合は偽を返却します。
- IEEE_SUPPORT_DENORMAL() は、すべての実数型*X*に対してIEEE_SUPPORT_DENORMAL(*X*) の値が真である場合、真を返却します。それ以外の場合は偽を返却します。

関数の結果の型は、基本論理型スカラです。

使用例

```
use, intrinsic :: ieee_arithmetic
logical ::result
result = ieee_support_denormal()
```

2.263 IEEE_SUPPORT_DIVIDE組込みモジュール関数

IEEE 除算のサポート判定

分類

問合せ関数

形式

result = IEEE_SUPPORT_DIVIDE()　　または
result = IEEE_SUPPORT_DIVIDE(*X*)

X

実数型でなければなりません。

result

基本論理型スカラです。

機能説明

IEEE_SUPPORT_DIVIDE は、以下の結果を返却します。

- IEEE_SUPPORT_DIVIDE(*X*) は、*X*と同じ種別型パラメタの実数型に対して、IEEE 国際規格で規定した精度をもつ除算を用意している場合に、真を返却します。それ以外の場合は偽を返却します。
- IEEE_SUPPORT_DIVIDE() は、すべての実数型*X* に対してIEEE_SUPPORT_DIVIDE(*X*) の値が真である場合、真を返却します。それ以外の場合は偽を返却します。

関数の結果の型は、基本論理型スカラです。

使用例

```
use, intrinsic :: ieee_arithmetic
logical ::result
result = ieee_support_divide()
```

2.264 IEEE_SUPPORT_FLAG組込みモジュール関数

IEEE 例外のサポート判定

分類

問合せ関数

形式

result = IEEE_SUPPORT_FLAG(*FLAG*)　　または
result = IEEE_SUPPORT_FLAG(*FLAG*,*X*)

FLAG

派生型IEEE_FLAG_TYPE のスカラでなければなりません。*FLAG* の値は、IEEE_INVALID、IEEE_OVERFLOW、IEEE_DIVIDE_BY_ZERO、IEEE_UNDERFLOW またはIEEE_INEXACT のいずれかでなければなりません。

X

実数型でなければなりません。

result

基本論理型スカラです。

機能説明

IEEE_SUPPORT_FLAG は、以下の結果を返却します。

- IEEE_SUPPORT_FLAG(*FLAG*, *X*) は、*X*と同じ種別型パラメタの実数型に対して、指定された例外を用意している場合に、真を返却します。それ以外の場合は偽を返却します。
- IEEE_SUPPORT_FLAG(*FLAG*) は、すべての実数型X に対してIEEE_SUPPORT_FLAG(*FLAG,X*) の値が真である場合、真を返却します。それ以外の場合は偽を返却します。

関数の結果の型は、基本論理型スカラです。

使用例

```
use, intrinsic :: ieee_arithmetic
real ::x
logical ::result
result = ieee_support_flag(ieee_invalid,x)
```

2.265 IEEE_SUPPORT_HALTING組込みモジュール関数

IEEE 停止モードのサポート判定

分類

問合せ関数

形式

result = IEEE_SUPPORT_HALTING(*FLAG*)

FLAG

派生型IEEE_FLAG_TYPE のスカラでなければなりません。*FLAG* の値は、IEEE_INVALID、IEEE_OVERFLOW、IEEE_DIVIDE_BY_ZERO、IEEE_UNDERFLOW またはIEEE_INEXACT のいずれかでなければなりません。

result

基本論理型スカラです。

機能説明

IEEE_SUPPORT_HALTING は、*FLAG*で指定した例外の発生後に、その後のプログラム実行を中断するか続行するかを制御できる場合、真を返却します。それ以外の場合は偽を返却します。

関数の結果の型は、基本論理型スカラです。

使用例

処理系がINVALID 例外を用意しているか問い合わせる例

```
use, intrinsic :: ieee_arithmetic
logical ::result
result = ieee_support_halting(ieee_invalid)
```

2.266 IEEE_SUPPORT_INF組込みモジュール関数

IEEE 無限大のサポート判定

分類

問合せ関数

形式

result = IEEE_SUPPORT_INF()　　または
result = IEEE_SUPPORT_INF(*X*)

X

実数型でなければなりません。

result

基本論理型スカラです。

機能説明

IEEE_SUPPORT_INF は、以下の結果を返却します。

- IEEE_SUPPORT_INF(*X*)は、*X*と同じ種別型パラメタの実数型に対して、IEEE の正負いずれの無限大も用意している場合、真を返却します。それ以外の場合は偽を返却します。
- IEEE_SUPPORT_INF()は、すべての実数型*X*に対してIEEE_SUPPORT_INF(*X*)の値が真である場合、真を返却します。それ以外の場合は偽を返却します。

関数の結果の型は、基本論理型スカラです。

使用例

```
use, intrinsic :: ieee_arithmetic
logical ::result
result = ieee_support_inf()
```

2.267 IEEE_SUPPORT_IO組込みモジュール関数

IEEE 書式処理のサポート判定

分類

問合せ関数

形式

result = IEEE_SUPPORT_IO()　　または
result = IEEE_SUPPORT_IO(*X*)

X

実数型でなければなりません。

result

基本論理型スカラです。

機能説明

IEEE_SUPPORT_IO は、以下の結果を返却します。

- IEEE_SUPPORT_IO(*X*) は、書式付き入出力において、*X*と同じ種別型パラメタの実数型に対して、IEEE 国際規格で規定されたとおりの入出力丸めモードUP、DOWN、ZERO およびNEAREST に対する基数変換を用意している場合、真を返却します。それ以外の場合は偽を返却します。
- IEEE_SUPPORT_IO() は、すべての実数型X に対してIEEE_SUPPORT_IO(*X*) の値が真である場合、真を返却します。それ以外の場合は偽を返却します。

関数の結果の型は、基本論理型スカラです。

使用例

```
use, intrinsic :: ieee_arithmetic
logical ::result
result = ieee_support_io()
```

2.268 IEEE_SUPPORT_NAN組込みモジュール関数

IEEE 非数のサポート判定

分類

問合せ関数

形式

result = IEEE_SUPPORT_NAN()　　または
result = IEEE_SUPPORT_NAN(*X*)

X

実数型でなければなりません。

result

基本論理型スカラです。

機能説明

IEEE_SUPPORT_NAN は、以下の結果を返却します。

- IEEE_SUPPORT_NAN(*X*) は、*X*と同じ種別型パラメタの実数型に対して、IEEE 国際規格の非数を用意している場合、真を返却します。それ以外の場合は偽を返却します。
- IEEE_SUPPORT_NAN() は、すべての実数型*X*に対してIEEE_SUPPORT_NAN(*X*) の値が真である場合、真を返却します。それ以外の場合は偽を返却します。

関数の結果の型は、基本論理型スカラです。

使用例

```
use, intrinsic :: ieee_arithmetic
logical ::result
result = ieee_support_nan()
```

2.269 IEEE_SUPPORT_ROUNDING組込みモジュール関数

IEEE 丸めモードのサポート判定

分類

問合せ関数

形式

result = IEEE_SUPPORT_ROUNDING(*ROUND_VALUE*)　　または
result = IEEE_SUPPORT_ROUNDING(*ROUND_VALUE,X*)

ROUND_VALUE

派生型IEEE_ROUND_TYPE でなければなりません。

X

実数型でなければなりません。

result

基本論理型スカラです。

機能説明

IEEE_SUPPORT_ROUNDING は、以下の結果を返却します。

- IEEE_SUPPORT_ROUNDING(*ROUND_VALUE,X*) は、*X*と同じ種別型パラメタの実数型に対して、*ROUND_VALUE*で指定した丸めモードを用意している場合、真を返却します。それ以外の場合は偽を返却します。
- IEEE_SUPPORT_ROUNDING(*ROUND_VALUE*) は、すべての実数型*X*に対して IEEE_SUPPORT_ROUNDING(*ROUND_VALUE,X*)の値が真である場合、真を返却します。それ以外の場合は偽を返却します。

関数の結果の型は、基本論理型スカラです。

使用例

```
use, intrinsic :: ieee_arithmetic
type(ieee_round_type) ::round_value
logical ::result
round_value = ieee_nearest
result = ieee_support_rounding(round_value)
```

2.270 IEEE_SUPPORT_SQRT組込みモジュール関数

IEEE 平方根のサポート判定

分類

問合せ関数

形式

result = IEEE_SUPPORT_SQRT()　　または
result = IEEE_SUPPORT_SQRT(*X*)

X

実数型でなければなりません。

result

基本論理型スカラです。

機能説明

IEEE_SUPPORT_SQRT は、以下の結果を返却します。

- IEEE_SUPPORT_SQRT(*X*) は、*X*と同じ種別型パラメタの実数型に対して、IEEE 国際規格の平方根を実装している場合、真を返却します。それ以外の場合は偽を返却します。
- IEEE_SUPPORT_SQRT() は、すべての実数型*X*に対してIEEE_SUPPORT_SQRT(*X*) の値が真である場合、真を返却します。それ以外の場合は偽を返却します。

関数の結果の型は、基本論理型スカラです。

使用例

```
use, intrinsic :: ieee_arithmetic
logical ::result
result = ieee_support_sqrt()
```

2.271 IEEE_SUPPORT_STANDARD組込みモジュール関数

IEEE 機能のサポート判定

分類

問合せ関数

形式

result = IEEE_SUPPORT_STANDARD()　　または
result = IEEE_SUPPORT_STANDARD(*X*)

X

実数型でなければなりません。

result

基本論理型スカラです。

機能説明

IEEE_SUPPORT_STANDARD は、IEEE の機能をすべて用意しているかどうかを判定します。以下の結果を返却します。

- IEEE_SUPPORT_STANDARD(*X*) は、関数群 IEEE_SUPPORT_DATATYPE(*X*)、IEEE_SUPPORT_DENORMAL(*X*)、IEEE_SUPPORT_DIVIDE(*X*)、有効なFLAG に対するIEEE_SUPPORT_FLAG(FLAG,*X*)、有効なFLAG に対するIEEE_SUPPORT_HALTING(*FLAG*)、IEEE_SUPPORT_INF(*X*)、IEEE_SUPPORT_NAN(*X*)、有効なROUND_VALUE に対するIEEE_SUPPORT_ROUNDING(ROUND_VALUE,*X*) およびIEEE_SUPPORT_SQRT(*X*) の結果の値がすべて真の場合、真を返却します。それ以外の場合は偽を返却します。
- IEEE_SUPPORT_STANDARD() は、すべての実数型*X*に対してIEEE_SUPPORT_STANDARD(*X*) の値が真である場合、真を返却します。それ以外の場合は偽を返却します。

関数の結果の型は、基本論理型スカラです。

使用例

```
use, intrinsic :: ieee_arithmetic
logical ::result
result = ieee_support_standard()
```

2.272 IEEE_SUPPORT_UNDERFLOW_CONTROL組込みモジュール関数

IEEE 下位けたあふれモードのサポート判定

分類

問合せ関数

形式

result = IEEE_SUPPORT_UNDERFLOW_CONTROL()　　または
result = IEEE_SUPPORT_UNDERFLOW_CONTROL(*X*)

X

実数型でなければなりません。

result

基本論理型スカラです。

機能説明

IEEE_SUPPORT_UNDERFLOW_CONTROL は、以下の結果を返却します。

- IEEE_SUPPORT_UNDERFLOW_CONTROL(*X*) は、*X*と同じ型についての浮動小数点演算に対して、下位けたあふれモードを制御する機能を用意している場合、真を返却します。それ以外の場合は偽を返却します。
- IEEE_SUPPORT_UNDERFLOW_CONTROL() は、すべての実数型*X*に対して、IEEE_SUPPORT_UNDERFLOW_CONTROL(*X*) の値が真である場合、真を返却します。それ以外の場合は偽を返却します。

関数の結果の型は、基本論理型スカラです。

使用例

```
use, intrinsic :: ieee_arithmetic
logical ::result
result = ieee_support_underflow_control()
```

2.273 IEEE_UNORDERED組込みモジュール関数

IEEE 非数の判定

分類

要素別処理関数

形式

result = IEEE_UNORDERED(*X*,*Y*)

X

実数型でなければなりません。

Y

実数型でなければなりません。

result

基本論理型スカラです。

機能説明

IEEE_UNORDERED は、*X*または*Y*のどちらか、または両方とも非数の場合、真を返却します。それ以外の場合は偽を返却します。

関数の結果の型は、基本論理型スカラです。

IEEE_SUPPORT_DATATYPE(*X*) またはIEEE_SUPPORT_DATATYPE(*Y*) が真でないとき、この手続を呼び出してはなりません。

使用例

```
use, intrinsic :: ieee_arithmetic
real :: x,y
logical :: result
x = 1.0
```

```
y = 2.0
if (ieee_support_datatype(x) .and. ieee_support_datatype(y)) then
  result = ieee_unordered(x,y)
end if
```

2.274 IEEE_VALUE組込みモジュール関数

IEEE 浮動小数点数の生成

分類

要素別処理関数

形式

result = IEEE_VALUE(*X*,*CLASS*)

X

実数型でなければなりません。

CLASS

派生型IEEE_CLASS_TYPE でなければなりません。値として以下を指定することができます。

- IEEE_SUPPORT_NAN(*X*) が真の場合:IEEE_SIGNALING_NAN または IEEE_QUIET_NAN
- IEEE_SUPPORT_INF(*X*) が真の場合:IEEE_NEGATIVE_INF または IEEE_POSITIVE_INF
- IEEE_SUPPORT_DENORMAL(*X*) が真の場合:IEEE_NEGATIVE_DENORMAL または IEEE_POSITIVE_DENORMAL
- 無条件で指定することができる値:IEEE_NEGATIVE_NORMAL、IEEE_NEGATIVE_ZERO、IEEE_POSITIVE_ZERO、またはIEEE_POSITIVE_NORMAL

result

*X*と同じです。

機能説明

IEEE_VALUE は、*CLASS* で指定したIEEE 実数を返却します。

関数の結果の型は、*X*と同じ型です。

IEEE_SUPPORT_DATATYPE(*X*) が真でないとき、この手続を呼び出してはなりません。

使用例

```
use, intrinsic :: ieee_arithmetic
real ::result,x
type(ieee_class_type) :: class
...
class = ieee_positive_normal
if (ieee_support_datatype(x)) then
  result = ieee_value(x,class)
end if
```

2.275 IEOR組込み関数

IEOR 関数は、引数同士の排他的論理和を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
IEOR	-----	2	整数型、または 非10進定数表現 , 整数型、または 非10進定数表現	整数型
IEOR、 XOR	-----	2	1バイトの整数型 , 1バイトの整数型	1バイトの整数型
	IIEOR		2バイトの整数型 , 2バイトの整数型	2バイトの整数型
	IEOR		4バイトの整数型 , 4バイトの整数型	4バイトの整数型
	XOR		4バイトの整数型 , 4バイトの整数型	4バイトの整数型
	JIEOR		4バイトの整数型 , 4バイトの整数型	4バイトの整数型
	-----		8バイトの整数型 , 8バイトの整数型	8バイトの整数型

result = IEOR (*I* , *J*)
result = XOR (*I* , *J*)

I

整数型でなければなりません。IEOR は非10進定数表現を指定できます。

J

*I*と同じ種別型パラメタをもつ整数型でなければなりません。IEOR は非10進定数表現を指定できます。

result

*I*または*J*と同じ型です。

機能説明

IEOR、XOR、IIEOR、およびJIEOR は、引数同士の排他的論理和を返す関数です。

総称名IEOR およびXOR は、すべての整数型の引数に使用することができます。

総称名IEOR の引数*I*または*J*のどちらかに非10進定数表現を指定した場合、非10進定数表現は、指定した整数型に変換されます。両方に非10進定数表現を指定してはなりません。

それぞれの関数の結果の型は、*I*または*J*と同じです。

使用例

```
i = 53
j = 45
k = ieor(i,j)              ! k には24が代入されます
```

2.276 IERRNOサービス関数

機能説明

現在のシステムエラー番号を返却します。

形式

iy = IERRNO ( )

関数結果

基本整数型スカラ。

使用例

```
      use service_routines,only:ierrno
      integer :: i
      do i=7,99
        write(unit=i,err=10) i
      end do
   10 print *,ierrno()
      end
```

2.277 IF構文

IF 構文は、それを構成するブロックのうち、多くても1つの実行を選択します。

IF 構文は、以下の形式です。

```
[ if-construct-name : ] IF ( scalar-logical-expr ) THEN
    block
[ ELSE IF ( scalar-logical-expr ) THEN [ if-construct-name ]
    block ]...
[ ELSE [ if-construct-name ]
    block ]
END IF [ if-construct-name ]
```

if-construct-name は、IF 構文名です。

IF THEN 文にIF 構文名を指定する場合、対応するEND IF 文にも同じIF 構文名を指定しなければなりません。IF THEN 文にIF 構文名を指定しない場合、対応するEND IF 文にIF 構文名を指定してはなりません。ELSE IF 文またはELSE 文にIF 構文名を指定する場合、対応するIF THEN 文にも同じIF 構文名を指定しなければなりません。

scalar-logical-expr は、スカラ論理式です。

block は、ブロックです。ブロックは、0個以上の実行文または実行構文の並びです。ブロックに実行文または実行構文が1つも含まれない場合、そのようなブロックの実行は、効果をもちません。

IF 構文中の、多くても1つのブロックが実行されます。IF 構文中にELSE 文があれば、そのIF 構文中ではちょうど1つのブロックが実行されます。スカラ論理式は、値が真となる式が見つかるか、またはELSE 文またはEND IF 文に到達するまで、IF 構文中に書いてある順に評価されます。真となる式またはELSE 文に到達した場合、その直後に続くブロックが実行され、それによってそのIF 構文の実行は完了し、それ以降のELSE IF 文に含まれるスカラ論理式は、評価されません。どの式も真とならず、かつELSE 文がない場合、IF 構文の実行は、そのIF 構文中のどのブロックも実行せずに完了します。

IF 構文の例:

```
if (a > b) then
  c = d
else if (a < b) then
  d = c
else ! a == b
  stop
end if
```

2.278 IF文

IF 文は、1つの実行文の実行を制御します。

IF 文は、以下の形式です。

```
IF ( scalar-logical-expr ) action-stmt
```

scalar-logical-expr は、スカラ論理式です。

action-stmt は実行文であり、他のIF 文またはEND 文であってはなりません。

IF 文を実行すると、そのスカラ論理式が評価されます。式の値が真であれば、*action-stmt* が実行されます。式の値が偽であれば、*action-stmt* は実行されません。

IF 文の例:

```
if ( a >= b ) a = -a
```

2.279 IF THEN文

IF THEN 文は、IF 構文の開始を示します。

IF THEN 文は、以下の形式です。

```
[ if-construct-name : ] IF ( scalar-logical-expr ) THEN
```

if-construct-name は、IF 構文名です。

scalar-logical-expr は、スカラ論理式です。

IF 構文の詳細については、“2.277 IF構文”を参照してください。

2.280 算術IF文（廃止予定事項）

算術IF 文は、数値式の評価結果により、指定されたいずれかの文番号をもつ飛び先文に制御移行します。

算術IF 文は、以下の形式です。

```
IF ( scalar-numeric-expr ) label , label , label
```

scalar-numeric-expr は、スカラ数値式です。スカラ数値式は複素数型であってはなりません。

label は文番号であり、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。

1つの算術IF 文中に、同じ文番号を2回以上書いてもかまいません。

算術IF 文を実行すると、数値式が評価され、続いて制御移行が起こります。数値式の値が負である場合、ゼロである場合、および正である場合に、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。

算術IF 文の例:

```
if (b) 10,20,30          ! goto 10 if b<0
                         ! goto 20 if b==0
                         ! goto 30 if b>0
```

2.281 IMAGE_INDEX組込み関数

IMAGE_INDEX 関数は共添字を像番号に変換します。

分類

問合せ関数

形式

result = IMAGE_INDEX(*COARRAY*, *SUB*)

COARRAY

共配列でなければなりません。

SUB

1次元の整数型配列でなければなりません。大きさは*COARRAY* の共次元数と同じでなければなりません。

result

基本整数型スカラです。*SUB*の値が*COARRAY*に対する共添字として有効である場合、*result*は対応する像番号となります。それ以外の場合は0です。

使用例

```
integer, save :: k[2,*]
if (this_image() == 1) then
  print *,image_index(k,[1,1]) ! 1が出力されます
  print *,image_index(k,[2,1]) ! 像の数が2以上なら2、そうでなければ0が出力されます
  print *,image_index(k,[1,2]) ! 像の数が3以上なら3、そうでなければ0が出力されます
  print *,image_index(k,[2,2]) ! 像の数が4以上なら4、そうでなければ0が出力されます
end if
```

2.282 IMPLICIT文

IMPLICIT 文は、有効域内において、この文で指定された英字または'$'を第1文字としてもつ名前のデータ要素が、暗黙的に型宣言されるときの型および型パラメタを指定します。その有効域内で暗黙の型規則を適用しないことも指定できます。

IMPLICIT 文は、以下の形式です。

```
IMPLICIT implicit-spec-list        または
IMPLICIT NONE
```

implicit-spec は、以下の形式です。

type-spec (*letter-spec-list*)

type-spec は宣言型指定子であり、以下の形式です。

```
組込み型指定子                 または
TYPE ( 組込み型指定子 )        または
TYPE ( type-name )             または
CLASS( type-name )             または
CLASS( * )                     または
TYPE ( * )
```

組込み型指定子は、以下の形式です。

```
INTEGER [ kind-selector ]          または
REAL [ kind-selector ]             または
DOUBLE PRECISION                   または
COMPLEX [ kind-selector ]          または
CHARACTER [ char-selector ]        または
LOGICAL [ kind-selector ]          または
UNDEFINED                          または
BYTE
```

kind-selector は以下の形式です。

```
( [ KIND = ] kind )          または
* mem-length
```

char-selector は、以下の形式です。

```
( LEN = char-length-parm , KIND = kind )              または
( char-length-parm , [ KIND = ] kind )                または
( KIND = kind [ , LEN = char-length-parm ] )          または
( [LEN = ] char-length-parm )                         または
* char-length （廃止予定事項）
```

char-length は、以下の形式です。

```
( char-length-parm )                    または
scalar-int-literal-constant
```

char-length-parm は、以下の形式です。

```
scalar-int-expr             または
*                           または
:
```

scalar-int-literal-constant はスカラ整定数表現です。

kind は種別型パラメタであり、整数型のスカラ定数式でなければなりません。

mem-length は、領域の長さを示し、整数型のスカラ定数式でなければなりません。型指定子がINTEGER、REAL、およびLOGICAL の場合、*mem-length* の値は*kind* の値と同じ意味をもちます。型指定子がCOMPLEX の場合、*mem-length* の値は、*kind* の値を2倍にしたものと同じ意味です。

kind および*mem-length* の指定可能な値については、“2.3 型宣言文”を参照してください。

scalar-int-expr は、宣言式です。

‘:’ の文字長パラメタ値は、無指定型パラメタであることを示します。

type-name は、派生型の型名であり、その型名はその有効域内で定義されるか、参照結合または親子結合により参照可能でなければなりません。

letter-spec は英字範囲指定で、以下の形式です。

```
letter [ - letter ]
```

letter は、英字‘A’から‘Z’、および通貨記号‘$’でなければなりません。

英字範囲として‘-’で区切られた2つの英字は、1番目の文字から2番目の文字までのアルファベット順の英字をすべて指定したのと同じ意味です。2番目の文字はアルファベット順で1番目の文字より後でなければなりません。通貨記号‘$’は、アルファベット順で英字‘Z’の次とします。1つの有効域内のすべてのIMPLICIT 文を通して、同じ英字を単独の英字または英字範囲として、2回以上指定してはなりません。

IMPLICIT NONE は、すべての英字に対して、型の対応付けを空とします。IMPLICIT NONE を指定する場合、どのPARAMETER 文よりも前に指定しなければなりません。また、その有効域内に、他のIMPLICIT 文を指定することはできません。

型指定子がUNDEFINED の場合、指定された英字を第1文字としてもつ名前のデータ要素が、その有効域内で暗黙の型規則を適用しないことを指定します。

プログラム単位または引用仕様本体において、対応付けが指定されていない英字は、‘I’から‘N’であれば基本整数型に、その他の英字および‘$’であれば基本実数型に対応付けられます。内部手続またはモジュール手続において、対応付けが指定されていない英字は、暗黙的に親有効域の対応付けに従います。

型宣言文で明示的に宣言されていなくて、組込み関数でもなく、さらに参照結合または親子結合によって参照可能になったのでもないデータ要素は、その名前の第1文字の対応付けが空でないとき、その対応付けに従った型および型パラメタをもつとして暗黙的に宣言されます。

FUNCTION 文中の明示的な型指定は、その関数副プログラムの結果変数名について、IMPLICIT文を上書きします。

CLASS 指定子の詳細については、“2.3 型宣言文”を参照してください。

IMPLICIT 文の例:

```
implicit character(c), integer(a-b,d-z)
```

この例では、第1文字が英字‘C’である名前は暗黙的に文字型となり、それ以外の英字を第1文字にもつ名前は暗黙的に整数型となります。

2.283 IMPORT文

IMPORT 文は、引用仕様本体において、親有効域の名前付き要素を親子結合によって参照可能にすることを宣言します。

IMPORT 文は、引用仕様本体にだけ指定することができます。

IMPORT 文は、以下の形式です。

```
IMPORT [ [ :: ] import-name-list ]
```

import-name-list は、輸入名の並びです。輸入名は、親有効域内の要素の名前でなければなりません。

親有効域において定義され、IMPORT 文によって参照結合される要素は、その引用本体よりも先に明示的に宣言しなければなりません。指定した名前は、その有効域内で局所要素と宣言してはなりません。

輸入名の並びを指定しないIMPORT 文を指定した場合、他のIMPORT 文において指定しなかった親有効域の要素も親子結合によって参照可能になります。

IMPORT 文の例:

```
module mod
  integer,parameter::i=1
end module

use mod
interface
subroutine sub(k)
  import::i        ! IMPORT 文によってi が参照可能になります。
  integer(i)::k
end subroutine
end interface
```

2.284 INCLUDE行

INCLUDE 行は、プログラム中に他のファイル中のソースプログラムを展開します。

INCLUDE 行は、以下の形式です。

```
INCLUDE char-const
```

char-const は、アポストロフィ‘’’または引用符‘"’でくくった形式の文字定数表現でなければなりません。*char-const* に指定された文字列は、ファイル名です。

INCLUDE 行は、文を指定できる個所に1行で書かなければなりません。INCLUDE 行には、注釈を指定することはできますが、文番号を指定することはできません。INCLUDE 行はFortran の文ではありません。

INCLUDE 行の例:

```
include "typedef.inc"
```

この例では、"typedef.inc" のファイルの内容が、このINCLUDE 行の位置に展開されます。

2.285 INDEX組込み関数

INDEX 関数は、文字列中の文字部分列の開始位置を返却します。

分類

要素別処理関数

形式

result = INDEX (*STRING* , *SUBSTRING* [, *BACK* , *KIND*])

STRING

文字型でなければなりません。

SUBSTRING

*STRING*と同じ種別型パラメタの文字型でなければなりません。

BACK（省略可能）

論理型でなければなりません。

KIND（省略可能）

スカラ整数定数式でなければなりません。

result

整数型です。*KIND*が指定された場合、種別型パラメタは*KIND*の指定に従います。*KIND*が省略された場合、種別型パラメタは基本整数型になります。

機能説明

INDEX は、文字列中の文字部分列の開始位置を返却します。

－ *BACK*に偽を指定している、または*BACK*を省略している場合

*STRING*が*SUBSTRING*中の文字列を含む場合、*STRING*中にある最も左の*SUBSTRING*の開始位置を返却します。
*STRING*が*SUBSTRING*を含まない場合、および*STRING*の長さが*SUBSTRING*の長さよりも短い場合は、0を返却します。
LEN(*SUBSTRING*) が0である場合、1を返却します。

－ *BACK*に真を指定している場合

*STRING*が*SUBSTRING*中の文字列を含む場合、*STRING*中にある最も右の*SUBSTRING*の開始位置を返却します。
*STRING*が*SUBSTRING*を含まない場合、および*STRING*の長さが*SUBSTRING*の長さよりも短い場合は、0を返却します。
LEN(*SUBSTRING*) が0である場合、LEN(*STRING*) + 1 を返却します。

関数の結果の型は、種別型パラメタ*KIND*が指定された場合、種別型パラメタ*KIND*の整数型となります。*KIND*が省略された場合、関数の結果の型は基本整数型です。

使用例

```
i = index('mississippi' , 'si' )                ! i には4が代入されます
i = index('mississippi' , 'si' , back=.true.)   ! i には7が代入されます
```

2.286 INMAXサービス関数

機能説明

整数型の最大値を返却します。

形式

iy = INMAX ( )

関数結果

基本整数型スカラ。

使用例

```
use service_routines,only:inmax
print *,inmax()                  ! 2147483647が出力されます。
end
```

2.287 INQUIRE文

INQUIRE 文は、特定のファイル、または装置の接続の性質について、問い合わせます。

INQUIRE 文は、以下の形式です。

```
INQUIRE ( inquire-spec-list )                          または
INQUIRE ( IOLENGTH = iolength ) output-item-list
```

inquire-spec-list は、コンマで区切られた問合せ指定子の並びです。

inquire-spec は、以下の形式です。

```
[ UNIT = ] external-file-unit          または
FILE = file-name-expr                  または
IOSTAT = io-stat                       または
ERR = err-label                        または
EXIST = exist                          または
OPENED = opened                        または
NUMBER = number                        または
NAMED = named                          または
NAME = name                            または
ACCESS = access                        または
SEQUENTIAL = sequential                または
DIRECT = direct                        または
FORM = form                            または
FORMATTED = formatted                  または
UNFORMATTED = unformatted              または
BINARY = binary                        または
RECL = recl                            または
NEXTREC = rextrec                      または
BLANK = blank                          または
POSITION = position                    または
ACTION = action                        または
READ = read                            または
WRITE = write                          または
READWRITE = readwrite                  または
DELIM = delim                          または
PAD = pad                              または
BLOCKSIZE = blocksize                  または
CONVERT = convert                      または
FLEN = flen                            または
ASYNCHRONOUS = asynchronous            または
DECIMAL = decimal                      または
ENCODING = encoding                    または
ID = id                                または
IOMSG = iomsg                          または
PENDING = pending                      または
POS = pos                              または
ROUND = round                          または
SIGN = sign                            または
SIZE = size                            または
STREAM = stream
```

UNIT 指定子を含むINQUIRE 文は、装置INQUIRE 文です。

FILE 指定子を含むINQUIRE 文は、ファイルINQUIRE 文です。

IOLENGTH 指定子を含むINQUIRE 文は、出力並びINQUIRE 文です。

問合せ指定子は、UNIT 指定子またはFILE 指定子のいずれか1つを含まなければなりません。
UNIT 指定子およびFILE 指定子の両方を含むことはできません。

各指定子は、問合せ指定子並びに2回以上指定してはなりません。

文字列‘UNIT=’をUNIT 指定子から省略する場合、UNIT 指定子は、問合せ指定子並びの最初の項目でなければなりません。

external-file-unit は、外部ファイル装置であり、スカラ整数式でなければなりません。

file-name-expr は、スカラ基本文字式です。

FILE 指定子の値は、問い合わせるファイルの名前です。後続の空白は無視されます。FILE 指定子に指定されたファイルは、存在していなくてもかまいませんし、装置と接続されていなくてもかまいません。

io-stat は、スカラ整変数です。

IOSTAT 指定子が指定された場合、IOSTAT 指定子に指定された変数には、以下の値が設定されます。

- 誤り条件、ファイル終了条件、および記録終了条件が検出されない場合は、0
- 誤り条件が検出された場合は、1または実行時の診断メッセージの番号

err-label は文番号であり、このINQUIRE 文と同じ有効域内の飛び先文の文番号でなければなりません。

ERR 指定子が指定され、INQUIRE 文の実行中に誤り条件が検出された場合、ERR 指定子に指定された文番号をもつ文が次に実行されます。

*exist*は、スカラ基本論理変数でなければなりません。ファイルINQUIRE 文を実行すると、指定された名前のファイルが存在する場合、*exist* に真の値が代入され、存在しない場合、偽の値が代入されます。装置INQUIRE 文を実行すると、指定した装置が存在する場合、真の値が代入され、存在しない場合、偽の値が代入されます。

*opened*は、スカラ基本論理変数でなければなりません。ファイルINQUIRE 文を実行すると、指定したファイルが装置に接続されている場合、*opened*に真の値が代入され、装置に接続されていない場合、偽の値が代入されます。装置INQUIRE 文を実行すると、指定した装置がファイルに接続されている場合、真の値が代入され、ファイルに接続されていない場合、偽の値が代入されます。

*number*は、スカラ整変数でなければなりません。*number*には、そのファイルが現在接続されている装置の外部装置識別子の値が代入されます。そのファイルに接続されている装置が存在しない場合、-1が代入されます。

*named*は、スカラ基本論理変数でなければなりません。*named*には、そのファイルが名前をもつ場合、真の値が代入されます。そうでない場合、偽の値が代入されます。

*name*は、スカラ基本文字変数でなければなりません。*name*は、そのファイルが名前をもつ場合、ファイルの名前の値が代入されます。そうでない場合、不定となります。

*access*は、スカラ基本文字変数でなければなりません。*access*には、そのファイルが順番探査として接続されている場合、'SEQUENTIAL' が代入されます。直接探査として接続されている場合、'DIRECT' が代入されます。流れ探査として接続されている場合、'STREAM' が代入されます。接続されていない場合、'UNDEFINED' が代入されます。

*sequential*は、スカラ基本文字変数でなければなりません。*sequential*には、そのファイルに対して順番探査入出力が許される場合、'YES' が代入されます。許されない場合、'NO' が代入されます。ファイルが装置と接続されていない場合'UNKNOWN' が代入されます。

*direct*は、スカラ基本文字変数でなければなりません。*direct*には、そのファイルに対して直接探査入出力が許される場合、'YES' が代入されます。許されない場合、'NO' が代入されます。ファイルが装置と接続されていない場合'UNKNOWN' が代入されます。

form は、スカラ基本文字変数でなければなりません。*form* には、そのファイルが書式付き入出力として接続されている場合、'FORMATTED' が代入されます。書式なし入出力として接続されている場合、'UNFORMATTED' が代入されます。BINARY 入出力として接続されている場合、'BINARY' が代入されます。接続されていない場合、'UNDEFINED' が代入されます。

*formatted*は、スカラ基本文字変数でなければなりません。*formatted*には、そのファイルに対して書式付き入出力が許される場合、'YES' が代入されます。許されない場合、'NO' が代入されます。ファイルが装置と接続されていない場合'UNKNOWN' が代入されます。

unformatted は、スカラ基本文字変数でなければなりません。*unformatted* には、そのファイルに対して書式なし入出力が許される場合、'YES' が代入されます。許されない場合、'NO' が代入されます。ファイルが装置と接続されていない場合'UNKNOWN' が代入されます。

*binary*は、スカラ基本文字変数でなければなりません。*binary*には、そのファイルに対してBINARY 入出力が許される場合、'YES' が代入されます。許されない場合、'NO' が代入されます。ファイルが装置と接続されていない場合'UNKNOWN' が代入されます。

*recl*は、スカラ整変数でなければなりません。*recl*には、直接探査として接続されているファイルの記録長の値、または順番探査として接続されているファイルの最大の記録長の値が代入されます。長さの単位は、バイトです。ファイルが接続されていない場合または流れ探査として接続されている場合、*recl* は不定となります。

*nextrec*は、スカラ整変数でなければなりません。*nextrec*には、直接探査として接続されているファイルにおいて、直前に読まれたか、または書かれた記録番号をnとして、n+1 の値が代入されます。ファイルが接続されていて、接続されて以後読み書きされた記録がない場合、1が代入されます。ファイルが直接探査として指定されていない場合、およびファイル位置が不定の場合、*nextrec* は不定となります。

*blank*は、スカラ基本文字変数でなければなりません。*blank*には、書式付き入出力と接続されているファイルに対して空白を無視する指定がなされている場合、'NULL' が代入されます。空白を0とする指定がなされている場合、'ZERO' が代入されます。接続されていない場合、および書式付き入出力以外で接続されている場合、'UNDEFINED' が代入されます。

position は、スカラ基本文字変数でなければなりません。*position* には、OPEN 文によってその位置を変更しないで接続されている場合、'ASIS' が代入されます。始点に位置付けられているファイルの場合、'REWIND' が代入されます。ファイル終了記録の直前または終点に位置付けられているファイルの場合、'APPEND' が代入されます。接続されていない場合および直接探査として接続されている場合、'UNDEFINED' が代入されます。

action は、スカラ基本文字変数でなければなりません。*action* には、そのファイルが入力専用として接続されている場合、'READ' が代入されます。出力専用として接続されている場合、'WRITE'が代入されます。入力と出力の両方が可能であるとして接続されている場合、'READWRITE' が代入されます。接続されていない場合、'UNDEFINED' が代入されます。

read は、スカラ基本文字変数でなければなりません。*read* には、そのファイルに対して入力が許される場合、'YES' が代入されます。許されない場合、'NO' が代入されます。ファイルが装置と接続されていない場合、'UNKNOWN' が代入されます。

write は、スカラ基本文字変数でなければなりません。*write* には、そのファイルに対して出力が許される場合、'YES' が代入されます。許されない場合'NO' が代入されます。ファイルが装置と接続されていない場合'UNKNOWN' が代入されます。

readwrite は、スカラ基本文字変数でなければなりません。*readwrite* には、そのファイルに対して入力および出力の両方が許される場合、'YES' が代入されます。許されない場合'NO' が代入されます。ファイルが装置と接続されていない場合'UNKNOWN' が代入されます。

delim は、スカラ基本文字変数でなければなりません。*delim* には、並び書式または変数群書式によって書かれるデータを囲むのにアポストロフィ‘’’が使用される場合、'APOSTROPHE' が代入されます。それらのデータを囲むのに引用符‘"’が使用される場合、'QUOTE' が代入されます。文字データを囲むのにそのどちらも使用されない場合、'NONE' が代入されます。接続されていない場合、および書式付き入出力以外で接続されている場合、'UNDEFINED' が代入されます。

*pad*は、スカラ基本文字変数でなければなりません。*pad*には、装置番号で指定されるファイルがPAD 指定子として'NO' をもつ場合、'NO' が代入されます。そうでない場合、'YES' が代入されます。

blocksize は、スカラ整変数でなければなりません。*blocksize* には、順番探査入出力、または流れ探査入出力として接続されている場合、バッファサイズの値が代入されます。単位はバイトです。接続されていない場合、または直接入出力として接続されている場合、0が代入されます。

*convert*は、スカラ基本文字変数でなければなりません。*convert*には、書式なし入出力として接続されているファイルが、CONVERT 指定子の値として'LITTLE_ENDIAN' をもつ場合、'LITTLE_ENDIAN' が代入されます。'BIG_ENDIAN' をもつ場合、'BIG_ENDIAN' が代入されます。'NATIVE' をもつ場合、'NATIVE' が代入されます。書式付き入出力として接続されている場合、'NATIVE' が代入されます。接続されていない場合、'UNKNOWN' が代入されます。

flen は、スカラ整変数でなければなりません。*flen* には、指定されたファイルの大きさの値が代入されます。単位は、バイトです。ファイルが装置と接続されていない場合、0が代入されます。

iolength は、スカラ整変数でなければなりません。*iolength* には、指定した入出力並び項目をもつ書式なし出力文を実行した結果生成される記録の記録長が代入されます。単位は、バイトです。

output-item-list は、コンマで区切られた出力項目の並びです。

output-item は、以下の形式です。

```
expr                    または
io-implied-do
```

expr は、式です。

io-implied-do は入出力DO 形反復であり、以下の形式です。

```
( output-item-list , io-implied-do-control )
```

io-implied-do-control は入出力DO 制御であり、以下の形式です。

```
do-variable = scalar-expr , scalar-expr [ , scalar-expr ]
```

*do-variable*はDO 変数であり、整数型、基本実数型、または倍精度実数型の名前付きスカラ変数でなければなりません。基本実数型および倍精度実数型のDO 変数は廃止事項です。

*scalar-expr*は、整数型、基本実数型、または倍精度実数型のスカラ式でなければなりません。基本実数型および倍精度実数型の*scalar-expr* は廃止事項です。

asynchronous は、スカラ基本文字変数でなければなりません。

asynchronous には、ファイルが接続されていてその装置で非同期入出力が許されている場合、'YES' が代入されます。ファイルが接続されていてその装置で非同期入出力が許されていない場合、'NO' が代入されます。ファイルが接続されていない場合、スカラ基本文字変数には、'UNDEFINED' が代入されます。

decimal は、スカラ基本文字変数でなければなりません。*decimal* には、書式付き入出力としての接続で有効な小数編集モードに対応して、'COMMA' または'POINT' が代入されます。接続されていない場合および書式付き入出力以外で接続されている場合、スカラ基本文字変数には、'UNDEFINED' が代入されます。

encoding は、スカラ基本文字変数でなければなりません。*encoding* には、ファイルが書式付き入出力で接続されている場合、'UNKNOWN' が代入されます。ファイルが書式なし入出力で接続されている場合、'UNDEFINED' が代入されます。接続されていない場合、'UNKNOWN' が代入されます。

id は、スカラ整数式でなければなりません。*id* の式の値は、指定した装置でデータ転送操作を識別するものです。この指定子は、*pending* と相互に作用します。

pending は、スカラ基本論理変数でなければなりません。*pending* は、事前に非同期データ転送が完了したかどうかを判断するために使用します。*id* があり、そこで指定したデータ転送操作が完了している場合、*pending* の変数には偽の値が代入され、INQUIRE 文は、指定したデータ転送に対して待機操作を実行します。

id がなく、その装置におけるデータ転送操作のすべてが完了している場合、*pending* の変数には偽の値が代入され、INQUIRE 文は、その装置におけるデータ転送操作のすべてに対して待機操作を実行します。

これら以外の場合、*pending* の変数には真の値が代入され、待機操作は実行されません。*pos* は、スカラ整変数でなければなりません。*pos* には、流れ探査として接続したファイルの現在の位置の直後であるファイル記憶単位の番号が代入されます。ファイルが終点に位置付けられている場合、変数には、ファイル中の最大番号のファイル記憶単位の番号より1だけ大きい値が代入されます。ファイルが流れ探査として接続されていないか、またはファイル位置がそれ以前の誤り条件のために不定である場合、変数は不定となります。

round は、スカラ基本文字変数でなければなりません。*round* には、書式付き入出力としての接続で有効となっている入出力丸めモードに対応して、'UP'、'DOWN'、'ZERO'、'NEAREST'、'COMPATIBLE' または'PROCESSOR_DEFINED' が代入されます。ファイルが接続されていない場合かまたは書式付き入出力として接続されていない場合、'UNDEFINED' が代入されます。現在有効な入出力丸めモードが'UP'、'DOWN'、'ZERO'、'NEAREST' および'COMPATIBLE' とは異なる振る舞いをする場合にだけ、'PROCESSOR_DEFINED' が代入されます。

sign は、スカラ基本文字変数でなければなりません。*sign* には、書式付き入出力としての接続で有効となっている符号モードに対応して、'PLUS'、'SUPPRESS' または'PROCESSOR_DEFINED'が代入されます。ファイルが接続されていない場合または書式付き入出力として接続されていない場合、'UNDEFINED' が代入されます。

size は、スカラ整変数でなければなりません。*size* には、ファイルの大きさが代入されます。ファイルの大きさを決定できない場合、-1 が代入されます。

stream は、スカラ基本文字変数でなければなりません。*stream* には、そのファイルが流れ探査として接続されている場合、'YES' が代入されます。流れ探査として接続されていない場合、'NO'が代入されます。決定できない場合、'UNKNOWN' が代入されます。

iomsg はスカラ基本文字変数でなければなりません。入出力文の実行中に以下の条件が発生した場合、*iomsg* には説明のためのメッセージが代入されます。

- 誤り条件
- ファイル終了条件
- 記録終了条件

それ以外の場合、*iomsg* の値は変更されません。

INQUIRE 文の例:

```
inquire (unit=8, access=acc, err=200)
            ! 装置番号8に対して探査方法を問い合わせます
inquire (this_unit, opened=opnd, direct=dir)
            ! 装置番号this_unit に対して、
            ! ファイルに接続されているか？
            ! 直接探査入出力が許されるか？
            ! を問い合わせます
inquire (file="myfile.dat", recl=record_length)
            ! ファイル"myfile.dat" の記録長を問い合わせます
```

2.287.1 出力並びINQUIRE文

出力並びINQUIRE 文は、1つのIOLENGTH 指定子と出力項目並びだけを含みます。

IOLENGTH 指定子のスカラ整変数には、指定した出力項目並びのデータを書式なしファイルに格納するために必要なバイト単位の値が代入されます。その値は、同じ並びを入力項目並びまたは出力項目並びとしてもつ入出力文があるファイルを、書式なし直接探査として接続するOPEN文において、RECL 指定子に指定できる値です。

2.288 INT組込み関数

INT 関数は、数値型データを整数型に変換します。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
INT1	-----	1	整数型、実数型、または複素数型	1バイトの整数型
INT2	-----	1	整数型、実数型、または複素数型	2バイトの整数型
	HFIX		単精度実数型	2バイトの整数型
	IINT		単精度実数型	2バイトの整数型
	IIFIX		単精度実数型	2バイトの整数型
	IIDINT		倍精度実数型	2バイトの整数型
INT4	-----	1	整数型、実数型、または複素数型	4バイトの整数型
JFIX	-----	1	整数型、実数型、または複素数型	4バイトの整数型
INT	-----	1 または 2	整数型、実数型、複素数型、または非10進定数表現 [, 整数型]	整数型
	-----	1	1バイトの整数型	基本整数型
	-----		2バイトの整数型	基本整数型
	-----		4バイトの整数型	基本整数型
	-----		8バイトの整数型	基本整数型
	INT		単精度実数型	基本整数型
	IFIX		単精度実数型	基本整数型
	JINT		単精度実数型	基本整数型
	JIFIX		単精度実数型	基本整数型
	IDINT		倍精度実数型	基本整数型
	JIDINT		倍精度実数型	基本整数型
	IQINT		4倍精度実数型	基本整数型
	-----		単精度複素数型	基本整数型
	-----		倍精度複素数型	基本整数型
	-----		4倍精度複素数型	基本整数型

result = INT (*A* [, *KIND*])

A

数値型、または非10進定数表現でなければなりません。

KIND（省略可能）

スカラ整数定数式でなければなりません。

result

整数型です。*KIND*が指定された場合、種別型パラメタは*KIND*の指定に従います。*KIND*が省略された場合、種別型パラメタは基本整数型の種別型パラメタと同じになります。

機能説明

INT、INT1、INT2、INT4、HFIX、IINT、IIFIX、IIDINT、IFIX、JINT、JFIX、JIFIX、IDINT、JIDINT、およびIQINT は、数値型データを整数型に変換します。

関数の結果は以下のとおりです。

- *A* が整数型の場合

  A を返却します。

- *A* が実数型の場合

  ABS(*A*) < 1.0 の場合、結果の値は0となります。それ以外の場合は、*A* の絶対値以下で最大の整数値に*A* の符号を付けた値となります。

- *A* が複素数型の場合

  A の実部に、*A* が実数型の場合の規則を適用した値となります。

- *A* が非10進定数表現の場合

  非10進定数表現を整数型に変換します。

関数の結果が、指定された整数型で表現できない場合、結果の値は不定となります。

総称名INT1 は、すべての整数型、実数型および複素数型の引数に使用することができます。関数の結果の型は、1バイトの整数型です。

総称名INT2 は、すべての整数型、実数型および複素数型の引数に使用することができます。関数の結果の型は、2バイトの整数型です。

HFIX は、単精度実数型の引数に使用することができます。関数の結果の型は、2バイトの整数型です。

総称名INT4 は、すべての整数型、実数型および複素数型の引数に使用することができます。関数の結果の型は、4バイトの整数型です。

総称名INT は、すべての整数型、実数型、複素数型および非10進定数表現の引数に使用することができます。INT の関数の結果の型は、種別型パラメタ*KIND*が指定された場合、種別型パラメタ*KIND*の整数型となります。*KIND*が省略された場合、関数の結果の型は基本整数型です。

使用例

```
i = int(-3.6)                ! i には-3が代入されます
```

2.289 INTEGER型宣言文

INTEGER 型宣言文は、整数型のデータ実体を宣言します。

INTEGER 型宣言文は、以下の形式です。

```
INTEGER [ kind-selector ] [ [ , attr-spec ]... :: ] entity-decl-list
```

型宣言文の詳細については、“2.3 型宣言文”を参照してください。

2.290 INTENT文

INTENT 文は、仮引数の授受特性を宣言します。

INTENT 文は、以下の形式です。

```
INTENT ( intent-spec ) [ :: ] dummy-arg-name-list
```

intent-spec は授受特性指定であり、以下の形式です。

```
IN                  または
OUT                 または
INOUT
```

dummy-arg-name-list は、コンマで区切られた仮引数名の並びです。*dummy-arg-name* は、仮手続または仮ポインタの名前であってはなりません。

INTENT 文は、副プログラムまたは引用仕様本体の宣言部にだけ指定できます。

INTENT(IN) 属性の仮引数がポインタ実体でない場合、手続を呼び出した有効域からデータを受け取ることができることを指定します。そのような仮引数は、手続の実行中に再確定したり、不定にしたりしてはなりません。

INTENT(IN)属性の仮引数がポインタ実体の場合、次の箇所に現れてはなりません。

- NULLIFY文中のポインタ実体
- ポインタ代入文のデータポインタ実体または手続ポインタ実体
- ALLOCATE文またはDEALLOCATE文中の割付け実体
- 仮引数にINTENT(OUT)属性またはINTENT(INOUT)属性のポインタがある手続を利用するとき、その仮引数に結合する実引数

INTENT(OUT) 属性の仮引数がポインタ実体でない場合、手続を呼び出した有効域にデータを返すことができることを指定します。そのような仮引数と結合する実引数は、確定可能でなければなりません。手続の呼出し時には、暗黙的初期値指定された派生型実体の成分を除いて不定であり、引用する前に確定しなければなりません。

INTENT(OUT)属性の仮引数がポインタ実体の場合、手続の呼出し時に、その仮引数のポインタ結合状態が不定になることを指定します。

INTENT(OUT)属性は、LOCK_TYPE 型(“1.18.3 LOCK_TYPE 型”)の変数、LOCK_TYPE型の成分をもつ変数、割付け共配列、または部分実体が割付け共配列の実体に指定してはなりません。

INTENT(INOUT) 属性は、その仮引数が、手続を呼び出した有効域からデータを受け取ったり、データを返したりできることを指定します。そのような仮引数と結合する実引数は、確定可能でなければなりません。

INTENT属性をもつ実体の部分実体は、実体と同じINTENT属性をもちます。

INTENT 文の例:

```
subroutine ex (a,b,c)
  real :: a,b,c
  intent(in) :: a
  intent(out) :: b
  intent(inout) :: c
```

2.291 INTERFACE文

INTERFACE 文は、引用仕様宣言を開始します。引用仕様宣言は、その手続を呼び出すときの引用の形を定義します。また、手続の総称引用仕様、利用者定義演算、および利用者定義代入を指定します。引用仕様の詳細については、“1.12.7 手続引用仕様”を参照してください。

INTERFACE 文は、以下の形式です。

```
INTERFACE [ generic-spec ]          または
ABSTRACT INTERFACE
```

generic-spec は、総称指定であり、以下の形式です。

```
generic-name                        または
OPERATOR ( defined-operator )       または
```

```
ASSIGNMENT ( = )                    または
dtio-generic-spec
```

generic-name は総称名です。

defined-operator は利用者定義演算子であり、以下の形式です。

```
intrinsic-operator                  または
. operator-name .
```

intrinsic-operator は組込み演算子です。

operator-name は、利用者が定義した240文字までの利用者定義演算子の名前です。

ASSIGNMENT(=) は、利用者定義代入です。

dtio-generic-spec は派生型入出力総称指定であり、以下の形式です。

```
READ ( FORMATTED )                  または
READ ( UNFORMATTED )                または
WRITE ( FORMATTED )                 または
WRITE ( UNFORMATTED )
```

総称名は、その引用仕様宣言中のすべての手続を引用できる1つの名前を指定します。総称名は、引用仕様宣言中の手続の名前と同じであってもかまいませんし、参照可能な他の総称名と同じであってもかまいません。

総称指定の中にOPERATOR を指定する場合には、その引用仕様宣言内で指定する手続は、すべて利用者定義演算として引用可能な関数でなければなりません。2つの引数をもつ関数の場合には、2項演算の規則が適用されます。1つの引数をもつ関数の場合には、単項演算の規則が適用されます。引数をもたない関数または3つ以上の引数をもつ関数には、OPERATOR を指定することはできません。仮引数は、INTENT(IN) を指定した省略可能でない仮データ実体でなければなりません。関数結果は、文字長引継ぎであってはなりません。演算子が組込み演算子である場合には、関数の引数の個数は、その演算子の組込み演算としての仕様に適合しなければなりません。利用者定義演算は、指定された関数の引用とみなされます。利用者定義単項演算に対しては、演算対象は、関数の仮引数に対応します。利用者定義2項演算に対しては、左側の演算対象は、関数の1番目の仮引数に対応し、右側の演算対象は、2番目の仮引数に対応します。利用者定義演算子は、総称名と同様に2つ以上の関数に適用でき、その解釈は、総称手続の場合と同じです。組込み演算子に対する総称性は、それが表現する組込み演算を含みます。関係演算子の2つの形式は、どちらも同じ解釈をもつので、1つの形式(例えば <=)を拡張すれば、両方の形式(<= および.LE.)を定義することになります。

INTERFACE 文にASSIGNMENT を指定する場合には、その引用仕様宣言内の手続は、すべて利用者定義代入として引用可能なサブルーチンでなければなりません。これらのサブルーチンは、ちょうど2つの仮引数をもたなければなりません。それぞれの引数は省略可能であってはなりません。1番目の引数は、INTENT(OUT) 属性またはINTENT(INOUT) 属性をもたなければならず、2番目の引数は、INTENT(IN) 属性をもたなければなりません。利用者定義代入は、1番目の引数として左辺をもち、2番目の引数として右辺を括弧でくくったものをもつようなサブルーチンの引用とみなされます。ASSIGNMENT 総称指定は、代入操作を拡張します。ただし、等号の両方が同じ派生型である場合には、代入操作を再定義します。

1つの有効域内において、2つの手続が同じ派生型入出力総称指定をもつ場合、それらの仮引数*dtv_arg*(“1.7.4 利用者定義派生型入出力”参照)は、異なる型または異なる種別型パラメタをもたなければなりません。

ABSTRACT INTERFACE による引用仕様宣言は、抽象引用仕様宣言です。

ABSTRACT INTERFACE では、FUNCTION 文の関数名またはSUBROUTINE 文のサブルーチン名は、組込み型と同じであってはなりません。

INTERFACE 文の例:

```
interface                           ! 総称指定のないINTERFACE 文
  subroutine ex(a,b,c)
    implicit none
    real, dimension (2,8) :: a,b,c
    intent(in) :: a
    intent(out) :: b
  end subroutine ex
  function why (t,f)
    implicit none
    logical, intent(in) :: t,f
```

```
    logical :: why
  end function why
end interface

interface swap                          ! 総称名swap
  subroutine real_swap(x,y)
    implicit none
    real , intent(inout) :: x,y
  end subroutine real_swap
  subroutine int_swap(x,y)
    implicit none
    integer, intent(inout) :: x,y
  end subroutine int_swap
end interface swap

interface operator (*)                  ! * 演算子の拡張
  function set_intersection(a,b)
    use set_module                      ! 派生型set が定義されているモジュール
    implicit none
    type(set) , intent(in) :: a,b
    type(set) :: set_intersection
  end function set_intersection
end interface operator(*)

interface assignment(=)                 ! 代入の拡張
  subroutine integer_to_bit (n,b)
    implicit none
    logical, intent(out) :: n
    integer, intent(in) :: b(:)
  end subroutine integer_to_bit
end interface assignment(=)
```

2.292 INTRINSIC文

INTRINSIC 文は、その名前が組込み手続であることを確認し、指定された個別組込み関数名を実引数として使用できるようにします。

INTRINSIC 文は、以下の形式です。

```
INTRINSIC [ :: ] intrinsic-procedure-name-list
```

intrinsic-procedure-name-list は、コンマで区切られた組込み手続名の並びです。*intrinsic-procedure-name* は、組込み手続名でなければなりません。

総称組込み手続名をINTRINSIC 文に書いても、その名前は総称性を失うことはありません。

組込み関数の個別名を実引数として使用する場合には、有効域内のINTRINSIC 文にその名前を書くか、または有効域内の型宣言文でその名前にINTRINSIC 属性を与えなければなりません。

INTRINSIC 文の例:

```
intrinsic :: dlog , dabs            ! dlog およびdabs は組込み関数の個別名であり、
                                    ! 実引数に指定することができます
call zee (a,b,dlog)
```

2.293 IOR組込み関数

IOR 関数は、引数同士の論理和を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
IOR	-----	2	整数型、または 非10進定数表現 , 整数型、または 非10進定数表現	整数型
IOR、 OR	-----	2	1バイトの整数型 , 1バイトの整数型	1バイトの整数型
	IIOR		2バイトの整数型 , 2バイトの整数型	2バイトの整数型
	IOR		4バイトの整数型 , 4バイトの整数型	4バイトの整数型
	OR		4バイトの整数型 , 4バイトの整数型	4バイトの整数型
	JIOR		4バイトの整数型 , 4バイトの整数型	4バイトの整数型
	-----		8バイトの整数型 , 8バイトの整数型	8バイトの整数型

result = IOR (*I* , *J*)
result = OR (*I* , *J*)

I

整数型でなければなりません。IOR は非10進定数表現を指定できます。

J

*I*と同じ種別型パラメタをもつ整数型でなければなりません。IOR は非10進定数表現を指定できます。

result

*I*または*J*と同じ型です。

機能説明

IOR、OR、IIOR、およびJIOR は、引数同士の論理和を返す関数です。

総称名IOR およびOR は、すべての整数型の引数に使用することができます。

総称名IOR の引数*I*または*J*のどちらかに非10進定数表現を指定した場合、非10進定数表現は、指定した整数型に変換されます。両方に非10進定数表現を指定してはなりません。

それぞれの関数の結果の型は、*I*または*J*と同じです。

使用例

```
i=53
j=45
k=ior(i,j)              ! k には61が代入されます
```

2.294 IOSTAT_MSGサービスサブルーチン

機能説明

実行時に出力される診断メッセージを取得します。

形式

CALL IOSTAT_MSG (*errnum* , *message*)

errnum

基本整数型スカラ。実行時の診断メッセージ番号を指定します。

message

基本文字型スカラ。*errnum* に対応する実行時の診断メッセージが設定されます。

使用例

```
use service_routines,only:iostat_msg
character(len=150) :: msg
call iostat_msg(111,msg)
write (*,*) msg
end
```

2.295 IPARITY組込み関数

IPARITY関数は、配列内の排他的論理和を求めます。

分類

変形関数

形式

result = IPARITY (*ARRAY* , *DIM* [, *MASK*])　または
result = IPARITY (*ARRAY* [, *MASK*])

ARRAY

整数型の配列でなければなりません。

DIM

整数型スカラであって、1 <= *DIM* <= *n* の範囲の値でなければなりません。ここで、n は、*ARRAY* の次元数とします。対応する実引数は、省略可能な仮引数であってはなりません。

MASK（省略可能）

論理型でなければならず、*ARRAY* と形状適合しなければなりません。

result

ARRAY と同じ型および同じ種別型パラメタです。*DIM* が省略された、または *ARRAY* が1次元配列の場合は、スカラとなります。それ以外の場合、*ARRAY* の形状を (d_1, d_2, ..., d_n) としたとき、形状は、(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_n) となります。

機能説明

IPARITY関数は、*ARRAY* 内の排他的論理和を求めます。

－ *DIM*を省略している場合

MASK を省略すると、*MASK* の全要素が真と等価です。

MASK 中の真の要素に対応する *ARRAY* 内の排他的論理和を返却します。*ARRAY* が大きさゼロであるとき、結果は ゼロ になります。

－ *DIM*を指定している場合

ARRAY が1次元の場合は、IPARITY (*ARRAY* [, *MASK*]) と等しく、結果はスカラとなります。

他の場合、結果の要素(S_1, S_2, ..., S_{DIM-1}, S_{DIM+1}, ..., S_n) は、IPARITY (*ARRAY*(S_1, S_2, ..., S_{DIM-1}, : , S_{DIM+1}, ..., S_n) [,MASK = *MASK*])となります。ここで、*n* は *ARRAY* の次元数、*MASK*は *MASK*(S_1, S_2, ..., S_{DIM-1}, : , S_{DIM+1}, ..., S_n) とします。

使用例

```
integer:: result
result = IPARITY ( [ 3, 7, 11 ] ) ! result には 15 が代入されます
```

2.296 IRANDサービス関数

機能説明

0から2147483647までの乱数を発生させます。

形式

iy = IRAND (*i*)

i

基本整数型スカラ。

関数結果

基本整数型スカラ。*i* の値に従って、関数結果値が異なります。

=0　　：乱数列内の次の値が関数値として返却されます。
=1　　：乱数列内の最初の値が関数値として返却されます。
上記以外：乱数列発生のためのシード値として、引数の値が使われます。
　　この新しく発生させられた乱数列の最初の値が関数値として返却されます。

使用例

```
use service_routines,only:irand
do i=1,10
  print *,irand(0)                ! 10個の乱数値を獲得
end do
end
```

2.297 ISATTYサービス関数

機能説明

装置番号に接続されているファイルが端末かどうかを判定します。

形式

ly = ISATTY (*unit*)

unit

基本整数型スカラ。装置番号を指定します。

関数結果

基本論理型スカラ。装置番号と端末装置が接続している場合は真、接続していない場合は偽が返却されます。

使用例

```
use service_routines,only:isatty
print *,isatty(5)                 ! T が出力されます
print *,isatty(10)                ! F が出力されます
end
```

2.298 ISHA組込み関数

ISHA 関数は、算術シフトを行います。

分類

要素別処理関数

形式

result = ISHA (*I* , *SHIFT*)

I

整数型でなければなりません。

SHIFT

整数型でなければなりません。ABS(*SHIFT*) <= BIT_SIZE(*I*) でなければなりません。

result

I と同じ型です。

機能説明

ISHA は、算術シフトを行います。

SHIFT が正の場合、*SHIFT* ビットの左シフトを行います。

SHIFT が負の場合、ABS(*SHIFT*) ビットの右算術シフトを行います。

SHIFT が0の場合、何も行いません。

ABS(*SHIFT*) > BIT_SIZE(*I*) の場合、結果は保証されません。

関数の結果の型は、*I* と同じです。

使用例

```
i = isha(223606,5)          ! i には7155392が代入されます
i = isha(-14142,-5)         ! i には-442が代入されます
```

2.299 ISHC組込み関数

ISHC 関数は、循環シフトを行います。

分類

要素別処理関数

形式

result = ISHC (*I* , *SHIFT*)

I

整数型でなければなりません。

SHIFT

整数型でなければなりません。ABS(*SHIFT*) <= BIT_SIZE(*I*) でなければなりません。

result

I と同じ型です。

機能説明

ISHC は、循環シフトを行います。

SHIFT が正のときは左循環シフト、*SHIFT* が負のときは右循環シフトを行います。

関数の結果の型は、*I* と同じです。

使用例

```
i = ishc(223606,5)          ! i には7155392が代入されます
i = ishc(-14142,-5)         ! i には402652742が代入されます
```

2.300 ISHFT組込み関数

ISHFT 関数は、論理シフトを行います。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
ISHFT	-----	2	1バイトの整数型 , 整数型	1バイトの整数型
	IISHFT		2バイトの整数型 , 整数型	2バイトの整数型
	ISHFT		4バイトの整数型 , 整数型	4バイトの整数型
	JISHFT		4バイトの整数型 , 整数型	4バイトの整数型
	-----		8バイトの整数型 , 整数型	8バイトの整数型

result = ISHFT (*I* , *SHIFT*)

I

整数型でなければなりません。

SHIFT

整数型でなければなりません。ABS(*SHIFT*) <= BIT_SIZE(*I*) でなければなりません。

result

*I*と同じ型です。

機能説明

ISHFT、IISHFT、およびJISHFT は、論理シフトを行います。

*SHIFT*が正の場合、*SHIFT*ビットの左シフトを行います。

*SHIFT*が負の場合、-*SHIFT*ビットの右論理シフトを行います。

*SHIFT*が0の場合、何も行いません。

総称名ISHFT は、すべての整数型の引数に使用することができます。

それぞれの関数の結果の型は、*I*と同じです。

使用例

```
i = ishft(3,2)          ! i には12が代入されます
```

2.301 ISHFTC組込み関数

ISHFTC 関数は、右端のビットから指定したビット列内で循環シフトを行います。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
ISHFTC	-----	2 または 3	1バイトの整数型 , 整数型 [, 整数型]	1バイトの整数型
	IISHFTC		2バイトの整数型 , 整数型[, 整数型]	2バイトの整数型

総称名	個別名	引数の数	引数の型	結果の型
	ISHFTC		4バイトの整数型 , 整数型[, 整数型]	4バイトの整数型
	JISHFTC		4バイトの整数型 , 整数型[, 整数型]	4バイトの整数型
	-----		8バイトの整数型 , 整数型[, 整数型]	8バイトの整数型

result = ISHFTC (*I* , *SHIFT* [, *SIZE*])

I

整数型でなければなりません。

SHIFT

整数型でなければなりません。ABS(*SHIFT*) <= *SIZE* でなければなりません。

SIZE（省略可能）

整数型でなければなりません。0 < *SIZE* <= BIT_SIZE(*I*) でなければなりません。省略された場合は、BIT_SIZE(*I*) が設定されます。

result

I と同じ型です。

機能説明

ISHFTC、IISHFTC、およびJISHFTC は、右端のビットから指定したビット列内で循環シフトを行います。

I の右端からの *SIZE* ビットを、*SHIFT* 分循環シフトを行います。*SHIFT* が正のときは左循環シフト、*SHIFT* が負のときは右循環シフトを行います。いかなるビットも失われることはありません。シフト対象以外のビットは変わりません。

総称名ISHFTC は、すべての整数型の引数に使用することができます。

それぞれの関数の結果の型は、*I* と同じです。

使用例

```
i = ishftc(13,-2,3)          ! i には11が代入されます
```

2.302 ISHL組込み関数

ISHL 関数は、論理シフトを行います。

分類

要素別処理関数

形式

result = ISHL (*I* , *SHIFT*)

I

整数型でなければなりません。

SHIFT

整数型でなければなりません。ABS(*SHIFT*) < BIT_SIZE(*I*) でなければなりません。

result

I と同じ型です。

機能説明

ISHL は、論理シフトを行います。

SHIFT が正の場合、*SHIFT* ビットの左論理シフトを行います。

*SHIFT*が負の場合、-*SHIFT*ビットの右論理シフトを行います。

*SHIFT*が0の場合、何も行いません。

ABS(*SHIFT*) >= BIT_SIZE(*I*) の場合、結果は保証されません。

関数の結果の型は、*I*と同じです。

使用例

```
i = ishl(223606,5)          ! i には7155392が代入されます
i = ishl(-14142,-5)         ! i には134217286が代入されます
```

2.303 ISO_C_BINDING組込みモジュール

組込みモジュールISO_C_BINDING では、以下が提供されています。

- C プログラムと相互利用のための名前付き定数
- C プログラムと相互利用のための型
- C プログラムと相互利用のための組込み手続

組込みモジュールISO_C_BINDING で提供される名前付き定数および型については、“Fortran使用手引書”を参照してください。C プログラムと相互利用のための組込み手続については、“2.106 C_ASSOCIATED組込みモジュール関数”、“2.108 C_F_POINTER組込みモジュールサブルーチン”、“2.109 C_F_PROCPOINTER組込みモジュールサブルーチン”、“2.107 C_FUNLOC組込みモジュール関数”、“2.110 C_LOC組込みモジュール関数”、および“2.111 C_SIZEOF組込みモジュール関数”を参照してください。

2.304 ISO_Fortran_binding.h

Cプログラムのヘッダーファイルです。次元引継ぎとの言語間結合で使用します。

このヘッダーをインクルードしたC言語プログラムは、このヘッダーで定義されていないCFI_で始まる名前は、使用してはなりません。

ISO_Fortran_binding.hに含まれる構造体定義、型定義名宣言、およびマクロ定義については、“Fortran使用手引書”を参照してください。

2.305 ISO_FORTRAN_ENV組込みモジュール

このモジュールは、名前付き定数、派生型、および手続を参照可能にします。

次の名前付き定数を参照可能にします。

名前付き定数	意味	型	値
CHARACTER_STORAGE_SIZE	文字記憶単位のビット数	4バイトの整数型	8
ERROR_UNIT	実行時に診断メッセージを出力するファイル装置番号	4バイトの整数型	0
FILE_STORAGE_SIZE	ファイル記憶単位のビット数	4バイトの整数型	8
INPUT_UNIT	READ文で装置*指定に対応するファイル装置番号	4バイトの整数型	5
IOSTAT_END	ファイル終了条件を検出した場合のIOSTAT指定子の値	4バイトの整数型	-1
IOSTAT_EOR	記録終了条件を検出した場合のIOSTAT指定子の値	4バイトの整数型	-2
IOSTAT_INQUIRE _INTERNAL_UNIT	INQUIRE文の装置番号が内部ファイルと結合した場合のIOSTAT指定子の値	4バイトの整数型	1124
NUMERIC_STORAGE_SIZE	数値記憶単位のビット数	4バイトの整数型	32

名前付き定数	意味	型	値
OUTPUT_UNIT	WRITE文で装置*指定に対応するファイル装置番号	4バイトの整数型	6
INT8	8ビットを占有する整数型の種別型パラメタ	4バイトの整数型	1
INT16	16ビットを占有する整数型の種別型パラメタ	4バイトの整数型	2
INT32	32ビットを占有する整数型の種別型パラメタ	4バイトの整数型	4
INT64	64ビットを占有する整数型の種別型パラメタ	4バイトの整数型	8
REAL16	16ビットを占有する実数型の種別型パラメタ	4バイトの整数型	2
REAL32	32ビットを占有する実数型の種別型パラメタ	4バイトの整数型	4
REAL64	64ビットを占有する実数型の種別型パラメタ	4バイトの整数型	8
REAL128	128ビットを占有する実数型の種別型パラメタ	4バイトの整数型	16
CHARACTER_KINDS	文字型としてサポートしている種別型パラメタの値	4バイトの整数型	1
INTEGER_KINDS	整数型としてサポートしている種別型パラメタの値	4バイトの整数型	(/1,2,4,8/)
LOGICAL_KINDS	論理型としてサポートしている種別型パラメタの値	4バイトの整数型	(/1,2,4,8/)
REAL_KINDS	実数型としてサポートしている種別型パラメタの値	4バイトの整数型	(/4,8,16/)
ATOMIC_INT_KIND	アトミック操作を整数型としてサポートしている種別型パラメタの値	4バイトの整数型	4
ATOMIC_LOGICAL_KIND	アトミック操作を論値型としてサポートしている変数の種別型パラメタの値	4バイトの整数型	4
STAT_LOCKED	ロック変数が実行中の像によってロックされていることを示す値	4バイトの整数型	1732
STAT_LOCKED_OTHER_IMAGE	ロック変数が他の像によってロックされていることを示す値	4バイトの整数型	1733
STAT_STOPPED_IMAGE	実行を終えた像との同期が必要になったことを示す値	4バイトの整数型	1731
STAT_UNLOCKED	ロック変数がロックされていないことを示す値	4バイトの整数型	1734

COMPILER_OPTIONS組込みモジュール関数(“2.83 COMPILER_OPTIONS組込みモジュール関数”)およびCOMPILER_VERSION組込みモジュール関数(“2.84 COMPILER_VERSION組込みモジュール関数”)を引用可能にします。

ロック変数のための派生型LOCK_TYPE(“1.18.3 LOCK_TYPE 型”)を参照可能にします。

使用例

```
use, intrinsic :: iso_fortran_env, only:input_unit
integer :: input
read(input_unit,*) input
```

2.306 IS_CONTIGUOUS組込み関数

機能説明

IS_CONTIGUOUS 組込み関数は、引数の配列が連続した記憶域をもつか否かを返却します(“2.88 CONTIGUOUS文”参照)。

分類

問合せ関数

形式

result = IS_CONTIGUOUS (*ARRAY*)

ARRAY

配列でなければなりません。空状態のポインタであってはなりません。大きさまたは長さがゼロであってはなりません。

result

基本論理型です。

関数結果

*ARRAY*が連続した記憶域をもつ場合、真が返却されます。連続しない場合、偽が返却されます。

使用例

```
integer,target,dimension(5)::target
integer,pointer,dimension(:)::ptr
logical::flag
ptr => target
flag = is_contiguous(ptr)     ! flagには真が代入されます
ptr => target(1:5:2)
flag = is_contiguous(ptr)     ! flagには偽が代入されます
```

2.307 IS_IOSTAT_END組込み関数

機能説明

IS_IOSTAT_END 組込み関数は、ファイル終了条件を示す値か否かを返却します。

分類

要素別処理関数

形式

result = IS_IOSTAT_END (*I*)

I

整数型でなければなりません。

result

基本論理型です。

関数結果

*I*がファイル終了条件を示すIOSTAT 指定子の値と一致する場合、真が返却されます。一致しない場合、偽が返却されます。

2.308 IS_IOSTAT_EOR組込み関数

機能説明

IS_IOSTAT_EOR 組込み関数は、記録終了条件を示す値か否かを返却します。

分類

要素別処理関数

形式

result = IS_IOSTAT_EOR (*I*)

I

整数型でなければなりません。

result

基本論理型です。

関数結果

*I*が記録終了条件を示すIOSTAT 指定子の値と一致する場合、真が返却されます。一致しない場合、偽が返却されます。

2.309 ITIMEサービスサブルーチン

機能説明

現在の時、分、秒を取得します。

形式

CALL ITIME (*ia*)

ia

基本整数型配列。現在の時、分、秒が設定されます。
要素数が3より小さい場合、サービスサブルーチンの動作は、保証されません。また、要素数が3より大きい場合、4要素目以降の値は変更されません。

使用例

```
use service_routines,only:itime
integer :: t(3)
call itime(t)
write(6,fmt="(1x,i2,':',i2,':',i2)") t
end
```

2.310 IVALUEサービスサブルーチン

機能説明

第3引数の値をそのまま、第1引数が示す領域に転送します。転送する長さは4バイトを単位として第2引数個分です。

形式

CALL IVALUE (*i* , *j* , *k*)

i

基本整数型スカラまたは基本整数型配列。*k* の値を転送する領域です。

j

基本整数型スカラ。転送する個数を指定します。

k

基本整数型スカラ。*i* に転送する値を指定します。

使用例

```
use service_routines,only:ivalue
integer :: i(3),k
open(10,file='x.dat')
write(10,*)1
rewind(10)
read(10,'(a)')k
```

```
call ivalue(i,3,k)
end
```

2.311 IZEXT組込み関数

IZEXT 関数は、ゼロ拡張を行います。

分類

要素別処理関数

形式

result = IZEXT (*A*)
result = IZEXT2 (*A*)
result = JZEXT (*A*)
result = JZEXT2 (*A*)
result = JZEXT4 (*A*)

A

IZEXT :1バイトの論理型でなければなりません。
IZEXT2 :2バイトの整数型でなければなりません。
JZEXT :4バイトの論理型でなければなりません。
JZEXT2 :2バイトの整数型でなければなりません。
JZEXT4 :4バイトの整数型でなければなりません。

result

IZEXT :2バイトの整数型です。
IZEXT2 :2バイトの整数型です。
JZEXT :4バイトの整数型です。
JZEXT2 :4バイトの整数型です。
JZEXT4 :4バイトの整数型です。

機能説明

IZEXT、IZEXT2、JZEXT、JZEXT2、およびJZEXT4 は、ゼロ拡張を行います。

結果の値は、*A* を0ビット目から複写し、空いたビットには0を設定した値です。

関数の結果の型は、IZEXT およびIZEXT2 の場合は2バイトの整数型、JZEXT、JZEXT2、およびJZEXT4 の場合は4バイトの整数型となります。

使用例

```
i = jzext2(-4_2)          ! i には65532が代入されます
```

2.312 JDATEサービス関数

機能説明

プログラムの実行年月日を

"yyddd　"

という文字列の形式で返却します。

yy は西暦年の下2けた、ddd は1年の通算日を表しています。

形式

ch = JDATE ( )

関数結果

長さ8の基本文字型スカラ。西暦年の下2けたと1年の通算日を5バイトで表し、後ろに3バイトの空白を付加した文字列を返却します。

使用例

```
use service_routines,only:jdate
character(len=8) :: dt
dt = jdate( )
end
```

2.313 KILLサービス関数

機能説明

プロセスまたはプロセスのグループにシグナルを送信します。

形式

iy = KILL (*pid* , *sig*)

pid

基本整数型スカラ。ユーザが所有するプロセスID を指定します。

sig

基本整数型スカラ。シグナル番号を指定します。

関数結果

基本整数型スカラ。正常のシグナルを送信したときは0、エラーが発生したときは0以外の値が返却されます。

使用例

```
use service_routines,only:kill,getpid
i = kill(getpid(),9)
end
```

2.314 KIND組込み関数

KIND 関数は、*X* の種別型パラメタ値を返却します。

分類

問合せ関数

形式

result = KIND (*X*)

X

どの組込み型でもかまいません。

result

基本整数型スカラです。

機能説明

KIND は、*X* の種別型パラメタ値を返却します。

関数の結果の型は、基本整数型です。

使用例

```
i = kind (0.0)              ! i には4が代入されます
```

2.315 LBOUND組込み関数

LBOUND 関数は、配列添字の下限を返却します。

分類

問合せ関数

形式

result = LBOUND (*ARRAY* [, *DIM* , *KIND*])

ARRAY

スカラであってはなりません。空状態のポインタまたは割り付けられていない割付け配列であってはなりません。

DIM (省略可能)

整数型スカラであって、1 <= *DIM* <= n の範囲の値でなければなりません。ここで、n は*ARRAY*の次元数とします。対応する実引数は、省略可能な仮引数であってはなりません。

KIND(省略可能)

スカラ整数定数式でなければなりません。

result

整数型です。*KIND*が指定された場合、種別型パラメタは*KIND*の指定に従います。*KIND*が省略された場合、種別型パラメタは基本整数型になります。*DIM*が指定されている場合はスカラとなり、省略された場合は大きさn の1次元の配列です。ここで、n は*ARRAY*の次元数とします。

機能説明

LBOUND は、配列添字の下限を返却します。

－ *DIM*を指定している場合

*ARRAY*の第*DIM*次元目の添字の下限値を返却します。第*DIM*次元目の大きさが0の場合は、1を返却します。

－ *DIM*を省略している場合

*ARRAY*のすべての添字の上限値(/(LBOUND(*ARRAY*,I),I=1,n)/) を返却します。ここで、n は*ARRAY*の次元数とします。

関数の結果の型は、種別型パラメタ*KIND*が指定された場合、種別型パラメタ*KIND*の整数型となります。*KIND*が省略された場合、関数の結果の型は基本整数型です。

使用例

```
integer, dimension (3,-4:0) :: i
integer :: k,j(2)
j = lbound(i)            ! j には(/1,-4/) が代入されます
k = lbound(i, 2)         ! k には-4が代入されます
```

2.316 LCOBOUND組込み関数

LCOBOUND 関数は、共配列の共下限を返却します。

分類

問合せ関数

形式

result = LCOBOUND (*COARRAY* [, *DIM*, *KIND*])

COARRAY

共配列でなければなりません。割付け可能である場合は割り付けられていなければなりません。

DIM (省略可能)

整数型スカラであって、1 <= *DIM* <= n の範囲でなければなりません。ここで、n は*COARRAY*の共次元数です。対応する実引数は、省略可能な仮引数であってはなりません。

KIND (省略可能)

スカラ整数型定数式でなければなりません。

result

整数型です。*KIND*が指定された場合、種別型パラメタは*KIND*の値となり、そうでない場合は基本整数型となります。*DIM*が指定された場合、*COARRAY*の共添字*DIM*に対する共下限を値とするスカラとなります。そうでない場合は大きさがn の1次元配列となります。ここで、n は*COARRAY*の共次元数です。その場合、各要素の値は、それぞれの共添字の共下限となります。

使用例

```
integer, save :: k[2,3:4,*]
if (this_image()==1) then
  print *,lcobound(k,dim=1) ! 1が出力されます
  print *,lcobound(k,dim=2) ! 3が出力されます
  print *,lcobound(k,dim=3) ! 1が出力されます
end if
```

2.317 LEADZ組込み関数

LEADZ 関数は、先行する0のビット数を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
LEADZ	-----	1	整数型	基本整数型

result = LEADZ (*I*)

I

整数型でなければなりません。

result

基本整数型です。

機能説明

先行する0のビット数を求めます。*I*の値が0である場合、結果は値BIT_SIZE(*I*) です。

関数の結果の型は、基本整数型です。

使用例

```
k = leadz( -1 )          ! k には0が代入されます
m = leadz( 0 )           ! m には32が代入されます
n = leadz( 1 )           ! n には31が代入されます
```

2.318 LEN組込み関数

LEN 関数は、文字列の文字長を返却します。

分類

問合せ関数

形式

result = LEN (*STRING* [, *KIND*])

STRING

文字型でなければなりません。スカラまたは配列でなければなりません。空状態のポインタまたは割り付けられていない割付け実体であるとき、その文字長パラメタは無指定であってはなりません。

KIND（省略可能）

スカラ整数定数式でなければなりません。

result

整数型です。*KIND* が指定された場合、種別型パラメタは*KIND* の指定に従います。
KIND が省略された場合、種別型パラメタは基本整数型になります。

機能説明

LEN は、*STRING* の文字要素の長さを返却します。

関数の結果の型は、種別型パラメタ*KIND* が指定された場合、種別型パラメタ*KIND* の整数型となります。*KIND* が省略された場合、関数の結果の型は基本整数型です。

使用例

```
i = len ('Fujitsu') ! i には7が代入されます
```

2.319 LEN_TRIM組込み関数

LEN_TRIM 関数は、文字列の末尾の空白を除いた文字長を返却します。

分類

要素別処理関数

形式

result = LEN_TRIM (*STRING* [, *KIND*])

STRING

文字型でなければなりません。

KIND（省略可能）

スカラ整数定数式でなければなりません。

result

整数型です。*KIND* が指定された場合、種別型パラメタは*KIND* の指定に従います。
KIND が省略された場合、種別型パラメタは基本整数型になります。

機能説明

LEN_TRIM は、*STRING* の末尾の空白を除いた文字長を返却します。

STRING の長さが0または*STRING* がすべて空白で構成されている場合、0を返却します。

関数の結果の型は、種別型パラメタ*KIND* が指定された場合、種別型パラメタ*KIND* の整数型となります。*KIND* が省略された場合、関数の結果の型は基本整数型です。

使用例

```
i = len_trim ('Fujitsu ')       ! i には7が代入されます
i = len_trim (' ')              ! i には0が代入されます
```

2.320 LGE組込み関数

LGE 関数は、文字列の大小比較を行います。

分類

要素別処理関数

形式

result = LGE (*STRING_A* , *STRING_B*)

STRING_A

基本文字型でなければなりません。

STRING_B

基本文字型でなければなりません。

result

基本論理型です。

機能説明

LGE は、ASCII 大小順序に基づいて*STRING_A* の内容が*STRING_B* の内容以上の場合に真を返却し、それ以外は偽を返却します。

STRING_A と*STRING_B* の長さがともに0の場合は真を返却します。

関数の結果の型は、基本論理型です。

使用例

```
logical :: l
l = lge('elephant','horse')          ! l には偽が代入されます
```

2.321 LGT組込み関数

LGT 関数は、文字列の大小比較を行います。

分類

要素別処理関数

形式

result = LGT (*STRING_A* , *STRING_B*)

STRING_A

基本文字型でなければなりません。

STRING_B

基本文字型でなければなりません。

result

基本論理型です。

機能説明

LGT は、ASCII 大小順序に基づいて*STRING_A* の内容が*STRING_B* の内容よりも大きい場合に真を返却し、それ以外は偽を返却します。

STRING_A と*STRING_B* の長さがともに0の場合は真を返却します。

関数の結果の型は、基本論理型です。

使用例

```
logical :: l
l = lgt('elephant','horse')          ! l には偽が代入されます
```

2.322 LINKサービス関数

機能説明

既存ファイルに関する新しいリンクを作成します。

形式

iy = LINK (*path1* , *path2*)

path1

基本文字型スカラ。既存のファイルを指定します。
指定するファイル名は既存ファイルでなければなりません。

path2

基本文字型スカラ。新たに作成するディレクトリエントリーのパス名を指定します。

関数結果

基本整数型スカラ。リンクを作成できたときは0、エラーが発生したときは0以外の値が返却されます。

使用例

```
use service_routines,only:link
i = link('libx.a','../libp.a')
end
```

2.323 LLE組込み関数

LLE 関数は、文字列の大小比較を行います。

分類

要素別処理関数

形式

result = LLE (*STRING_A* , *STRING_B*)

STRING_A

基本文字型でなければなりません。

STRING_B

基本文字型でなければなりません。

result

基本論理型です。

機能説明

LLEは、ASCII 大小順序に基づいて*STRING_A* の内容が*STRING_B* の内容以下の場合に真を返却し、それ以外は偽を返却します。

STRING_A と*STRING_B* の長さがともに0の場合は真を返却します。

関数の結果の型は、基本論理型です。

使用例

```
logical :: l
l = lle('elephant','horse')        ! l には真が代入されます
```

2.324 LLT組込み関数

LLT 関数は、文字列の大小比較を行います。

分類

要素別処理関数

形式

result = LLT (*STRING_A* , *STRING_B*)

STRING_A

基本文字型でなければなりません。

STRING_B

基本文字型でなければなりません。

result

基本論理型です。

機能説明

LLT は、ASCII 大小順序に基づいて*STRING_A* の内容が*STRING_B* の内容より小さい場合に真を返却し、それ以外は偽を返却します。

STRING_A と*STRING_B* の長さがともに0の場合は真を返却します。

関数の結果の型は、基本論理型です。

使用例

```
logical :: l
l = llt('elephant','horse')        ! l には真が代入されます
```

2.325 LNBLNKサービス関数

機能説明

文字列の末尾の空白を除いた文字長を返却します。

形式

iy = LNBLNK (*string*)

string

基本文字型スカラ。

関数結果

基本整数型スカラ。*string* の末尾の空白を除いた文字長が返却されます。*string* がすべて空白である場合、結果の値は0になります。

使用例

```
use service_routines,only:lnblnk
character(len=10) :: ch1
integer :: i
read (*,*) ch1
i = lnblnk(ch1)
write(*,fmt='(a)') ch1(1:i)
end
```

2.326 LOC組込み関数

LOC 関数は、*X* の先頭アドレスを返却します。

分類

アドレス取得関数

形式

result = LOC (*X*)

X

どの組込み型でもかまいません。ベクトル添字をもつ配列を指定してはなりません。

result

8バイトの整数型スカラです。

機能説明

LOC は、*X* の先頭アドレスを返却します。

関数の結果の型は8バイトの整数型です。

使用例

```
i = loc(a)          ! a のアドレスを取得します
```

2.327 LOCK文

LOCK文は、以下の形式です。

```
LOCK ( lock-variable [ , lock-stat-list ] )
```

*lock-variable*は、ロック変数です。LOCK_TYPE 型でなければなりません。LOCK_TYPE 型については“1.18.3 LOCK_TYPE 型”を参照してください。

lock-stat-list は、コンマで区切られたロック状態指定子の並びです。*lock-stat*は以下の形式です。

```
ACQUIRED_LOCK = scalar-logical-variable     または

sync-stat
```

sync-stat は同期状態指定子であり、以下の形式です。

```
STAT = stat-variable            または

ERRMSG = errmsg-variable
```

stat-variable および*errmsg-variable* については“1.18.2 同期状態指定子”を参照してください。各指定子は、*lock-stat-list* に2回以上指定してはなりません。

ロック変数がある像においてLOCK文によってロックされ、その像でUNLOCK文によってロック解除されていないとき、そのロック変数はその像にロックされています。

ACQUIRED_LOCK指定子のないLOCK文の実行に成功すると、ロック変数はその像によってロックされます。ロック変数がすでに他の像によってロックされている場合、他の像がロック変数をロック解除した後にLOCK文によりロック変数が定義されます。

ロック変数がロック解除されていると、ACQUIRED_LOCK指定子のあるLOCK文の実行が成功した場合、ロック変数がその像にロックされ、ACQUIRED_LOCK指定子に指定された変数が真になります。ロック変数がすでに他の像によってロックされている場合、ACQUIRED_LOCK指定子のあるLOCK文が成功した場合にロック変数は変更されず、ACQUIRED_LOCK変数が偽になります。

ロック変数が像MにおいてUNLOCK文の実行によりロック解除され、その後、像TにおいてLOCK文の実行によりロックされた場合、像MのUNLOCK文に先行するセグメント(“1.18.1 セグメント”)は、像TのLOCK文より後のセグメントに先行します。ロック変数をロックしないLOCK文の実行は実行順に影響しません。

LOCK文のロック変数が実行中の像によってすでにロックされている場合、誤り条件が発生します。LOCK文の実行中に誤り条件が発生した場合、ロック変数の値およびACQUIRED_LOCK変数の値は変更されません。

LOCK 文の例:

```
use iso_fortran_env, only : lock_type
type(lock_type) :: x[*]
if (this_image() == 4) then
  lock(x)     ! ロック変数xは像4によってロックされます
end if
end
```

2.328 LOG組込み関数

LOG 関数は、実数型データまたは複素数型データの自然対数を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
LOG	-----	1	実数型または複素数型	実数型または複素数型
	LOG		単精度実数型	単精度実数型
	ALOG		単精度実数型	単精度実数型
	DLOG		倍精度実数型	倍精度実数型
	QLOG		4倍精度実数型	4倍精度実数型
	CLOG		単精度複素数型	単精度複素数型
	CDLOG		倍精度複素数型	倍精度複素数型
	CQLOG		4倍精度複素数型	4倍精度複素数型

result = LOG（ *X* ）

X

実数型または複素数型でなければなりません。*X*が実数型の場合、*X*> 0.0 でなければなりません。*X*が複素数型の場合、*X*=(0.0, 0.0) であってはなりません。

result

*X*と同じ型です。

機能説明

LOG、ALOG、DLOG、QLOG、CLOG、CDLOG、およびCQLOG は、実数型データまたは複素数型データの自然対数を求めます。

*X*が実数型の場合、*X*<= 0.0 であってはなりません。*X*が複素数型の場合、*X*が(0.0, 0.0) であってはなりません。

総称名LOG は、すべての実数型または複素数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
x = log (3.7)
```

2.329 LOG10組込み関数

LOG10 関数は、実数型データの10.0を底とする対数を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
LOG10	-----	1	実数型	実数型
	LOG10		単精度実数型	単精度実数型
	ALOG10		単精度実数型	単精度実数型
	DLOG10		倍精度実数型	倍精度実数型
	QLOG10		4倍精度実数型	4倍精度実数型

result = LOG10（ *X* ）

X

実数型でなければなりません。*X* > 0.0 でなければなりません。

result

X と同じ型です。

機能説明

LOG10、ALOG10、DLOG10、およびQLOG10 は、実数型データの10.0を底とする対数を求めます。

X <= 0.0 であってはなりません。

総称名LOG10 は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*X* と同じです。

使用例

```
x = log10 (3.7)
```

2.330 LOG2組込み関数

LOG2 関数は、実数型データの2.0を底とする対数を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
LOG2	-----	1	実数型	実数型
	LOG2		単精度実数型	単精度実数型
	ALOG2		単精度実数型	単精度実数型
	DLOG2		倍精度実数型	倍精度実数型
	QLOG2		4倍精度実数型	4倍精度実数型

result = LOG2 (*X*)

X

実数型でなければなりません。*X* > 0.0 でなければなりません。

result

X と同じ型です。

機能説明

LOG2、ALOG2、DLOG2、およびQLOG2 は、実数型データの2.0を底とする対数を求めます。

X <= 0.0 であってはなりません。

総称名LOG2 は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*X* と同じです。

使用例

```
x = log2(3.7)
```

2.331 LOGICAL組込み関数

LOGICAL 関数は、*L* の値を種別型パラメタ *KIND* に従って変換します。

分類

要素別処理関数

形式

result = LOGICAL (*L* [, *KIND*])

L

論理型でなければなりません。

KIND（省略可能）

スカラ整数定数式でなければなりません。

result

論理型です。*KIND* が指定された場合、種別型パラメタは *KIND* に従います。*KIND* が省略された場合、種別型パラメタは基本論理型になります。

機能説明

LOGICAL は、*L* の値を種別型パラメタ *KIND* に従って変換します。

関数の結果の型は、種別型パラメタ *KIND* が指定された場合、種別型パラメタ *KIND* の論理型となります。*KIND* が省略された場合、関数の結果の型は基本論理型です。

使用例

```
logical(kind=2) :: l
l = logical(.true.,2) ! l には種別型パラメタ2をもつ真の値が代入されます
```

2.332 LOGICAL型宣言文

LOGICAL 型宣言文は、論理型のデータ実体を宣言します。

LOGICAL 型宣言文は、以下の形式です。

LOGICAL [*kind-selector*] [[, *attr-spec*]... ::] *entity-decl-list*

型宣言文の詳細については、“2.3 型宣言文”を参照してください。

2.333 LOG_GAMMA組込み関数

LOG_GAMMA 関数は、実数型データに対応するガンマ関数値の絶対値の対数を計算します。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
LOG_GAMMA	-----	1	実数型	実数型
LGAMMA	-----	1	実数型	実数型
	LGAMMA		単精度実数型	単精度実数型
	ALGAMA		単精度実数型	単精度実数型
	DLGAMA		倍精度実数型	倍精度実数型
	QLGAMA		4倍精度実数型	4倍精度実数型

result = LOG_GAMMA(*X*)

X

実数型でなければなりません。

result

*X*と同じ型です。

機能説明

LOG_GAMMA、LGAMMA、ALGAMA、DLGAMA、およびQLGAMA は、実数型データに対応するガンマ関数値の絶対値の対数を計算します。

単精度実数型の引数の場合、引数は *X* < 4.03711E+36 でなければなりません。

倍精度実数型の引数の場合、引数は *X* < 2.55634D+305 でなければなりません。

4倍精度実数型の引数の場合、引数は *X* < 1.048Q+4928 でなければなりません。

X の値は、負の整数、またはゼロであってはなりません。

総称名LOG_GAMMA、およびLGAMMA は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*X* と同じです。

使用例

```
a = log_gamma(.5)
```

2.334 LONGサービス関数

機能説明

2バイトの整数型スカラを基本整数型スカラに変換します。

形式

iy = LONG (*ix*)

ix

2バイトの整数型スカラ。

関数結果

基本整数型スカラ。2バイトの整数型スカラを基本整数型スカラに変換した値が返却されます。

使用例

```
use service_routines,only:long
integer(2) :: ix
integer :: ix4
ix4 = long(ix)
end
```

2.335 LRSHFT組込み関数

LRSHFT 関数は、右論理シフトを行います。

分類

要素別処理関数

形式

result = LRSHFT (*I* , *SHIFT*)

I

整数型でなければなりません。

SHIFT

整数型でなければなりません。*SHIFT* < BIT_SIZE(*I*) でなければなりません。また、負の値であってはなりません。

result

I と同じ型です。

機能説明

LRSHFT は、右論理シフトを行います。

I を *SHIFT* ビット分だけ右論理シフトを行います。

関数の結果の型は、*I* と同じです。

使用例

```
i = lrshft (7,2)          ! i には1が代入されます
```

2.336 LSHIFT組込み関数

LSHIFT 関数は、左論理シフトを行います。

分類

要素別処理関数

形式

result = LSHIFT(*I* , *SHIFT*)

I

整数型でなければなりません。

SHIFT

整数型でなければなりません。*SHIFT* < BIT_SIZE(*I*) でなければなりません。また、負の値であってはなりません。

result

I と同じ型です。

機能説明

LSHIFT は、左論理シフトを行います。

I を *SHIFT* ビット分だけ左シフトを行います。

関数の結果の型は、*I* と同じです。

使用例

```
i = lshift (7,2) ! i には28が代入されます
```

2.337 LSTATサービス関数

機能説明

ファイルの状態に関する情報を返却します。

形式

iy = LSTAT (*name* , *status*)

name

基本文字型スカラ。情報を返却するファイル名を指定します。

status

基本整数型配列。ファイルの状態に関する情報が設定されます。

name がシンボリックリンクの場合、リンクについての情報が返却されます。

指定する配列変数は要素数が13以上の配列変数でなければなりません。配列の要素数が12以下の場合の動作は保証されません。

status (1)　：ファイル・モード
status (2)　：ファイルのiノード番号
status (3)　：デバイスのiノード番号
status (4)　：デバイス識別子（特殊ファイルだけ設定されます）
status (5)　：ファイルに対するハード・リンク数
status (6)　：オーナーのユーザID
status (7)　：オーナーのグループID
status (8)　：ファイルの合計サイズ（バイト単位）
status (9)　：ファイルの最後のアクセス時間
status (10)　：ファイルの最後の変更時間
status (11)　：ファイルの最後のステータス変更時間
status (12)　：ファイル・システムの最適なブロック・サイズ
status (13)　：実際に割り当てられたブロック数

関数結果

基本整数型スカラ。情報が返却できた時は0、エラーが発生した場合は0以外を返却します。

使用例

```
use service_routines,only:lstat
integer :: st(13)
print *,lstat('f90.dat',st)
end
```

2.338 LSTAT64サービス関数

機能説明

ファイルの状態に関する情報を返却します。

形式

iy = LSTAT64 (*name* , *status*)

name

基本文字型スカラ。情報を返却するファイル名を指定します。

status

8バイトの整数型配列。ファイルの状態に関する情報が設定されます。

name がシンボリックリンクの場合、リンクについての情報が返却されます。

指定する配列変数は要素数が13以上の配列変数でなければなりません。配列の要素数が12以下の場合の動作は保証されません。

status (1)　：ファイル・モード
status (2)　：ファイルのiノード番号
status (3)　：デバイスのiノード番号
status (4)　：デバイス識別子（特殊ファイルだけ設定されます）
status (5)　：ファイルに対するハード・リンク数
status (6)　：オーナーのユーザID
status (7)　：オーナーのグループID
status (8)　：ファイルの合計サイズ（バイト単位）
status (9)　：ファイルの最後のアクセス時間
status (10)　：ファイルの最後の変更時間
status (11)　：ファイルの最後のステータス変更時間
status (12)　：ファイル・システムの最適なブロック・サイズ
status (13)　：実際に割り当てられたブロック数

関数結果

基本整数型スカラ。情報が返却できた時は0、エラーが発生した場合は0以外を返却します。

使用例

```
use service_routines,only:lstat64
integer(kind=8) :: st(13)
print *,lstat64('f90.dat',st)
end
```

2.339 LTIMEサービスサブルーチン

機能説明

システム時間をローカル時間に従って、秒、分、時間、日、月、年、曜日、1月1日からの通算日付、夏時間かどうかを表す情報を取得します。

形式

```
CALL LTIME ( time , t )
```

time

基本整数型スカラ。システム時間を指定します。

t

基本整数型配列。*time* で指定したシステム時間をローカル時間に従って、以下の配列に設定します。

要素数が9より小さい場合、サービスサブルーチンの動作は、保証されません。また、引数*t*の要素数が9より大きい場合、10要素目以降の値は変更されません。

配列	値
t(1)	秒 (0-59)
t(2)	分 (0-59)
t(3)	時 (0-23)
t(4)	日 (1-31)
t(5)	月 (0-11)
t(6)	1900からの通算年
t(7)	日曜日からの通算曜日 (0-6)
t(8)	1月1日からの通算日(0-365)
t(9)	夏時間を示すフラグ(標準時間は0、夏時間は1)

使用例

```
use service_routines,only:ltime,time
integer :: t(9)
call ltime(time( ),t)
write(6,fmt="(1x,9i4)") t
end
```

2.340 MALLOCサービス関数

機能説明

領域を確保し、そのアドレスを返却します。

形式

```
iy = MALLOC ( size )
```

size

8バイトの整数型スカラです。確保する領域の大きさを設定します。

関数結果

8バイトの整数型スカラです。領域が獲得できたときは獲得した領域の先頭番地、エラーが発生したときは0を返却します。

注意事項

MALLOC サービス関数で確保した領域は、初期化されません。

使用例

```
use service_routines,only:free,malloc
integer(8) :: i
i = malloc(20_8)
...
call free(i)
end
```

2.341 MAP文

MAP 文は、STRUCTURE 文による派生型定義内において、共用体宣言のブロックを開始します。派生型定義の詳細については、“1.5.11.1 派生型定義”を参照してください。

MAP 文は、以下の形式です。

```
MAP
```

MAP 文の例:

```
structure /complex_element/
  union
    map
      real :: real,imag
    end map
    map
      complex :: complex
    end map
  end union
end structure
record /complex_element/ x
x%real = 2.0
x%imag = 3.0
print *,x%complex       ! 複素数型の要素complex は、(2.0,3.0) で確定しています
```

2.342 MASKL組込み関数

MASKL 関数は、左から*I*ビットまで1にした値を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
MASKL	-----	1	整数型	基本整数型
		2	整数型 , 整数型	整数型

result = MASKL (*I* [, *KIND*])

I

整数型でなければなりません。結果の型のビットサイズ以下の値でなければなりません。また、負の値であってはなりません。

KIND（省略可能）

スカラ整数定数式でなければなりません。

result

整数型です。*KIND* が指定された場合、種族型パラメタは*KIND* の指定に従います。

KIND が省略された場合、種別型パラメタは基本整数型になります。

機能説明

結果の値は、左端の*I*ビット列が1となり、残りのビット列が0となります。

関数の結果の型は、種別型パラメタ*KIND* が指定された場合、種別型パラメタ*KIND* の整数型となります。*KIND* が省略された場合、基本整数型です。

使用例

```
k = maskl( 31 )          ! k には-2が代入されます
```

2.343 MASKR組込み関数

MASKR 関数は、右から*I*ビットまで1にした値を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
MASKR	-----	1	整数型	基本整数型
		2	整数型 , 整数型	整数型

result = MASKR (*I* [, *KIND*])

I

整数型でなければなりません。結果の型のビットサイズ以下の値でなければなりません。また、負の値であってはなりません。

KIND（省略可能）

スカラ整数定数式でなければなりません。

result

整数型です。*KIND* が指定された場合、種族型パラメタは*KIND* の指定に従います。

KIND が省略された場合、種別型パラメタは基本整数型になります。

機能説明

結果の値は、右端の*I*ビット列が1となり、残りのビット列が0となります。

関数の結果の型は、種別型パラメタ*KIND* が指定された場合、種別型パラメタ*KIND* の整数型となります。*KIND* が省略された場合、基本整数型です。

使用例

```
k = maskr( 2 )           ! k には3が代入されます
```

2.344 MATMUL組込み関数

MATMUL 関数は、行列の積を計算します。

分類

変形関数

形式

result = MATMUL (*MATRIX_A* , *MATRIX_B*)

MATRIX_A

数値型または論理型です。1次元配列または2次元配列でなければなりません。

MATRIX_B が1次元配列の場合、2次元配列でなければなりません。

MATRIX_B

MATRIX_A が数値型のときは数値型、論理型のときは論理型でなければなりません。1次元配列または2次元配列でなければなりません。

MATRIX_A が1次元配列の場合、2次元配列でなければなりません。

MATRIX_B の最初の次元の寸法は、*MATRIX_A* の最後の次元の寸法に等しくなければなりません。

result

引数が数値型の場合、*MATRIX_A* * *MATRIX_B* の型および種別型パラメタとします。引数が論理型の場合、*MATRIX_A* .and. *MATRIX_B* の型および種別型パラメタとします。

MATRIX_A、*MATRIX_B*、および結果の形状の関係は以下の表のとおりです。

MATRIX_A	MATRIX_B	result
(n,m)	(m,k)	(n,k)
(m)	(m,k)	(k)
(n,m)	(m)	(n)

機能説明

MATMUL は、行列の積を計算します。

－ *MATRIX_A* の形状が (n,m) で*MATRIX_B* の形状が (m,k) の場合

引数が数値型の場合、結果の要素(I, J) は、
SUM(*MATRIX_A* (I, :) * *MATRIX_B* (:, J)) になります。
引数が論理型の場合、結果の要素(I, J) は、
ANY(*MATRIX_A* (I, :) .and. *MATRIX_B* (:, J)) になります。

－ *MATRIX_A* の形状が (m) で*MATRIX_B* の形状が (m,k) の場合

引数が数値型の場合、結果の要素 (J) は、
SUM(*MATRIX_A* (:) * *MATRIX_B* (:, J)) になります。
引数が論理型の場合、結果の要素 (J) は、
ANY(*MATRIX_A* (:) .and. *MATRIX_B* (:, J)) になります。

－ *MATRIX_A* の形状が (n,m) で*MATRIX_B* の形状が (m) の場合

引数が数値型の場合、結果の要素 (I) は、
SUM(*MATRIX_A* (I, :) * *MATRIX_B* (:)) になります。
引数が論理型の場合、結果の要素 (I) は、
ANY(*MATRIX_A* (I, :) .and. *MATRIX_B* (:)) になります。

使用例

```
integer a(2,3), b(3), c(2)
a = reshape((/1,2,3,4,5,6/), (/2,3/))          ! a は ┌       ┐
                                               !      | 1 3 5 |
                                               !      | 2 4 6 |
                                               !      └       ┘
b = (/1,2,3/)                                  ! b は(/1,2,3/)
c = matmul(a,b)                                ! c には(/22,28/) が代入されます
```

2.345 MAX組込み関数

MAX 関数は、すべての引数の中の最大値を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
MAX	-----	2以上	1バイトの整数型	1バイトの整数型
	I2MAX0		2バイトの整数型	2バイトの整数型
	IMAX0		2バイトの整数型	2バイトの整数型
	MAX		4バイトの整数型	4バイトの整数型
	MAX0		4バイトの整数型	4バイトの整数型
	JMAX0		4バイトの整数型	4バイトの整数型
	-----		8バイトの整数型	8バイトの整数型
	-----		実数型	実数型
	AMAX1		単精度実数型	単精度実数型
	DMAX1		倍精度実数型	倍精度実数型
	QMAX1		4倍精度実数型	4倍精度実数型
	-----		文字型	文字型
-----	AIMAX0		2バイトの整数型	単精度実数型
	AMAX0		4バイトの整数型	単精度実数型
	AJMAX0		4バイトの整数型	単精度実数型
	IMAX1		単精度実数型	2バイトの整数型
	MAX1		単精度実数型	4バイトの整数型
	JMAX1		単精度実数型	4バイトの整数型

result = MAX (*A1* , *A2* [, *A3* , ...])

A1 , *A2* [, *A3* , ...]

整数型、実数型または文字型でなければなりません。すべての引数は同じ型および種別型パラメタでなければなりません。

result

MAX、I2MAX0、IMAX0、MAX0、JMAX0、AMAX1、DMAX1、およびQMAX1 の場合、引数と同じ型です。

AIMAX0、AMAX0、およびAJMAX0 の場合、単精度実数型です。

IMAX1 の場合、2バイトの整数型です。

MAX1 およびJMAX1 の場合、4バイトの整数型です。

引数が文字型のとき、結果の文字列の長さは、引数の中で最も長い引数の長さとなります。

機能説明

MAX、I2MAX0、IMAX0、MAX0、JMAX0、AMAX1、DMAX1、QMAX1、AIMAX0、AMAX0、AJMAX0、IMAX1、MAX1、およびJMAX1 は、すべての引数の中の最大値を求めます。

引数が文字型のとき、結果は組込み関係演算子の適用によって選択される値になります。選択された引数が最も長い引数より短い場合、最も長い引数の長さ分だけ右に空白を詰めて広げられます。

－ MAX、I2MAX0、IMAX0、MAX0、JMAX0、AMAX1、DMAX1、およびQMAX1 を使用する場合

総称名MAX は、すべての整数型および実数型の引数に使用することができます。それぞれの関数の結果の型は、引数と同じです。

－ AIMAX0、AMAX0、AJMAX0、IMAX1、MAX1、およびJMAX1 を使用する場合

AIMAX0 は2バイトの整数型の引数、AMAX0 およびAJMAX0 は4バイトの整数型の引数に使用することができます。それぞれの関数の結果の型は、単精度実数型です。

IMAX1、MAX1、およびJMAX1 は、単精度実数型の引数に使用することができます。それぞれの関数の結果の型は、IMAX1 の場合2バイトの整数型、MAX1 およびJMAX1 の場合4バイトの整数型となります。

使用例

```
character(len=7) :: c
k = max(-14,3,0,-2,19,1)              ! k には19が代入されます
c = max("abcdefg","d","xyz")          ! c には"xyz    " が代入されます。
```

2.346 MAXEXPONENT組込み関数

MAXEXPONENT 関数は、*X* と同じ型および種別型パラメタの数体系で最大の指数を返却します。

分類

問合せ関数

形式

result = MAXEXPONENT (*X*)

X

実数型でなければなりません。

result

基本整数型スカラです。

機能説明

MAXEXPONENT は、*X* と同じ型および種別型パラメタの数体系で最大の指数を返却します。

関数の結果の型は、基本整数型です。

以下の値の固定値となります。

引数の型	結果の型
半精度実数型	16
単精度実数型	128
倍精度実数型	1024
4倍精度実数型	16384

使用例

```
real :: r
integer :: i
i = maxexponent(r)          ! i には128が代入されます
```

2.347 MAXLOC組込み関数

MAXLOC 関数は、配列内で最大値となる要素の添字値を調べます。

分類

変形関数

形式

result = MAXLOC (*ARRAY* [, *MASK* , *KIND* , *BACK*])　　または
result = MAXLOC (*ARRAY* , *DIM* [, *MASK* , *KIND* , *BACK*])

ARRAY

整数型、実数型または文字型でなければなりません。スカラであってはなりません。

DIM

整数型スカラであって、1 <= *DIM*<= n の範囲の値でなければなりません。ここで、n は*ARRAY*の次元数とします。対応する実引数は、省略可能な仮引数であってはなりません。

MASK（省略可能）

論理型でなければならず、*ARRAY*と形状適合しなければなりません。

KIND（省略可能）

スカラ整数定数式でなければなりません。

BACK（省略可能）

論理型スカラでなければなりません。

result

整数型です。*KIND*を指定した時、種別型パラメタは*KIND*に従います。*KIND*を省略したとき、種別型パラメタは基本整数型になります。*DIM*が省略された場合は、大きさ n の1次元配列となります。n は*ARRAY*の次元数です。*DIM*が指定された場合は、*ARRAY*が1次元配列の場合はスカラ、それ以外の場合は、*ARRAY*の形状を (d_1, d_2, ..., d_n) としたとき、
(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_n) となります。

機能説明

MAXLOC は、*ARRAY*の第*DIM*次元の中でもっとも大きい値となる要素の添字を調査します。

－ *DIM*を省略している場合

*MASK*を省略すると、*MASK*の全要素が真であるとして、以下の処理が行われます。

*MASK*中の真の要素に対応する、*ARRAY*の全要素の最大値が現れる各次元の添字値を要素とする1次元配列を返却します。*ARRAY*の大きさが0または*MASK*の全要素が偽である場合は、すべて0の1次元配列を返却します。

－ *DIM*を指定している場合

*ARRAY*が1次元の場合は、MAXLOC(*ARRAY*[,*MASK*]) の要素となります。
*ARRAY*が2次元以上の場合、結果の要素(S_1, S_2, ..., S_{DIM-1}, S_{DIM+1}, ..., S_n) は、
MAXLOC(*ARRAY*(S_1, S_2, ..., S_{DIM-1}, : , S_{DIM+1}, ..., S_n), DIM = 1[,MASK = *MASK*])
となります。ここで、n は*ARRAY*の次元数、*MASK*は*MASK*(S_1, S_2, ..., S_{DIM-1}, : , S_{DIM+1}, ..., S_n) とします。

－ *BACK*に偽を指定している、または省略している場合

最初に現れる最大値の各次元の添字値が要素となります。

－ *BACK*に真を指定している場合

最後に現れる最大値の各次元の添字値が要素となります。

関数の結果の型は、種別型パラメタ*KIND*が指定された場合、種別型パラメタ*KIND*の整数型となります。*KIND*が省略された場合、関数の結果の型は基本整数型です。

*ARRAY*が文字型のとき、関数の結果は組込み関係演算子の適用によって選択される値になります。*ARRAY*が大きさゼロであるとき、結果のすべての要素はゼロになります。また、*MASK*のすべての要素の値が偽であるとき、結果のすべての要素はゼロになります。

使用例

```
integer, dimension(1) :: i
i = maxloc ((/3,0,4,4/))          ! i には(/3/) が代入されます
```

2.348 MAXVAL組込み関数

MAXVAL 関数は、配列内の最大の値を求めます。

分類

変形関数

形式

result = MAXVAL (*ARRAY* [, *MASK*])　　または
result = MAXVAL (*ARRAY* , *DIM* [, *MASK*])

ARRAY

整数型、実数型または文字型でなければなりません。スカラであってはなりません。

DIM

整数型スカラであって、1 <= *DIM*<= n の範囲の値でなければなりません。ここで、n は*ARRAY*の次元数とします。対応する実引数は、省略可能な仮引数であってはなりません。

MASK（**省略可能**）

論理型でなければならず、*ARRAY*と形状適合しなければなりません。

result

*ARRAY*と同じ型および同じ種別型パラメタです。*DIM*が省略されている、または*ARRAY*が1次元配列の場合は、スカラとなります。それ以外の場合は、*ARRAY*の形状を $(d_1, d_2, ..., d_n)$ としたとき、$(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_n)$ となります。n は*ARRAY*の次元数です。

機能説明

MAXVAL は、*ARRAY*の第*DIM*次元の中でもっとも大きい値を調査します。

－ *DIM*を省略している場合

*MASK*中の真の要素に対応する、*ARRAY*の全要素中の最大値を返却します。*ARRAY*の大きさが0または*MASK*中の全要素が偽である場合、*ARRAY*が整数型の場合は、-HUGE(*ARRAY*)-1、実数型の場合は、-HUGE(*ARRAY*)を返却します。*ARRAY*が大きさゼロで文字型であるときの結果の値は、引数の長さと等しく、各文字が組込み関数
CHAR(0,KIND=KIND(*ARRAY*)) になります。

－ *DIM*を指定している場合

*ARRAY*が1次元の場合は、MAXVAL(*ARRAY*[,*MASK*]) となります。

*ARRAY*が2次元以上の場合、結果の要素 $(S_1, S_2, ..., S_{DIM-1}, S_{DIM+1}, ..., S_n)$ は、MAXVAL(*ARRAY*$(S_1, S_2, ..., S_{DIM-1}, :, S_{DIM+1}, ..., S_n)$, DIM = 1[, MASK = *MASK*]) となります。ここで、n は*ARRAY*の次元数、*MASK*は$MASK(S_1, S_2, ..., S_{DIM-1}, :, S_{DIM+1}, ..., S_n)$ とします。

関数の結果の型は、*ARRAY*と同じです。

*ARRAY*が文字型のとき、結果の値は組込み関係演算子の適用によって選択される値になります。すなわち、引数の種別型パラメタによる文字の大小順序が適用されます。

使用例

```
integer j(2)
integer, dimension (2,2) :: m = reshape((/1,2,3,4/),(/2,2/))
                          ! m は ┌     ┐
                          !      | 1 3 |
                          !      | 2 4 |
                          !      └     ┘
i = maxval(m)             ! i には4が代入されます
j = maxval(m,dim=1)       ! j には(/2,4/) が代入されます
k = maxval(m,mask=m<3)    ! k には2が代入されます
```

2.349 MERGE組込み関数

MERGE 関数は、*MASK*の値に従ってどちらか一方の値を選択します。

分類

要素別処理関数

形式

result = MERGE (*TSOURCE* , *FSOURCE* , *MASK*)

TSOURCE

どの型でもかまいません。

FSOURCE

*TSOURCE*と同じ型および同じ種別型パラメタでなければなりません。

MASK

論理型でなければなりません。

result

*TSOURCE*と同じ型です。

機能説明

*MERGE*は、*MASK*の値に従ってどちらか一方の値を選択します。

*MASK*が真の場合は、*TSOURCE*を返却し、*MASK*が偽の場合は、*FSOURCE*を返却します。

関数の結果の型は、*TSOURCE*と同じです。

使用例

```
integer, dimension(2,2) :: r
integer, dimension(2,2) :: m = reshape((/1,2,3,4/),(/2,2/))
                     ! m は  ┌     ┐
                     !       | 1 3 |
                     !       | 2 4 |
                     !       └     ┘
integer, dimension(2,2) :: n = reshape((/3,3,3,3/),(/2,2/))
                     ! n は  ┌     ┐
                     !       | 3 3 |
                     !       | 3 3 |
                     !       └     ┘
r = merge(m,n,m<n)   ! r には ┌     ┐
                     !       | 1 3 |
                     !       | 2 3 |
                     !       └     ┘ が代入されます
```

2.350 MERGE_BITS組込み関数

MERGE_BITS 関数は、*MASK*のビット値に従ってどちらかの一方のビットを選択します。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
MERGE_BITS	-----	3	整数型または 非10進定数表現 , 整数型または 非10進定数表現 , 整数型または 非10進定数表現	整数型

result = MERGE_BITS (*I* , *J* , *MASK*)

I

整数型または非10進定数表現でなければなりません。

J

整数型または非10進定数表現でなければなりません。

MASK

整数型または非10進定数表現でなければなりません。

result

*I*または*J*と同じ整数型です。

機能説明

MERGE_BITS 関数は、*MASK* のビット値に従ってどちらかの一方のビットを選択します。
IOR(IAND(*I*, *MASK*), IAND(*J*, NOT(*MASK*))) と同じです。

I および *J* が整数型の場合、*I* と *J* は同じ型でなければなりません。

I または *J* のどちらかに非10進定数表現を指定した場合、非10進定数表現は、指定した整数型に変換されます。*I* と *J* の両方に非10進定数表現を指定してはなりません。

MASK が整数型の場合、それは *I* または *J* と同じ整数型でなければなりません。非10進定数表現を指定した場合、非10進定数表現は、*I* または *J* の指定した整数型に変換されます。

関数の結果の型は、*I* または *J* と同じ整数型です。

使用例

```
k = merge_bits(18,13,22)          ! k には27が代入されます
```

2.351 MIN組込み関数

MIN 関数は、すべての引数の中の最小値を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
MIN	-----	2以上	1バイトの整数型	1バイトの整数型
	I2MIN0		2バイトの整数型	2バイトの整数型
	IMIN0		2バイトの整数型	2バイトの整数型
	MIN		4バイトの整数型	4バイトの整数型
	MIN0		4バイトの整数型	4バイトの整数型
	JMIN0		4バイトの整数型	4バイトの整数型
	-----		8バイトの整数型	8バイトの整数型
	-----		実数型	実数型
	AMIN1		単精度実数型	単精度実数型
	DMIN1		倍精度実数型	倍精度実数型
	QMIN1		4倍精度実数型	4倍精度実数型
	-----		文字型	文字型
-----	AIMIN0		2バイトの整数型	単精度実数型
	AMIN0		4バイトの整数型	単精度実数型

総称名	個別名	引数の数	引数の型	結果の型
	AJMIN0		4バイトの整数型	単精度実数型
	IMIN1		単精度実数型	2バイトの整数型
	MIN1		単精度実数型	4バイトの整数型
	JMIN1		単精度実数型	4バイトの整数型

result = MIN (*A1* , *A2* [, *A3* , ...])

A1 , *A2* [, *A3* , ...]

整数型、実数型、または文字型でなければなりません。すべての引数は同じ型および種別型パラメタでなければなりません。

result

MIN、I2MIN0、IMIN0、MIN0、JMIN0、AMIN1、DMIN1、およびQMIN1 の場合、引数と同じ型です。

AIMIN0、AMIN0、およびAJMIN0 の場合、単精度実数型です。

IMIN1 の場合、2バイトの整数型です。

MIN1 およびJMIN1 の場合、4バイトの整数型です。

引数が文字型のとき、結果の文字列の長さは最も長い引数の長さとなります。

機能説明

MIN、I2MIN0、IMIN0、MIN0、JMIN0、AMIN1、DMIN1、QMIN1、AIMIN0、AMIN0、AJMIN0、IMIN1、MIN1、およびJMIN1 は、すべての引数の中の最小値を求めます。

引数が文字型のとき、結果は組込み関係演算子の適用によって選択される値になります。選択された引数が最も長い引数より短い場合、最も長い引数の長さ分だけ右に空白を詰めて広げられます。

- MIN、I2MIN0、IMIN0、MIN0、JMIN0、AMIN1、DMIN1、およびQMIN1 を使用する場合

  総称名MIN は、すべての整数型および実数型の引数に使用することができます。それぞれの関数の結果の型は、引数と同じです。

- AIMIN0、AMIN0、AJMIN0、IMIN1、MIN1、およびJMIN1 を使用する場合

  AIMIN0は2バイトの整数型の引数、AMIN0およびAJMIN0は4バイトの整数型の引数に使用することができます。それぞれの関数の結果の型は、単精度実数型です。

  IMIN1、MIN1、およびJMIN1 は、単精度実数型の引数に使用することができます。それぞれの関数の結果の型は、IMIN1 の場合2バイトの整数型、MIN1 およびJMIN1 の場合4バイトの整数型となります。

使用例

```
character(len=7) :: c
k = min(-14,3,0,-2,19,1)              ! k には-14が代入されます
c = min("abc","defghij","xyz")        ! c には"abc    " が代入されます。
```

2.352 MINEXPONENT組込み関数

MINEXPONENT 関数は、*X* と同じ型および種別型パラメタの数体系で最小の指数を返却します。

分類

要素別処理関数

形式

result = MINEXPONENT (*X*)

X

実数型でなければなりません。

result

基本整数型スカラです。

機能説明

MINEXPONENT は、*X* と同じ型および種別型パラメタの数体系で最小の指数を返却します。

関数の結果の型は、基本整数型です。

以下の値の固定値となります。

引数の型	結果の値
半精度実数型	-13
単精度実数型	-125
倍精度実数型	-1021
4倍精度実数型	-16381

使用例

```
real :: r
integer :: i
i = minexponent (r)        ! i には-125が代入されます
```

2.353 MINLOC組込み関数

MINLOC 関数は、配列内で最小値となる要素の添字値を調べます。

分類

変形関数

形式

result = MINLOC (*ARRAY* [, *MASK* , *KIND* , *BACK*])　　または
result = MINLOC (*ARRAY*, *DIM* [, *MASK* , *KIND* , *BACK*])

ARRAY

整数型、実数型または文字型でなければなりません。スカラであってはなりません。

DIM

整数型スカラであって、1 <= *DIM* <= n の範囲の値でなければなりません。ここで、n は *ARRAY* の次元数とします。対応する実引数は、省略可能な仮引数であってはなりません。

MASK（省略可能）

論理型でなければならず、*ARRAY* と形状適合しなければなりません。

KIND（省略可能）

スカラ整数定数式でなければなりません。

BACK（省略可能）

論理型スカラでなければなりません。

result

整数型です。*KIND* を指定した時、種別型パラメタは *KIND* に従います。*KIND* を省略したとき、種別型パラメタは基本整数型になります。*DIM* が省略された場合は、大きさnの1次元配列となります。n は *ARRAY* の次元数です。*DIM* が指定された場合は、*ARRAY* が1次元配列の場合はスカラ、それ以外の場合は、*ARRAY* の形状を (d_1, d_2, ..., d_n) としたとき、
(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_n) となります。

機能説明

MINLOC は、*ARRAY* の第 *DIM* 次元の中でもっとも小さい値となる要素の添字を調査します。

－ *DIM*を省略している場合

*MASK*を省略すると、要素の値が真であるとして、以下の処理が行われます。
*MASK*中の真の要素に対応する、*ARRAY*の全要素の最小値が現れる各次元の添字値を要素とする1次元配列を返却します。
*ARRAY*の大きさが0または*MASK*の全要素が偽である場合は、すべて0の1次元配列を返却します。

－ *DIM*を指定している場合

*ARRAY*が1次元の場合は、MINLOC(*ARRAY*[,*MASK*]) の要素となります。
*ARRAY*が2次元以上の場合、結果の要素 (S_1, S_2, ..., S_{DIM-1}, S_{DIM+1}, ..., S_n) は、
MINLOC(*ARRAY*(S_1, S_2, ..., S_{DIM-1}, : , S_{DIM+1}, ..., S_n) , DIM = 1[,MASK = *MASK*]) となります。ここで、n は*ARRAY*の次元数、*MASK*はMASK(S_1, S_2, ..., S_{DIM-1}, : , S_{DIM+1}, ..., S_n)とします。

－ *BACK*に偽を指定している、または省略している場合

最初に現れる最小値の各次元の添字値が要素となります。

－ *BACK*に真を指定している場合

最後に現れる最小値の各次元の添字値が要素となります。

関数の結果の型は、種別型パラメタ*KIND*が指定された場合、種別型パラメタ*KIND*の整数型となります。*KIND*が省略された場合、関数の結果の型は基本整数型です。

*ARRAY*が文字型のとき、関数の結果は組込み関係演算子の適用によって選択される値になります。*ARRAY*が大きさゼロであるとき、結果のすべての要素はゼロになります。また、*MASK*のすべての要素の値が偽であるとき，結果のすべての要素はゼロになります。

使用例

```
integer, dimension(1) :: i
i = minloc ((/3,0,4,4/))          ! i には(/2/) が代入されます
```

2.354 MINVAL組込み関数

MINVAL 関数は、配列内の最小の値を求めます。

分類

変形関数

形式

result = MINVAL (*ARRAY* [, *MASK*])　　または
result = MINVAL (*ARRAY* , *DIM* [, *MASK*])

ARRAY

整数型、実数型または文字型でなければなりません。スカラであってはなりません。

DIM

整数型スカラであって、1 <= *DIM* <= n の範囲の値でなければなりません。ここで、n は*ARRAY*の次元数とします。対応する実引数は、省略可能な仮引数であってはなりません。

MASK（省略可能）

論理型でなければならず、*ARRAY*と形状適合しなければなりません。

result

*ARRAY*と同じ型および同じ種別型パラメタです。*DIM*が省略されている、または*ARRAY*が1次元配列の場合は、
スカラとなります。それ以外の場合は、*ARRAY*の形状を (d_1, d_2, ..., d_n) としたとき、
(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_n) となります。n は*ARRAY*の次元数です。

機能説明

MINVAL は、*ARRAY*の第*DIM*次元の中でもっとも小さい値を調査します。

ー *DIM*を省略している場合

*MASK*中の真の要素に対応する、*ARRAY*の全要素中の最小値を返却します。*ARRAY*の大きさが0または*MASK*中の全要素が偽である場合、HUGE(*ARRAY*)を返却します。*ARRAY*が大きさゼロで文字型であるときの結果の値は、引数の長さと等しく、各文字は組込み関数 CHAR(255,KIND=1) になります。

ー *DIM*を指定している場合

*ARRAY*が1次元の場合は、MINVAL(*ARRAY*[,*MASK*]) となります。
*ARRAY*が2次元以上の場合、結果の要素 (S_1, S_2, ..., S_{DIM-1}, S_{DIM+1}, ..., S_n) は、
MINVAL(*ARRAY* (S_1, S_2, ..., S_{DIM-1}, : , S_{DIM+1}, ..., S_n), DIM = 1[,MASK =*MASK*]) となります。ここで、n は*ARRAY*の次元数、*MASK*はMASK(S_1, S_2, ..., S_{DIM-1}, :, S_{DIM+1}, ..., S_n) とします。

関数の結果の型は、*ARRAY*と同じです。

*ARRAY*が文字型のとき、関数の結果は組込み関係演算子の適用によって選択される値になります。すなわち、引数の種別型パラメタによる文字の大小順序が適用されます。

使用例

```
integer :: j(2)
integer, dimension (2,2) :: m = reshape((/1,2,3,4/),(/2,2/))
                            ! m は     ┌     ┐
                            !          | 1 3 |
                            !          | 2 4 |
                            !          └     ┘
i = minval(m)               ! i には1が代入されます
j = minval(m,dim=1)         ! j には(/1,3/) が代入されます
k = minval(m,mask=m<3)      ! k には1が代入されます
```

2.355 MOD組込み関数

MOD 関数は、余りを求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
MOD	-----	2	1バイトの整数型 , 1バイトの整数型	1バイトの整数型
	I2MOD		2バイトの整数型 , 2バイトの整数型	2バイトの整数型
	IMOD		2バイトの整数型 , 2バイトの整数型	2バイトの整数型
	MOD		4バイトの整数型 , 4バイトの整数型	4バイトの整数型
	JMOD		4バイトの整数型 , 4バイトの整数型	4バイトの整数型
	-----		8バイトの整数型 , 8バイトの整数型	8バイトの整数型
	-----		実数型 , 実数型	実数型
	AMOD		単精度実数型 , 単精度実数型	単精度実数型
	DMOD		倍精度実数型 , 倍精度実数型	倍精度実数型

総称名	個別名	引数の数	引数の型	結果の型
	QMOD		4倍精度実数型 , 4倍精度実数型	4倍精度実数型

result = MOD (*A* , *P*)

A

整数型または実数型でなければなりません。

P

A と同じ型および種別型パラメタでなければなりません。0であってはなりません。

result

A と同じ型です。

機能説明

MOD、I2MOD、IMOD、JMOD、AMOD、DMOD、およびQMOD は、余りを求めます。

結果の値は、*A* - INT(*A* / *P*) × *P* となります。

P は0であってはなりません。

総称名MOD は、すべての整数型および実数型の引数に使用することができます。

それぞれの関数の結果の型は、*A* と同じです。

使用例

```
r = mod(23.0,4.0)       ! r には3.0が代入されます
i = mod(-23,4)          ! i には-3が代入されます
j = mod(23,-4)          ! j には3が代入されます
k = mod(-23,-4)         ! k には-3が代入されます
```

2.356 MODULE文

MODULE 文は、モジュールを開始します。モジュールについては、“1.11.2 モジュール”を参照してください。

MODULE 文は、以下の形式です。

```
MODULE module-name
```

module-name は、モジュール名です。モジュール名は、プログラム中で大域的であり、そのプログラム中の他のプログラム単位、外部手続、または共通ブロックの名前と同じであってはなりません。また、そのモジュール中のどの局所要素とも、同じ名前であってはなりません。

MODULE 文の例:

```
module mod
  implicit none
  type mytype
    real :: a,b(2,4)
    integer :: n,o,p
  end type mytype
end module

subroutine zee( )
  use mod
  implicit none
  type(mytype) :: bee,dee
    ...
end subroutine zee
```

2.357 MODULE PROCEDURE副プログラム文

MODULE PROCEDURE 副プログラム文は、MODULE PROCEDURE 副プログラムを開始します。MODULE PROCEDURE 副プログラムについては、“1.11.5 MODULE PROCEDURE副プログラム”を参照してください。

MODULE PROCEDURE 副プログラム文は、以下の形式です。

```
MODULE PROCEDURE procedure-name
```

*procedure-name*は手続名です。

MODULE PROCEDURE 副プログラム文の例:

```
module mod
  interface
    module subroutine sub1(val)  ! 分離引用仕様本体
      integer, intent(in) :: val
    end subroutine sub1

    module subroutine sub2(val)  ! 分離引用仕様本体
      integer, intent(in) :: val
    end subroutine sub2
  end interface
end module

submodule(mod)submod
  contains
    module subroutine sub1(val)   ! 分離モジュール手続
      integer, intent(in) :: val  ! 特性および仮引数名は上記のsub1と一致させる
    end subroutine sub1

    module procedure sub2         ! MODULE PROCEDURE 文を使用すると
    end procedure sub2            ! 特性および仮引数名を省略できる
end submodule submod
```

2.358 MODULO組込み関数

MODULO 関数は、剰余を求めます。

分類

要素別処理関数

形式

result = MODULO (*A* , *P*)

A

整数型または実数型でなければなりません。

P

A と同じ型および種別型パラメタでなければなりません。0であってはなりません。

result

A と同じ型です。

機能説明

MODULO は、剰余を求めます。

－ *A* が整数型の場合

P ≠ 0 のとき、*A* = Q × *P* + R となる R を返却します。Q は *P* > 0 のとき、0 <= R < *P* を満たし、*P* < 0 のとき、*P* < R <= 0 を満たします。

－ *A* が実数型の場合

P ≠ 0.0 のとき、*A* - FLOOR(*A* / *P*) × *P*を返却します。

関数の結果の型は、*A* と同じです。

使用例

```
r = modulo(23.0,4.0)          ! r には3.0が代入されます
i = modulo(-23,4)             ! i には1が代入されます
j = modulo(23,-4)             ! j には-1が代入されます
k = modulo(-23,-4)            ! k には-3が代入されます
```

2.359 MOVE_ALLOC組込みサブルーチン

MOVE_ALLOC 組込みサブルーチンは、ある割付け実体から他の割付け実体へ割付けを移動します。

分類

純粋サブルーチン

形式

CALL MOVE_ALLOC (*FROM* , *TO*)

FROM

どの型およびどの次元数であってもよく、割付け実体でなければなりません。INTENT(INOUT) の引数です。

TO

FROM と同じ次元数、同じ型、同じ共次元数の割付け実体でなければなりません。INTENT(OUT) の引数です。*TO* が文字型の場合、その文字長パラメタは、*FROM* の文字長パラメタと同じでなければなりません。

機能説明

FROM がMOVE_ALLOC の入口で割り付けられていないとき、*TO* の割付け状態は、未割付けとなります。それ以外のとき、*TO* は、*FROM* が MOVE_ALLOC の入口でもっている配列上下限、共上下限、および同一の値をもって割り付けられます。

TO がTARGET 属性をもつとき、MOVE_ALLOC の入口で*FROM* と結合しているどのポインタも対応して*TO* と結合します。*TO* が TARGET 属性をもたないとき、入口で*FROM* と結合しているどのポインタの結合状態も不定となります。

FROM の割付け状態は、未割付けになります。

引数が共配列であるMOVE_ALLOCを実行したとき、すべての像が暗黙のうちに同期します。各々の像上で、すべての像が文を実行するまで、この文に続くセグメントの実行は遅延します。

使用例

```
integer, allocatable :: a(:), b(:)
allocate(a(1:10))
allocate(b(1:100))
call move_alloc(b, a)           ! a は解放され、b のもつ配列上下限で割り付けられます
                                ! b は未割付けになります
```

2.360 MVBITS組込みサブルーチン

MVBITS サブルーチンは、ビット列を複写します。

分類

要素別処理サブルーチン

形式

CALL MVBITS (*FROM* , *FROMPOS* , *LEN* , *TO* , *TOPOS*)

FROM

整数型でなければなりません。INTENT(IN) の引数です。

FROMPOS

整数型でなければなりません。INTENT(IN) の引数です。

FROMPOS + *LEN* <= BIT_SIZE(*FROM*) でなければなりません。

FROMPOS >= 0 でなければなりません。

LEN

整数型でなければなりません。INTENT(IN) の引数です。*LEN* >= 0 でなければなりません。

TO

FROMと同じ種別型パラメタ値をもつ整数型でなければなりません。INTENT(INOUT) の引数です。*FROM*の*FROMPOS*番目のビットから*LEN*ビット分を*TO*の*TOPOS*の位置へ複写します。

TOPOS

整数型でなければなりません。INTENT(IN) の引数です。
TOPOS + *LEN* <= BIT_SIZE(*TO*) でなければなりません。*TOPOS* >= 0 でなければなりません。

使用例

```
i = 17
j = 3
call mvbits(i,3,3,j,1)          ! j は5になります
```

2.361 NAMELIST文

NAMELIST 文は、入出力文において、1つの名前で参照できる名前付きデータ実体群を指定します。

NAMELIST 文は、以下の形式です。

```
NAMELIST / namelist-group-name / namelist-group-object-list ■
■ [ [ , ] / namelist-group-name / namelist-group-object-list ]...
```

namelist-group-name は、変数群名です。

変数群名は、参照結合によって参照可能になった名前であってはなりません。

同じ変数群名を、有効域内で2つ以上のNAMELIST 文に書いてもかまいません。この場合、その変数群要素をその順に1つにつないだ並びとして扱われます。

namelist-group-object-list は、コンマで区切られた変数群要素の並びです。

namelist-group-object は、以下の形式です。

```
variable
```

variable は、変数名です。

変数群要素は、末端成分にPOINTER属性、ALLOCATABLE属性、または長さ型パラメタをもつ派生型であってはなりません。

変数群要素は、参照結合または親子結合によって参照されるか、またはその型、型パラメタおよび形状が、同じ有効域内の宣言文で前もって指定されるか、またはその有効域の暗黙の型規則で決められたものでなければなりません。変数群要素が暗黙の型規則によっているならば、その後の型宣言文は、その暗黙の型および型パラメタを確認するものでなければなりません。

1つの変数群要素は、2つ以上の変数群に含めてもかまいません。

変数群名がPUBLIC 属性をもつとき、変数群要素の各項目は、PRIVATE 属性または非公開の成分をもってはなりません。

NAMELIST 文で指定した変数の順序が、出力時の値の順序になります。

NAMELIST 文の例:

```
namelist /mylist/ x,y,z
```

2.362 NARGSサービス関数

機能説明

実行時コマンドと実行時オプションまたは利用者定義オプションに指定した文字列の個数を返却します。

形式

iy = NARGS ( )

関数結果

基本整数型スカラ。

使用例

```
use service_routines,only:nargs
print *,nargs( )
end
```

2.363 NEAREST組込み関数

NEAREST 関数は、*S*と同符号の無限大の方向で最も*X*に近い値を返却します。

分類

要素別処理関数

形式

result = NEAREST (*X* , *S*)

X

実数型でなければなりません。

S

実数型でなければなりません。0.0であってはなりません。

result

*X*と同じ型です。

機能説明

NEAREST は、*S*と同符号の無限大の方向で最も*X*に近い値を返却します。

関数の結果は、*X* + SIGN(SPACING(*X*),*S*) となり、その型は*X*と同じです。

*X*の絶対値がHUGE(*X*) と等しく*X*と*S*の符号が同じ場合、結果は*X*の符号をもつInf となります。

使用例

```
a = nearest(34.3,-2.0)
```

2.364 NEW_LINE組込み関数

NEW_LINE 関数は、改行文字を返却します。

分類

問合せ関数

形式

result = NEW_LINE (*A*)

A

文字型でなければなりません。スカラまたは配列です。

result

A と同じ種別型パラメタで文字長1の文字型スカラです。

機能説明

結果はCHAR(10,KIND(*A*)) になります。

使用例

```
character(len=50) :: char_string
character(len=1) :: nl
nl = new_line(char_string)            ! nl にはCHAR(10) の値が代入されます
```

2.365 NINT組込み関数

NINT 関数は、実数型データの四捨五入整数化を行います。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
NINT	-----	1 または 2	実数型[, 整数型]	整数型
	NINT	1	単精度実数型	基本整数型
	JNINT		単精度実数型	基本整数型
	IDNINT		倍精度実数型	基本整数型
	JIDNNT		倍精度実数型	基本整数型
	IQNINT		4倍精度実数型	基本整数型
I2NINT	-----	1	実数型	2バイトの整数型
	ININT		単精度実数型	2バイトの整数型
	IIDNNT		倍精度実数型	2バイトの整数型

result = NINT (*A* [, *KIND*])

A

実数型でなければなりません。

KIND（省略可能）

スカラ整数定数式でなければなりません。

result

整数型です。*KIND*が指定された場合、種別型パラメタは*KIND*の指定に従います。*KIND*が省略された場合、種別型パラメタは基本整数型になります。

機能説明

NINT、ININT、IDNINT、JIDNNT、IQNINT、I2NINT、JNINT、およびIIDNNT は、実数型データの四捨五入整数化を行います。

関数の結果は、*A* が0.0または正の場合、結果の値は INT(*A* + 0.5[,*KIND*]) となります。*A* が負の場合は、INT(*A* - 0.5[,*KIND*]) となります。関数の結果が指定された整数型で表現できない場合、結果の値は不定となります。

― NINT、JNINT、IDNINT、JIDNNT、およびIQNINTを使用している場合

総称名NINT は、すべての実数型の引数に使用することができます。
それぞれの関数の結果の型は、種別型パラメタ*KIND*が指定された場合、種別型パラメタ*KIND*の整数型となります。*KIND*が省略された場合、関数の結果の型は基本整数型です。

－ I2NINT、ININT、およびIIDNNTを使用している場合

総称名I2NINT は、すべての実数型の引数に使用することができます。
それぞれの関数の結果の型は、2バイトの整数型となります。

使用例

```
i = nint (7.73)          ! i には8が代入されます
i = nint (-4.2)          ! i には-4が代入されます
i = nint (-7.5)          ! i には-8が代入されます
i = nint (2.50)          ! i には3が代入されます
```

2.366 NORM2組込み関数

NORM2関数は、配列の二乗和平方根を求めます。

分類

変形関数

形式

result = NORM2 (*X* [, *DIM*])

X

実数型の配列でなければなりません。

DIM（省略可能）

整数型スカラであって、1 <= *DIM* <= *n* の範囲の値でなければなりません。ここで、n は*X* の次元数とします。対応する実引数は、省略可能な仮引数であってはなりません。

result

*X*と同じ型および同じ種別型パラメタです。*DIM*が省略された、または*X*が1次元配列の場合は、スカラとなります。それ以外の場合、*X* の形状を (d_1, d_2, ..., d_n) としたとき、形状は(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_n) となります。

機能説明

NORM2関数は、二乗和平方根を求めます。

－ *DIM*を省略している場合

結果はスカラとなります。

－ *DIM*を指定している場合

X が1次元の場合、結果はスカラとなります。

他の場合、結果の要素(S_1, S_2, ..., S_{DIM-1}, S_{DIM+1}, ..., S_n) は、NORM2 (*X*(S_1, S_2, ..., S_{DIM-1}, : , S_{DIM+1}, ..., S_n))となります。ここで、n は*X*の次元数とします。

使用例

```
real:: result
result = norm2 ( [ 3.0, 4.0] )       ! resultには5.0が代入されます
```

2.367 NOT組込み関数

NOT 関数は、引数の論理否定を返す関数です。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
NOT	-----	1	1バイトの整数型	1バイトの整数型
	INOT		2バイトの整数型	2バイトの整数型
	NOT		4バイトの整数型	4バイトの整数型
	JNOT		4バイトの整数型	4バイトの整数型
	-----		8バイトの整数型	8バイトの整数型

result = NOT (*I*)

I

整数型でなければなりません。

result

*I*と同じ型です。

機能説明

NOT、INOT、およびJNOT は、引数の論理否定を返す関数です。

総称名NOT は、すべての整数型の引数に使用することができます。

それぞれの関数の結果の型は、*I*と同じです。

使用例

```
i = not(5)          ! i には-6が代入されます
```

2.368 NULL組込み関数

NULL 関数は、空状態のポインタまたは未割付けの割付けを返却します。

分類

変形関数

形式

result = NULL ([*MOLD*])

MOLD（省略可能）

ポインタまたは割付けでなければなりません。どの型でもかまいません。

MOLD がポインタの場合、ポインタ結合状態は、不定状態、空状態、結合状態のいずれでもかまいません。

MOLD が割付けの場合、割付け状態は、割付け、未割付けのどちらでもかまいません。

result

MOLD が指定されている場合、*MOLD* と同じ型になります。*MOLD* が省略された場合、以下のとおりとなります。

NULL()の出現場所	型および型パラメタ
ポインタ代入の右辺	左辺のポインタ
宣言中の実体の初期化	その実体
成分の暗黙的初期値指定	その成分
構造体構成子	対応する成分
実引数	対応する仮引数
DATA文の中	対応するポインタ実体

機能説明

NULL は、空状態のポインタ、または未割付けの割付けを返却します。

使用例

```
real ,pointer,dimension(:) :: a=>null( )          ! a は未結合状態になります
```

2.369 NULLIFY文

NULLIFY 文は、ポインタの結合状態を空状態にします。

NULLIFY 文は、以下の形式です。

```
NULLIFY ( pointer-object-list )
```

pointer-object-list は、コンマで区切られたポインタ実体の並びです。

pointer-object は、以下の形式です。

```
variable-name              または
structure-component        または
proc-pointer-name
```

variable-name は、変数名です。

structure-component は、構造体成分です。

proc-pointer-name は、手続ポインタ名です。

ポインタ実体は、POINTER 属性をもたなければなりません。

NULLIFY 文の例:

```
real, pointer :: a,b,c
real, target :: t,u,v
a=>t; b=>u; c=>v           ! a、b、およびc は、それぞれt、u、およびv と結合します
nullify (a, b, c)          ! a、b、およびc は、空状態になります
```

2.370 NUM_IMAGES組込み関数

NUM_IMAGES 関数は、像の数を返却します。

分類

変形関数

形式

```
result = NUM_IMAGES ( )
```

result

基本整数型のスカラです。値は像の数です。

使用例

```
k = num_images() ! k に像の数が代入されます
```

2.371 OMP_LIB非標準組込みモジュール

このモジュールは、OpenMP 仕様の名前付き定数および明示的引用仕様を参照可能にします。

2.372 OPEN文

OPEN 文は、外部ファイルと装置を接続したり、接続を修正したりします。

OPEN 文は、以下の形式です。

```
OPEN ( connect-spec-list )
```

connect-spec-list は、コンマで区切られた接続指定子の並びです。

connect-spec は、以下の形式です。

```
[ UNIT = ] external-file-unit          または
IOSTAT = io-stat                       または
ERR = err-label                        または
FILE = file-name-expr                  または
STATUS = status                        または
ACCESS = access                        または
FORM = form                            または
RECL = recl                            または
BLANK = blank                          または
POSITION = position                    または
ACTION = action                        または
DELIM = delim                          または
PAD = pad                              または
TOTALREC = totalrec                    または
BLOCKSIZE = blocksize                  または
CONVERT = convert                      または
ASYNCHRONOUS = asynchronous            または
DECIMAL = decimal                      または
ENCODING = encoding                    または
IOMSG = iomsg                          または
ROUND = round                          または
SIGN = sign                            または
NEWUNIT = newunit
```

UNIT 指定子は、NEWUNIT指定子を指定した場合を除き、指定しなければなりません。文字列‘UNIT=’をUNIT 指定子から省略する場合、UNIT 指定子は、接続指定子並びの最初の項目でなければなりません。

external-file-unit は、外部ファイル装置であり、スカラ整数式でなければなりません。

io-stat は、スカラ整変数です。

IOSTAT 指定子が指定された場合、IOSTAT 指定子に指定された変数には、以下の値が設定されます。

- 誤り条件、ファイル終了条件、および記録終了条件が検出されない場合は、0
- 誤り条件が検出された場合は、1または実行時の診断メッセージの番号

err-label は文番号であり、このOPEN 文と同じ有効域内の飛び先文の文番号でなければなりません。

ERR 指定子が指定され、OPEN 文の実行中に誤り条件が検出された場合、ERR 指定子に指定された文番号をもつ文が次に実行されます。

file-name-expr は、スカラ基本文字式です。

FILE 指定子の値は、指定した装置に接続するファイル名です。後続の空白は無視されます。この指定子を省略し装置がファイルと接続されていない場合、STATUS 指定子には'SCRATCH' を指定しなければなりません。この場合、装置は名前なしファイルまたは実行時の環境変数(“Fortran使用手引書”参照)で指定したファイル名と接続します。

status はスカラ基本文字式であり、その値は'OLD'、'NEW'、'SCRATCH'、'REPLACE'、'UNKNOWN'、または'SHR' でなければなりません。

STATUS 指定子は、接続するファイルの状態を指定します。'OLD' または'SHR' を指定した場合、ファイルは存在しなければなりません。'NEW' を指定した場合、ファイルは存在してはなりません。

STATUS 指定子に'NEW'、'REPLACE'、または'SHR'を指定した場合、FILE 指定子も指定しなければなりません。'SCRATCH'を指定した場合、FILE 指定子を指定してはなりません。'OLD'を指定した場合、装置が現在接続されており、接続先のファイルが存在するとき以外は、FILE 指定子も指定しなければなりません。

STATUS 指定子に'NEW'を指定したOPEN 文の実行が成功すると、ファイルが生成され、状態が'OLD' に変更されます。'REPLACE' を指定し、ファイルがまだ存在しない場合、ファイルが生成され、状態が'OLD' に変更されます。'REPLACE' を指定し、ファイルが存在する場合、そのファイルは削除され、新しいファイルが同じ名前で生成され、状態が'OLD' に変更されます。
'SCRATCH'を指定した場合、ファイルが生成され、プログラムで使えるように装置と接続されますが、同じ装置を参照するCLOSE 文の実行、またはプログラムの終了によって削除されます。

STATUS 指定子に'UNKNOWN' が指定された場合、FILE 指定子が指定されており、そのファイルが存在しない場合は、'NEW' を指定したのと同じです。FILE 指定子が指定されており、そのファイルが存在する場合は、'OLD' が指定されたのと同じです。FILE 指定子が指定されておらず、実行時の環境変数による割当て(“Fortran使用手引書”参照)がない場合は、'SCRATCH' が指定されたのと同じです。FILE 指定子が指定されておらず、実行時の環境変数による割当てがある場合は、'OLD' を指定したのと同じです。

STATUS 指定子を省略すると、'UNKNOWN' が想定されます。

access はスカラ基本文字式であり、その値は'SEQUENTIAL'、'DIRECT'、'STREAM'、または'APPEND' でなければなりません。

ACCESS 指定子は、ファイルの接続の探査方法を指定します。'SEQUENTIAL' が指定された場合、順番探査として接続します。'DIRECT' が指定された場合、直接探査として接続します。'DIRECT'を指定した場合、RECL 指定子も指定しなければなりません。'STREAM' が指定された場合は、流れ探査として接続します。'APPEND' の指定は、POSITION 指定子に'APPEND' を指定したのと同じです。ACCESS 指定子を省略すると、'SEQUENTIAL' が想定されます。

form はスカラ基本文字式であり、その値は'FORMATTED'、'UNFORMATTED'、または'BINARY' でなければなりません。

FORM 指定子は、接続するファイルの記録形式を指定します。'FORMATTED' を指定した場合、このファイルに入出力する記録は、書式付きFortran 記録、並びによるFortran 記録、または変数群Fortran 記録でなければなりません。'UNFORMATTED' を指定した場合、このファイルに入出力する記録は、書式なしFortran 記録でなければなりません。'BINARY' を指定した場合、このファイルに入出力する記録は、BINARY Fortran 記録でなければなりません。FORM 指定子を省略すると、ファイルが直接探査として接続されるならば'UNFORMATTED'、順番探査として接続されるならば'FORMATTED' が想定されます。

recl はスカラ整数式であり、正の値でなければなりません。

RECL 指定子は、直接探査として接続されるファイルの場合、その記録の長さを指定し、順番探査として接続される場合、その記録の最大の長さを指定します。ファイルが流れ探査として接続される場合、この指定子を書いてはなりません。長さの単位は、バイトです。RECL 指定子は、ファイルが直接探査として接続される場合、必ず指定しなければなりません。ファイルが順番探査として接続され、RECL 指定子を省略すると、記録の長さとして2147483647が想定されます。

blank はスカラ基本文字式であり、その値は'NULL' または'ZERO' でなければなりません。

BLANK 指定子は、書式付き入出力として接続されるファイルに対してだけ指定できます。'NULL' を指定すると、装置上の数値書式の入力欄中の空白は、すべて無視されます。'ZERO' を指定すると、先行する空白以外の空白は、すべて0とみなされます。BLANK 指定子を省略すると、'NULL' が想定されます。

position はスカラ基本文字式であり、その値は'ASIS'、'REWIND'、または'APPEND' でなければなりません。

POSITION 指定子を指定する場合、ファイルの接続は、順番探査または流れ探査でなければなりません。

POSITION 指定子は、ファイル位置を指定します。それまでにファイルが存在しなかった場合、その始点に位置付けられます。'REWIND' を指定すると、存在するファイルの始点に位置付けられます。存在するファイルにファイル終了記録が存在する場合に'APPEND' を指定すると、ファイル終了記録が直後記録となるように位置付けられます。存在するファイルにファイル終了記録が存在しない場合に'APPEND' を指定すると、終点に位置付けられます。ファイルが存在し、すでに接続されている場合に'ASIS' を指定すると、位置は変わりません。ファイルが存在していても接続されていない場合、'ASIS' の位置付けは無視されます。POSITION 指定子を省略すると、'ASIS' が想定されます。

action はスカラ基本文字式であり、その値は'READ'、'WRITE'、'READWRITE'、または'BOTH' でなければなりません。

ACTION 指定子は、この接続に許される入出力の種類を指定します。'READ' は、WRITE 文、PRINT 文、またはENDFILE 文が、この接続を参照してはならないことを指定します。'WRITE' は、READ 文がこの接続を参照してはならないことを指定します。'READWRITE' はすべての入出力文がこの接続を参照できることを指定します。'BOTH' の指定は、'READWRITE' の指定と同じです。ACTION 指定子を省略すると、'READWRITE' が想定されます。

delim はスカラ基本文字式であり、その値は'APOSTROPHE'、'QUOTE'、または'NONE' でなければなりません。

DELIM 指定子は、書式付き入出力として接続されるファイルに対してだけ指定することができます。書式付き記録の入力においては、DELIM 指定子は無視されます。

DELIM 指定子は、出力される文字の値の囲み記号を指定します。'APOSTROPHE' を指定すると、並び書式または変数群書式によって書かれる文字の値を囲むのに、アポストロフィ‘’’が用いられます。文字列中の1つのアポストロフィ‘’’は、連続する2つのアポストロフィ‘’’で表現されます。'QUOTE' を指定すると、並び書式または変数群書式によって書かれる文字の値を囲むのに、引用符‘"’が用いられます。文字列中の1つの引用符‘"’は、連続する2つの引用符‘""’で表現されます。'NONE' を指定すると、書かれる文字の値は、囲み記号はなく、そのまま出力されます。DELIM 指定子を省略すると、'NONE' が想定されます。

pad はスカラ基本文字式であり、その値は'YES' または'NO' でなければなりません。

PAD 指定子は、書式付き入出力として接続されるファイルに対してだけ指定することができます。書式付き記録の出力においては、PAD 指定子は無視されます。

PAD 指定子は、書式付き入力記録に空白を補うかどうかを指定します。'YES' を指定した場合に、入力項目並びが指定され、書式仕様が記録に含まれるより多くのデータを要求すると、書式付き入力記録に空白が補われます。'NO' を指定した場合には、入力項目並びおよび書式仕様は、記録に含まれるより多くの文字を要求してはなりません。

totalrec はスカラ整数式であり、正の値でなければなりません。

TOTALREC 指定子は、直接探査として接続されるファイルに対してだけ指定することができます。ファイルがすでに存在している場合、TOTALREC 指定子は無視されます。

TOTALREC 指定子は、直接探査で接続されるファイルを新しく生成する場合の記録の総数を指定します。TOTALREC 指定子を省略すると、2147483647が想定されます。

blocksize はスカラ整数式であり、正の値でなければなりません。

BLOCKSIZE 指定子は、順番探査、または流れ探査として接続されるファイルに対して、指定することができます。

BLOCKSIZE 指定子は、順番探査入出力、または流れ探査入出力のバッファサイズをバイト数で指定します。BLOCKSIZE指定子を省略した場合に想定される値は、“Fortran使用手引書”を参照してください。

convert は、スカラ基本文字式であり、その値は'LITTLE_ENDIAN'、'BIG_ENDIAN'、または'NATIVE' でなければなりません。

CONVERT 指定子は、書式なしとして接続されるファイルに対してだけ指定することができます。

CONVERT 指定子は、書式なし入出力で扱うデータの形式を指定します。'LITTLE_ENDIAN' を指定すると、接続されたファイルの書式なしFortran 記録は、リトルエンディアンデータの形式として入出力されます。'BIG_ENDIAN' を指定すると、接続されたファイルの書式なしFortran 記録は、ビッグエンディアンデータの形式として入出力されます。'NATIVE' が指定されると、データの変換は行われません。CONVERT 指定子が省略された場合、'NATIVE' が想定されます。

asynchronous は、スカラ基本文字式でなければなりません。

ASYNCHRONOUS 指定子は、その入出力文が同期か非同期かを指定します。*asynchronous* の値は、'YES' または'NO' でなければなりません。'YES' を指定した場合、その装置に非同期の入出力が行えることを指定します。'NO' を指定した場合、その装置に非同期の入出力を行えないことを指定します。この指定子を省略すると、'NO' が指定されたとみなします。

decimal は、スカラ基本文字式でなければなりません。*decimal* の値は、'COMMA' または'POINT'でなければなりません。DECIMAL 指定子は、書式付き入出力として接続されるファイルに対してだけ指定できます。この指定子は、この接続での小数編集モードの現在の値を指定します。このモードは、変更可能なモードとします。接続を開始するOPEN 文でこの指定子を省略すると、'POINT' が指定されたものとみなします。

encoding は、スカラ基本文字式でなければなりません。*encoding* の値は、'DEFAULT' でなければなりません。ENCODING 指定子は、書式付き入出力として接続されるファイルに対してだけ指定できます。この指定子を省略すると、'DEFAULT'が指定されたとみなします。

iomsg はスカラ基本文字変数でなければなりません。入出力文の実行中に誤り条件が発生した場合、*iomsg* には説明のためのメッセージが代入されます。

誤り条件が発生しない場合、*iomsg* の値は変更されません。

round は、スカラ基本文字式であり、その値は、'UP'、'DOWN'、'ZERO'、'NEAREST'、'COMPATIBLE'、または'PROCESSOR_DEFINED' でなければなりません。

ROUND 指定子は、書式付き入出力として接続されるファイルに対してだけ指定できます。この指定子は、この接続に対する入出力丸めモードの現在の値を指定します。このモードは、変更可能なモードです。

sign は、スカラ基本文字式でなければなりません。*sign* の値は、'PLUS'、'SUPPRESS'、または'PROCESSOR_DEFINED' のいずれかです。

SIGN 指定子は、書式付き入出力として接続されるファイルに対してだけ指定できます。この指定子は、接続に対する符号モードの現在の値を指定します。このモードは、変更可能なモードです。接続を開始するOPEN 文でこの指定子を省略すると、'PROCESSOR_DEFINED' が指定されたものとみなします。

*newunit*はスカラ変数です。OPEN文の実行が正常に終了した場合、本処理系が決定したNEWUNIT値が*newunit*に設定されます。NEWUNIT 指定子を指定した場合、UNIT指定子は指定してはなりません。

NEWUNIT 指定子を指定した場合、FILE指定子またはSCRATCHの値をもつSTATUS指定子を指定しなければなりません。

OPEN 文は、存在するファイルを装置に接続するか、事前接続されているファイルを生成するか、ファイルを生成しそれに接続するか、またはファイルと装置との接続の指定子を変更するのに使用します。

装置に接続されるファイルが、その装置に接続されているファイルと同一である場合、BLANK指定子、DELIM 指定子、PAD 指定子、ERR 指定子、およびIOSTAT 指定子だけは、現在有効になっている値と異なってもかまいません。

ファイルがある装置と接続している場合、そのファイルを異なる装置に接続するOPEN 文を実行してはなりません。

OPEN 文の例:

```
open (10, file='info.dat',status='new')
```

2.373 OPTIONAL文

OPTIONAL 文は、手続の引用において、仮引数が必ずしも実引数と結合しなくてもよいことを指定します。

OPTIONAL 文は、以下の形式です。

```
OPTIONAL [ :: ] dummy-arg-name-list
```

dummy-arg-name-list は、コンマで区切られた仮引数名の並びです。

OPTIONAL 文は、副プログラムまたは引用仕様本体の宣言部にだけ指定できます。

OPTIONAL 文の例:

```
subroutine sub(a,b)
optional :: b                ! 仮引数b は、実引数と結合しなくてもよい
```

2.374 PACK組込み関数

PACK 関数は、多次元配列の1次元化を行います。

分類

変形関数

形式

result = PACK (*ARRAY* , *MASK* [, *VECTOR*])

ARRAY

どの型でもかまいません。スカラであってはなりません。

MASK

論理型でなければなりません。*ARRAY*と形状適合しなければなりません。

VECTOR（**省略可能**）

*ARRAY*と同じ型および同じ種別型パラメタでなければなりません。1次元配列でなければなりません。*MASK* 中の真の要素と同数以上の要素をもたなければなりません。
MASK がスカラであってその値が真のときは、*ARRAY*の要素数以上の要素をもたなければなりません。

result

*ARRAY*と同じ型および同じ種別型パラメタの1次元配列です。*VECTOR* が指定された場合は、*VECTOR* と同じ大きさになります。
VECTOR が省略された場合は、*MASK* 中の真の要素数と等しい大きさになります。

ただし、*MASK* が真の値のスカラのとき、結果の大きさは*ARRAY*の大きさになります。

機能説明

PACK は、配列の要素を*MASK* に従って1次元配列に分布させます。

- *VECTOR*を省略している場合

  MASK 中の真の要素に対応する*ARRAY* の値を先頭から1次元配列としたものを結果の値とします。

- *VECTOR*を指定している場合

  VECTOR の要素数が*MASK* 中の真の値の総数よりも多い場合は、*VECTOR* の先頭から、*MASK* 中の真の要素に対応する*ARRAY* の値を複写したものを結果の値とします。

使用例

```
integer, dimension(3,3) :: c
integer, dimension(6) :: d
integer, dimension(9) :: e
c = reshape((/0,3,2,4,3,2,5,1,2/),(/3,3/))
                              ! c は      ┌       ┐
                              !           │ 0 4 5 │
                              !           │ 3 3 1 │
                              !           │ 2 2 2 │
                              !           └       ┘
d = pack(c,mask=c.ne.2)       ! d には(/0,3,4,3,5,1/) が代入されます
e = pack(c,mask=.true.)       ! e には(/0,3,2,4,3,2,5,1,2/) が代入されます
```

2.375 PARAMETER文

PARAMETER 文は、名前付き定数を宣言します。

PARAMETER 文は、以下の形式です。

```
PARAMETER ( named-constant-def-list )
```

named-constant-def-list は、コンマで区切られた名前付き定数定義並びです。

named-constant-def は、以下の形式です。

```
constant-name = init-expr
```

constant-name は、定義される名前付き定数です。

init-expr は、定数式です。

名前付き定数は、型、型パラメタ、および形状をもち、それらはその宣言部で前もって宣言するか、または暗黙の型規則に従うかしなければなりません。名前付き定数の型が暗黙の型規則によっているならば、その後の宣言部での宣言は、その暗黙の型および型パラメタを確認するものでなければなりません。

それぞれの名前付き定数は、組込み代入規則(“2.1 代入文”参照)に従って、等号の右辺の定数式によって与えられる値で確定になります。

PARAMETER 文の例:

```
real :: freezing_point , conv_factor
parameter (freezing_point = 32.0 , conv_factor = 9./5.)
```

2.376 PARITY組込み関数

PARITY関数は、*MASK*の第*DIM*次元の要素に真が奇数個あるかどうかを判定します。

分類

変形関数

形式

result = PARITY (*MASK* [, *DIM*])

MASK

論理型の配列でなければなりません。

DIM (**省略可能**)

整数型スカラであって、1 <= *DIM*<= *n*の範囲の値でなければなりません。ここで、nは*MASK*の次元数とします。対応する実引数は、省略可能な仮引数であってはなりません。

result

*MASK*と同じ型および同じ種別型パラメタです。*DIM*が省略された、または、*MASK*が1次元配列の場合は、スカラとなります。他の場合、*MASK* の形状を (d_1, d_2, ..., d_n) としたとき、形状は、(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_n) となります。

機能説明

PARITYは、*MASK*の第*DIM*次元の要素に真が奇数個あるかどうかを判定します。

- *DIM*を省略している場合

  結果はスカラとなります。

- *DIM*を指定している場合

  MASK が1次元の場合は、結果はスカラとなります。
  他の場合、結果の要素(S_1, S_2, ..., S_{DIM-1}, S_{DIM+1}, ..., S_n) は、PARITY (*MASK*(S_1, S_2, ..., S_{DIM-1}, : , S_{DIM+1}, ..., S_n))となります。ここで、n は *MASK* の次元数とします。

使用例

```
logical,parameter::T=.true., F=.false.
logical:: result
result = parity ( [ T,T,F,T] ) ! resultには .true. が代入されます
```

2.377 PAUSE文(廃止事項)

PAUSE 文は、プログラムの実行の一時中断を指定します。

PAUSE 文は、以下の形式です。

PAUSE [*stop-code*]

stop-code は終了符号であり、以下の形式です。

scalar-char-constant または
digit [*digit* [*digit* [*digit* [*digit*]]]]

scalar-char-constant は、スカラ基本文字定数です。

digit は、数字です。

PAUSE 文を実行すると、プログラムの実行が一時中断され、標準エラー出力ファイルに対して診断メッセージが出力されます。終了符号の指定があれば、その終了符号も出力されます。メッセージが端末に出力され、かつ標準入力が端末の場合は、応答待ちになり、プログラムの実行を中断します。それ以外の場合は、PAUSE 文を無視し、プログラムの実行を継続します。

応答待ちのプログラムの実行を再開させるには、標準入力に対して任意のデータを入力してください。

PAUSE 文の例:

```
pause "pausing"
```

2.378 PERRORサービスサブルーチン

機能説明

最後に検出されたシステムエラーに続いて、指定した文字列を標準エラー出力ファイルに出力します。

形式

CALL PERROR (*string*)

string

基本文字型スカラ。

使用例

```
      use service_routines,only:perror
      open(10,file='x.dat',err=10)
      write(10,*,err=10) 'Fortran program'
      stop
10    call perror('Fortran i/o error')
      stop 200
      end
```

2.379 POINTER文

POINTER 文は、POINTER 属性をもつ実体および手続要素に対して、POINTER 属性を宣言します。

POINTER 文は、以下の形式です。

POINTER [::] *pointer-dcl-list*

pointer-dcl-list はポインタ宣言並びです。ポインタ宣言は以下の形式です。

object-name [(*deferred-shape-spec-list*)]　　または
procedure-entity-name

object-name はPOINTER 属性をもつ実体の名前です。

deferred-shape-spec-list は無指定上下限の並びです。*deferred-shape-spec* は以下の形式(コロンのみ)です。

:

ポインタは、データポインタまたは手続ポインタです。

ポインタは、ポインタ代入またはALLOCATE 文の実行の結果として引用可能または確定可能な指示先実体と結合されない限り、引用または確定してはなりません。

データポインタが結合され、ポインタに無指定型パラメタがあれば、その値は、指示先の対応する型パラメタの値になります。

手続ポインタは、指示先の手続とポインタ結合されない限り、引用してはなりません。

object-name のDIMENSION 属性が有効域内の他の場所で指定されている場合、その配列形状指定は、無指定上下限並びでなければなりません。

object-name はPARAMETER 属性をもってはなりません。

procedure-entity-name は、手続要素名であり、EXTERNAL 属性を明示的に指定して宣言されたものでなければなりません。

POINTER 文の例:

```
real :: next,previous,value
pointer :: next,previous
```

2.380 POINTER文（CRAY 仕様）

POINTER 文（CRAY 仕様）は、アドレス保持変数と、そのアドレスによって指されるポインタ変数の組を宣言します。

POINTER 文（CRAY 仕様）は、以下の形式です。

```
POINTER ( ptr , var ) [ , ( ptr , var ) ]...
```

ptr は、アドレス保持変数であり、暗黙的に、8バイトの整数型になります。スカラ変数でなければなりません。*ptr* は型宣言文に指定することはできません。

*var*は、ポインタ変数名です。*var*は、仮引数名、共通ブロック実体の名前、結合実体の名前、SAVE 属性をもつ実体名、変数群要素の名前、関数結果の名前、割付け変数の名前であってはなりません。*var* は、DO 変数およびASSIGN 文のスカラ変数に指定してはなりません。宣言部に指定した*var* は名前、型、型パラメタ、および属性を宣言するだけであり、*var* に対応する記憶域は確保されません。

POINTER 文は、アドレス保持変数*ptr*の指す領域を、ポインタ変数*var*の名前、型、型パラメタ、および属性をもつ変数として引用することを宣言します。ポインタ変数*var*を、さらにアドレス保持変数として宣言することはできません。ポインタ変数が配列である場合、形状明示配列でなければなりません。

アドレス保持変数に実際のアドレス値を設定するために、組込み関数LOC（“2.326 LOC組込み関数”参照）またはサービス関数MALLOC（“2.340 MALLOCサービス関数”参照）を使用することができます。また、サービス関数MALLOC によって確保した記憶域は、サービスサブルーチンFREE（“2.194 FREEサービスサブルーチン”参照）によって解放することができます。

POINTER 文（CRAY 仕様）の例:

```
real ,dimension(10) :: var
real ,dimension(20) :: array
pointer (ptr,var)
ptr = loc(array)              ! アドレス保持変数ptr にarray の先頭番地を設定
var(1) = 1.0                  ! var の引用はarray の引用に対応し
print *,array(1)              ! array(1) は、値1.0で確定している
end
```

2.381 POPCNT組込み関数

POPCNT 関数は、1であるビットの数を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
POPCNT	-----	1	整数型	基本整数型

result = POPCNT (*I*)

I

整数型でなければなりません。

result

基本整数型でなければなりません。

機能説明

*I*のビット列の中の1のビットの数を求めます。

関数の結果の型は、基本整数型です。

使用例

```
k = popcnt(9) ! k には2が代入されます
```

2.382 POPPAR組込み関数

POPPAR 関数は、0と1でのパリティを求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
POPPAR	-----	1	整数型	基本整数型

result = POPPAR (*I*)

I

整数型でなければなりません。

result

基本整数型でなければなりません。

機能説明

POPCNT(*I*) の結果が奇数の場合、値1、偶数の場合、値0になります。

関数の結果の型は、基本整数型です。

使用例

```
k = poppar(2)          ! k には1が代入されます
n = poppar(3)          ! n には0が代入されます
```

2.383 PRECFILLサービスサブルーチン

機能説明

F形、E形、EN形、ES形、D形、G形、およびQ*w.d*形の出力において、実定数表現の有効けた数を超えた部分に出力される文字を決定します。

形式

CALL PRECFILL (*letter*)

letter

長さ1の基本文字型スカラ。出力する文字を指定します。

使用例

```
use service_routines,only:precfill
write(*,'(d24.17)') 1.0d+01
call precfill('*')
write(*,'(d24.17)') 1.0d+01
end
```

2.384 PRECISION組込み関数

PRECISION 関数は、10進精度を求めます。

分類

問合せ関数

形式

result = PRECISION (*X*)

X

実数型または複素数型でなければなりません。スカラまたは配列です。

result

基本整数型スカラです。

機能説明

PRECISION は*X*を数体系の数として表現したときの10進精度の値を返却します。

関数の結果は、INT((DIGITS(*X*)-1) × LOG10(RADIX(*X*))) となり、その型は基本整数型です。

以下の値の固定値となります。

引数の型	結果の値
半精度実数型	3
単精度実数型	6
倍精度実数型	15
4倍精度実数型	33
半精度複素数型	3
単精度複素数型	6
倍精度複素数型	15
4倍精度複素数型	33

使用例

```
i = precision (4.2)          ! i には6が代入されます
```

2.385 PRESENT組込み関数

PRESENT 関数は、省略可能な仮引数の存在の有無を判定します。

分類

問合せ関数

形式

result = PRESENT (*A*)

A

省略可能な仮引数でなければなりません。スカラ、配列、仮手続など、どの型でもかまいません。INTENT 属性はもちません。

result

基本論理型スカラです。

機能説明

PRESENTは、省略可能な仮引数の実体があるかどうかを調査します。*A*が実在するとき真とし、*A*が実在しないとき偽とします。関数の結果の型は、基本論理型スカラです。

使用例

```
call zee(a, b)
contains
  subroutine zee (x,y,z)
    implicit none
    real, intent(inout) :: x, y
    real, intent (in), optional :: z
    logical :: r
```

```
r = present(z)              ! r には偽が代入されます
  ...
```

2.386 PRINT文

PRINT 文は、出力項目並びおよび書式仕様で指定されたデータ要素から、ファイルに値を転送します。

PRINT 文は、以下の形式です。

```
PRINT format [ , output-item-list ]
```

format は書式識別子であり、以下の形式です。

```
default-char-expr                   または
label                               または
*                                   または
scalar-default-int-variable
```

default-char-expr は、基本文字式です。基本文字式の値は、有効な書式仕様でなければなりません。書式仕様については、“1.8.1 書式仕様”を参照してください。

label は文番号です。文番号は、このPRINT 文を含む有効域内のFORMAT 文の文番号でなければなりません。

scalar-default-int-variable は、スカラ基本整変数です。スカラ基本整変数には、ASSIGN 文により、このPRINT 文を含む有効域内のFORMAT 文の文番号が割り当てられていなければなりません。

output-item-list は、コンマで区切られた出力項目の並びです。

output-item は、以下の形式です。

```
expr                or
io-implied-do
```

expr は、式です。

io-implied-do は入出力DO 形反復であり、以下の形式です。

```
(output-item-list , io-implied-do-control )
```

io-implied-do-control は入出力DO 制御であり、以下の形式です。

```
do-variable = scalar-expr , scalar-expr [ , scalar-expr ]
```

do-variable はDO 変数であり、整数型、基本実数型、または倍精度実数型の名前付きスカラ変数でなければなりません。基本実数型および倍精度実数型のDO 変数は廃止事項です。

scalar-expr は、整数型、基本実数型、または倍精度実数型のスカラ式でなければなりません。基本実数型および倍精度実数型の*scalar-expr* は廃止事項です。

入出力DO 形反復における、ループの初期化および実行は、DO 構文(“2.120 DO構文”参照)と同じです。

ほかの入出力DO 形反復を含む入出力DO 形反復のDO 変数は、含まれる方の入出力DO 形反復のDO 変数に現れたり結合されたりしてはなりません。

出力項目がポインタの場合、ポインタは、指示先に結合されていなければならず、データは指示先からファイルに転送されます。

出力項目が割付け変数の場合、その割付け変数は割り付けられていなければなりません。

出力項目として配列を指定した場合、配列要素順序(“1.5.8.3 配列要素順序”参照)の順序で、すべての配列要素が指定されたかのように扱われます。

派生型が末端成分にポインタ成分または割付け成分をもつ場合、この派生型の実体は、出力項目の評価の結果として現れてはなりません。

派生型実体の名前を出力項目並びに指定した場合、そのすべての成分が派生型の定義におけるのと同じ順序で指定されたものとして扱われます。

PRINT 文の例:

```
    print *,"hello world"
    print 100,i,j,k
100 format (3i8)
```

2.387 PRIVATE文

PRIVATE 文は、言語要素が非公開であり、そのモジュール内でだけ参照可能であることを宣言します。PRIVATE 文は、モジュールの宣言部にだけ指定可能です。

PRIVATE 文は、以下の形式です。

```
PRIVATE [ [ :: ] access-id-list ]
```

access-id-list は、コンマで区切られた参照対象の並びです。

access-id は、以下の形式です。

```
use-name                または
generic-spec
```

use-name は、モジュール内で宣言される名前付き変数、手続、派生型、名前付き定数、または変数群の名前でなければなりません。

generic-spec は総称指定であり、以下の形式です。

```
generic-name                        または
OPERATOR ( defined-operator )       または
ASSIGNMENT ( = )                    または
dtio-generic-spec
```

generic-name は、総称名です。

defined-operator は利用者定義演算子であり、以下の形式です。

```
intrinsic-operator                  または
. operator-name .
```

intrinsic-operator は組込み演算子です。

operator-name は、利用者が定義した240文字までの利用者定義演算子の名前です。

ASSIGNMENT(=) は、利用者定義代入です。

dtio-generic-spec は、派生型入出力手続です。以下の形式です。

```
READ ( FORMATTED )                  または
READ ( UNFORMATTED )                または
WRITE ( FORMATTED )                 または
WRITE ( UNFORMATTED )
```

*access-id-list*が省略されたPRIVATE 文は、そのモジュールの暗黙の参照許可属性を非公開とします。*access-id*が指定された場合、指定された言語要素を非公開とします。*access-id-list*が省略されたPRIVATE 文の指定がないモジュールの暗黙の参照許可属性は公開です。

PRIVATE 文は、モジュール内の派生型定義中にも指定できます。派生型定義の中のPRIVATE 文は、その型がPUBLIC 属性をもつ公開の型であっても、その型の成分の名前が、そのモジュール内でだけ参照可能であることを宣言します。同様に、その型の構造体構成子はその型定義のあるモジュール内だけで参照することができます。

派生型定義中のPRIVATE 文は、以下の形式です。

```
PRIVATE
```

派生型定義中のPRIVATE 文については、“1.5.11.1 派生型定義”を参照してください。

PRIVATE 文の例:

```
module ex
implicit none
public                          ! 暗黙の参照許可属性を公開とします。
real :: a,b,c
private a                       ! a は非公開であり、このモジュールの外では参照できません。
                                ! b およびc は公開であり、このモジュールの外でも参照する
                                ! ことができます。
type zee                        ! zee は公開であり、型自身はこのモジュールの外でも参照できますが、
private                         ! PRIVATE 文の指定があるため、構造体構成子、および
integer :: l,m                  ! 成分の引用はこのモジュールの外では利用できません。
end type zee
end module ex
```

2.388 PRNSETサービスサブルーチン

機能説明

精度縮小機能の精度縮小値を指定した値に変更します。

形式

```
CALL PRNSET ( i )
```

i

基本整数型スカラ。0から15までの値を指定します。

注意事項

PRNSET サービスサブルーチンを使用する場合、精度縮小機能が有効になっていなければなりません。精度縮小機能が有効でない場合、PRNSET サービスサブルーチンは無効となります。精度縮小機能については、“Fortran使用手引書”を参照してください。

使用例

```
use service_routines,only:prnset
real :: r4
call prnset(0)
r4 = -1.234567
write(1,*) r4
call prnset(1)
write(1,*) r4
end
```

2.389 PROCEDURE文

PROCEDURE 文には、以下があります。

- 手続成分定義文(“1.5.11.1 派生型定義”参照)
- 個別束縛(“1.5.11.3 型束縛手続”参照)
- 引用仕様部構文
- 手続宣言文(“2.390 手続宣言文”参照)

引用仕様部構文のPROCEDURE 文は、総称指定をもつ引用仕様宣言内に指定することができ、その総称引用仕様をもつ手続を列挙して指定します。

PROCEDURE 文は、以下の形式です。

```
[MODULE] PROCEDURE [ :: ] procedure-name-list
```

procedure-name-list は、コンマで区切られた手続名の並びです。指定された名前は、明示的な引用仕様をもたなければなりません。また、参照可能な手続ポインタ、外部手続、仮手続、モジュール手続、または内部手続でなければなりません。

PROCEDURE 文は、総称指定をもつ引用仕様宣言にだけ指定できます。

PROCEDURE 文にMODULE を指定する場合、その文の各手続名は、現在の有効範囲内でモジュール手続として参照可能でなければなりません。

手続名には、それより前に、同じ総称識別子をもつ参照可能な引用仕様内のPROCEDURE 文で宣言される手続を指定してはなりません。

引用仕様宣言については、“1.12.7.2 手続引用仕様宣言”を参照してください。

PROCEDURE 文の例:

```
module names
  interface bill
    module procedure fred, jim
  end interface
  contains
  function fred(i)
    integer :: i
    ...
  end function fred
  function jim(r)
    real :: r
    ...
  end function jim
end module names
```

2.390 手続宣言文

手続宣言文は、手続ポインタ、仮手続、および外部手続を宣言し、EXTERNAL 属性を宣言します。

手続宣言文は、以下の形式です。

```
PROCEDURE ( [ proc-interface ] ) [ [ , proc-attr-spec ] ... :: ] proc-decl-list
```

proc-interface は、手続引用仕様です。以下の形式です。

```
interface-name                  または
declaration-type-spec
```

interface-name は、引用仕様名です。

declaration-type-spec は、宣言型指定子です。

proc-attr-spec は、手続属性指定です。以下の形式です。

```
access-spec                     または
proc-language-binding-spec      または
INTENT ( intent-spec )          または
OPTIONAL                        または
POINTER                         または
PROTECTED                       または
SAVE
```

access-spec は、参照許可指定子です。

proc-language-binding-spec は、手続言語束縛指定子で以下の形式です。

```
BIND ( C [, NAME = スカラ文字定数式] )
```

intent-spec は、授受特性指定です。

proc-decl-list は、手続宣言並びです。手続宣言は以下の形式です。

procedure-entity-name [=> *proc-pointer-init*]

proc-pointer-init は、以下の形式です。

null-init または
initial-proc-target

procedure-entity-name は、手続ポインタ、仮手続、または外部手続名の手続要素名です。

null-init は、引数のない組込み関数NULL の引用でなければなりません。

initial-proc-target は、初期指示先手続の手続名です。

引用仕様名は、抽象引用仕様の名前または明示的な引用仕様をもつ手続の名前でなければなりません。

引用仕様名が、手続宣言文で宣言される場合、あらかじめ宣言しなければなりません。引用仕様名が、組込み手続を表す場合、以下のいずれかでなければなりません。

ABS、ACOS、AIMAG、AINT、ALOG、ALOG10、AMOD、ANINT、ASIN、ATAN、ATAN2、CABS、CCOS、CEXP、CLOG、CONJG、COS、COSH、CSIN、CSQRT、DABS、DACOS、DASIN、DATAN、DATAN2、DCOS、DCOSH、DDIM、DEXP、DIM、DINT、DLOG、DLOG10、DMOD、DNINT、DPROD、DSIGN、DSIN、DSINH、DSQRT、DTAN、DTANH、EXP、IABS、IDIM、IDNINT、INDEX、ISIGN、LEN、MOD、NINT、SIGN、SIN、SINH、SQRT、TAN、またはTANH です。

引用仕様名は、組込み型を宣言するキーワードと同じであってはなりません。

INTENT 属性またはSAVE 属性を指定する場合、POINTER 属性も指定しなければなりません。

手続引用仕様に要素別処理手続の指定がある場合、それぞれの手続要素名は外部手続を指定するものでなければなりません。

手続宣言に=> が指定される場合、POINTER 属性を指定しなければなりません。

NAME 指定子をもつ手続言語束縛指定子を指定する場合、手続宣言並びには手続宣言をひとつだけ指定しなければなりません。その場合の手続宣言は、POINTER 属性を指定してはならず、仮手続であってもなりません。

手続言語束縛指定子を指定する場合、手続引用仕様を指定しなければならず、その手続引用仕様は引用仕様名でなければならず、引用仕様名は手続言語束縛指定子を指定して宣言しなければなりません。

手続引用仕様が指定され、それが引用仕様名からなる場合、それは宣言される手続または手続ポインタに対する明示的な個別引用仕様を宣言します。その抽象引用仕様は、引用仕様名によって名前付けられた引用仕様で指定されます。

手続引用仕様が指定され、宣言型指定子からなる場合、宣言される手続または手続ポインタが暗黙の引用仕様および指定された結果の型をもつ関数であることを宣言します。外部手続に型が指定される場合、その関数定義には同じ結果の型および種別型パラメタを指定しなければなりません。

手続引用仕様が指定されない場合、手続宣言文は、宣言される手続または手続ポインタがサブルーチンであるか関数であるかを宣言しません。

手続宣言文の手続宣言に“ =>*null-init*”を指定した場合、その手続要素の初期結合状態は空状態です。

手続宣言文の手続宣言に“ => *initial-proc-target*”を指定した場合、その手続要素は初期指示先手続と初期結合します。

手続要素が明示的引用仕様(“1.12.7.1 明示的引用仕様”参照)をもつ場合、その特性は、初期指示先手続と同じでなければなりません。

手続要素と初期指示先手続が関数の場合、これらの関数結果は同じ特性をもたなければなりません。

“=>”の指定は、SAVE 属性をもつことを意味します。手続宣言文におけるSAVE 属性の明示的な指定またはSAVE 文によって再度宣言することもできます。

2.391 PRODUCT組込み関数

PRODUCT 関数は、配列内の積を求めます。

分類

変形関数

形式

result = PRODUCT (*ARRAY* [, *MASK*])　　または
result = PRODUCT (*ARRAY* , *DIM* [, *MASK*])

ARRAY

整数型、実数型、または複素数型でなければなりません。スカラであってはなりません。

DIM

整数型スカラであって、1 <= *DIM*<= n の範囲の値でなければなりません。ここで、n は*ARRAY*の次元数とします。対応する実引数は、省略可能な仮引数であってはなりません。

MASK（省略可能）

論理型でなければならず、*ARRAY*と形状適合しなければなりません。

result

*ARRAY*と同じ型および同じ種別型パラメタです。*DIM*が省略されている、または*ARRAY*が1次元配列の場合は、スカラとなります。それ以外の場合は、*ARRAY*の形状を $(d_1, d_2, ..., d_n)$ としたとき、$(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_n)$ となります。n は*ARRAY*の次元数です。

機能説明

PRODUCT は*ARRAY*の第*DIM*次元の要素の積を計算します。

－ *DIM*を省略している場合

MASK 中の真の要素に対応する、*ARRAY*の全要素の積を返却します。
*ARRAY*の大きさが0または*MASK* 中の全要素が偽である場合、*ARRAY*が整数型または実数型の場合は1、複素数型の場合は (1.0, 0.0) を返却します。

－ *DIM*を指定している場合

*ARRAY*が1次元の場合は、PRODUCT(*ARRAY*[,*MASK*]) となります。
*ARRAY*が2次元以上の場合、結果の要素 $(S_1, S_2, ..., S_{DIM-1}, S_{DIM+1}, ..., S_n)$ は、
PRODUCT(*ARRAY*($S_1, S_2, ..., S_{DIM-1}, :, S_{DIM+1}, ..., S_n$), DIM = 1[,MASK = *MASK*]) となります。ここで、n は*ARRAY*の次元数、MS は、*MASK* がスカラの場合は*MASK*、配列の場合は*MASK*($S_1, S_2, ..., S_{DIM-1}, :, S_{DIM+1}, ..., S_n$) とします。

関数の結果の型は、*ARRAY*と同じです。

使用例

```
integer, dimension (2,2) :: m = reshape((/1,2,3,4/),(/2,2/))
integer :: j(2)
                                  ! m は    ┌     ┐
                                  !         | 1 3 |
                                  !         | 2 4 |
                                  !         └     ┘
i = product(m)                    ! i には24が代入されます
j = product(m,dim=1)              ! j には(/2,12/) が代入されます
k = product(m,mask=m>2)           ! k には12が代入されます
```

2.392 PROGRAM文

PROGRAM 文は、主プログラムを開始します。主プログラムについては、“1.11.1 主プログラム”を参照してください。

PROGRAM 文は、以下の形式です。

PROGRAM *program-name*

program-name はプログラム名です。プログラム名は、プログラム中で大域的であり、そのプログラム中の他のプログラム単位、外部手続、または共通ブロックの名前と同じであってはなりません。また、主プログラム中のどの局所名とも、同じであってはなりません。

PROGRAM 文の例:

```
program main
  ...
end program main
```

2.393 PROMPTサービスサブルーチン

機能説明

標準入力ファイルからデータを入力するとき、指定した促進メッセージを出力します。PROMPTサービスサブルーチンは、TSS で使用した場合だけ、有効となります。

形式

CALL PROMPT (*string*)

string

基本文字型スカラ。出力する促進メッセージを指定します。

文字長150を超えた場合、151文字以上の文字は出力されません。

使用例

```
use service_routines,only:prompt
read(*,*) i
call prompt('>>>')
read(*,*) i
end
```

2.394 PROTECTED文

PROTECTED 文は、モジュールの宣言部で変数または手続ポインタを指定し、その変数または手続ポインタを参照結合した有効範囲で引用できる箇所を限定します。

PROTECTED 文は、以下の形式です。

```
PROTECTED [ :: ] entity-name-list
```

entity-name-list はコンマで区切られた要素名の並びです。

PROTECTED 文は、モジュールの宣言部にだけ指定できます。

PROTECTED 文の例:

```
module mod
integer :: i, j, k
protected :: i          ! 要素名i に、protected 属性を指定しているので、
                        ! モジュールmod の有効域以外では変更できません。
end module
```

2.395 PUBLIC文

PUBLIC 文は、言語要素が公開であり、言語要素が他のプログラム単位からUSE 文によって参照可能であることを宣言します。PUBLIC 文は、モジュールの宣言部にだけ指定可能です。

PUBLIC 文は、以下の形式です。

```
PUBLIC [ [ :: ] access-id-list ]
```

access-id-list は、コンマで区切られた参照対象の並びです。

access-id は、以下の形式です。

use-name または
generic-spec

use-name は、モジュール内で宣言される名前付き変数、手続、派生型、名前付き定数、または変数群の名前でなければなりません。

generic-spec は総称指定であり、以下の形式です。

generic-name または
OPERATOR (*defined-operator*) または
ASSIGNMENT (=) または
dtio-generic-spec

generic-name は、総称名です。

defined-operator は利用者定義演算子であり、以下の形式です。

intrinsic-operator または
. *operator-name* .

intrinsic-operator は組込み演算子です。

operator-name は、利用者が定義した240文字までの利用者定義演算子の名前です。

ASSIGNMENT(=) は、利用者定義代入です。

dtio-generic-spec は、派生型入出力手続です。以下の形式です。

READ (FORMATTED) または
READ (UNFORMATTED) または
WRITE (FORMATTED) または
WRITE (UNFORMATTED)

モジュールの暗黙の参照許可属性は、*access-id-list* が省略されたPRIVATE 文の指定がなければ公開です。*access-id-list* が省略されたPUBLIC 文の指定は、暗黙の参照許可属性が公開であることを確認します。*access-id* が指定された場合、指定された言語要素を公開とします。

PUBLIC 文の例:

```
module zee
  implicit none
  private                ! 暗黙の参照許可属性を非公開とします
  real :: a,b,c
  public a               ! a は公開であり、このモジュールの外でも参照できます
                         ! b およびc は非公開であり、このモジュールの外では参照できません
end module zee
```

2.396 PUTCサービス関数

機能説明

標準出力ファイルに1文字を書き出します。

形式

iy = PUTC (*ch*)

ch

長さ1の基本文字型スカラ。書き込むデータ1文字を設定します。

関数結果

基本整数型スカラ。正常に書き込んだときは0、エラーが発生したときは0以外の値が返却されます。

使用例

```
use service_routines,only:putc
integer :: x
character(len=6) :: pr='input>'
do i=1,6
  if (putc(pr(i:i)) .ne. 0) stop 10
end do
read(*,*) x
end
```

2.397 QEXT組込み関数

QEXT 関数は、4倍精度実数型への変換を行います。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
QEXT	-----	1	1バイトの整数型	4倍精度実数型
	-----		2バイトの整数型	4倍精度実数型
	QFLOAT		4バイトの整数型	4倍精度実数型
	-----		8バイトの整数型	4倍精度実数型
	-----		実数型、複素数型	4倍精度実数型
	QEXT		単精度実数型	4倍精度実数型
	QEXTD		倍精度実数型	4倍精度実数型
	-----		4倍精度実数型	4倍精度実数型
	-----		単精度複素数型	4倍精度実数型
	-----		倍精度複素数型	4倍精度実数型
	QREAL		4倍精度複素数型	4倍精度実数型

result = QEXT (*A*)

A

整数型、実数型、または複素数型でなければなりません。

result

4倍精度実数型です。

機能説明

QEXT、QFLOAT、QEXTD、およびQREAL は、数値型データを4倍精度実数型に変換します。

*A*が整数型または実数型の場合、*A*を4倍精度実数型に変換した値を返却します。*A*が複素数型の場合、*A*の実部を4倍精度実数型に変換した値を返却します。

総称名QEXT は、すべての整数型、実数型および複素数型の引数に使用することができます。

それぞれの関数の結果の型は、4倍精度実数型です。

使用例

```
real(kind=16) :: q
q = qext(2.0)                    ! q には2.0q0 が代入されます
```

2.398 QPROD組込み関数

QPROD 関数は、倍精度実数の4倍精度化乗算を行います。

分類

要素別処理関数

形式

result = QPROD (*X* , *Y*)

X

倍精度実数型でなければなりません。

Y

倍精度実数型でなければなりません。

result

4倍精度実数型です。

機能説明

QPROD は、2つの倍精度実数型データを4倍精度実数型に変換した上で乗算を行います。

関数の結果は、QEXTD(*X*) × QEXTD(*Y*) となります。

関数の結果の型は、4倍精度実数型です。

使用例

```
real(kind=8) :: a,b
real(kind=16) :: c
a = 2.0d0
b = 3.0d0
c = qprod(a,b)              ! c には6.0q0 が代入されます
```

2.399 QSORTサービスサブルーチン

機能説明

1次元配列のクイックソートを行います。

形式

CALL QSORT (*array* , *nel* , *width* , *compare*)

array

任意の型の1次元配列。ソートする領域を指定します。
本サービスサブルーチンの実行終了後は、ソートされた結果が格納されます。

nel

基本整数型スカラ。ソートする1次元配列の要素数を指定します。

width

基本整数型スカラ。*array* の一要素の大きさを指定します。

compare

2バイトの整数型を復帰値とする外部関数名を指定します。
この関数は、*array* の2つの要素を引数とする関数であり、以下の値を復帰値として返却します。

1. 引数1が引数2の前にある場合は、負の値を返却します。
2. 引数1と引数2の順序が同じである場合は、0を返却します。
3. 引数1が引数2の後ろにある場合は、正の値を返却します。

使用例

```
! 第1引数を昇順でソートする例
   use service_routines,only:qsort
   integer(4),dimension(10) :: array
   integer(2),external :: compare4
   array=(/4,6,8,2,10,5,3,1,7,9/)
   write(6,*) array                    ! 4 6 8 2 10 5 3 1 7 9 が出力されます
   call qsort(array,10,4,compare4)
   write(6,*) array                    ! 1 2 3 4 5 6 7 8 9 10 が出力されます
   end
   integer(2) function compare4(i1,i2)
   integer(4) i1,i2
   if(i1-i2) 10,20,30
10 compare4 = -1
   return
20 compare4 = 0
   return
30 compare4 = 1
   return
   end
```

2.400 RADIX組込み関数

RADIX 関数は、基数を求めます。

分類

問合せ関数

形式

result = RADIX (*X*)

X

整数型または実数型でなければなりません。スカラまたは配列です。

result

基本整数型スカラです。

機能説明

RADIX は、*X*を数体系の数として表現したときの基数の値を返却します。

関数の結果の型は、基本整数型です。

常に2が返却されます。

使用例

```
i = radix(2.3)              ! i には2が代入されます
```

2.401 RANサービス関数

機能説明

0.0から1.0までの乱数を発生させます。

形式

y = RAN (*seed*)

seed

基本整数型スカラ。乱数列発生のためシード値を設定します。*seed* の値は、RAN サービス関数実行後、再設定されます。

関数結果

単精度実数型スカラ。新しく発生させられた乱数列の最初の値が関数値として返却されます。

使用例

```
use service_routines,only:ran
real(4) :: x(10)
integer(4) :: seed=123456
do i=1,10
  x(i)=ran(seed) ! 10個の乱数を発生させます
end do
end
```

2.402 RANDサービス関数

機能説明

0.0から1.0までの乱数を発生させます。

形式

y = RAND (*i*)

i

基本整数型スカラ。

関数結果

単精度実数型スカラ。*i* の値に従って、関数結果値が異なります。

```
=0       : 乱数列内の次の値が関数値として返却されます。
=1       : 乱数列内の最初の値が関数値として返却されます。
上記以外 : 乱数列発生のためのシード値として、引数の値が使われます。
           この新しく発生させられた乱数列の最初の値が関数値として返却されます。
```

使用例

```
use service_routines,only:rand
do i=1,10
  print *,rand(0)                    ! 10個の乱数値を獲得
end do
end
```

2.403 RANDOM_NUMBER組込みサブルーチン

RANDOM_NUMBER サブルーチンは、擬似乱数を取得します。

分類

サブルーチン

形式

CALL RANDOM_NUMBER (*HARVEST*)

HARVEST

実数型でなければなりません。INTENT(OUT) の引数です。スカラまたは配列変数でなければなりません。
0.0 <= *HARVEST* < 1.0 に分布する擬似乱数が設定されます。

機能説明

HARVEST に分布する擬似乱数が設定されます。

使用例

```
real, dimension(8) :: x
call random_number(x)              ! x のすべての要素には擬似乱数が代入されます
```

2.404 RANDOM_SEED組込みサブルーチン

RANDOM_SEED サブルーチンは、RANDOM_NUMBER で使われる種子の初期化、設定、または問合せを行います。

分類

サブルーチン

形式

CALL RANDOM_SEED ([*SIZE* , *PUT* , *GET*])

SIZE（省略可能）

基本整数型スカラでなければなりません。INTENT(OUT) の引数です。種子を保持するのに用いられる個数を返却します。本処理系では、この値は2です。

PUT（省略可能）

基本整数型で*SIZE*以上の大きさをもつ1次元の配列でなければなりません。INTENT(IN) の引数です。*PUT*の値を種子の値として設定します。

GET（省略可能）

基本整数型で*SIZE* 以上の大きさをもつ1次元の配列でなければなりません。

INTENT(OUT) の引数です。現在の種子の値を*GET* に設定します。

すべての引数が省略された場合、種子を初期化します。

引数は0個または1個でなければなりません。

機能説明

RANDOM_NUMBER で使われる種子の初期化、設定、または問合せを行います。

使用例

```
integer, dimension(100) :: seed,old
call random_seed                        ! 種子の初期化を行います
call random_seed(size=k)                ! k にseed の個数が設定
call random_seed(put=seed(1:k))         ! seed に設定
call random_seed(get=old(1:k))          ! seed を取出す
```

2.405 RANGE組込み関数

RANGE 関数は、10進指数範囲を求めます。

分類

要素別処理関数

形式

result = RANGE (*X*)

X

整数型、実数型または複素数型でなければなりません。スカラまたは配列です。

result

基本整数型スカラです。

機能説明

RANGE は、10進指数範囲を返却します。

－ *X* が整数型の場合

関数の結果は、INT(LOG10(HUGE(*X*))) となります。

－ *X* が実数型または複素数型の場合

関数の結果は、INT(MIN(LOG10(HUGE(*X*)),-LOG10(TINY(*X*)))) となります。

関数の結果の型は、基本整数型です。

以下の値の固定値となります。

引数の型	結果の値
1バイトの整数型	2
2バイトの整数型	4
4バイトの整数型	9
8バイトの整数型	18
半精度実数型	4
単精度実数型	37
倍精度実数型	307
4倍精度実数型	4931
半精度複素数型	4
単精度複素数型	37
倍精度複素数型	307
4倍精度複素数型	4931

使用例

```
i = range(4.2)              ! i には37が代入されます
```

2.406 RANK組込み関数

RANK 関数は、データ実体の次元数を求めます。

分類

問合せ関数

形式

result = RANK （ *A* ）

A

任意の型のデータ実体でなければなりません。

result

基本整数型スカラです。

機能説明

RANKは、*A*の次元数を返却します。

使用例

```
integer, dimension(2,2,2) :: array
integer :: i
i = rank (array) ! i には3が代入されます
```

2.407 READ文

READ 文は、ファイルから入力項目並びで指定されたデータ要素、または変数群要素に値を転送します。

READ 文は、以下の形式です。

```
READ ( io-control-spec-list ) [ input-item-list ]          または
READ format [ , input-item-list ]
```

io-control-spec-list はコンマで区切られたデータ転送指定子の並びです。

io-control-spec は、以下の形式です。

```
[ UNIT = ] io-unit                      または
[ FMT = ] format                        または
[ NML = ] namelist-group-name           または
REC = record-number                     または
IOSTAT = io-stat                        または
ERR = err-label                         または
END = end-label                         または
ADVANCE = advance                       または
SIZE = size                             または
EOR = eor-label                         または
NUM = record-len                        または
ASYNCHRONOUS = asynchronous             または
BLANK = blank                           または
DECIMAL = decimal                       または
ID = id                                 または
IOMSG = iomsg                           または
PAD = pad                               または
POS = pos                               または
ROUND = round
```

UNIT 指定子は、必ず指定しなければなりません。文字列‘UNIT=’をUNIT 指定子から省略する場合、UNIT 指定子は、データ転送指定子並びの最初の項目でなければなりません。

io-unit は装置識別子であり、以下の形式です。

```
external-file-unit          または
*                           または
internal-file-unit
```

external-file-unit は、外部ファイル装置であり、スカラ整数式でなければなりません。

internal-file-unit は、内部ファイル装置であり、基本文字変数でなければなりません。

UNIT 指定子に内部ファイル装置を指定した場合、REC指定子またはPOS指定子を指定してはなりません。

UNIT 指定子に * を指定した場合、像番号が1の像だけで有効です。

FMT 指定子とNML 指定子をともに指定することはできません。

文字列‘FMT=’をFMT 指定子から省略する場合、FMT 指定子は、データ転送指定子並びの2番目の項目でなければならず、最初の項目は、文字列‘UNIT=’を省略したUNIT 指定子でなければなりません。

format は書式識別子であり、以下の形式です。

default-char-expr　　または
label　　または
*　　または
scalar-default-int-variable

default-char-expr は、基本文字式です。基本文字式の値は、有効な書式仕様でなければなりません。書式仕様については、“1.8.1 書式仕様”を参照してください。

label は文番号です。文番号は、このREAD 文を含む有効域内のFORMAT 文の文番号でなければなりません。

scalar-default-int-variable は、スカラ基本整変数です。スカラ基本整変数には、ASSIGN 文により、このREAD 文を含む有効域内のFORMAT 文の文番号が割り当てられていなければなりません。

文字列‘NML=’をNML 指定子から省略する場合、NML 指定子は、データ転送指定子並びの2番目の項目でなければならず、最初の項目は、文字列‘UNIT=’を省略したUNIT 指定子でなければなりません。

namelist-group-name は、変数群名です。

NML 指定子を指定した場合、入力項目並びを指定してはなりません。

record-number は、直接探査入力文で入力する記録の番号を指定します。*record-number* は、スカラ整数式でなければなりません。

REC 指定子を指定した場合、END 指定子、POS 指定子およびNML 指定子を指定してはならず、FMT 指定子に並び入力を意味する星印‘*’を指定してはなりません。

io-stat は、スカラ基本整変数です。

IOSTAT 指定子が指定された場合、IOSTAT 指定子に指定された変数には、以下の値が設定されます。

- 誤り条件、ファイル終了条件、および記録終了条件が検出されない場合は、0
- 誤り条件が検出された場合は、1または実行時の診断メッセージの番号
- ファイル終了条件が検出された場合は、-1
- 記録終了条件が検出された場合は、-2

err-label は文番号であり、このREAD 文と同じ有効域内の飛び先文の文番号でなければなりません。

ERR 指定子が指定され、READ 文の実行中に誤り条件が検出された場合、ERR 指定子に指定された文番号をもつ文が次に実行されます。

end-label は文番号であり、このREAD 文と同じ有効域内の飛び先文の文番号でなければなりません。

END 指定子が指定され、READ 文の実行中にファイル終了条件が検出された場合、END 指定子に指定された文番号をもつ文が次に実行されます。

advance は、スカラ基本文字式であり、その値は'YES' または'NO' でなければなりません。

ADVANCE 指定子は、停留入力を行うかどうかを指定します。'NO' を指定すると停留入力が行われます。'YES' を指定すると、停留入力は行われません。ADVANCE 指定子の省略値は、'YES' です。

ADVANCE 指定子は、UNIT 指定子に内部ファイル装置を指定せず、FMT 指定子に明示的な書式仕様をもつ書式付き順番探査にだけ指定できます。

size は、スカラ基本整変数です。

SIZE 指定子は、ADVANCE 指定子が指定されており、その値が'NO' である停留入力文にだけ指定することができます。停留入出力文が終了したとき、SIZE 指定子で指定した変数には、その入力文の実行中に、データ編集記述子によって転送された文字数が設定されます。

eor-label は文番号であり、このREAD 文と同じ有効域内の飛び先文の文番号でなければなりません。

EOR 指定子は、ADVANCE 指定子が指定されており、その値が'NO' である停留入力文にだけ指定することができます。

EOR 指定子が指定され、READ 文の実行中に記録終了条件が検出された場合、EOR 指定子に指定された文番号をもつ文が次に実行されます。

record-len は、スカラ基本整変数です。

NUM 指定子が指定された場合、NUM 指定子に指定された変数には、その入力文の実行によって実際に転送されたFortran 記録の長さが、バイトを単位として、設定されます。

NUM 指定子を指定した場合、FMT 指定子およびNML 指定子を指定してはなりません。

input-item-list は、コンマで区切られた入力項目の並びです。

input-item は、以下の形式です。

variable または
io-implied-do

variable は、変数です。変数は、大きさ引継ぎ配列の全体配列であってはなりません。

io-implied-do は入出力DO 形反復であり、以下の形式です。

(*input-item-list* , *io-implied-do-control*)

io-implied-do-control は入出力DO 制御であり、以下の形式です。

do-variable = *scalar-expr* , *scalar-expr* [, *scalar-expr*]

do-variable はDO 変数であり、整数型、基本実数型、または倍精度実数型の名前付きスカラ変数でなければなりません。基本実数型および倍精度実数型のDO 変数は廃止事項です。

scalar-expr は、整数型、基本実数型、または倍精度実数型のスカラ式でなければなりません。基本実数型および倍精度実数型の *scalar-expr* は廃止事項です。

入出力DO 形反復における、ループの初期化および実行は、DO 構文(“2.120 DO構文”参照)と同じです。

ほかの入出力DO 形反復を含む入出力DO 形反復のDO 変数は、含まれる方の入出力DO 形反復のDO 変数に現れたり結合されたりしてはなりません。

入力項目がポインタの場合、ポインタは、指示先に結合されていなければならず、データはファイルから結合された指示先に転送されます。

入力項目が割付け変数の場合、その割付け変数は割り付けられていなければなりません。

入力項目として配列を指定した場合、配列要素順序(“1.5.8.3 配列要素順序”参照)の順序で、すべての配列要素が指定されたかのように扱われます。

派生型が末端成分にポインタ成分または割付け成分をもつ場合、この派生型の実体は、利用者定義の派生型入出力手続で処理されなければなりません。

利用者定義の派生型入出力手続で処理されなく、かつ、派生型実体の名前を入力項目並びに指定した場合、そのすべての成分が派生型の定義におけるのと同じ順序で指定されたものとして扱われます。

asynchronous はスカラ基本文字定数式でなければなりません。

ASYNCHRONOUS 指定子は、その入出力文が同期か非同期かを指定します。*asynchronous* は値が'YES' または'NO' でなければなりません。

'YES' を指定したとき、その文および入出力操作は非同期です。'NO' を指定するかまたはこの指定子を省略したとき、その文および入出力操作は同期します。

装置識別子がファイル装置番号でない場合、ASYNCHRONOUS 指定子に'YES' を指定できません。

ID 指定子を指定する場合、'YES' を指定したASYNCHRONOUS 指定子も必要です。

非同期入出力は、OPEN 文のASYNCHRONOUS 指定子に'YES' を指定した外部ファイルに対してだけ許されます。

ASYNCHRONOUS 指定子に'YES' を指定したファイルに対しては、同期入出力および非同期入出力の両方ができます。
ASYNCHRONOUS 指定子に'NO' を指定したファイル、ASYNCHRONOUS指定子を省略したファイル、OPEN 文なしで参照する事前接続したファイルおよび内部ファイルには、同期入出力だけができます。
外部ファイルに対して非同期データ転送文で読み書きされた記録およびファイル記憶単位は、データ転送文が同期であった場合と同じ順序で読み書きできます。
非同期データ転送文で変数を次のいずれかに指定した場合、そのデータ参照の実体には、そのデータ転送文の有効域内で暗黙的にASYNCHRONOUS 属性が与えられます。この属性は、明示的な宣言で確定することができます。

- 入出力項目並び
- 変数群要素
- SIZE 指定子

blank はスカラ基本文字式でなければなりません。*blank* の値は、'NULL' または'ZERO' です。BLANK 指定子は、その接続の空白解釈モードを一時的に変更します。この指定子を省略すると、モードは変更されません。

decimal はスカラ基本文字式でなければなりません。*decimal* の値は、'COMMA' または'POINT' です。DECIMAL 指定子は、その接続の小数編集モードを一時的に変更します。この指定子を省略すると、モードは変更されません。

id はスカラ整変数でなければなりません。ID 指定子のある非同期データ転送文の実行が成功すると、ID 指定子に指定した変数は、確定となります。この値は、後続するWAIT 文またはINQUIRE文で、特定のデータ転送操作を識別するために使用することができます。ID 指定子のあるデータ転送文の実行中に誤りが発生すると、ID 指定子に指定した変数は不定となります。

iomsg はスカラ基本文字変数でなければなりません。入出力文の実行中に以下の条件が発生した場合、*iomsg* には説明のためのメッセージが代入されます。

- 誤り条件
- ファイル終了条件
- 記録終了条件

それ以外の場合、*iomsg* の値は変更されません。

pad はスカラ基本文字式でなければなりません。*pad* の値は、'YES' または'NO' です。PAD 指定子は、その接続の空白補充モードを一時的に変更します。この指定子を省略すると、モードは変更されません。

pos はスカラ整数式でなければなりません。POS 指定子は、ファイル記憶単位の中のファイル位置を指定します。この指定子は、流れ探査として接続した装置を指定している場合だけ、データ転送文に指定することができます。
round はスカラ基本文字式でなければなりません。*round* の値は、OPEN 文のROUND 指定子で規定した値の1つです。ROUND 指定子は、その接続の入出力丸めモードを一時的に変更します。この指定子を省略すると、モードは変更されません。

DECIMAL 指定子、BLANK 指定子、PAD 指定子、またはROUND 指定子を指定する場合、書式識別子または変数群名も指定する必要があります。

READ 文の例:

```
read *, a,b,c                    ! 並び出力により、a、b、およびc に入力します
read (3, fmt= "(e7.4)") x        ! 装置参照番号3よりE 形編集により、x に入力します
read 10, i,j,k                   ! 文番号10のFORMAT 文の書式仕様により、
                                 ! i、j、およびk に入力します
```

2.408 REAL組込み関数

REAL 関数は、実数型への変換を行います。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
REAL	-----	1 または 2	整数型、実数型、複素数型、または非10進定数表現 [, 整数型]	実数型
	-----	1	1バイトの整数型	単精度実数型
	FLOATI		2バイトの整数型	単精度実数型
	REAL		4バイトの整数型	単精度実数型
	FLOAT		4バイトの整数型	単精度実数型
	FLOATJ		4バイトの整数型	単精度実数型
	-----		8バイトの整数型	単精度実数型
	-----		単精度実数型	単精度実数型
	SNGL		倍精度実数型	単精度実数型

総称名	個別名	引数の数	引数の型	結果の型
	SNGLQ		4倍精度実数型	単精度実数型
	-----		単精度複素数型	単精度実数型
	-----		倍精度複素数型	倍精度実数型
	-----		4倍精度複素数型	4倍精度実数型

result = REAL (*A* [, *KIND*])

A

整数型、実数型、複素数型、または非10進定数表現でなければなりません。

KIND（省略可能）

スカラ整数定数式でなければなりません。

result

実数型です。*KIND*が指定された場合、種別型パラメタは*KIND*の指定に従います。*KIND*が省略された場合、種別型パラメタは基本実数型になります。

機能説明

REAL、FLOATI、FLOAT、FLOATJ、SNGL、およびSNGLQ は、数値型データを実数型に変換します。

*A*が整数型または実数型の場合、*A*を実数型に変換した近似値を返却します。*A*が複素数型の場合、*A*の実部を実数型に変換した値の近似値を返却します。*A*が非10進定数表現の場合、実数型の変数が非10進定数表現によって指定されたビット列をもったときと同じ値を返却します。

関数の結果の絶対値が HUGE(1.0[,*KIND*]) を超えた場合、結果の値は不定となります。

総称名REAL は、すべての整数型、実数型、複素数型および非10進定数表現の引数に使用することができます。

それぞれの関数の結果の型は以下のとおりとなります。

- *A* が整数型または実数型の場合

  種別型パラメタ*KIND*が指定された場合、*KIND*の大きさをもつ実数型となります。*KIND*が省略された場合、関数の結果の型は基本実数型です。

- *A* が複素数型の場合

  種別型パラメタ*KIND*が指定された場合、*KIND*の大きさをもつ実数型となります。*KIND*が省略された場合、関数の結果の型は*A*の種別型パラメタをもつ実数型です。

- *A* が非10進定数表現の場合

  種別型パラメタ*KIND*が指定された場合、*KIND*の大きさをもつ実数型となります。*KIND*が省略された場合、関数の結果の型は基本実数型です。

使用例

```
b = real(-3)                ! b には-3.0が代入されます
```

2.409 REAL型宣言文

REAL 型宣言文は、実数型のデータ実体を宣言します。

REAL 型宣言文は、以下の形式です。

```
REAL [ kind-selector ] [ [ , attr-spec ]... :: ] entity-decl-list
```

型宣言文の詳細については、“2.3 型宣言文”を参照してください。

2.410 RECORD文

RECORD 文は、指定された派生型‘*type-name*’のデータ実体を宣言します。

RECORD 文は、以下の形式です。

```
RECORD / type-name / entity-decl-list [ , / type-name / entity-decl-list ] ...
```

type-name は、派生型の型名であり、その型名は有効域内で前もって定義されているか、参照結合または親子結合により参照可能でなければなりません。

entity-decl-list はコンマで区切られたデータ要素宣言の並びです。*entity-decl* は、以下の形式です。

```
object-name [ ( array-spec ) ]          または
function-name
```

array-spec は、配列形状指定です。詳細については“2.119 DIMENSION文”を参照してください。

object-name は、実体名です。

function-name は関数名であり、外部関数、組込み関数、仮手続関数、または文関数の名前でなければなりません。

RECORD 文の例:

```
structure /complex_element/
  union
    map
      real :: real,imag
    end map
    map
      complex :: complex
    end map
  end union
end structure
record /complex_element/ x
x%real = 2.0
x%imag = 3.0
print *,x%complex          ! 複素数型の要素complex は、(2.0,3.0) で確定しています
```

2.411 REDLENサービスサブルーチン

機能説明

直前に入力された1つのFortran 記録の長さを返却します。

形式

```
CALL REDLEN ( i , retcd )
```

i

基本整数型スカラ。直前に入力されたFortran 記録の長さが設定されます。

retcd

基本整数型スカラ。常に0が設定されます。

使用例

```
use service_routines,only:redlen
integer :: i,retcd
character(len=10) :: ch
open(10,file='fort.txt')
read(10,*) ch
call redlen(i,retcd)
write(6,*) i
```

```
close(10)
end
```

2.412 RENAMEサービス関数

機能説明

ファイル名を変更します。

形式

iy = RENAME (*old* , *new*)

old

基本文字型スカラ。変更前のファイル名を指定します。
指定するファイル名の長さは、システムのインクルードファイルに定義されている最大長を超えてはなりません。

new

基本文字型スカラ。変更後のファイル名を指定します。
このファイルが既存である場合、削除した後、ファイル名を変更します。
指定するファイル名の長さは、システムのインクルードファイルに定義されている最大長を超えてはなりません。

関数結果

基本整数型スカラ。ファイル名を変更できたときは0、エラーが発生したときは0以外の値が返却されます。

使用例

```
use service_routines,only:rename
iy = rename('xx','yy')
end
```

2.413 REPEAT組込み関数

REPEAT 関数は、文字列の複写を行います。

分類

変形関数

形式

result = REPEAT (*STRING* , *NCOPIES*)

STRING

文字型スカラでなければなりません。

NCOPIES

整数型スカラでなければなりません。*NCOPIES* >= 0 でなくてはなりません。

result

STRING と同じ種別型パラメタをもつ文字型スカラです。長さは LEN(*STRING*) × *NCOPIES* です。

機能説明

REPEAT は、*STRING* を *NCOPIES* 回複写して連結した文字列を返却します。

NCOPIES が0のときは長さ0の文字型スカラを返却します。

関数の結果の型は、*STRING* と同じ種別型パラメタをもつ文字型スカラで、その長さは *STRING* の文字長に *NCOPIES* を掛けた値です。

使用例

```
character (len=6) :: n
n = repeat('ho',3)                ! n には'hohoho' が代入されます
```

2.414 RESHAPE組込み関数

RESHAPE 関数は、配列形状を変更します。

分類

変形関数

形式

result = RESHAPE (*SOURCE* , *SHAPE* [, *PAD* , *ORDER*])

SOURCE

どの型でもかまいません。スカラであってはなりません。
PAD が省略されているまたは大きさが0の場合、*SOURCE* の大きさはPRODUCT(*SHAPE*) の値以上でなければなりません。

SHAPE

整数型の1次元配列で、その大きさは定数でなければなりません。大きさは、1～30の正の数でなければなりません。どの要素も負であってはなりません。

PAD **(省略可能)**

SOURCE と同じ型および同じ型パラメタでなければなりません。スカラであってはなりません。

ORDER **(省略可能)**

整数型でなければなりません。形状は*SHAPE* と同じでなければなりません。
ORDER の値は、*SHAPE* の大きさをn としたとき、1 ～ n の値を並び替えたものでなければなりません。
ORDER が省略された場合、(/ (I, I = 1, n) /) が設定されたものとします。

result

SOURCE と同じ型および同じ種別型パラメタで、形状は*SHAPE* と同じです。

機能説明

RESHAPE は、指定された配列を*SHAPE* の形状に再構成します。

結果の要素 $r(S_1, S_2, ..., S_n)$ を、$r(S_{ORDER(1)}, S_{ORDER(2)}, ..., S_{ORDER(n)})$ で並び替えたものは、*SOURCE* の要素を配列要素順序で並べ、足りなければ*PAD* の要素を繰り返し配列要素順序で並べたものと等しいものとします。

関数の結果の型は、*SOURCE* と同じです。

使用例

```
integer x(3,2)
x = reshape((/1,2,3,4/),(/3,2/),pad=(/0/))          ! x     には ┌     ┐
                                                    !            | 1 4 |
                                                    !            | 2 0 |
                                                    !            | 3 0 |
                                                    !            └     ┘ が代入されます
```

2.415 RETURN文

RETURN 文は、関数副プログラムまたはサブルーチン副プログラムの実行を終了します。

RETURN 文は、以下の形式です。

```
RETURN [ scalar-int-expr ]
```

scalar-int-expr はスカラ整数式であり、サブルーチン副プログラムの有効域内だけで指定できます。

*scalar-int-expr*を指定した場合、その値n が1以上で、仮引数並びの星印'*'の個数以下であると、そのサブルーチンを呼び出したCALL文から、実引数並び中のn 番目の選択戻り指定子によって識別される文に制御が移行します。スカラ整数式を省略したか、またはスカラ整数式の値が制限された範囲外であった場合には、選択戻りへの制御の移行は発生しません。

RETURN 文の例:

```
subroutine zee (a,b)
implicit none
real, intent(inout) :: a,b
  ...
if (a>b) then
  return                        ! サブルーチンzee の終了
else
  a=a*b
  return                        ! サブルーチンzee の終了
end if
end subroutine zee
```

2.416 REWIND文

REWIND 文は、指定されたファイルをその始点に位置付けます。

REWIND 文の実行は、指定した装置における非同期データ転送操作のすべてに対して、待機操作を行います。

REWIND 文は、以下の形式です。

```
REWIND external-file-unit                または
REWIND ( position-spec-list )
```

external-file-unit は外部ファイル装置であり、スカラ整数式でなければなりません。

position-spec-list は、コンマで区切られた位置付け指定子の並びです。

position-spec は、以下の形式です。

```
[ UNIT= ] external-file-unit             または
IOMSG = iomsg                            または
IOSTAT= io-stat                          または
ERR= err-label
```

位置付け指定子並びにおいて、UNIT 指定子は必ず1つ指定しなければならず、他の指定子はそれぞれ1つ指定することができます。

文字列‘UNIT=’をUNIT 指定子から省略する場合、UNIT 指定子は、位置付け指定子並びの最初の項目でなければなりません。

io-stat は、スカラ整変数です。

IOSTAT 指定子が指定された場合、IOSTAT 指定子に指定された変数には、以下の値が設定されます。

- 誤り条件、ファイル終了条件、および記録終了条件が検出されない場合は、0
- 誤り条件が検出された場合は、1または実行時の診断メッセージの番号

err-label は文番号であり、このREWIND 文と同じ有効域内の飛び先文の文番号でなければなりません。

ERR 指定子が指定され、REWIND 文の実行中に誤り条件が検出された場合、ERR 指定子に指定された文番号をもつ文が次に実行されます。

接続されていて存在しないファイルに対して、REWIND 文を実行してもかまいませんが、何も効果がありません。

iomsg はスカラ基本文字変数でなければなりません。入出力文の実行中に誤り条件が発生した場合、*iomsg* には説明のためのメッセージが代入されます。

誤り条件が発生しない場合、*iomsg* の値は変更されません。

直接探査として接続されているファイルを、REWIND 文で参照してはなりません。

REWIND 文の例:

```
rewind 10                   ! 装置番号10に接続されているファイルを始点に位置付けます
rewind (10,err=99)          ! 装置番号10に接続されているファイルを始点に位置付け、
                            ! 誤り条件が発生した場合には、文番号99の文が次に実行されます
```

2.417 RINDEXサービス関数

機能説明

文字列中の最も右にある文字列の先頭の位置を返却します。

形式

iy = RINDEX (*string* , *substring*)

string

基本文字型スカラ。文字列を指定します。

substring

基本文字型スカラ。検索する文字列を指定します。

関数結果

基本整数型スカラ。*string* が *substring* を含む場合、*string* 中の最も右の *substring* の先頭の位置を返却します。*string* が *substring* を含まない場合または *string* の長さが *substring* の長さよりも短い場合、0が返却されます。*substring* の長さが0である場合、*string* の長さ+1が返却されます。

使用例

```
use service_routines,only:rindex
character(len=50) string
character(len=7) :: substr
integer(4) :: offset
string = "Fortran 95 Fortran 90 Fortran77 string scan"
substr = "Fortran"
offset = rindex(string,substr)                    ! offset には23が代入されます
end
```

2.418 RRSPACING組込み関数

RRSPACING 関数は、相対的な間隔の逆数を求めます。

分類

要素別処理関数

形式

result = RRSPACING (*X*)

X

実数型でなければなりません。

result

X と同じ型です。

機能説明

RRSPACING は、*X* を数体系の数として表現したときの引数値前後の相対的な間隔の逆数を返却します。そのような値が2つあるときは、絶対値の大きいほうの値とします。*X* がIEEE 無限大である場合、結果は非数になります。*X* がIEEE 非数である場合、結果は非数になります。

関数の結果は、ABS($X \times \mathrm{RADIX}(X)^{(-\mathrm{EXPONENT}(X))} \times \mathrm{RADIX}(X)^{\mathrm{DIGITS}(X)}$) となり、その型は *X* と同じです。

使用例

```
r = rrspacing(-4.7)
```

2.419 RSHIFT組込み関数

RSHIFT 関数は、右算術シフトを行います。

分類

要素別処理関数

形式

result = RSHIFT (*I* , *SHIFT*)

I

整数型でなければなりません。

SHIFT

整数型でなければなりません。*SHIFT* < BIT_SIZE(*I*) でなければなりません。また、負の値であってはなりません。

result

I と同じ型です。

機能説明

RSHIFT は、右算術シフトを行います。

I を *SHIFT* ビット分だけ右算術シフトを行います。

関数の結果の型は、*I* と同じです。

使用例

```
i = rshift (-7,2)                    ! i には-2が代入されます
```

2.420 RTCサービス関数

機能説明

1970 年1月1日 午前0時からのUTC の通算秒を返却します。

形式

y = RTC ( )

関数結果

倍精度実数型スカラ。1970年1月1日 午前0時からのUTC の通算秒が返却されます。時間が取得できなかった場合、-1.0が返却されます。

使用例

```
use service_routines,only:rtc
real(8) :: time1,time2
time1 = rtc( )
call sub( )
time2 = rtc( )
print *,'time spent = ', time2-time1
end
```

2.421 SAME_TYPE_AS組込み関数

機能説明

SAME_TYPE_AS 組込み関数は、第1引数の動的な型が第2引数の動的な型と同じかどうかを問い合わせます。

分類

問合せ関数

形式

result = SAME_TYPE_AS (*A* , *B*)

A

拡張可能型の実体でなければなりません。ポインタの場合、結合状態が不定であってはなりません。

B

拡張可能型の実体でなければなりません。ポインタの場合、結合状態が不定であってはなりません。

result

基本論理型スカラです。

関数結果

A の動的な型が*B* の動的な型と同じとき真を返却します。

空状態のポインタの動的な型または割り付けられていない割付けの型は、その実体の宣言時の型です。

使用例

```
type base
  integer :: base01
end type
type,extends(base):: ext
  integer :: ext01
end type
type (ext),target :: t01a, t01b
class(base),pointer:: p01,p02
p01=>t01a
p02=>t01b
print *,same_type_as(p01, p02)
end
```

2.422 SAVE文

SAVE 文は、RETURN 文またはEND 文の実行後も結合状態、割付け状態、定義状態、および値を保持することを指定します。

SAVE 文は、以下の形式です。

SAVE [[::] *saved-entity-list*]

saved-entity-list はコンマで区切られた保存要素の並びであり、*saved-entity* は以下の形式です。

object-name または
procedure-pointer-name または
/ *common-block-name* /

object-name は、実体名です。*object-name* は、共通ブロック中の実体の名前、仮引数名、手続名、関数結果名、自動割付け変数名、および名前付き定数であってはなりません。

procedure-pointer-name は、手続ポインタの名前でなければなりません。

common-block-name は、共通ブロック名です。

saved-entity-list のないSAVE 文は、その有効域内のすべての可能な項目を指定したかのように扱われます。*saved-entity-list* のないSAVE 文が有効域内にあるとき、同じ有効域内に他に明示的なSAVE 文およびSAVE 属性があってはなりません。

副プログラム内でSAVE 属性をもつ実体は、その副プログラムのすべての引用で共有されます。

SAVE 文に共通ブロック名を指定したとき、主プログラムを除いて、その共通ブロックが現れるすべての有効域内で、その共通ブロック名をSAVE 文に指定しなければなりません。

SAVE 文の例:

```
subroutine sub( )
save i,j,/myblock/,k       ! i、j、k および共通ブロック名myblock はSAVE 属性をもちます
common /myblock/ x1, x2
```

2.423 SCALE組込み関数

SCALE 関数は、実数と基数の整数べきとの積を求めます。

分類

要素別処理関数

形式

result = SCALE (*X* , *I*)

X

実数型でなければなりません。

I

整数型でなければなりません。

result

*X*と同じ型です。

機能説明

SCALE は、実数と基数の整数べきとの積を返却します。

関数の結果は、$X \times \mathrm{RADIX}(X)^I$となり、その型は*X*と同じです。

関数の結果の絶対値がHUGE(*X*) を超えた場合、結果の値は不定となります。

使用例

```
x = scale(1.5,3)                ! x には12.0が代入されます
```

2.424 SCAN組込み関数

SCAN 関数は、文字列の検索を行います。

分類

要素別処理関数

形式

result = SCAN (*STRING* , *SET* [, *BACK* , *KIND*])

STRING

文字型でなければなりません。

SET

*STRING*と同じ種別型パラメタの文字型でなければなりません。

BACK **(省略可能)**

論理型でなければなりません。

KIND **(省略可能)**

スカラ整数定数式でなければなりません。

result

整数型です。*KIND*が指定された場合、種別型パラメタは*KIND*の指定に従います。*KIND*が省略された場合、種別型パラメタは基本整数型になります。

機能説明

SCAN は、文字列中に部分文字列が含まれているかどうかを調査します。

－ *BACK*に偽を指定している、または省略している場合

*STRING*が*SET*中の文字を含む場合、両方にある最も左の文字の位置を返却します。*STRING*が*SET*中の文字を1文字も含まない場合および、*STRING*または*SET*の文字長が0の場合は、0を返却します。

－ *BACK*に真を指定している場合

*STRING*が*SET*中の文字を含む場合、両方にある最も右の文字の位置を返却します。*STRING*が*SET*中の文字を1文字も含まない場合および、*STRING*または*SET*の文字長が0の場合は、0を返却します。

関数の結果の型は、種別型パラメタ*KIND*が指定された場合、種別型パラメタ*KIND*の整数型となります。*KIND*が省略された場合、関数の結果の型は基本整数型です。

使用例

```
i = scan("Lalalalala","la")                    ! i には2が代入されます
i = scan("LalalaLALA","la",back=.true.)        ! i には6が代入されます
```

2.425 SECNDSサービス関数

機能説明

午前0時からのシステム時間の経過秒数から第1引数で指定した秒数を引いた秒数を返却します。

形式

y = SECNDS (*sec*)

sec

実数型スカラ。午前0 時からのシステム時間の経過秒数から引く値を秒単位で指定します。

関数結果

実数型スカラ。午前0 時からのシステム時間の経過秒数から*sec*秒を引いた秒数を返却します。システムが時間を返却できないときは、-1.0が返却されます。

使用例

```
use service_routines,only:secnds
real :: zero=0.0,sec0,sec1
sec0 = secnds(zero)
call sub1( )
sec1 = secnds(sec0)
write(6,*) sec1                ! サブルーチンsub1 が要した秒数
end
```

2.426 SECONDサービス関数

機能説明

実行可能プログラムの実行開始時からのユーザCPU 時間を返却します。ユーザCPU 時間は現行プロセスとそのプロセス内の全スレッドで利用されたユーザCPU 時間です。

形式

y = SECOND ( )

関数結果

実数型スカラ。実行可能プログラムの実行開始時からのユーザCPU 時間が秒単位で返却されます。エラーが発生したときは、-1.0が返却されます。

使用例

```
use service_routines,only:second
real :: y1,y2
y1 = second( )
do i=1,5000
  write(*,*) i,i*i
end do
y2 = second( )
print *,'user time=',y2-y1
end
```

2.427 SELECT CASE文

SELECT CASE 文は、CASE 構文の開始を示します。

SELECT CASE 文は、以下の形式です。

[*case-construct-name* :] SELECT CASE (*case-expr*)

case-construct-name は、CASE 構文名です。

case-expr は場合式であり、スカラ整数式、スカラ文字式、またはスカラ論理式でなければなりません。

CASE 構文の詳細については、“2.63 CASE構文”を参照してください。

2.428 SELECTED_CHAR_KIND組込み関数

SELECTED_CHAR_KIND 組込み関数は、引数によって名前付けられた文字集合の種別型パラメタ値を返却します。

分類

変形関数

形式

result = SELECTED_CHAR_KIND (*NAME*)

NAME

基本文字型スカラでなければなりません。

result

基本整数型スカラです。

機能説明

NAME が値'DEFAULT' の場合、結果の値は基本文字型の種別型パラメタの値です。*NAME* が値'ASCII' の場合、結果の値は基本文字型の種別型パラメタの値です。そうでないとき、結果の値は-1です。
NAME に指定された文字列中の大小文字は区別されず、後続空白は読み飛ばされます。

使用例

```
character (kind=selected_char_kind("ASCII")) :: c          ! c は基本文字型です。
```

2.429 SELECTED_INT_KIND組込み関数

SELECTED_INT_KIND 関数は、整数型の種別型パラメタを求めます。

分類

変形関数

形式

result = SELECTED_INT_KIND (*R*)

R

整数型スカラでなければなりません。

result

基本整数型スカラです。

機能説明

SELECTED_INT_KIND は、$-10^R < n < 10^R$ の範囲内のすべての値n を表現できる整数型の種別型パラメタ値を返却します。

対応する種別型パラメタが存在しない場合、-1を返却します。

関数の結果の型は、基本整数型です。

使用例

```
integer (kind=selected_int_kind(3)) :: i,j
!    i とj は少なくとも-1000から1000までの数値を表現できる整数型の変数です
```

2.430 SELECTED_REAL_KIND組込み関数

SELECTED_REAL_KIND 関数は、実数型の種別型パラメタを求めます。

分類

変形関数

形式

result = SELECTED_REAL_KIND ([*P* , *R* , *RADIX*])

P **(省略可能)**

整数型スカラでなければなりません。省略した場合、値0を指定したとみなします。

R **(省略可能)**

整数型スカラでなければなりません。省略した場合、値0を指定したとみなします。

RADIX **(省略可能)**

整数型スカラでなければなりません。省略した場合、値2を指定したとみなします。

result

基本整数型スカラです。

機能説明

SELECTED_REAL_KINDは、組込み関数PRECISIONによって返されるような*P*けた以上の10進精度、組込み関数RANGEによって返されるような*R*以上の10進指数範囲、および組込み関数RADIXによって返されるような*RADIX*の基数をもつ実数型の種別型パラメタ値を返却します。

少なくとも1つの引数を指定しなければなりません。

*RADIX*は、値2だけが有効です。

10進精度のみ利用できない場合は、-1を返却します。10進指数範囲のみ利用できない場合は、-2を返却します。10進精度および10進指数範囲が利用できない場合は、-3を返却します。*RADIX*が値2でない場合は、-5を返却します。

関数の結果の型は、基本整数型です。

使用例

```
real (kind=selected_real_kind(3,3)) :: a,b
!    a とb は、少なくとも-1000から1000までの数値が表現可能であり
!    小数点以下3けたの有効範囲をもつ実数型の変数です
```

2.431 SELECT TYPE構文

SELECT TYPE 構文は、それを構成するブロックのうち、1つの実行を選択します。

その選択は、*selector* の実行時の型に基づいて行われます。*selector* への名前の結合は、ASSOCIATE 構文と同じ方法で行われます。

SELECT TYPE 構文は以下の形式です。

```
[ select-construct-name: ] SELECT TYPE ( [ associate-name => ] selector)
    [ type-guard-stmt
      block] ...
END SELECT TYPE [ select-construct-name ]
```

selector は選択子です。*selector* が名前付き変数でない場合、構文要素“*associate-name* =>”がなければなりません。

selector が変数でない、またはベクトル添字をもつ変数でない場合、結合名は変数確定の文脈に現れてはなりません。

selector が変数の場合、共添字付き実体であってはなりません。

SELECT TYPE 文中の*selector* は、多相的でなければなりません。

associate-name は結合名です。結合名の属性規則については、“2.29.1 ASSOCIATE 構文の形”を参照してください。

type-guard-stmt は、型保持文です。以下の形式です。

```
TYPE IS( type-spec ) [ select-construct-name ]                  または
CLASS IS( derived-type-spec ) [ select-construct-name ]         または
CLASS DEFAULT [ select-construct-name ]
```

type-spec は、型指定子です。型指定子には、組込み型指定子および派生型指定子があります。

derived-type-spec は、派生型指定子です。

組込み型指定子については、“2.3 型宣言文”を参照してください。また、派生型指定子については、“1.5.11.8 派生型指定子”を参照してください。

type-spec および*derived-type-spec* は、各長さ型パラメタが引き継がれることを指定しなければなりません。

type-spec および*derived-type-spec* は、連続派生型またはBIND 属性をもつ型を指定できません。

selector が無制限多相的でない場合、*type-spec* および*derived-type-spec* はその*selector* の宣言時の型の直接拡張型を指定しなければなりません。

1つのSELECT TYPE 構文について、同一の型および種別型パラメタ値を2つ以上のTYPE IS で始まる型保持文で指定してはならず、かつ2つ以上のCLASS IS で始まる型保持文で指定してはなりません。

1つのSELECT TYPE 構文で、CLASS DEFAULT は、1つ以下でなければなりません。

SELECT TYPE 構文のSELECT TYPE 文に*select-construct-name* を指定する場合は、対応するEND SELECT TYPE 文にも同じ*select-construct-name* を指定しなければなりません。SELECT TYPE 構文のSELECT TYPE 文に*select-construct-name* を指定しない場合は、対応するEND SELECT TYPE文に*select-construct-name* を指定してはなりません。型保持文に*select-construct-name* を指定する場合は、対応するSELECT TYPE 文にも同じ*select-construct-name* を指定しなければなりません。

SELECT TYPE 構文の結合名が指定されていない場合は、結合名は、*selector* を構成する名前になります。

SELECT TYPE 構文を実行すると、*selector* が変数でない場合は、選択された式が評価されます。

SELECT TYPE 構文では、最大1つのブロックが実行されます。そのブロックの実行中は、*associate-name* は*selector* と結合された実体を指します。

型保持文がTYPE IS で始まる場合、その文と*selector*の実行時の型および型パラメタ値が一致します。CLASS IS で始まる場合、その文の実行時の型が、その型の直接拡張型である*selector*の実行時の型と一致します。その場合、指定された種別型パラメタ値は、選択子の実行時の型の対応する型パラメタ値と同じです。

実行されるブロックは次の順で選択されます。

1. *selector* がTYPE IS で始まる型保持文と一致する場合、その文の後続のブロックが実行されます。
2. 1. でなく、かつ*selector* がCLASS IS で始まる型保持文のうちただ1つと一致する場合、その文の後続のブロックが実行されます。
3. 1. および2. でなく、*selector*とCLASS IS で始まる型保持文に一致するものが複数ある場合、そのうちの1つがそれ以外で指定された型のすべての直接拡張型であれば、それに対応する文の後続のブロックが実行されます。
4. 上記以外でCLASS DEFAULT で始まる型保持文がある場合、それに後続する文が実行されます。

TYPE IS で始まる型保持文の後続ブロック内では、結合要素は、多相的ではなく、型保持文内で名前付けられた型をもち、かつ*selector*と同じ型パラメタ値をもちます。

CLASS IS で始まる型保持文の後続ブロック内では、結合要素は、多相的であって、型保持文で名前付けられた宣言時の型をもちます。その型パラメタ値は、対応する*selector* の型パラメタ値です。

CLASS DEFAULT で始まる型保持文の後続ブロック内では、結合要素は、多相的であって、*selector*と同じ宣言時の型をもちます。その型パラメタ値は、対応する*selector* の宣言時の型の型パラメタ値です。

型保持文を飛び先文としてはなりません。END SELECT TYPE 文への飛び越しができるのは、そのSELECT TYPE 構文の内部からだけです。

SELECT TYPE 構文の例:

```
type p1
  real :: x,y
end type p1
type, extends(p1) :: p2_a
  real :: z
end type p2_a
type, extends(p1) :: p2_b
  integer :: c
end type p2_b
type(p1), target :: t_p1
type(p2_a), target :: t_p2_a
type(p2_b), target :: t_p2_b
class(p1), pointer :: p_p1
p_p1 => t_p2_b
...
select type ( p_p1 )                ! p_p1 の実行時の型は、p2_b です
class is ( p1 )                     ! p2_b は、p1 の直接拡張型です。
print *, p_p1%x, p_p1%y             ! このブロックが実行されます。
type is ( p2_a ) !
  print *, p_p1%x, p_p1%y, p_p1%z
end select
end
```

2.432 SEQUENCE文

SEQUENCE 文は、派生型定義中に指定し、その型の成分を指定された順に格納することを宣言します。

SEQUENCE 文は以下の形式です。

```
SEQUENCE
```

SEQUENCE 文の指定がある派生型を連続型といいます。

派生型定義にSEQUENCE 文の指定がある場合、成分定義中で指定される派生型は、すべて連続型でなければなりません。

派生型定義の詳細については、“1.5.11.1 派生型定義”を参照してください。

SEQUENCE 文の例:

```
type zee
  sequence                ! zee は連続型です
  real :: a,b,c           ! a、b、c は、指定された順に格納されます
end type zee
```

2.433 SERVICE_ROUTINES非標準組込みモジュール

このモジュールは、サービスルーチン(“1.12.8 サービスルーチン”参照)の明示的引用仕様を参照可能にします。

2.434 SETBITサービスサブルーチン

機能説明

第3引数の値に従って、1つのビットにオンまたはオフを設定します。

形式

```
CALL SETBIT ( pos , i , status )
```

pos

基本整数型スカラ。*status* の値を設定するビットを指定します。0未満または*i*のビット数以上の値を指定した場合、*i*の値は変更されません。

i

基本整数型スカラ。*status* の値が設定されます。

status

基本整数型スカラ。*i* に設定する値を指定します。

```
=0          : i のpos 番目のビットを0にした値を i に設定します。
=0 以外     : i のpos 番目のビットを1にした値を i に設定します。
```

使用例

```
use service_routines,only:setbit
integer :: i
i = 10
call setbit(3,i,0)              ! i に2が設定されます
write(6,fmt="(1x,i4)") i
i = 10
call setbit(0,i,1)              ! i に11が設定されます
write(6,fmt="(1x,i4)") i
end
```

2.435 SETRCDサービスサブルーチン

機能説明

指定した値の下位1バイトをFortran プログラムの復帰コードとして設定し、Fortran プログラムの実行を終了します。

形式

```
CALL SETRCD ( i )
```

i

基本整数型スカラ。復帰コードを指定します。指定した値の下位1バイト(0 から255)が、Fortran プログラムの復帰コードとなります。

使用例

```
use service_routines,only:setrcd
read(*,*) ia
```

```
if (ia .eq. 99) then
  call setrcd(ia)
else
  print *,'read OK'
end if
end
```

2.436 SET_EXPONENT組込み関数

SET_EXPONENT 関数は、指数部の入替えを行います。

分類

要素別処理関数

形式

result = SET_EXPONENT (*X* , *I*)

X

実数型でなければなりません。

I

整数型でなければなりません。

result

*X*と同じ型です。

機能説明

SET_EXPONENT は、小数部が*X*で、指数部が*I*の実数型の値を返却します。

関数の結果は、$X \times \mathrm{RADIX}(X)^{(I-\mathrm{EXPONENT}(X))}$ となり、その型は*X*と同じです。

関数の結果の絶対値がHUGE(*X*) を超えた場合、結果の値は不定となります。

使用例

```
a = set_exponent(4.6,2)
```

2.437 SHサービス関数

機能説明

Bourne シェルの入力としてコマンドを発行します。

形式

iy = SH (*command*)

command

基本文字型スカラ。Bourne シェルの入力として実行するコマンドを指定します。

関数結果

基本整数型スカラ。Bourne シェルの終了ステータスが返却されます。

注意事項

現在のプロセスは、コマンドの実行が完了するまで待ち状態となります。

使用例

```
use service_routines,only:sh
if ( sh('x.sh') /= 0 ) stop 'shell error'
call sub( )
end
```

2.438 SHAPE組込み関数

SHAPE 関数は、配列またはスカラの形状を求めます。

分類

問合せ関数

形式

result = SHAPE (*SOURCE* [, *KIND*])

SOURCE

どの型でもかまいません。配列またはスカラです。空状態のポインタまたは割り付けられていない割付け実体であってはなりません。大きさ引継ぎ配列であってはなりません。

KIND（省略可能）

スカラ整数定数式でなければなりません。

result

整数型です。*KIND*が指定された場合、種別型パラメタは*KIND*の指定に従います。*KIND*が省略された場合、種別型パラメタは基本整数型になります。*SOURCE*の次元数に等しい大きさの基本整数型の1次元配列です。*SOURCE*がスカラの場合、大きさ0となります。

機能説明

SHAPE は、*SOURCE*の形状を1次元配列として返却します。

関数の結果の型は、種別型パラメタ*KIND*が指定された場合、種別型パラメタ*KIND*の整数型となります。*KIND*が省略された場合、関数の結果の型は基本整数型です。

使用例

```
integer i(3)
integer b(10,-2:10,10)
i = shape(b(1:9,-2:3,:))            ! i には(/9,6,10/) が代入されます
```

2.439 SHIFTA組込み関数

SHIFTA 関数は、右方向へ算術シフトします。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
SHIFTA	-----	2	整数型 , 整数型	整数型

result = SHIFTA (*I* , *SHIFT*)

I

整数型でなければなりません。

SHIFT

整数型でなければなりません。*SHIFT* <= BIT_SIZE(*I*) でなければなりません。また、負の値であってはなりません。

result

*I*と同じ整数型です。

機能説明

SHIFTA は、*SHIFT*ビットの右算術シフトをします。

関数の結果の型は、*I*と同じ整数型です。

使用例

```
k = shifta(-14142,5)          ! k には-442が代入されます
```

2.440 SHIFTL組込み関数

SHIFTL 関数は、左方向へシフトします。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
SHIFTL	-----	2	整数型 , 整数型	整数型

result = SHIFTL (*I* , *SHIFT*)

I

整数型でなければなりません。

SHIFT

整数型でなければなりません。*SHIFT* <= BIT_SIZE(*I*) でなければなりません。また、負の値であってはなりません。

result

*I*と同じ整数型です。

機能説明

SHIFTL は、*SHIFT*ビットの左シフトをします。

関数の結果の型は、*I*と同じ整数型です。

使用例

```
k = shiftl(223606,5)          ! k には7155392が代入されます
```

2.441 SHIFTR組込み関数

SHIFTR 関数は、右方向へシフトします。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
SHIFTR	-----	2	整数型 , 整数型	整数型

result = SHIFTR (*I* , *SHIFT*)

I

整数型でなければなりません。

SHIFT

整数型でなければなりません。*SHIFT* <= BIT_SIZE(*I*) でなければなりません。また、負の値であってはなりません。

result

*I*と同じ整数型です。

機能説明

SHIFTR は、*SHIFT* ビットの右シフトをします。

関数の結果の型は、*I* と同じ整数型です。

使用例

```
k = shiftr(-14142,5)          ! k には134217286が代入されます
```

2.442 SHORTサービス関数

機能説明

基本整数型スカラを2バイトの整数型に変換します。

形式

iy = SHORT (*ix*)

ix

基本整数型スカラ。

関数結果

2バイトの整数型スカラ。基本整数型スカラを2バイトの整数型スカラに変換した値が返却されます。

使用例

```
use service_routines,only:short
integer(kind=2) :: ix
integer :: ix4
ix = short(ix4)
end
```

2.443 SIGN組込み関数

SIGN 関数は、符号の付け替えを行います。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
SIGN	-----	2	1バイトの整数型 , 1バイトの整数型	1バイトの整数型
	I2SIGN		2バイトの整数型 , 2バイトの整数型	2バイトの整数型
	IISIGN		2バイトの整数型 , 2バイトの整数型	2バイトの整数型
	ISIGN		4バイトの整数型 , 4バイトの整数型	4バイトの整数型
	JISIGN		4バイトの整数型 , 4バイトの整数型	4バイトの整数型
	-----		8バイトの整数型 , 8バイトの整数型	8バイトの整数型
	-----		実数型	実数型
	SIGN		単精度実数型 , 単精度実数型	単精度実数型

総称名	個別名	引数の数	引数の型	結果の型
	DSIGN		倍精度実数型， 倍精度実数型	倍精度実数型
	QSIGN		4倍精度実数型， 4倍精度実数型	4倍精度実数型

result = SIGN (*A* , *B*)

A

整数型または実数型でなければなりません。

B

A と同じ型および種別型パラメタでなければなりません。

result

A と同じ型です。

機能説明

SIGN、I2SIGN、IISIGN、ISIGN、JISIGN、DSIGN、およびQSIGN は、*A* の絶対値に*B* のオペランドの符号を付けた値を求めます。

B >= 0 のとき、結果の値はABS(*A*) となり、*B* < 0 のときは -ABS(*A*) となります。

総称名SIGN は、すべての整数型または実数型の引数に使用することができます。

それぞれの関数の結果の型は、*A* と同じです。

使用例

```
i = sign (30,-2)          ! i には-30が代入されます
```

2.444 SIGNALサービス関数

機能説明

異常終了事象または例外事象が発生したときに、アクションを行うルーチンを登録したり、シグナルを無視することができます。

形式

iy = SIGNAL (*i* , *func* , *flag*)

i

基本整数型スカラ。シグナル番号を指定します。

func

シグナル番号*i* が補足されたときに制御を渡すハンドリングルーチンの名前です。

flag

基本整数型スカラ。

<0 : *func* がシグナル出口関数名であることを示します。
=0 : システムのデフォルトのアクションを使用します。
=1 : シグナルを無視します。
>1 : シグナルの動作として、システムに通知されます。

関数結果

基本整数型スカラ。以前のシグナルに対する*func* または*flag* の値が返却されます。

- 以前の*flag* の値が0または1の場合には、その値
- 以前の*flag* の値が負数の場合は、以前の*func* の値
- *i* の値が無効の場合、または*flag* の値が1より大きい場合は、システムのエラーコードを反転した値

使用例

```
use service_routines,only:signal
integer(kind=2),external :: func
integer :: iy
iy = signal(8,func,-1)
end
```

2.445 SIN組込み関数

SIN 関数は、ラジアン値を引数とする正弦を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
SIN	-----	1	実数型 または 複素数型	実数型 または 複素数型
	SIN		単精度実数型	単精度実数型
	DSIN		倍精度実数型	倍精度実数型
	QSIN		4倍精度実数型	4倍精度実数型
	CSIN		単精度複素数型	単精度複素数型
	CDSIN		倍精度複素数型	倍精度複素数型
	CQSIN		4倍精度複素数型	4倍精度複素数型

result = SIN (*X*)

X

実数型または複素数型でなければなりません。

result

*X*と同じ型です。

機能説明

SIN、DSIN、QSIN、CSIN、CDSIN、およびCQSIN は、実数型データまたは複素数型データの正弦を求めます。

引数はラジアン値を指定します。

単精度実数型の引数の場合、引数は ABS(*X*) < 8.23E+05 でなければなりません。

倍精度実数型の引数の場合、引数はDABS(*X*) < 3.53D+15 でなければなりません。

4倍精度実数型の引数の場合、引数は QABS(*X*) < 2.0Q0^{62} × π でなければなりません。

単精度複素数型の引数の場合、ABS(REAL(*X*)) < 8.23E+05 かつ ABS(IMAG(*X*)) < 89.415E0 でなければなりません。

倍精度複素数型の引数の場合、DABS(DREAL(*X*)) < 3.53D+15 かつ DABS(DIMAG(*X*)) < 710.475D0 でなければなりません。

4倍精度複素数型の引数の場合、QABS(QREAL(*X*)) < 2.0Q0^{62} × π かつ QABS(QIMAG(*X*)) < 11357.125Q0 でなければなりません。

総称名SIN は、すべての実数型または複素数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
r = sin(.5)
```

2.446 SIND組込み関数

SIND 関数は、度数値を引数とする正弦を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
SIND	-----	1	実数型	実数型
	SIND		単精度実数型	単精度実数型
	DSIND		倍精度実数型	倍精度実数型
	QSIND		4倍精度実数型	4倍精度実数型

result = SIND (*X*)

X

実数型でなければなりません。

result

*X*と同じ型です。

機能説明

SIND、DSIND、およびQSIND は、実数型データの正弦を求めます。

引数は度数値を指定します。

結果の値は、SIND(*X*) = SIN(π / 180 × *X*)となります。

単精度実数型の引数の場合、引数は ABS(*X*) < 4.72E+07 でなければなりません。

倍精度実数型の引数の場合、引数は DABS(*X*) < 2.03D+17 でなければなりません。

4倍精度実数型の引数の場合、引数は QABS(*X*) < 2.0Q0^{62} × 180 でなければなりません。

総称名SIND は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
r = sind(.5)
```

2.447 SINH組込み関数

SINH 関数は、双曲線正弦を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
SINH	-----	1	実数型 または 複素数型	実数型 または 複素数型
	SINH		単精度実数型	単精度実数型
	DSINH		倍精度実数型	倍精度実数型
	QSINH		4倍精度実数型	4倍精度実数型

result = SINH (*X*)

X

実数型または複素数型でなければなりません。

result

*X*と同じ型です。

機能説明

SINHは、実数型データまたは複素数型データの双曲線正弦を求めます。DSINHおよびQSINHは、実数型データの双曲線正弦を求めます。

単精度実数型の引数の場合、引数はABS(*X*) < 89.415E0でなければなりません。

倍精度実数型の引数の場合、引数はDABS(*X*) < 710.475D0でなければなりません。

4倍精度実数型の引数の場合、引数はQABS(*X*) < 11357.125Q0でなければなりません。

単精度複素数型の引数の場合、引数はABS(REAL(*X*)) < 89.415E0かつABS(AIMAG(*X*)) < 8.23E+05でなければなりません。

倍精度複素数型の引数の場合、引数はDABS(DREAL(*X*)) < 710.475D0かつDABS(DIMAG(*X*)) < 3.53D+15でなければなりません。

4倍精度複素数型の引数の場合、引数はQABS(QREAL(*X*)) < 11357.125Q0かつQABS(QIMAG(*X*)) < $2.0Q0^{62} \times \pi$でなければなりません。

総称名SINHは、すべての実数型および複素数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
r = sinh(.5)
```

2.448 SINQ組込み関数

SINQ関数は、象限値を引数とする正弦を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
SINQ	-----	1	実数型 または 複素数型	実数型 または 複素数型
	SINQ		単精度実数型	単精度実数型
	DSINQ		倍精度実数型	倍精度実数型
	QSINQ		4倍精度実数型	4倍精度実数型
	CSINQ		単精度複素数型	単精度複素数型
	CDSINQ		倍精度複素数型	倍精度複素数型
	CQSINQ		4倍精度複素数型	4倍精度複素数型

result = SINQ (*X*)

X

実数型または複素数型でなければなりません。

result

*X*と同じ型です。

機能説明

SINQ、DSINQ、QSINQ、CSINQ、CDSINQ、およびCQSINQは、実数型データまたは複素数型データの正弦を求めます。

引数は象限値を指定します。

結果の値は、SINQ(*X*) = SIN(π / 2 × *X*) となります。

単精度複素数型の引数の場合、ABS(REAL(*X*)) < 5.24E+05 かつ ABS(IMAG(*X*)) < 56.92E0 でなければなりません。

倍精度複素数型の引数の場合、DABS(DREAL(*X*)) < 2.25D+15 かつ DABS(DIMAG(*X*)) < 452.305D0 でなければなりません。

4倍精度複素数型の引数の場合、QABS(QREAL(*X*)) < $2.0Q0^{63}$かつ QABS(QIMAG(*X*)) < 7230.125Q0 でなければなりません。

総称名SINQ は、すべての実数型および複素数型の引数に使用することができます。

それぞれの関数の結果の型は、*X* と同じです。

使用例

```
r = sinq(.5)
```

2.449 SIZE組込み関数

SIZE 関数は、配列要素の数を求めます。

分類

問合せ関数

形式

result = SIZE (*ARRAY* [, *DIM* , *KIND*])

ARRAY

どの型でもかまいません。スカラであってはなりません。空状態のポインタまたは割り付けられていない割付け実体であってはなりません。

DIM (省略可能)

基本整数型スカラでなければなりません。1 <= *DIM* <= n の範囲の値でなければなりません。ここで、n は*ARRAY* の次元数です。*ARRAY* が大きさ引継ぎ配列であるときは、*DIM* は省略できず、*ARRAY* の次元数より小さい値でなければなりません。

KIND (省略可能)

スカラ整数定数式でなければなりません。

result

整数型スカラです。*KIND* が指定された場合、種別型パラメタは*KIND* の指定に従います。*KIND* が省略された場合、種別型パラメタは基本整数型になります。

機能説明

SIZE は、配列の要素の総数または指定した次元の寸法を返却します。

－ *DIM* を省略している場合

ARRAY の要素の総数を返却します。

－ *DIM* を指定している場合

ARRAY の *DIM* 次元目の大きさを返却します。

関数の結果の型は、種別型パラメタ *KIND* が指定された場合、種別型パラメタ *KIND* の整数型となります。*KIND* が省略された場合、関数の結果の型は基本整数型です。

使用例

```
integer, dimension(3,-4:0) :: i
integer :: k,j
j = size (i)                    ! j には15が代入されます
k = size (i,2)                  ! k には5が代入されます
```

2.450 SIZEOF組込み関数

SIZEOF 関数は、引数のサイズをバイト単位で返却します。

分類

問合せ関数

形式

result = SIZEOF (*X*)

X

型引継ぎを除き、どの型でもかまいません。大きさ引継ぎ配列、次元引継ぎ、空状態のポインタ、または割り付けられていない割付け実体であってはなりません。

result

8バイトの整数型のスカラです。

機能説明

SIZEOF は、引数のサイズをバイト単位で返却します。

引数がスカラの場合、実体のサイズを返却します。

引数が配列の場合、1要素のサイズと配列の要素数の乗数を返却します。

使用例

```
integer, dimension(3,-4:0) :: i
integer :: k
k = sizeof (i)                    ! k には60が代入されます。
```

2.451 SLEEPサービスサブルーチン

機能説明

現在のプロセスを中断させます。

形式

CALL SLEEP (*i*)

i

基本整数型スカラ。中断する時間を秒単位で指定します。

注意事項

実際に中断される時間は、指定した時間と一致しない場合があります。

使用例

```
use service_routines,only:sleep
call sleep(200)
end
```

2.452 SLITEサービスサブルーチン

機能説明

センスライトの点灯、消灯を行います。

形式

CALL SLITE (*i*)

i

基本整数型スカラ。センスライトを指定します。指定する値は以下の値です。以下の値以外を指定したときは、センスライトの状態は変更されません。

=0：4個のセンスライトを消灯します。
=1：センスライト1を点灯します。
=2：センスライト2を点灯します。
=3：センスライト3を点灯します。
=4：センスライト4を点灯します。

使用例

```
use service_routines,only:slite
call slite(1)
end
```

2.453 SLITETサービスサブルーチン

機能説明

センスライトの状態を確認します。その後、センスライトは消灯します。

形式

CALL SLITET（ *i* , *j* ）

i

基本整数型スカラ。センスライトを指定します。

指定する値は以下の値です。以下の値以外を指定したときは、センスライトの状態は変更されません。

=1：センスライト1の状態を確認します。
=2：センスライト2の状態を確認します。
=3：センスライト3の状態を確認します。
=4：センスライト4の状態を確認します。

j

基本整数型スカラ。センスライトの状態が設定されます。

以下の値が設定されます。*i* の値が誤っている場合は、不定となります。

=1：センスライトが点灯されています。
=2：センスライトが消灯されています。

使用例

```
use service_routines,only:slitet
integer :: j
call slitet(1,j)
print *,j
end
```

2.454 SPACING組込み関数

SPACING 関数は、引数前後の間隔を求めます。

分類

要素別処理関数

形式

result = SPACING（ *X* ）

X

実数型でなければなりません。

result

*X*と同じ型です。

機能説明

SPACING は、*X*を数体系の数として表現したときの引数値前後の間隔の絶対値を返却します。

関数の結果は、RADIX(*X*)$^{(EXPONENT(X)-DIGITS(X))}$ となり、その型は*X*と同じです。

*X*がIEEE無限大である場合、結果は非数になります。

*X*がIEEE非数である場合、結果は非数になります。

使用例

```
x = spacing(4.7)
```

2.455 SPREAD組込み関数

SPREAD 関数は、配列の複写を行います。

分類

変形関数

形式

result = SPREAD (*SOURCE* , *DIM* , *NCOPIES*)

SOURCE

どの型でもかまいません。スカラまたは6次元以下の配列でなければなりません。

DIM

整数型スカラであって、1 <= *DIM* <= n+1 の範囲の値でなければなりません。ここで、n は*SOURCE* の次元数とします。

NCOPIES

整数型スカラでなければなりません。

result

*SOURCE*と同じ型および同じ種別型パラメタです。*SOURCE* がスカラの場合、結果の形状は大きさ MAX(*NCOPIES*, 0) の1次元配列です。*SOURCE* が配列で、その形状が (d_1, d_2, ..., d_n) としたとき、結果の形状は
(d_1, d_2, ..., d_{DIM-1}, MAX(*NCOPIES*, 0), d_{DIM}, ..., d_n) となります。

機能説明

SPREAD は、配列を複製して次元を付け加えます。

結果の*DIM* 次元目に*SOURCE* を*NCOPIES* 回複写して、1次元だけ大きい配列を形成した配列を作成します。

－ *SOURCE* がスカラの場合

結果の各要素は*SOURCE* に等しい値となります。

－ *SOURCE* が配列の場合

結果の要素 (r_1, r_2, ..., r_{n+1}) は、*SOURCE*(r_1, r_2, ..., r_{DIM-1}, r_{DIM+1}, ..., r_{n+1}) となります。

使用例

```
integer, dimension(2) :: b=(/1,2/)
integer, dimension(2,3) :: a
a = spread(b,2,3)              ! a には       ┌       ┐
                               !              | 1 1 1 |
```

```
!                  | 2 2 2 |
!                  └       ┘ が代入されます
```

2.456 SQRT組込み関数

SQRT 関数は、平方根を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
SQRT	-----	1	実数型 または 複素数型	実数型 または 複素数型
	SQRT		単精度実数型	単精度実数型
	DSQRT		倍精度実数型	倍精度実数型
	QSQRT		4倍精度実数型	4倍精度実数型
	CSQRT		単精度複素数型	単精度複素数型
	CDSQRT		倍精度複素数型	倍精度複素数型
	CQSQRT		4倍精度複素数型	4倍精度複素数型

result = SQRT (*X*)

X

実数型または複素数型でなければなりません。*X* が実数型の場合、*X* >= 0.0 でなければなりません。

result

X と同じ型です。

機能説明

SQRT、DSQRT、QSQRT、CSQRT、CDSQRT、およびCQSQRT は、実数型データまたは複素数型データの平方根を求めます。

X が実数型の場合、*X* >= 0.0 でなければなりません。

総称名SQRT は、すべての実数型または複素数型の引数に使用することができます。

それぞれの関数の結果の型は、*X* と同じです。

使用例

```
x = sqrt(16.0)                ! x には4.0が代入されます
```

2.457 STATサービス関数

機能説明

ファイルの状態に関する情報を返却します。

形式

iy = STAT (*name* , *status*)

name

基本文字型スカラ。ファイル名を指定します。ファイル名の長さは、システムインクルードで定義されている最大長を超えてはなりません。

status

基本整数型配列。ファイルの状態に関する情報が設定されます。指定する配列変数は要素数が13以上の配列変数でなければなりません。配列の要素数が12以下の場合の動作は保証されません。

status (1)　: ファイル・モード
status (2)　: ファイルのiノード番号
status (3)　: デバイスのiノード番号
status (4)　: デバイス識別子（特殊ファイルだけ設定されます）
status (5)　: ファイルに対するハード・リンク数
status (6)　: オーナーのユーザID
status (7)　: オーナーのグループID
status (8)　: ファイルの合計サイズ（バイト単位）
status (9)　: ファイルの最後のアクセス時間
status (10)　: ファイルの最後の変更時間
status (11)　: ファイルの最後のステータス変更時間
status (12)　: ファイル・システムの最適なブロック・サイズ
status (13)　: 実際に割り当てられたブロック数

関数結果

基本整数型スカラ。ファイル情報が返却できた時は0、エラーが発生したときは0以外の値が返却されます。

使用例

```
use service_routines,only:stat
integer :: st(13)
print *,stat('xxx.dat',st)
end
```

2.458 STAT64サービス関数

機能説明

ファイルの状態に関する情報を返却します。

形式

iy = STAT64 (*name* , *status*)

name

基本文字型スカラ。ファイル名を指定します。ファイル名の長さは、システムインクルードで定義されている最大長を超えてはなりません。

status

8バイトの整数型配列。ファイルの状態に関する情報が設定されます。指定する配列変数は要素数が13以上の配列変数でなければなりません。配列の要素数が12以下の場合の動作は保証されません。

status (1)　: ファイル・モード
status (2)　: ファイルのiノード番号
status (3)　: デバイスのiノード番号
status (4)　: デバイス識別子（特殊ファイルだけ設定されます）
status (5)　: ファイルに対するハード・リンク数
status (6)　: オーナーのユーザID
status (7)　: オーナーのグループID
status (8)　: ファイルの合計サイズ（バイト単位）
status (9)　: ファイルの最後のアクセス時間
status (10)　: ファイルの最後の変更時間
status (11)　: ファイルの最後のステータス変更時間
status (12)　: ファイル・システムの最適なブロック・サイズ
status (13)　: 実際に割り当てられたブロック数

関数結果

基本整数型スカラ。ファイル情報が返却できた時は0、エラーが発生したときは0以外の値が返却されます。

使用例

```
use service_routines,only:stat64
integer(kind=8) :: st(13)
```

```
print *,stat64('xxx.dat',st)
end
```

2.459 STATIC文

STATIC 文は、実体をメモリ上に割り付け、それらがSAVE 属性をもつことを指定します。

STATIC 文は、以下の形式です。

```
STATIC [ [ :: ] object-name-list ]
```

object-name-list はコンマで区切られた実体名の並びです。*object-name* は、仮引数名、手続名、関数結果の名前、自動割付け変数の名前、結合実体の名前、および共通ブロック実体の名前であってはなりません。

object-name-list のないSTATIC 文は、その有効域内のすべての可能な項目を指定したかのように扱われます。

STATIC 属性をもつ実体は、暗黙的にSAVE 属性をもちます。

STATIC 文はモジュールの宣言部および初期値設定プログラム単位には指定できません。

STATIC 文の例:

```
subroutine sub
static :: i,j              ! i およびj は、メモリ上に割り付けられます
```

2.460 STOP文

STOP 文は、プログラムの実行を正常終了します。

STOP 文は、以下の形式です。

```
STOP [ stop-code ]
```

stop-code は終了符号であり、以下の形式です。

```
scalar-char-constant                                  または
digit [ digit [ digit [ digit [ digit ] ] ] ]         または
scalar-default-char-initialization-expr               または
scalar-int-initialization-expr
```

scalar-char-constant は、スカラ文字定数です。スカラ文字定数は、基本文字型でなければなりません。

digit は、数字です。

scalar-default-char-initialization-expr は、スカラ基本文字型の定数式です。

scalar-int-initialization-expr は、スカラ整数型の定数式です。

STOP 文を実行すると、プログラムの実行を正常終了します。終了符号が指定されている場合は、標準エラー出力ファイルに対して診断メッセージを出力します。

STOP 文の例:

```
if (a > b) then
stop                    ! プログラムの実行を終了します
end if
```

2.461 STORAGE_SIZE組込み関数

STORAGE_SIZE 関数は、サイズをビット単位で返却します。

分類

問合せ関数

形式

result = STORAGE_SIZE (*A* [, *KIND*])

A

どの型でもかまいません。多相的である場合、それは不定なポインタであってはなりません。無指定型パラメタを含む場合、未割付けの割付け変数、空状態ポインタまたは不定ポインタであってはなりません。

KIND (省略可能)

スカラ整数定数式でなければなりません。

result

整数型のスカラです。*KIND*が指定された場合、種別型パラメタは*KIND*の指定に従います。*KIND*が省略された場合、種別型パラメタは基本整数型になります。

機能説明

A の動的な型のビット単位の大きさを求めます。配列の場合、1要素のビット単位の大きさです。

使用例

```
integer,dimension(100)::i
k = storage_size(i)                ! k には32が代入されます
```

2.462 STRUCTURE文

STRUCTURE 文は、派生型の定義を開始します。派生型定義の詳細については、“1.5.11.1 派生型定義”を参照してください。

STRUCTURE 文は、以下の形式です。

```
STRUCTURE [ / type-name / ] [ component-list ]
```

type-name は、このSTRUCTURE 文で定義される派生型名です。/ *type-name* / は、派生型定義が入れ子になっている場合の内側のSTRUCTURE 文に対してだけ、省略することができます。

component-list は、入れ子になっている場合の内側のSTRUCTURE 文に対してだけ指定することができます。*component* はこのSTRUCTURE 文で定義される型をもつ成分です。

STRUCTURE 文によって定義される派生型は、連続型です。

STRUCTURE 文の例:

```
structure /complex_element/
  union
    map
      real :: real,imag
    end map
    map
      complex :: complex
    end map
  end union
end structure
record /complex_element/ x
x%real = 2.0
x%imag = 3.0
print *,x%complex              ! 複素数型の要素complex は、(2.0,3.0) で確定しています
```

2.463 SUBMODULE文

SUBMODULE文は、サブモジュールを開始します。サブモジュールについては、“1.11.3 サブモジュール”を参照してください。

SUBMODULE文は、以下の形式です。

```
SUBMODULE ( parent-identifier ) submodule-name
```

*parent-identifier*は、親識別子です。

*submodule-name*は、サブモジュール名です。

サブモジュール名は、大域名と局所名のどちらでもありません。

SUBMODULE文の例:

```
module mod
  interface
    module subroutine sub1(val)  ! 分離引用仕様本体
      integer, intent(in) :: val
    end subroutine sub1

    module subroutine sub2(val)  ! 分離引用仕様本体
      integer, intent(in) :: val
    end subroutine sub2
  end interface
end module

submodule(mod)submod
  contains
    module subroutine sub1(val)   ! 分離モジュール副プログラム
      integer, intent(in) :: val  ! 特性および仮引数名は上記のsub1と一致させる
    end subroutine sub1

    module procedure sub2         ! module procedure文を使用すると
    end procedure sub2            ! 特性および引数を省略できる
end submodule submod
```

2.464 SUBROUTINE文

SUBROUTINE 文は、サブルーチン副プログラムを開始し、サブルーチンの特性を宣言します。サブルーチン副プログラムについては、“1.12.2 サブルーチン副プログラム”を参照してください。

SUBROUTINE 文は、以下の形式です。

```
[ prefix-spec ]... SUBROUTINE subroutine-name [ ( [ dummy-arg-list ] ) proc-language-binding-spec ]
```

prefix-spec は、以下の形式です。

```
ELEMENTAL              または
IMPURE                 または
MODULE                 または
PURE                   または
RECURSIVE
```

ELEMENTAL は、そのサブルーチンが要素別処理サブルーチンであることを宣言します。

‘IMPURE’は、そのサブルーチンが純粋サブルーチンでないことを宣言します。

‘MODULE’を指定したサブルーチン副プログラムは、分離モジュール手続です(“1.11.3 サブモジュール”参照)。

‘PURE’は、そのサブルーチンが純粋サブルーチンであることを宣言します。

‘RECURSIVE’は、そのサブルーチンが再帰的サブルーチンであることを宣言します。

‘ELEMENTAL’と‘RECURSIVE’の両方を指定することはできません。

‘IMPURE’と‘PURE’の両方を指定することはできません。

詳細については、“1.12.3 再帰的引用”、“1.12.4 純粋手続”、または“1.12.5 要素別処理手続”を参照してください。

subroutine-name は、サブルーチン名です。

dummy-arg-list は、コンマで区切られた仮引数の並びです。

dummy-arg は、以下の形式です。

dummy-arg-name または
* （廃止予定事項）

dummy-arg-name は、仮引数名です。

proc-language-binding-spec は手続言語束縛指定子であり、以下の形式です。

BIND （ C [, NAME = スカラ文字定数式] ）

SUBROUTINE 文の例:

```
pure subroutine zee (var1,var2)
  ...
end subroutine
```

2.465 SUM組込み関数

SUM 関数は、配列内の和を求めます。

分類

変形関数

形式

result = SUM （ *ARRAY* [, *MASK*] ） または
result = SUM （ *ARRAY* , *DIM* [, *MASK*] ）

ARRAY

整数型、実数型または複素数型でなければなりません。スカラであってはなりません。

DIM

整数型スカラであって、1 <= *DIM*<= n の範囲の値でなければなりません。ここで、n は*ARRAY*の次元数とします。対応する実引数は、省略可能な仮引数であってはなりません。

MASK **(省略可能)**

論理型でなければならず、*ARRAY*と形状適合しなければなりません。

result

*ARRAY*と同じ型および同じ種別型パラメタです。*DIM*が省略されている、または*ARRAY*が1次元配列の場合は、スカラとなります。それ以外の場合は、*ARRAY*の形状を(d_1, d_2, ..., d_n)としたとき、(d_1, d_2, ..., d_{DIM-1}, d_{DIM+1}, ..., d_n) となります。n は*ARRAY*の次元数です。

機能説明

SUM は、*ARRAY*の第*DIM* 次元の要素の和を計算します。

－ *DIM*を省略している場合

MASK 中の真の要素に対応する、*ARRAY*の全要素の和を返却します。
*ARRAY*の大きさが0または*MASK* 中の全要素が偽である場合、*ARRAY*が整数型または実数型の場合は0、複素数型の場合は (0.0, 0.0) を返却します。

－ *DIM*を指定している場合

*ARRAY*が1次元の場合は、SUM(*ARRAY* [,*MASK*]) となります。
*ARRAY*が2次元以上の場合、結果の要素(S_1, S_2, ..., S_{DIM-1}, S_{DIM+1}, ..., S_n) は、
SUM(*ARRAY*(S_1, S_2, ..., S_{DIM-1}, : , S_{DIM+1}, ..., S_n), DIM = 1[,MASK = *MASK*]) となります。ここで、n は*ARRAY*の次元数、MS は、*MASK* がスカラの場合は*MASK*、配列の場合は*MASK*(S_1, S_2, ..., S_{DIM-1}, : , S_{DIM+1}, ..., S_n) とします。

関数の結果の型は、*ARRAY*と同じです。

使用例

```
integer :: j(2)
integer ,dimension(2,2) :: m = reshape((/1,2,3,4/),(/2,2/))
                           ! m は          ┌    ┐
                           !               | 1 3 |
                           !               | 2 4 |
                           !               └    ┘
i = sum(m)                 ! i には10が代入されます
j = sum(m,dim=1)           ! j には(/3,7/) が代入されます
k = sum(m,mask=m>2)        ! k には7が代入されます
```

2.466 SYMLNKサービス関数

機能説明

シンボリックリンクを作成します。

形式

iy = SYMLNK (*path1* , *path2*)

path1

基本文字型スカラ。既存のファイルを指定します。指定するファイル名は既存ファイルでなければなりません。

path2

基本文字型スカラ。シンボリックリンク名を指定します。

関数結果

基本整数型スカラ。シンボリックリンクを作成できたときは0、エラーが発生したときは0以外の値が返却されます。

使用例

```
use service_routines,only:symlnk
i=symlnk('libx.so.1','../libx.so')
end
```

2.467 SYNC ALL文

SYNC ALL 文は、以下の形式です。

```
SYNC ALL [ ( [ sync-stat-list ] ) ]
```

sync-stat-list はコンマで区切られた同期状態指定子の並びです。*sync-stat* は以下の形式です。

```
STAT = stat-variable
ERRMSG = errmsg-variable
```

stat-variable は状態変数です。*errmsg-variable* は誤り通報変数です。1つの*sync-stat-list* に2回以上同じ指定子が現れてはなりません。*stat-variable* および*errmsg-variable* については“1.18.2 同期状態指定子”を参照してください。

SYNC ALL 文を実行すると、すべての像で同期を取ります。像M におけるSYNC ALL 文の後のセグメントの実行は、像M が実行してきたSYNC ALL 文の回数と同じだけ各像がSYNC ALL 文を実行するまで延期されます。SYNC ALL 文より前に実行されるセグメントは、他の像のSYNC ALL 文より後のセグメントに先行します。

SYNC ALL 文の例:

```
integer :: x[*]
x = 1
sync all
```

```
print *,x[1]  ! すべての像で1が出力されます
end
```

2.468 SYNC IMAGES文

SYNC IMAGES 文は以下の形式です。

```
SYNC IMAGES ( image-set [ , sync-stat-list ] )
```

image-set は像集合であり、以下の形式です。

```
int-expr                    または
*
```

int-expr は整数式であり、スカラまたは1次元でなければなりません。

sync-stat は同期状態指定子であり、以下の形式です。

```
STAT = stat-variable          または
ERRMSG = errmsg-variable
```

stat-variable および*errmsg-variable* については“1.18.2 同期状態指定子”を参照してください。

像集合が配列式の場合、各要素の値は正でなければならず、像の数より大きくてはならず、かつ同じ値が2回以上現れてはなりません。

像集合がスカラ式の場合、値は正でなければならず、かつ像の数より大きくてはなりません。

像集合に星印‘*’を指定した場合、すべての像を現します。

SYNC IMAGES 文を実行すると、像集合に現れる各像と同期を取ります。像Mにおいて像集合にTをもつSYNC IMAGES 文を実行した回数と、像Tにおいて像集合にMをもつSYNC IMAGES 文を実行した回数が同じであるとき、像Tと像MのSYNC IMAGES 文は対応します。SYNC IMAGES 文より先に実行されるセグメントは、他の像の対応するSYNC IMAGES 文より後に実行されるセグメントに先行します。

SYNC IMAGES文の例:

```
if (this_image() == 1 .or. this_image() == 2) then
  sync images([1,2])   ! 像1と像2は互いに同期を取ります
end if
end
```

2.469 SYNC MEMORY文

SYNC MEMORY 文は以下の形式です。

```
SYNC MEMORY [ ( [ sync-stat-list ] ) ]
```

sync-stat-list はコンマで区切られた同期状態指定子の並びです。*sync-stat* は以下の形式です。

```
STAT = stat-variable
ERRMSG = errmsg-variable
```

stat-variable は状態変数です。*errmsg-variable* は誤り通報変数です。1つの*sync-stat-list* に2回以上同じ指定子が現れてはなりません。*stat-variable* および*errmsg-variable* については“1.18.2 同期状態指定子”を参照してください。

SYNC MEMORY 文はセグメントを終了し、新しいセグメントを開始します。終了したセグメントと開始したセグメントは、他の像のそれらのセグメントに対して使用者定義の方法で順番付けをできます。セグメントについては“1.18.1 セグメント”を参照してください。

像P の文の実行によって以下の状態になった場合、像Q の変数X に対する操作は、像Q の変数Y に対する操作に先行します。

- 像Q の変数X が定義、参照、または未定義になるか、割付け状態、ポインタ結合の状態、配列の上下限、動的な型、または型パラメタが文の実行によって変更されるか、または問い合わされる。かつ、

- その文が、SYNC MEMORY 文に先行する。かつ、
- 像Q の変数Y が定義、参照、または未定義になるか、割付け状態、ポインタ結合の状態、配列の上下限、動的な型、または型パラメタが、そのSYNC MEMORY 文より後の文の実行によって変更されるか、または問い合わされる。

以下の場合、像P のセグメントP_i が像Q のセグメントQ_j に先行するという使用者定義の順番付けが発生します。

- 像Pがセグメント P_i を終わらせる像制御文を実行し、その後、像P と像Q の同期する文を実行する。かつ
- 像Q が像P との同期を完了させ、その後セグメントQ_j を開始する像制御文を実行する。

像P と像Q の同期の実行は、像P で同期を開始する文が、像Q で同期を完了させる文に先行することを強制する依存を含まなければなりません。

SYNC MEMORY文の例:

```
integer, save :: i[*], k[*]
i=1
sync memory
k=1             ! 各像において、必ず i=1が先に実行されます
end
```

2.470 SYSTEMサービス関数

機能説明

第1引数で指定したコマンドをシェルの入力として実行します。

形式

iy = SYSTEM (*ch*)

ch

基本文字型スカラ。実行するコマンド名を指定します。

関数結果

基本整数型スカラ。シェルの終了ステータスが返却されます。

注意事項

現在のプロセスは、コマンドの実行が完了するまで待ち状態となります。

使用例

```
use service_routines,only:system
if ( system('x.sh') /= 0 ) stop 'shell error'
call sub( )
end
```

2.471 SYSTEM_CLOCK組込みサブルーチン

実時間時計から数値データを返します。

分類

サブルーチン

形式

CALL SYSTEM_CLOCK ([*COUNT* , *COUNT_RATE* , *COUNT_MAX*])

COUNT (省略可能)

整数型スカラでなければなりません。INTENT(OUT) の引数です。午前0時から現在までに処理系のクロックが刻んだ回数が設定されます。

COUNT の値を取得できない場合は-HUGE (*COUNT*) が設定されます。

COUNT_RATE (省略可能)

整数型または実数型のスカラでなければなりません。INTENT(OUT) の引数です。1秒間に処理系が刻む回数が設定されます。

COUNT の値を取得できない場合は0が設定されます。

COUNT_MAX (省略可能)

整数型スカラでなければなりません。INTENT(OUT) の引数です。*COUNT* の最大値が設定されます。

COUNT の値を取得できない場合は0が設定されます。

使用例

```
integer c,cr,cm
call system_clock(c,cr,cm) ! c には処理系のクロックが刻んだ回数が、
                           ! cr には1秒間に処理系が刻む回数が、
                           ! cm にはCOUNTの最大値が、
                           ! 設定されます
```

2.472 TAN組込み関数

TAN 関数は、ラジアン値を引数とする正接を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
TAN	-----	1	実数型 または 複素数型	実数型 または 複素数型
	TAN		単精度実数型	単精度実数型
	DTAN		倍精度実数型	倍精度実数型
	QTAN		4倍精度実数型	4倍精度実数型

result = TAN (*X*)

X

実数型または複素数型でなければなりません。

result

X と同じ型です。

機能説明

TAN は、実数型データまたは複素数型データの正接を求めます。DTAN およびQTAN は、実数型データの正接を求めます。

引数はラジアン値を指定します。

単精度実数型の引数の場合、引数は ABS(*X*) < 8.23E+05 でなければなりません。

倍精度実数型の引数の場合、引数は DABS(*X*) < 3.53D+15 でなければなりません。

4倍精度実数型の引数の場合、引数は QABS(*X*) < 2.0Q0^{62} × π でなければなりません。

単精度複素数型の引数の場合、引数は ABS(REAL(*X*)) < 8.23E+05 でなければなりません。

倍精度複素数型の引数の場合、引数は DABS(DREAL(*X*)) < 3.53D+15 でなければなりません。

4倍精度複素数型の引数の場合、引数は QABS(QREAL(*X*)) < 2.0Q0^{62} × π でなければなりません。

総称名TAN は、すべての実数型および複素数型の引数に使用することができます。

それぞれの関数の結果の型は、*X* と同じです。

使用例

```
r = tan(.5)
```

2.473 TAND組込み関数

TAND 関数は、度数値を引数とする正接を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
TAND	-----	1	実数型	実数型
	TAND		単精度実数型	単精度実数型
	DTAND		倍精度実数型	倍精度実数型
	QTAND		4倍精度実数型	4倍精度実数型

result = TAND (*X*)

X

実数型でなければなりません。

result

*X*と同じ型です。

機能説明

TAND、DTAND、およびQTAND は、実数型データの正接を求めます。

引数は度数値を指定します。

結果の値は、TAND(*X*) = TAN(π / 180 × *X*) となります。

単精度実数型の引数の場合、引数は ABS(*X*) < 4.72E+07 でなければなりません。

倍精度実数型の引数の場合、引数は DABS(*X*) < 2.03D+17 でなければなりません。

4倍精度実数型の引数の場合、引数は QABS(*X*) < $2.0Q0^{62}$ × 180 でなければなりません。

総称名TAND は、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
r = tand(.5)
```

2.474 TANH組込み関数

TANH 関数は、双曲線正接を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
TANH	-----	1	実数型 または 複素数型	実数型 または 複素数型
	TANH		単精度実数型	単精度実数型

総称名	個別名	引数の数	引数の型	結果の型
	DTANH		倍精度実数型	倍精度実数型
	QTANH		4倍精度実数型	4倍精度実数型

result = TANH (*X*)

X

実数型または複素数型でなければなりません。

result

*X*と同じ型です。

機能説明

TANHは、実数型データまたは複素数型データの双曲線正接を求めます。DTANHおよびQTANHは、実数型データの双曲線正接を求めます。

単精度複素数型の引数の場合、引数はABS(AIMAG(*X*)) < 8.23E+05でなければなりません。

倍精度複素数型の引数の場合、引数はDABS(DIMAG(*X*)) < 3.53D+15でなければなりません。

4倍精度複素数型の引数の場合、引数はQABS(QIMAG(*X*)) < 2.0Q0^{62} × π でなければなりません。

総称名TANHは、すべての実数型および複素数型の引数に使用することができます。

それぞれの関数の結果の型は、*X*と同じです。

使用例

```
r = tanh(.5)
```

2.475 TANQ組込み関数

TANQ関数は、象限値を引数とする正接を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
TANQ	-----	1	実数型	実数型
	TANQ		単精度実数型	単精度実数型
	DTANQ		倍精度実数型	倍精度実数型
	QTANQ		4倍精度実数型	4倍精度実数型

result = TANQ (*X*)

X

実数型でなければなりません。

result

*X*と同じ型です。

機能説明

TANQ、DTANQ、およびQTANQは、実数型データの正接を求めます。

引数は象限値を指定します。

結果の値は、TANQ(*X*) = TAN(π / 2 × *X*)となります。

総称名TANQは、すべての実数型の引数に使用することができます。

それぞれの関数の結果の型は、X と同じです。

使用例

```
r = tanq(.5)
```

2.476 TARGET文

TARGET 文は、ポインタと結合できるTARGET 属性をもつ実体を宣言します。

TARGET 文は、以下の形式です。

```
TARGET [ :: ] target-decl-list
```

target-decl-list は、TARGET属性宣言並びです。*target-decl* は、以下の形式です。

```
object-name [ ( array-spec ) ] [ left-square-bracket coarray-spec right-square-bracket ]
```

object-name は実体名です。

array-spec は、配列形状指定です。配列形状指定の詳細については、“2.119 DIMENSION文”を参照してください。

coarray-spec は、共配列指定です。詳細については“1.17.1 共配列指定”を参照してください。

left-square-bracket は、左角括弧です。左角括弧は、[です。

right-square-bracket は、右角括弧です。右角括弧は、] です。

TARGET 文の例:

```
target :: a,b,c                ! a、b、およびc は、TARGET 属性をもつ実体です
```

2.477 THIS_IMAGE組込み関数

THIS_IMAGE関数は呼び出した像に対応する共添字を返却します。

分類

変形関数

形式

```
result = THIS_IMAGE ( )                    または

result = THIS_IMAGE ( COARRAY [, DIM ] )
```

COARRAY

共配列でなければなりません。割付け可能である場合は割り付けられていなければなりません。

DIM **(省略可能)**

基本整数型のスカラであって、1 <= *DIM* <= nの範囲の値でなければなりません。ここで、nは*COARRAY*の共次元数でなければなりません。対応する実引数は、省略可能であってはなりません。

result

基本整数型です。*COARRAY*が省略された場合、関数を呼び出した像番号を値としたスカラとなります。*COARRAY*のみ指定された場合、*COARRAY*に対応する呼び出した像を示す共添字を値とする1次元配列になります。*COARRAY*と*DIM*が指定された場合、*COARRAY*の共添字*DIM*に指定すると呼び出した像を指定するような値をとるスカラとなります。

使用例

```
if (this_image() == 1) then
  print *,'image 1' ! 1番目の像の実行だけで出力されます
end if
```

2.478 TIMEサービス関数

機能説明

00:00:00 GMT(1970年1月1日)から経過した秒数を返却します。

形式

iy = TIME ( )

関数結果

基本整数型スカラ。00:00:00 GMT(1970年1月1日)から経過した秒数が秒単位で返却されます。

使用例

```
use service_routines,only:time
integer :: iy
iy = time( )
end
```

2.479 TIMEFサービス関数

機能説明

直前に呼ばれたTIMEF サービス関数からの経過時間を返却します。

形式

dy = TIMEF ( )

関数結果

倍精度実数型スカラ。エラーが発生した場合、-1.0が返却されます。

返却される実行時間は、以下のようになります。

- 最初にTIMEF サービス関数を実行した場合、0.0D0が返却されます。
- 2回目以降は、直前に実行されたTIMEF サービス関数からの実行時間が秒単位で返却されます。

使用例

```
use service_routines,only:timef
real(kind=8) :: y
y = timef( )
do i=1,5000
  write(*,*) i,i*i
end do
y = timef( )
end
```

2.480 TIMERサービスサブルーチン

機能説明

午前0時からの通算1/100秒を取得します。

形式

CALL TIMER (*ix*)

ix

基本整数型スカラ。取得した時間を設定する領域です。時間が取得できない場合、-1が設定されます。

使用例

```
use service_routines,only:timer
integer :: ix
call timer(ix)
write(6,*) ix
end
```

2.481 TINY組込み関数

TINY 関数は、正の最小の値を求めます。

分類

問合せ関数

形式

result = TINY (*X*)

X

実数型でなければなりません。スカラまたは配列です。

result

*X*と同じ型および種別型パラメタをもつスカラです。

機能説明

TINY は、*X*と同じ型および種別型パラメタが取りうる正の最小値を返却します。

関数の結果は RADIX(*X*)$^{(MINEXPONENT(X)-1)}$ となり、その型は*X*と同じです。

以下の値の固定値となります。

引数の型	結果の値
半精度実数型	6.1035E-05
単精度実数型	1.17549435E-38
倍精度実数型	2.225073858507201D-308
4倍精度実数型	3.362103143112093506262677817321752 6Q-4932

使用例

```
a=tiny(4.0)          ! a には1.17549435E-38が代入されます
```

2.482 TRAILZ組込み関数

TRAILZ 関数は、後続する0のビット数を求めます。

分類

要素別処理関数

形式

総称名	個別名	引数の数	引数の型	結果の型
TRAILZ	-----	1	整数型	基本整数型

result = TRAILZ (*I*)

I

整数型でなければなりません。

result

基本整数型です。

機能説明

後続する0のビット数を求めます。*I*の値が0である場合、結果は値BIT_SIZE(*I*) です。

関数の結果の型は、基本整数型です。

使用例

```
k = trailz( -1 )          ! k には0が代入されます
m = trailz( 0 )           ! m には32が代入されます
n = trailz( 8 )           ! n には3が代入されます
```

2.483 TRANSFER組込み関数

TRANSFER 関数は、物理表現が同じであるデータの作成を行います。

分類

変形関数

形式

result = TRANSFER (*SOURCE* , *MOLD* [, *SIZE*])

SOURCE

どの型でもかまいません。

MOLD

どの型でもかまいません。

SIZE（省略可能）

整数型スカラでなければなりません。対応する実引数は、省略可能な仮引数であってはなりません。

result

*MOLD*と同じ型および同じ種別型パラメタです。

*SIZE*を省略し、*MOLD*がスカラである場合、スカラです。*SIZE*を省略し、*MOLD*が配列である場合、物理表現が*SOURCE*よりも短くならない範囲の大きさの1次元配列です。

SIZE が指定された場合、大きさが*SIZE* の1次元配列です。

機能説明

TRANSFER は、物理表現が*SOURCE*と同一であるデータを*MOLD* の種別型パラメタに変換した値を返却します。

－ 結果の物理表現が、*SOURCE* の長さと同じ、または短い場合

結果の物理表現は*SOURCE* の先頭部分と同じです。

－ 結果の物理表現が、*SOURCE* の長さよりも長い場合

結果の先頭部分の物理表現は*SOURCE* と同じです。残りの部分は不定となります。

関数の結果の型は、*MOLD*と同じです。

使用例

```
real :: a
integer :: i
a = transfer(i,a)         ! a には物理表現されたi が代入されます
```

2.484 TRANSPOSE組込み関数

TRANSPOSE 関数は、行列を転置します。

分類

変形関数

形式

result = TRANSPOSE (*MATRIX*)

MATRIX

どの型でもかまいません。2次元配列でなければなりません。

result

*MATRIX*と同じ型および同じ種別型パラメタの2次元配列です。
*MATRIX*の形状を(n,m) としたとき、結果の形状は(m,n) となります。

機能説明

TRANSPOSE は、*MATRIX*の転置行列を作成します。

関数の結果の要素(I,J) は、*MATRIX*(J,I) となります。

関数の結果の型は、*MATRIX*と同じです。

使用例

```
integer, dimension(2,3):: a = reshape((/1,2,3,4,5,6/),(/2,3/))
                  ! a は          ┌       ┐
                  !               | 1 3 5 |
                  !               | 2 4 6 |
                  !               └       ┘
integer, dimension(3,2) :: b
b = transpose(a)  ! b には        ┌     ┐
                  !               | 1 2 |
                  !               | 3 4 |
                  !               | 5 6 |
                  !               └     ┘ が代入されます
```

2.485 TRIM組込み関数

TRIM 関数は、末尾の空白を取り除きます。

分類

変形関数

形式

result = TRIM (*STRING*)

STRING

文字型スカラでなければなりません。

result

*STRING*と同じ型および同じ種別型パラメタの文字型です。文字長は、*STRING*の文字長から*STRING*の末尾の空白をすべて取り除いた長さと同じです。

機能説明

TRIM は、*STRING*の末尾の空白をすべて取り除いた文字列を返却します。*STRING*がすべて空白である場合、結果の文字長は0とします。

関数の結果の型は、*STRING*と同じ種別型パラメタをもつ文字型です。

使用例

```
character(len=6) shorter
shorter = trim("longer ")          ! shorter に"longer" が代入されます
```

2.486 TTYNAMサービス関数

機能説明

装置番号と結合している端末装置名のパス名を返却します。

形式

name = TTYNAM (*unit*)

unit

基本整数型スカラ。装置番号を指定します。

関数結果

基本文字型スカラ。装置番号と結合している端末装置名のパス名を返却します。

使用例

```
use service_routines,only:ttynam,isatty
if (isatty(6)) print *,ttynam(6)
end
```

2.487 TYPE文(派生型定義)

TYPE 文は、派生型の定義を開始します。派生型定義の詳細については、"1.5.11.1 派生型定義"を参照してください。

TYPE 文は、以下の形式です。

```
TYPE [ [ , type-attr-spec-list ] :: ] type-name [ ( type-param-name-list ) ]
```

type-attr-spec-list は、以下の形式です。

```
access-spec                     または
EXTENDS ( parent-type-name )    または
ABSTRACT                        または
BIND ( C )
```

access-spec は参照許可指定子であり、以下の形式です。

```
PUBLIC                          または
PRIVATE
```

access-spec は、モジュールの宣言部の型定義にだけ指定できます。

parent-type-name は、親型名です。

type-name は、このTYPE 文で定義される派生型名です。*type-name* は、組込み型の名前とも、参照可能な他の派生型の型名とも同じであってはなりません。*type-param-name-list* に型パラメタ名並びを指定できます。

TYPE 文の例:

```
type coordinates
real :: x,y = 40.0          ! y には、暗黙的初期値指定として40.0を設定します
end type coordinates
```

2.488 TYPE型宣言文

TYPE 型宣言文は、指定された派生型'*type-name*'のデータ実体を宣言します。

TYPE 型宣言文は、以下の形式です。

```
TYPE ( type-name ) [ [ , attr-spec ]... :: ] entity-decl-list
```

型宣言文の詳細については、“2.3 型宣言文”を参照してください。

2.489 TYPE IS文

TYPE IS 文は、SELECT TYPE 構文中に指定します。

TYPE IS 文は、以下の形式です。

```
TYPE IS( type-spec ) [ select-construct-name ]
```

type-spec は、型指定子です。型指定子には、組込み型指定子および派生型指定子があります。

組込み型指定子については、“2.3 型宣言文”を参照してください。派生型指定子については、“1.5.11.8 派生型指定子”を参照してください。

select-construct-name は、SELECT 構文名です。

SELECT TYPE 構文の詳細については、“2.431 SELECT TYPE構文”を参照してください。

2.490 UBOUND組込み関数

UBOUND 関数は、配列添字の上限を求めます。

分類

問合せ関数

形式

```
result = UBOUND ( ARRAY [ , DIM , KIND ] )
```

ARRAY

スカラであってはなりません。空状態のポインタまたは割り付けられていない割付け配列であってはなりません。

DIM（省略可能）

整数型スカラであって、1 <= *DIM*<= n の範囲の値でなければなりません。ここで、n は*ARRAY*の次元数とします。対応する実引数は、省略可能な仮引数であってはなりません。

ARRAY が大きさ引継ぎ配列の場合、*DIM* の値は*ARRAY* の次元数未満でなければなりません。

KIND（省略可能）

スカラ整数定数式でなければなりません。

result

整数型です。*KIND* が指定された場合、種別型パラメタは*KIND* の指定に従います。*KIND* が省略された場合、種別型パラメタは基本整数型になります。*DIM* が指定されている場合はスカラとなり、それ以外のときは大きさn の1次元の配列です。ここで、n は *ARRAY* の次元数とします。

機能説明

UBOUND は、配列添字の上限を返却します。

－ *DIM*を指定している場合

ARRAY の第*DIM* 次元目の添字の上限値を返却します。第*DIM* 次元目の大きさが0の場合は、0を返却します。

－ *DIM*を省略している場合

*ARRAY*のすべての添字の上限値 (/(UBOUND (*ARRAY*, I), I = 1,n)/)を返却します。ここで、n は*ARRAY* の次元数とします。

関数の結果の型は、種別型パラメタ*KIND* が指定された場合、種別型パラメタ*KIND* の整数型となります。*KIND* が省略された場合、関数の結果の型は基本整数型です。

使用例

```
integer, dimension (3,-4:0) :: i
integer :: k,j(2)
```

```
j = ubound(i)              ! j には(/3,0/) が代入されます
k = ubound(i,2)            ! k には0が代入されます
```

2.491 UCOBOUND組込み関数

UCOBOUND 関数は、共配列の共上限を返却します。

分類

問合せ関数

形式

result = UCOBOUND (*COARRAY* [, *DIM*, *KIND*])

COARRAY

共配列でなければなりません。割付け可能である場合は割り付けられていなければなりません。

DIM (省略可能)

整数型スカラであって、1 <= *DIM*<= n の範囲でなければなりません。ここで、n は*COARRAY*の共次元数です。対応する実引数は、省略可能な仮引数であってはなりません。

KIND (省略可能)

スカラ整数型定数式でなければなりません。

result

整数型です。*KIND*が指定された場合、種別型パラメタは*KIND*の値となり、そうでない場合は基本整数型となります。*DIM*が指定された場合、*COARRAY*の共添字*DIM*に対する共上限を値とするスカラとなります。そうでない場合は大きさがn の1次元配列となります。ここで、n は*COARRAY* の共次元数です。その場合、各要素の値は、それぞれの共添字の共上限となります。

使用例

```
integer, save :: k[2,3:4,*]
if (num_images()==10) then
  print *,ucobound(k,dim=1) ! 2が出力されます
  print *,ucobound(k,dim=2) ! 4が出力されます
  print *,ucobound(k,dim=3) ! 3が出力されます
end if
```

2.492 UNION文

UNION 文は、STRUCTURE 文による派生型定義内において、共用体宣言を開始します。派生型定義の詳細については、“1.5.11.1 派生型定義”を参照してください。

UNION 文は、以下の形式です。

```
UNION
```

UNION 文の例:

```
structure /complex_element/
  union
    map
      real :: real,imag
    end map
    map
      complex :: complex
    end map
  end union
end structure
record /complex_element/ x
x%real = 2.0
```

```
x%imag = 3.0
print *,x%complex              ! 複素数型の要素complex は、(2.0,3.0) で確定しています
```

2.493 UNLINKサービス関数

機能説明

第1引数で指定されたディレクトリエントリーを取り除き、ディレクトリエントリーによって参照されているファイルのリンク総数を減らします。

形式

iy = UNLINK (*ch*)

ch

基本文字型スカラ。ディレクトリ名を指定します。
ディレクトリ名の長さは、システムインクルードで定義されている最大長を超えてはなりません。

関数結果

基本整数型スカラ。ディレクトリエントリーを削除できたときは0、エラーが発生したときは-1が返却されます。

使用例

```
use service_routines,only:unlink
integer :: iy
iy = unlink('test/xx')
end
```

2.494 UNLOCK文

UNLOCK 文は、以下の形式です。

```
UNLOCK ( lock-variable [ , sync-stat-list ] )
```

*lock-variable*は、ロック変数です。LOCK_TYPE 型でなければなりません。LOCK_TYPE 型については“1.18.3 LOCK_TYPE 型”を参照してください。

sync-stat は同期状態指定子であり、以下の形式です。

```
STAT = stat-variable           または
ERRMSG = errmsg-variable
```

stat-variable および*errmsg-variable* については“1.18.2 同期状態指定子”を参照してください。

ロック変数が、ある像においてLOCK 文によってロックされ、その像でUNLOCK 文によってロック解除されていないとき、そのロック変数はその像にロックされています。

UNLOCK 文の実行に成功すると、ロック変数はロック解除されます。

ロック変数が像M においてUNLOCK 文の実行によりロック解除され、その後、像T においてLOCK 文の実行によりロックされた場合、像MのUNLOCK 文に先行するセグメントは像T のLOCK 文より後のセグメントに先行します。

UNLOCK 文のロック変数が実行中の像によってロックされていない場合は、誤り条件が発生します。UNLOCK 文の実行中に誤り条件が発生した場合、ロック変数の値は変更されません。

UNLOCK 文の例:

```
use iso_fortran_env, only : lock_type
type(lock_type) :: x[*]
if (this_image() == 4) then
  lock(x)     ! ロック変数xは像4によってロックされます
  unlock(x)   ! ロック変数xは像4によってロック解除されます
end if
end
```

2.495 UNPACK組込み関数

UNPACK 関数は、1次元配列の多次元化を行います。

分類

変形関数

形式

result = UNPACK (*VECTOR* , *MASK* , *FIELD*)

VECTOR

どの型でもかまいません。1次元配列でなければなりません。*MASK* の真の要素数以上の要素をもたなければなりません。

MASK

論理型配列でなければなりません。

FIELD

VECTOR と同じ型および同じ種別型パラメタでなければなりません。*MASK* と形状適合しなければなりません。

result

FIELD と同じ型、同じ種別型パラメタおよび同じ形状です。

機能説明

UNPACK は、1次元配列の要素を*MASK* に従って多次元配列に分布させます。

MASK 中の真の要素に対応する*FIELD* の値を、同じ位置にある*VECTOR* の値に置換した値を返却します。

関数の結果の型は、*FIELD* と同じです。

使用例

```
integer, dimension(9) :: c = (/0,3,2,4,3,2,5,1,2/)
logical, dimension(2,2) :: d
integer, dimension(2,2) :: e
d = reshape( (/.false.,.true.,.true.,.false./), (/2, 2/) )
e = unpack(c,mask=d,field=-1)
! e            には ┌      ┐
!                  | -1 3 |
!                  | 0 -1 |
!                  └      ┘ が代入されます
```

2.496 USE文

USE 文は、その有効域内から指定されたモジュール内の公開要素を参照可能にします。

USE 文は、以下の形式です。

```
USE [ [ , module-nature ] :: ] module-name [ , rename-list ]            または
USE [ [ , module-nature ] :: ] module-name , ONLY : [ only-list ]
```

module-nature は、モジュール性質であり、以下の形式です。

```
INTRINSIC             または
NON_INTRINSIC
```

module-name は、モジュール名です。

rename-list は、コンマで区切られた仮称指定の並びです。*rename* は、以下の形式です。

```
local-name => use-name                                                  または
OPERATOR ( local-defined-operator ) => OPERATOR ( use-defined-operator )
```

local-name は、局所名です。

use-name は、参照対象名です。

local-defined-operator は、局所利用者定義演算であり、以下の形式です。

defined-unary-operator または
defined-binary-operator

use-defined-operator は、参照利用者定義演算であり、以下の形式です。

defined-unary-operator または
defined-binary-operator

defined-unary-operator は利用者定義単項演算子であり、*defined-binary-operator* は利用者定義2項演算子であり、ともに以下の形式です。

. *operator-name* .

operator-name は、利用者が定義した240文字までの利用者定義演算子の名前です。

only-list は、コンマで区切られた参照限定の並びです。*only* は、以下の形式です。

generic-spec または
only-use-name または
only-rename

generic-spec は総称指定であり、総称束縛であってはなりません。以下の形式です。

generic-name または
OPERATOR (*defined-operator*) または
ASSIGNMENT (=) または
dtio-generic-spec

generic-name は、総称名です。

defined-operator は利用者定義演算子であり、総称束縛であってはなりません。
以下の形式です。

intrinsic-operator または
. *operator-name* .

intrinsic-operator は組込み演算子です。

operator-name は、利用者が定義した240文字までの利用者定義演算子の名前です。

ASSIGNMENT(=) は、利用者定義代入です。

dtio-generic-spec は派生型入出力手続であり、以下の形式です。

READ (FORMATTED) または
READ (UNFORMATTED) または
WRITE (FORMATTED) または
WRITE (UNFORMATTED)

only-use-name は、参照限定対象名であり、以下の形式です。

use-name

only-rename は、参照限定仮称指定であり、以下の形式です。

local-name => *use-name*

モジュール性質がINTRINSIC の場合、モジュール名は組込みモジュールの名前でなければなりません。

モジュール性質がNON_INTRINSIC の場合、モジュール名は組込みでないモジュールの名前でなければなりません。

有効域から、組込みモジュールおよび同じ名前をもつ組込みでないモジュールを参照してはなりません。

各参照利用者定義演算、各総称指定および各参照対象名は、モジュール内の公開要素でなければなりません。

モジュール性質をもたないUSE 文は、組込みモジュールまたは組込みでないモジュールのいずれかを参照可能にします。モジュール名が組込みモジュールおよび組込みでないモジュールの両方の名前である場合、組込みでないモジュールが参照されます。

ONLY 句をもたないUSE 文は、指定したモジュール内のすべての公開要素を参照可能にします。

ONLY 句をもつUSE 文は、参照限定並び中に総称指定、参照対象名、または参照利用者定義演算として書いてある要素だけを参照可能にします。

1つの有効域内で1つのモジュールに対して複数のUSE 文を書いてもかまいません。それらのうちいずれかのUSE 文がONLY 句をもたない場合、そのモジュール内のすべての公開要素が参照可能になります。それらすべてのUSE 文がONLY 句をもっている場合、参照限定並びで指定した要素だけが参照可能になります。

引用されるモジュール中の参照可能な要素は、次に示す1つ以上の局所名をもちます。

- 引用されるモジュールに対するいずれかの参照限定中に参照限定対象名または総称指定の利用者定義演算として書かれた名前の場合、そのモジュール内の要素の名前
- そのモジュールに対する仮称指定または参照限定仮称指定に指定されている要素の各局所名または局所利用者定義演算
- 引用されるモジュールに対するどの仮称指定および参照限定仮称指定の参照対象名または参照利用者定義演算としても書かれていない名前の場合、そのモジュール内の要素の名前

総称引用仕様および利用者定義演算以外の言語要素については、その有効域内でその名前を要素の参照に用いない場合にだけ、2つ以上の参照可能な要素が同じ名前をもってもかまいません。総称引用仕様および利用者定義演算については、同じ総称引用仕様をもつ場合、1つの総称引用仕様と解釈されます。これらの場合を除いて、USE 文で参照可能になった要素の局所名は、USE文または他の手段でその有効域内で参照可能になっている他のすべての要素の局所名と異なっていなければなりません。

参照結合した1つの要素が、2つ以上の局所名で参照可能でもかまいません。

USE 文で参照可能になった局所名は、そのUSE 文を含む有効域内でその要素の属性の再定義を引き起こす他の宣言文中に書いてはなりません。ただし、モジュールの有効域の場合は、有効域内のPUBLIC 文またはPRIVATE 文には書いてもかまいません。

参照結合しているモジュール内の要素がASYNCHRONOUS 属性またはVOLATILE 属性のいずれももたないときでも、局所有効域ではASYNCHRONOUS 属性またはVOLATILE 属性をもってかまいません。

USE 文の例:

```
module types
  type pair_int
    integer :: i1,i2
  end type pair_int
  type pair_real
    real :: r1,r2
  end type pair_real
end module types

module pair_data
  use types                          ! モジュールtypes の全要素を参照可能にします
  type(pair_int) :: ints_data
  type(pair_real) :: reals_data
end module pair_data

program main
  use types , ints => pair_int       ! モジュールtypes の全要素を参照可能にし、
                                     ! pair_int はints という局所名にします
  use pair_data ,only:ints_data      ! モジュールpair_data のints_data だけを
                                     ! 参照可能にします
```

2.497 VAL組込み関数

VAL 関数は、引数の値渡しを行います。

分類

引数値渡し関数

形式

CALL *subroutine-name* (VAL (*X*))
function-name (VAL (*X*))

X

1バイトの整数型、2バイトの整数型、4バイトの整数型、8バイトの整数型、1バイトの論理型、2バイトの論理型、4バイトの論理型、8バイトの論理型、単精度実数型、または倍精度実数型のスカラ式でなければなりません。

機能説明

VAL は、手続引用において引数の値渡しを行います。
この関数は、手続引用の実引数にだけ指定することができます。

使用例

```
i = my_c_function(val(a))          ! 引数a は値渡しとなります
```

2.498 VALUE文

VALUE 文は、仮引数が、実引数の値を初期値とする確定可能である一時的な領域と結合することを指定します。仮引数の値および定義状態のその後の変更は、実引数に影響を与えません。

VALUE 文は、以下の形式です。

```
VALUE [ :: ] dummy-arg-name-list
```

dummy-arg-name-list は、コンマで区切られた仮引数名の並びです。

dummy-arg-name は、仮手続、または仮ポインタの名前であってはなりません。

VALUE 属性を指定したとき、PARAMETER 属性、EXTERNAL 属性、POINTER 属性、ALLOCATABLE 属性、INTENT(INOUT) 属性、およびINTENT(OUT) 属性は同時に指定できません。

VALUE 属性を指定したとき、大きさ引継ぎ配列、共配列、共配列末端成分、多相的、型引継ぎ、および次元引継ぎは同時に指定できません。

VALUE 属性は、純粋サブルーチンおよび後始末サブルーチンの仮引数に指定してはなりません。

VALUE 文の例:

```
subroutine csub(i)
value :: i                  ! 引数i は、実引数の値を初期値とする確定可能である一時的な
                            ! 領域と結合します
```

2.499 VERIFY組込み関数

VERIFY 関数は、文字の包含の検査を行います。

分類

要素別処理関数

形式

result = VERIFY (*STRING* , *SET* [, *BACK* , *KIND*])

STRING

文字型でなければなりません。

SET

*STRING*と同じ種別型パラメタの文字型でなければなりません。

BACK（省略可能）

論理型でなければなりません。

KIND（省略可能）

スカラ整数定数式でなければなりません。

result

整数型です。*KIND*が指定された場合、種別型パラメタは*KIND*の指定に従います。

*KIND*が省略された場合、種別型パラメタは基本整数型になります。

機能説明

VERIFY は、文字列中に文字が含まれているかどうかを調査します。

－ *BACK*に偽を指定している、または省略している場合

*STRING*が*SET*中の文字を含む場合、*SET*中にない文字の*STRING*中での最も左の文字の位置を返却します。*STRING*が*SET*中の文字を1文字も含まない場合および、*STRING*または*SET*の文字長が0の場合、0を返却します。

－ *BACK*に真を指定している場合

*STRING*が*SET*中の文字を含む場合、*SET*中にない文字の*STRING*中での最も右の文字の位置を返却します。*STRING*が*SET*中の文字を1文字も含まない場合および、*STRING*または*SET*の文字長が0の場合、0を返却します。

関数の結果の型は、種別型パラメタ*KIND*が指定された場合、種別型パラメタ*KIND*の整数型となります。*KIND*が省略された場合、関数の結果の型は基本整数型です。

使用例

```
i = verify("Lalalalala","l")                ! i には1が代入されます
i = verify("LalalaLALA","LA",back=.true.)   ! i には6が代入されます
```

2.500 VOLATILE文

VOLATILE 文は、実体を最適化の対象としないことを指定します。

VOLATILE 文は、以下の形式です。

```
VOLATILE [ :: ] object-name-list
```

*object-name-list*は、コンマで区切られた実体名の並びです。

*object-name*は、名前付き定数および組込み手続名であってはなりません。

VOLATILE 文の例:

```
program main
volatile :: v,w          ! 変数v およびw は最適化の対象となりません
```

2.501 WAITサービス関数

機能説明

呼出し元がシグナルを受信するか、その子プロセスが終了するまで、現プロセスを中断します。WAIT サービス関数が呼び出される前に、子プロセスが停止しているか、終了していた場合、WAIT サービス関数は何も行わず復帰します。

形式

iy = WAIT (*status*)

status

基本整数型スカラ。子プロセス終了時のステータス値が設定されます。

関数結果

基本整数型スカラ。正の場合は子プロセスID を示しています。負の場合は、結果値を反転させた値がシステムエラーコードを示しています。

使用例

```
      use service_routines,only:wait,fork
      integer :: pid,stat
      pid = fork( )
      if (pid>0) goto 30
      call sub
   30 if (wait(stat).eq.-1) write(*,*) 'wait error'
      call sub1
      end
```

2.502 WAIT文

WAIT 文は、非同期データ転送操作に対して待機操作を行います。

WAIT 文は以下の形式です。

```
WAIT ( wait-spec-list )
```

wait-spec-list は待機指定子並びです。以下の形式です。

```
[ UNIT = ] external-file-unit       または
END = end-label                     または
EOR = eor                           または
ERR = err-label                     または
ID = id                             または
IOMSG = iomsg                       または
IOSTAT = io-stat
```

各指定子は、待機指定子並びに2回以上指定してはなりません。

external-file-unit は、外部ファイル装置であり、スカラ整数式でなければなりません。

ファイル装置番号は、必ず指定しなければなりません。省略可能な文字列UNIT= を省略する場合、ファイル装置番号は、待機指定子並びの最初の項目でなければなりません。

io-stat は、スカラ整変数でなければなりません。

IOSTAT 指定子が指定された場合、IOSTAT 指定子に指定された変数には、以下の値が設定されます。

- 誤り条件、ファイル終了条件、および記録終了条件が検出されない場合は、0
- 誤り条件が検出された場合は、1または実行時の診断メッセージの番号
- ファイル終了条件が検出された場合は、-1
- 記録終了条件が検出された場合は、-2

err-label は文番号であり、このWAIT 文と同じ有効域内の飛び先文の文番号でなければなりません。

ERR 指定子が指定され、WAIT 文の実行中に誤り条件が検出された場合、ERR 指定子に指定された文番号をもつ文が次に実行されます。

end-label は文番号であり、このWAIT 文と同じ有効域内の飛び先文の文番号でなければなりません。

END 指定子が指定され、WAIT 文の実行中にファイル終了条件が検出された場合、END 指定子に指定された文番号をもつ文が次に実行されます。

eor-label は文番号であり、このWAIT 文と同じ有効域内の飛び先文の文番号でなければなりません。

EOR 指定子が指定され、WAIT 文の実行中に記録終了条件が検出された場合、EOR 指定子に指定された文番号をもつ文が次に実行されます。

id は、スカラ整数式です。

ID 指定子に指定した式の値は、指定した装置におけるデータ転送操作の識別子でなければなりません。ID 指定子を指定した場合、指定したデータ転送操作に対する待機操作を行います。ID指定子を省略した場合、指定した装置におけるすべてのデータ転送操作に対する待機操作を行います。

iomsg はスカラ基本文字変数でなければなりません。入出力文の実行中に以下の条件が発生した場合、*iomsg* には説明のためのメッセージが代入されます。

- 誤り条件
- ファイル終了条件
- 記録終了条件

それ以外の場合、*iomsg* の値は変更されません。

存在しないか、ファイルと接続していないかまたは非同期入出力として開いていない装置を指定したWAIT 文を実行することもできます。そのWAIT 文はID 指定子を省略したものとみなされます。このようなWAIT 文は、誤り条件またはファイル終了条件とはなりません。

2.503 WHERE構文

WHERE 構文は、複数の配列代入および入れ子になった単純WHERE 文またはWHERE 構文の評価および代入を選別します。

WHERE 構文は、以下の形式です。

```
[ where-construct-name : ] WHERE ( mask-expr )
    [ where-body-construct ]...
[ ELSEWHERE ( mask-expr ) [ where-construct-name ]
    [ where-body-construct ]... ]...
[ ELSEWHERE [ where-construct-name ]
    [ where-body-construct ]... ]
END WHERE [ where-construct-name ]
```

where-construct-name は、WHERE 構文名です。

構造WHERE 文にWHERE 構文名を指定する場合は、対応するEND WHERE 文にも同じWHERE 構文名を指定しなければなりません。構造WHERE 文にWHERE 構文名を指定しない場合は、対応するEND WHERE 文にWHERE 構文名を指定してはなりません。ELSEWHERE 文または選別ELSEWHERE 文にWHERE 構文名を指定する場合は、対応する構造WHERE 文にも同じWHERE 構文名を指定しなければなりません。

mask-expr は、選別式です。

選別式は、論理型の配列式でなければなりません。

where-body-construct はWHERE 本体構文であり、以下の形式です。

```
assignment-stmt          または
単純WHERE 文             または
WHERE 構文
```

assignment-stmt は代入文です。

選別式の指定をもつELSEWHERE 文は、選別ELSEWHERE 文です。

WHERE 構文が単純WHERE 文、選別ELSEWHERE 文、または別のWHERE 構文を含む場合、WHERE 構文に含まれる選別式は、すべて同じ形状でなければなりません。

代入文で確定する変数は、選別式と同じ形状の配列でなければなりません。

代入文が利用者定義代入文であるとき、その代入文は要素別処理でなければなりません。

構造WHERE 文を実行すると、選別式が評価され、その結果が制御配列となります。また、値‘.NOT. 制御配列’が、否定制御配列となります。

WHERE 構文中の文は1つ1つ順番に実行されます。

選別ELSEWHERE 文を実行すると、制御配列は、‘直前までの否定制御配列.AND. 選別ELSEWHERE 文の選別式’の値となります。否定制御配列は、‘.NOT. 新しい制御配列’となります。

ELSEWHERE 文を実行すると、制御配列は、直前までの否定制御配列の値になります。

WHERE 本体構文の一部である構造WHERE 文または単純WHERE 文を実行すると、制御配列は、‘直前までの制御配列.AND. 選別式’の値となり、否定制御配列は、‘.NOT. 新しい制御配列’となります。

END WHERE 文を実行すると、制御配列および否定制御配列は、対応する構造WHERE 文を実行する前の値となります。WHERE 本体構文として書いた単純WHERE 文の実行後、制御配列は、単純WHERE 文を実行する前の値となります。

WHERE 構文中の代入文は、制御配列の真である要素に対応する右辺の式および左辺の変数中の式が評価され、制御配列の真である要素に対応する右辺の式の値が、左辺の変数の対応する要素に代入されます。

WHERE 構文の例:

```
integer,dimension(10) :: a,b,c
a = (/ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9/)
b = (/-5,-3,-1, 1, 3, 5, 7, 9,11,13/)
where (a > b)
  c = a
elsewhere (a == b)
  c = 0
elsewhere
  c = b
endwhere
  ! c は(/0,1,2,3,4,0,7,9,11,13/) になります
```

2.504 構造WHERE文

構造WHERE 文は、WHERE 構文の開始を示します。

構造WHERE 文は、以下の形式です。

```
[ where-construct-name : ] WHERE ( mask-expr )
```

where-construct-name は、WHERE 構文名です。

mask-expr は、選別式です。

WHERE 構文の詳細については、“2.503 WHERE構文”を参照してください。

2.505 単純WHERE文

単純WHERE 文は、論理配列式の値に従って、配列代入文における式の評価および代入を選別します。

単純WHERE 文は、以下の形式です。

```
WHERE ( mask-expr ) where-assignment-stmt
```

mask-expr は、選別式です。

選別式は、論理型の配列式でなければなりません。

where-assignment-stmt は、以下の形式です。

```
assignment-stmt
```

assignment-stmt は代入文です。

代入文で確定する変数は、選別式と同じ形状の配列でなければなりません。

代入文が利用者定義代入文であるとき、その代入文は要素別処理でなければなりません。

単純WHERE 文を実行すると、選別式が評価され、その結果が制御配列となります。単純WHERE 文中の代入文は、制御配列の真である要素に対応する右辺の式および左辺の変数中の式が評価され、制御配列の真である要素に対応する右辺の式の値が、左辺の変数の対応する要素に代入されます。

単純WHERE 文の例:

```
integer,dimension(10) :: a,b,c=0
a = (/ 0, 1, 2, 3, 4, 5, 6, 7, 8, 9/)
b = (/-5,-3,-1, 1, 3, 5, 7, 9,11,13/)
where (a > b) c = a
           ! c は(/0,1,2,3,4,0,0,0,0,0/) になります
```

2.506 WRITE文

WRITE 文は、出力項目並びおよび書式仕様で指定されたデータ要素から、または変数群からファイルに値を転送します。

WRITE 文は、以下の形式です。

```
WRITE ( io-control-spec-list ) [ output-item-list ]
```

io-control-spec-list はコンマで区切られたデータ転送指定子の並びです。

io-control-spec は、以下の形式です。

```
[ UNIT = ] io-unit              または
[ FMT = ] format                または
[ NML = ] namelist-group-name   または
REC = record-number             または
IOSTAT = io-stat                または
ERR = err-label                 または
ADVANCE = advance               または
NUM = record-len                または
ASYNCHRONOUS = asynchronous     または
DECIMAL = decimal               または
DELIM = delim                   または
ID = id                         または
IOMSG = iomsg                   または
POS = pos                       または
ROUND = round                   または
SIGN = sign
```

UNIT 指定子は、必ず指定しなければなりません。文字列‘UNIT=’をUNIT 指定子から省略する場合、UNIT 指定子は、データ転送指定子並びの最初の項目でなければなりません。

io-unit は装置識別子であり、以下の形式です。

```
external-file-unit              または
*                               または
internal-file-unit
```

external-file-unit は、外部ファイル装置であり、スカラ整数式でなければなりません。

internal-file-unit は、内部ファイル装置であり、基本文字変数でなければなりません。

UNIT 指定子に内部ファイル装置を指定した場合、REC 指定子またはPOS 指定子を指定してはなりません。

FMT 指定子とNML 指定子をともに指定することはできません。

文字列‘FMT=’をFMT指定子から省略する場合、FMT指定子は、データ転送指定子並びの2番目の項目でなければならず、最初の項目は、文字列‘UNIT=’を省略したUNIT指定子でなければなりません。

*format*は書式識別子であり、以下の形式です。

default-char-expr または
label または
* または
scalar-default-int-variable

*default-char-expr*は、基本文字式です。基本文字式の値は、有効な書式仕様でなければなりません。書式仕様については、“1.8.1 書式仕様”を参照してください。

*label*は文番号です。文番号は、このWRITE 文を含む有効域内のFORMAT 文の文番号でなければなりません。

*scalar-default-int-variable*は、スカラ基本整変数です。スカラ基本整変数には、ASSIGN 文により、このWRITE 文を含む有効域内のFORMAT 文の文番号が割り当てられていなければなりません。

文字列‘NML=’をNML指定子から省略する場合、NML指定子は、データ転送指定子並びの2番目の項目でなければならず、最初の項目は、文字列‘UNIT=’を省略したUNIT指定子でなければなりません。

*namelist-group-name*は、変数群名です。

NML指定子を指定した場合、出力項目並びを指定してはなりません。

*record-number*は、直接探査出力文で出力する記録の番号を指定します。*record-number*は、スカラ整数式でなければなりません。

REC指定子を指定した場合、NML指定子、POS指定子を指定してはならず、FMT指定子に並び出力を意味する星印‘*’を指定してはなりません。

*io-stat*は、スカラ整変数です。

IOSTAT指定子が指定された場合、IOSTAT指定子に指定された変数には、以下の値が設定されます。

- 誤り条件、ファイル終了条件、および記録終了条件が検出されない場合は、0
- 誤り条件が検出された場合は、1または実行時の診断メッセージの番号

*err-label*は文番号であり、このWRITE 文と同じ有効域内の飛び先文の文番号でなければなりません。

ERR指定子が指定され、WRITE 文の実行中に誤り条件が検出された場合、ERR指定子に指定された文番号をもつ文が次に実行されます。

*advance*は、スカラ基本文字式であり、その値は'YES' または'NO' でなければなりません。

ADVANCE指定子は、停留出力を行うかどうかを指定します。'NO'を指定すると停留出力が行われます。'YES'を指定すると、停留出力は行われません。ADVANCE指定子の省略値は、'YES' です。

ADVANCE指定子は、UNIT指定子に内部ファイル装置を指定せず、FMT指定子に明示的な書式仕様をもつ書式付き順番探査にだけ指定できます。

*record-len*は、スカラ基本整変数です。

NUM指定子を指定された場合、NUM指定子に指定された変数には、その出力文の実行によって実際に転送されたFortran記録の長さが、バイトを単位として、設定されます。

NUM指定子を指定した場合、FMT指定子およびNML指定子を指定してはなりません。

*asynchronous*はスカラ基本文字定数式でなければなりません。

ASYNCHRONOUS指定子は、その入出力文が同期か非同期かを指定します。'YES'を指定したとき、その文および入出力操作は非同期です。'NO' を指定するかまたはこの指定子を省略したとき、その文および入出力操作は同期です。

非同期入出力は、OPEN 文のASYNCHRONOUS指定子に'YES' を指定した外部ファイルに対してだけ許されます。

ASYNCHRONOUS 指定子に'YES' を指定したファイルに対しては、同期入出力および非同期入出力の両方ができます。ASYNCHRONOUS指定子に'NO'を指定したファイル、ASYNCHRONOUS指定子を省略して開いたファイル、OPEN 文なしで参照する事前接続したファイルおよび内部ファイルには、同期入出力だけができます。

*decimal*はスカラ基本文字式でなければなりません。*decimal*の値は、'COMMA' または'POINT' でなければなりません。DECIMAL指定子は、その接続の小数編集モードを一時的に変更します。この指定子を省略すると、モードは変更されません。

delim はスカラ基本文字式でなければなりません。*delim* の値は、'APOSTROPHE'、'QUOTE' または'NONE' でなければなりません。DELIM 指定子は、その接続の囲み記号モードを一時的に変更します。この指定子を省略すると、モードは変更されません。

*id*はスカラ整変数でなければなりません。ID 指定子のある非同期データ転送文の実行が成功すると、ID 指定子に指定した変数は、確定となります。この値は、後続するWAIT 文またはINQUIRE文で、特定のデータ転送操作を識別するために使用することができます。

ID 指定子のあるデータ転送文の実行中に誤りが発生すると、ID 指定子に指定した変数は不定となります。

iomsg はスカラ基本文字変数でなければなりません。入出力文の実行中に以下の条件が発生した場合、*iomsg* には説明のためのメッセージが代入されます。

- 誤り条件
- ファイル終了条件
- 記録終了条件

それ以外の場合、*iomsg* の値は変更されません。

pos はスカラ整変数でなければなりません。POS 指定子は、ファイル記憶単位の中のファイル位置を指定します。この指定子は、流れ探査として接続した装置を指定している場合だけ、データ転送文に指定することができます。

*round*はスカラ基本文字式でなければなりません。*round*の値は、OPEN 文のROUND 指定子で規定した値の1つです。ROUND 指定子は、その接続の入出力丸めモードを一時的に変更します。この指定子を省略すると、モードは変更されません。

*sign*はスカラ基本文字式でなければなりません。*sign*の値は、'PLUS'、'SUPPRESS' または'PROCESSOR_DEFINED' です。SIGN 指定子は、その接続の符号モードを一時的に変更します。この指定子を省略すると、モードは変更されません。

output-item-list は、コンマで区切られた出力項目の並びです。

output-item は、以下の形式です。

```
expr                    または
io-implied-do
```

expr は、式です。

io-implied-do は入出力DO 形反復であり、以下の形式です。

```
( output-item-list , io-implied-do-control )
```

io-implied-do-control は入出力DO 制御であり、以下の形式です。

```
do-variable = scalar-expr , scalar-expr [ , scalar-expr ]
```

*do-variable*はDO 変数であり、整数型、基本実数型、または倍精度実数型の名前付きスカラ変数でなければなりません。基本実数型および倍精度実数型のDO 変数は廃止事項です。

*scalar-expr*は、整数型、基本実数型、または倍精度実数型のスカラ式でなければなりません。基本実数型および倍精度実数型の*scalar-expr*は廃止事項です。

入出力DO 形反復における、ループの初期化および実行は、DO 構文(“2.120 DO構文”参照)と同じです。

ほかの入出力DO 形反復を含む入出力DO 形反復のDO 変数は、含まれる方の入出力DO 形反復のDO 変数に現れたり結合されたりしてはなりません。

出力項目は、利用者定義の派生型入出力手続で処理される場合を除いて、多相的であってはなりません。

出力項目がポインタの場合、ポインタは、指示先に結合されていなければならず、データは指示先からファイルに転送されます。

出力項目が割付け変数の場合、その割付け変数は割り付けられていなければなりません。

出力項目として配列を指定した場合、配列要素順序(“1.5.8.3 配列要素順序”参照)の順序で、すべての配列要素が指定されたかのように扱われます。

派生型が末端成分にポインタ成分または割付け成分をもつ場合、この派生型の実体は、利用者定義の派生型入出力手続で処理されなければなりません。

利用者定義の派生型入出力手続で処理されなく、かつ、派生型実体の名前を出力項目並びに指定した場合、そのすべての成分が派生型の定義におけるのと同じ順序で指定されたものとして扱われます。

WRITE 文の例:

```
write (*,*) a,b,c                  ! 並び出力により、a、b、およびc を出力します
write (3,fmt= "(e7.4)") x          ! 装置参照番号3にE 形編集により、x を出力します
write (*,10) i,j,k                 ! 文番号10のFORMAT 文の書式仕様により、
                                   ! i、j、およびk を出力します
```

付録A　組込み手続一覧

組込み手続には、次の手続があります。

- 要素別処理組込み手続
- 問合せ組込み関数(“表A.6 問合せ組込み関数”参照)
- 日付および時刻サブルーチン(“表A.7 日付および時刻サブルーチン”参照)
- アドレス取得関数(“表A.8 アドレス取得関数”参照)
- 共配列組込み手続(“表A.9 共配列組込み手続”参照)
- その他の手続(“表A.10 その他の手続”参照)

要素別処理組込み手続には、次の手続があります。

- 数値関数(“表A.1 数値関数”参照)
- 数学関数(“表A.2 数学関数”参照)
- 文字関数(“表A.3 文字関数”参照)
- ビット手続(“表A.4 ビット関数”参照)
- ビット複写サブルーチン(“表A.5 ビット複写サブルーチン”参照)

表A.1 数値関数

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
整数型への型変換 y=int(x) (*3)	INT (A[,KIND])	－ － － － INT IFIX JINT JIFIX IDINT JIDINT IQINT － － － －	1	I1 I2 I4 I8 R R R R D D Q C DC QC nd					I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4 I4	要素別処理関数
		－ － － － － － － － － －	2	I1 I2 I4 I8 R D Q C DC QC nd	i i i i i i i i i i i				i (*1) i (*1) i (*1) i (*1) i (*1) i (*1) i (*1) i (*1) i (*1) i (*1) i (*1)	

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
	INT4	― ― ― ― ― ― ― ― ― ―	1	I1 I2 I4 I8 R D Q C DC QC					I4 I4 I4 I4 I4 I4 I4 I4 I4 I4	
	JFIX	― ― ― ― ― ― ― ― ― ―	1	I1 I2 I4 I8 R D Q C DC QC					I4 I4 I4 I4 I4 I4 I4 I4 I4 I4	
1バイトの整数型への型変換 y=int(x) (*3)	INT1	― ― ― ― ― ― ― ― ― ―	1	I1 I2 I4 I8 R D Q C DC QC					I1 I1 I1 I1 I1 I1 I1 I1 I1 I1	要素別処理関数
2バイトの整数型への型変換 y=int(x) (*3)	INT2	HFIX IINT IIFIX IIDINT	1	R R R D					I2 I2 I2 I2	要素別処理関数
		― ― ― ― ― ― ― ― ― ―	1	I1 I2 I4 I8 R D Q C DC QC					I2 I2 I2 I2 I2 I2 I2 I2 I2 I2	
実数型への型変換 (*4)	REAL (A[,KIND])	― ― REAL ― FLOATI FLOAT FLOATJ	1	I1 I2 I4 I8 I2 I4 I4					R R R R R R R	要素別処理関数

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
		― SNGL SNGLQ ― ― ― ―		R D Q C DC QC nd					R R R R D Q R	
		― ― ― ― ― ― ― ― ― ― ―	2	I1 I2 I4 I8 R D Q C DC QC nd	i i i i i i i i i i i				r (*1) r (*1) r (*1) r (*1) r (*1) r (*1) r (*1) r (*1) r (*1) r (*1) r (*1)	
倍精度実数型への型変換 (*5)	DBLE (A)	― ― DFLOTI DFLOAT DFLOTJ ― DBLE ― DBLEQ ― DREAL ― ―	1	I1 I2 I2 I4 I4 I8 R D Q C DC QC nd					D D D D D D D D D D D D D D	要素別処理関数
4倍精度実数型への型変換	QEXT(A)	― ― QFLOAT ― QEXT QEXTD ― ― ― QREAL	1	I1 I2 I4 I8 R D Q C DC QC					Q Q Q Q Q Q Q Q Q Q	要素別処理関数
複素数型への型変換 (*6)	CMPLX (X[,Y][,KIND])	― ― ― ― ― ― ― ―	1 or 2	I1 I2 I4 I8 R D Q nd	in in in in in in in in				C C C C C C C C	要素別処理関数

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
		― ― ― ― ― ― ― ―	2 or 3	I1 I2 I4 I8 R D Q nd	in in in in in in in in	i i i i i i i i			c (*1) c (*1) c (*1) c (*1) c (*1) c (*1) c (*1) c (*1)	
	CMPLX (X[,KIND])	― ― ―	1	C DC QC					C C C	
		― ― ―	2	C DC QC	i i i				c (*1) c (*1) c (*1)	
倍精度複素数型への型変換 (*7)	DCMPLX(X[,Y])	― ― ― ― ― ― ―	1 or 2	I1 I2 I4 I8 R D Q	ir ir ir ir ir ir ir				DC DC DC DC DC DC DC	要素別処理関数
		― ― ―	1	C DC QC					DC DC DC	
4倍精度複素数型への型変換	QCMPLX(X[,Y])	― ― ― ― ― ― ―	1 or 2	I1 I2 I4 I8 R D Q	ir ir ir ir ir ir ir				QC QC QC QC QC QC QC	要素別処理関数
		― ― ―	1	C DC QC					QC QC QC	
切捨て y=int(x) (*3)	AINT (A[,KIND])	AINT DINT QINT	1	R D Q					R D Q	要素別処理関数
		― ― ―	2	R D Q	i i i				r (*1) r (*1) r (*1)	
四捨五入 x >= 0 ならば y = int(x + 0.5) x < 0 ならば y = int(x - 0.5) (*3)	ANINT (A[, KIND])	ANINT DNINT QNINT	1	R D Q					R D Q	要素別処理関数
		― ― ―	2	R D Q	i i i				r (*1) r (*1) r (*1)	

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
四捨五入整数化 x >= 0 ならば int(x + 0.5) x < 0 ならば int(x - 0.5) (*3)	NINT (A[,KIND])	NINT JNINT IDNINT JIDNNT IQNINT	1	R R D D Q					I4 I4 I4 I4 I4	要素別処理関数
		― ― ―	2	R D Q	i i i				i (*1) i (*1) i (*1)	
	I2NINT	― ININT IIDNNT ―	1	R R D Q					I2 I2 I2 I2	
絶対値 y = |x|	ABS (A)	― ― IABS I2ABS IIABS IABS JIABS ― ABS DABS QABS CABS CDABS CQABS	1	I1 I2 I2 I2 I2 I4 I4 I8 R D Q C DC QC					I1 I2 I2 I2 I2 I4 I4 I8 R D Q R D Q	要素別処理関数
余り x1 - int(x1/x2) × x2 ただし x2 ≠ 0 (*8)	MOD (A,P)	― I2MOD IMOD MOD JMOD MOD ― AMOD DMOD QMOD	2	I1 I2 I2 I2 I4 I4 I8 R D Q	I1 I2 I2 I2 I4 I4 I8 R D Q				I1 I2 I2 I2 I4 I4 I8 R D Q	要素別処理関数
符号の付け替え B >= 0 ならば |A| B < 0 ならば -|A| (*2)	SIGN (A,B)	― ― I2SIGN IISIGN ISIGN ISIGN JISIGN ― SIGN DSIGN QSIGN	2	I1 I2 I2 I2 I2 I4 I4 I8 R D Q	I1 I2 I2 I2 I2 I4 I4 I8 R D Q				I1 I2 I2 I2 I2 I4 I4 I8 R D Q	要素別処理関数
超過分 x1 > x2 ならば x1 - x2 x1 <= x2 ならば 0	DIM (X,Y)	― ― I2DIM IDIM	2	I1 I2 I2 I2	I1 I2 I2 I2				I1 I2 I2 I2	要素別処理関数

機能	総称名 （引数 キーワード）	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
		IIDIM IDIM JIDIM — DIM DDIM QDIM		I2 I4 I4 I8 R D Q	I2 I4 I4 I8 R D Q				I2 I4 I4 I8 R D Q	
倍精度化乗算 y = DBLE(x1) × DBLE(x2)	DPROD (X,Y)	DPROD	2	R	R				D	要素別処理関数
4倍精度化乗算 y = QEXTD(x1) × QEXTD(x2)	—	QPROD	2	D	D				Q	要素別処理関数
最大値 y = max(x1, x2, ..) x1, x2, ... の内の最大値を y とする	MAX (A1,A2 [,A3,...])	— MAX MAX I2MAX0 IMAX0 MAX0 JMAX0 MAX0 — AMAX1 DMAX1 QMAX1 —	2 < =	I1 I2 I4 I2 I2 I2 I4 I4 I8 R D Q CH	I1 I2 I4 I2 I2 I2 I4 I4 I8 R D Q CH	I1 I2 I4 I2 I2 I2 I4 I4 I8 R D Q CH	I1 I2 I4 I2 I2 I2 I4 I4 I8 R D Q CH	I1 I2 I4 I2 I2 I2 I4 I4 I8 R D Q CH	I1 I2 I4 I2 I2 I2 I4 I4 I8 R D Q CH	要素別処理関数
	—	AIMAX0 AMAX0 AJMAX0 AMAX0 IMAX1 JMAX1 MAX1	2 < =	I2 I2 I4 I4 R R R	I2 I2 I4 I4 R R R	I2 I2 I4 I4 R R R	I2 I2 I4 I4 R R R	I2 I2 I4 I4 R R R	R R R R I2 I4 I4	
最小値 y = min(x1, x2, ..) x1, x2, ... の内の最小値を y とする	MIN (A1,A2 [,A3,...])	— MIN MIN I2MIN0 IMIN0 MIN0 JMIN0 MIN0 — AMIN1 DMIN1 QMIN1 —	2 < =	I1 I2 I4 I2 I2 I2 I4 I4 I8 R D Q CH	I1 I2 I4 I2 I2 I2 I4 I4 I8 R D Q CH	I1 I2 I4 I2 I2 I2 I4 I4 I8 R D Q CH	I1 I2 I4 I2 I2 I2 I4 I4 I8 R D Q CH	I1 I2 I4 I2 I2 I2 I4 I4 I8 R D Q CH	I1 I2 I4 I2 I2 I2 I4 I4 I8 R D Q CH	要素別処理関数
	—	AIMIN0 AMIN0 AJMIN0	2 < =	I2 I2 I4	I2 I2 I4	I2 I2 I4	I2 I2 I4	I2 I2 I4	R R R	

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
		AMIN0 IMIN1 JMIN1 MIN1		I4 R R R	I4 R R R	I4 R R R	I4 R R R	I4 R R R	R I2 I4 I4	
虚部 x の虚部が関数値y である	AIMAG (Z)	IMAG AIMAG DIMAG QIMAG	1	C C DC QC					R R D Q	要素別処理関数
共役複素数 CMPLX(REAL(x),-IMAG(x))	CONJG (Z)	CONJG DCONJG QCONJG	1	C DC QC					C DC QC	要素別処理関数
剰余	MODULO (A,P)	― ― ― ― ― ― ―	2	I1 I2 I4 I8 R D Q	I1 I2 I4 I8 R D Q				I1 I2 I4 I8 R D Q	要素別処理関数
引数以上の最小整数値	CEILING (A[,KIND])	― ― ―	1	R D Q					I4 I4 I4	要素別処理関数
		― ― ―	2	R D Q	i i i				i (*1) i (*1) i (*1)	
引数以下の最大整数値	FLOOR (A[,KIND])	― ― ―	1	R D Q					I4 I4 I4	要素別処理関数
		― ― ―	2	R D Q	i i i				i (*1) i (*1) i (*1)	
モデル数指数部	EXPONENT (X)	―	1	r					I4	要素別処理関数
モデル表現小数部	FRACTION (X)	― ― ―	1	R D Q					R D Q	要素別処理関数
引数に最も近い数	NEAREST (X,S)	― ― ―	2	R D Q	r r r				R D Q	要素別処理関数
引数値に近いモデル数の 相対間隔の逆数	RRSPACING (X)	― ― ―	1	R D Q					R D Q	要素別処理関数
引数値に近いモデル数の 絶対間隔	SPACING (X)	― ― ―	1	R D Q					R D Q	要素別処理関数
実数にその基数を整数の べき数だけ乗算	SCALE (X,I)	― ― ―	2	R D Q	i i i				R D Q	要素別処理関数

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
指数部を設定	SET_EXPONENT(X,I)	－ － －	2	R D Q	i i i				R D Q	要素別処理関数
マージ	MERGE (TSOURCE, FSOURCE, MASK)	－	3	ay	a1	l			a1	要素別処理関数
論理型の変換	LOGICAL (L[,KIND])	－	1	l					L4	要素別処理関数
		－	2	l	i				l(*1)	
ファイル終了条件を検査	IS_IOSTAT_END(I)	－	1	i					L4	要素別処理関数
記録終了条件を検査	IS_IOSTAT_EOR(I)	－	1	i					L4	要素別処理関数

－ ：対応する総称名、または個別名が存在しないことを示します。
I1 ：1バイトの整数型。
I2 ：2バイトの整数型。
I4 ：4バイトの整数型。
I8 ：8バイトの整数型。
i ：1バイトの整数型、2バイトの整数型、4バイトの整数型、および8バイトの整数型。
R ：単精度実数型、および総称名の半精度実数型。
D ：倍精度実数型。
Q ：4倍精度実数型。
r ：半精度実数型、単精度実数型、倍精度実数型、および4倍精度実数型。
C ：単精度複素数型、および総称名の半精度複素数型。
DC ：倍精度複素数型。
QC ：4倍精度複素数型。
c ：半精度複素数型、単精度複素数型、倍精度複素数型、および4倍精度複素数型。
L4 ：4バイトの論理型。
l ：1バイトの論理型、2バイトの論理型、4バイトの論理型、および8バイトの論理型。
a1 ：第1引数と同じ型。
ir ：1バイトの整数型、2バイトの整数型、4バイトの整数型、8バイトの整数型、半精度実数型、単精度実数型、倍精度実数型、および4倍精度実数型。
ay ：すべての型。
nd ：非10進定数表現。
in ：1バイトの整数型、2バイトの整数型、4バイトの整数型、8バイトの整数型、半精度実数型、単精度実数型、倍精度実数型、4倍精度実数型、および非10進定数表現。
CH ：基本文字型。
*1) 引数でKIND を指定したとき、結果の種別型パラメタはKIND の指示に従います。
*2) 本処理系は、実数型の+0.0および-0.0を識別します。
*3) x が1バイトの整数型、2バイトの整数型、4バイトの整数型、または8バイトの整数型のとき、int(x)は、x です。x が半精度実数型、単精度実数型、倍精度実数型、または4倍精度実数型のとき、int(x)は、xの絶対値を超えない最大の整数値にxの符号を付けたものです。
xが半精度複素数型、単精度複素数型、倍精度複素数型、または4倍精度複素数型のとき、int(x) は、x の実部に上記の規則を適用して得られた値です。
*4) x が単精度実数型のとき、REAL(x) は、x です。x が整数型、半精度実数型、倍精度実数型、または4倍精度実数型のとき、REAL(x) は、x を単精度実数型のデータとして表したものです。
x が半精度複素数型、単精度複素数型、倍精度複素数型、または4倍精度複素数型のとき、REAL(x) は、x の実部を単精度実数型のデータとして表したものです。
*5) x が倍精度実数型のとき、DBLE(x) は、x です。x が整数型、半精度実数型、単精度実数型、または4倍精度実数型のとき、DBLE(x) は、x を倍精度実数型のデータとして表したものです。

x が半精度複素数型、単精度複素数型、倍精度複素数型、または4倍精度複素数型のとき、DBLE(x) は、
x の実部を倍精度実数型のデータとして表したものです。

*6) x が単精度複素数型のとき、CMPLX(x) は、x です。x、x1、およびx2 が整数型、半精度実数型、単精度実数型、倍精度実数型、または4倍精度実数型のとき、CMPLX(x) は、実部がREAL(x) で虚部が0の単精度複素数型の値であり、CMPLX(x1,x2) は、実部がREAL(x1) で虚部がREAL(x2)の単精度複素数型の値です。xが半精度複素数型、倍精度複素数型、または4倍精度複素数型のとき、CMPLX(x) は、実部がREAL(x) で虚部がREAL(IMAG(x)) の単精度複素数型の値です。

*7) x が倍精度複素数型のとき、DCMPLX(x) は、x です。x、x1、およびx2 が整数型、半精度実数型、単精度実数型、倍精度実数型、または4倍精度実数型のとき、DCMPLX(x) は、実部がDBLE(x) で虚部が0の倍精度複素数型の値であり、DCMPLX(x1,x2) は、実部がDBLE(x1) で虚部がDBLE(x2)の倍精度複素数型の値です。xが半精度複素数型、単精度複素数型、または4倍精度複素数型のとき、DCMPLX(x) は、実部がDBLE(x) で虚部がDBLE(IMAG(x)) の倍精度複素数型の値です。

*8) AMOD、DMOD、およびQMOD において、int(x1/x2) の演算結果が4バイトの整数型で表現できない場合は、AMOD、DMOD、およびQMOD の演算結果は不定になります。

備考1. x、x1、x2、... は、実引数を表し、xj はj 番目の実引数であることを示します。
xr は実引数の実部、xi は実引数の虚部を表します。

備考2. y は関数値を表します。

備考3. 以下の個別名をもつ組込み関数が手続引用の実引数として指定された場合に、それと結合される仮引数（仮手続）の引用における実引数は、4バイトの整数型だけが許されます。
IABS、MOD、ISIGN、IDIM、FLOAT、DFLOAT、QFLOAT

備考4. 以下の個別名をもつ組込み関数は、手続引用の実引数として指定できません。
MAX、I2MAX0、IMAX0、MAX0、JMAX0、AMAX1、DMAX1、QMAX1、AIMAX0、AMAX0、AJMAX0、IMAX1、MAX1、JMAX1、MIN、I2MIN0、IMIN0、MIN0、JMIN0、AMIN1、DMIN1、QMIN1、AIMIN0、AMIN0、AJMIN0、IMIN1、MIN1、JMIN1

表A.2 数学関数

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
平方根 $y = x^{\frac{1}{2}}$ x >= 0 $y = x^{\frac{1}{2}}$ $-\pi$ <xの偏角 <= πによる関数値	SQRT (X)	SQRT DSQRT QSQRT CSQRT CDSQRT CQSQRT	1	R D Q C DC QC					R D Q C DC QC	要素別処理関数
立方根 $y = x^{\frac{1}{3}}$	CBRT (X)	CBRT DCBRT QCBRT	1	R D Q					R D Q	要素別処理関数
指数 $y = e^x$ x < 88.722 x < 709.782 x < 11356.5 xr < 88.722 |xi| < 8.23e5 xr < 709.782 |xi| < 3.53e15 xr < 11356.5 |xi| < $2^{62} \times \pi$	EXP (X)	EXP DEXP QEXP CEXP CDEXP CQEXP	1	R D Q C DC QC					R D Q C DC QC	要素別処理関数
指数 $y = 10^x$ x < 38.531	EXP10 (X)	EXP10	1	R					R	

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
x < 308.254 x < 4932.0625		DEXP10 QEXP10		D Q					D Q	
指数 $y = 2^x$ x < 128.0 x < 1024.0 x < 16384.0	EXP2 (X)	 EXP2 DEXP2 QEXP2	1	 R D Q					 R D Q	
対数 y = log(x) x>0 y = PVlog(x) PV: 主値 x ≠ 0 + 0i	LOG (X)	 LOG ALOG DLOG QLOG CLOG CDLOG CQLOG	1	 R R D Q C DC QC					 R R D Q C DC QC	要素別処理関数
対数 y = log10(x) x > 0	LOG10 (X)	LOG10 ALOG10 DLOG10 QLOG10	1	R R D Q					R R D Q	
対数 y = log2(x) x > 0	LOG2 (X)	LOG2 ALOG2 DLOG2 QLOG2	1	R R D Q					R R D Q	
正弦 y = sin(x) |x| < 8.23e5 |x| < 3.53e15 $\|x\| < 2^{62} \times \pi$ |xr| < 8.23e5 |xi| < 89.415 |xr| < 3.53e15 |xi| < 710.475 $\|xr\| < 2^{62} \times \pi$ |xi| < 11357.125	SIN (X) (引数の単位はラジアン)	 SIN DSIN QSIN CSIN CDSIN CQSIN	1	 R D Q C DC QC					 R D Q C DC QC	要素別処理関数
正弦 y = sind(x) $= \sin(\pi / 180 \times x)$ |x| < 4.72e7 |x| < 2.03e17 $\|x\| < 2^{62} \times 180$	SIND (X) (引数の単位は度)	 SIND DSIND QSIND	1	 R D Q					 R D Q	
正弦 y = sinq(x) $= \sin(\pi / 2 \times x)$ |xr| < 5.24e5 |xi| < 56.92 |xr| < 2.25e15 |xi| < 452.30	SINQ (X) (引数の単位は象限)	 SINQ DSINQ QSINQ CSINQ CDSINQ	1	 R D Q C DC					 R D Q C DC	

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
|xr| < 2^{63} |xi| < 7230.125		CQSINQ		QC					QC	
余弦 y = cos(x) |x| < 8.23e5 |x| < 3.53e15 |x| < $2^{62} \times \pi$ |xr| < 8.23e5 |xi| < 89.415 |xr| < 3.53e15 |xi| < 710.475 |xr| < $2^{62} \times \pi$ |xi| < 11357.125	COS (X) (引数の単位はラジアン)	 COS DCOS QCOS CCOS CDCOS CQCOS	1	 R D Q C DC QC					 R D Q C DC QC	要素別処理関数
余弦 y = cosd(x) = cos($\pi/180 \times$ x) |x| < 4.72e7 |x| < 2.03e17 |x| < $2^{62} \times 180$	COSD (X) (引数の単位は度)	 COSD DCOSD QCOSD	1	 R D Q					 R D Q	
余弦 y = cosq(x) = cos($\pi/2 \times$ x) |xr| < 5.24e5 |xi| < 56.92 |xr| < 2.25e15 |xi| < 452.30 |xr| < 2^{63} |xi| < 7230.12	COSQ (X) (引数の単位は象限)	 COSQ DCOSQ QCOSQ CCOSQ CDCOSQ CQCOSQ	1	 R D Q C DC QC					 R D Q C DC QC	
正接 y = tan(x) |x| < 8.23e5 |x| < 3.53e15 |x| < $2^{62} \times \pi$	TAN (X) (引数の単位はラジアン)	 TAN DTAN QTAN － － －	1	 R D Q C DC QC					 R D Q C DC QC	要素別処理関数
正接 y = tand(x) = tan($\pi/180 \times$ x) |x| < 4.72e7 |x| < 2.03e17 |x| < $2^{63} \times 90$	TAND (X) (引数の単位は度)	 TAND DTAND QTAND	1	 R D Q					 R D Q	
正接 y = tanq(x) = tan($\pi/2 \times$ x)	TANQ (X) (引数の単位は象限)	TANQ DTANQ QTANQ	1	R D Q					R D Q	
余接 y = cot(x) |x| < 8.23e5	COTAN (X) (引数の単位はラジアン)	 COTAN	1	 R					 R	要素別処理関数

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
|x| < 3.53e15 |x| < 2^{62} × π		DCOTAN QCOTAN		D Q					D Q	
余接 y = cotd(x) = cot(π /180 × x) |x| < 4.72e7 |x| < 2.03e17 |x| < 2^{63} × 90	COTAND (X) (引数の単位は度)	COTAND DCOTAND QCOTAND	1	R D Q					R D Q	
余接 y = cotq(x) = cot(π /2 × x)	COTANQ (X) (引数の単位は象限)	COTANQ DCOTANQ QCOTANQ	1	R D Q					R D Q	
逆正弦 y = arc sin(x) |x| <= 1	ASIN (X) (関数結果の単位はラジアン)	ASIN DASIN QASIN － － －	1	R D Q C DC QC					R D Q C DC QC	要素別処理関数
逆正弦 y = arc sind(x) = 180 / π × arc sin(x) |x| <= 1	ASIND (X) (関数結果の単位は度)	ASIND DASIND QASIND	1	R D Q					R D Q	
逆正弦 y = arc sinq(x) = 2 / π × arc sin(x) |x| <= 1	ASINQ (X) (関数結果の単位は象限)	ASINQ DASINQ QASINQ	1	R D Q					R D Q	
逆余弦 y = arc cos(x) |x| <= 1	ACOS (X) (関数結果の単位はラジアン)	ACOS DACOS QACOS － － －	1	R D Q C DC QC					R D Q C DC QC	要素別処理関数
逆余弦 y = arc cosd(x) = 180 / π × arc cos(x) |x| <= 1	ACOSD (X) (関数結果の単位は度)	ACOSD DACOSD QACOSD	1	R D Q					R D Q	
逆余弦 y = arc cosq(x) = 2 / π × arc cos(x) |x| <= 1	ACOSQ (X) (関数結果の単位は象限)	ACOSQ DACOSQ QACOSQ	1	R D Q					R D Q	
逆正接 y = arc tan(x)	ATAN (X) (関数結果の単位はラジアン)	ATAN DATAN QATAN － － －	1	R D Q C DC QC					R D Q C DC QC	要素別処理関数
逆正接 y = arc tan(x1/x2) x1 ≠ 0, x2 ≠ 0	ATAN (Y,X) (関数結果の単位はラジアン)	－ － －	2	R D Q	R D Q				R D Q	

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
逆正接 y = arc tan(x1/x2) x1 ≠ 0, x2 ≠ 0	ATAN2 (Y,X) (関数結果の単位はラジアン)	ATAN2 DATAN2 QATAN2	2	R D Q	R D Q				R D Q	
逆正接 y = arc tand(x) = 180/π × arc tan(x)	ATAND (X) (関数結果の単位は度)	ATAND DATAND QATAND	1	R D Q					R D Q	
逆正接 y = arc tand(x1/x2) x1 ≠ 0, x2 ≠ 0	ATAN2D (Y,X) (関数結果の単位は度)	ATAN2D DATAN2D QATAN2D	2	R D Q	R D Q				R D Q	
逆正接 y = arc tanq(x) = 2/π × arc tan(x)	ATANQ (X) (関数結果の単位は象限)	ATANQ DATANQ QATANQ	1	R D Q					R D Q	
逆正接 y = arc tanq(x1/x2) x1 ≠ 0, x2 ≠ 0	ATAN2Q (Y,X) (関数結果の単位は象限)	ATAN2Q DATAN2Q QATAN2Q	2	R D Q	R D Q				R D Q	
双曲線正弦 y = sinh(x) |x| < 89.415 |x| < 710.475 |x| < 11357.125	SINH (X)	SINH DSINH QSINH － － －	1	R D Q C DC QC					R D Q C DC QC	要素別処理関数
双曲線余弦 y = cosh(x) |x| < 89.415 |x| < 710.475 |x| < 11357.125	COSH (X)	COSH DCOSH QCOSH － － －	1	R D Q C DC QC					R D Q C DC QC	要素別処理関数
双曲線正接 y = tanh(x)	TANH (X)	TANH DTANH QTANH － － －	1	R D Q C DC QC					R D Q C DC QC	要素別処理関数
逆双曲線正弦 y = arc sinh(x)	ASINH (X) (関数結果の単位はラジアン)	－ －	1	r c					r c	要素別処理関数
逆双曲線余弦 y = arc cosh(x) x >= 1	ACOSH (X) (関数結果の単位はラジアン)	－ －	1	r c					r c	要素別処理関数
逆双曲線正接 y = arc tanh(x) |x| <= 1.0	ATANH (X)	－ －	1	r c					r c	要素別処理関数

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
誤差関数 $y = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ 積分範囲は 0 ～ x	ERF (X)	ERF DERF QERF	1	r R D Q					r R D Q	要素別処理関数
相補誤差関数 $y = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt$ 積分範囲は x ～ ∞	ERFC (X)	ERFC DERFC QERFC	1	r R D Q					r R D Q	要素別処理関数
スケーリング相補誤差関数 $y = e^{x^2} \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt$ 積分範囲は x ～ ∞	ERFC _SCALED(X)	－	1	r					r	要素別処理関数
ガンマ関数 $y = \int_0^\infty e^{-t} t^{x-1} dt$ 積分範囲は 0 ～ ∞ x < 35.039860 x < 171.6243 x < 1755.0 x ≠ 0, x ≠ 負の整数	GAMMA (X)	GAMMA DGAMMA QGAMMA	1	r R D Q					r R D Q	要素別処理関数
ガンマ関数の絶対値の対数 $y = \log \left\| \int_0^\infty e^{-t} t^{x-1} dt \right\|$ 積分範囲は 0 ～ ∞ x < 0.403711e37 x < 2.55634e305 x < 1.048q+4928 x ≠ 0, x ≠ 負の整数	LOG_GAMMA (X)	－	1	r					r	要素別処理関数
	LGAMMA (X)	LGAMMA ALGAMA DLGAMA QLGAMA	1	R R D Q					R R D Q	要素別処理関数
ユークリッド距離関数 $\sqrt{x^2 + y^2}$	HYPOT (X,Y)	－	2	r	r				r	要素別処理関数
第1種ベッセル関数 (次数は0)	BESSEL_J0 (X)	－	1	r					r	要素別処理関数
第1種ベッセル関数 (次数は1)	BESSEL_J1 (X)	－	1	r					r	要素別処理関数
第1種ベッセル関数 (次数はN)	BESSEL_JN(N, X)	－	2	i	r				r	要素別処理関数
第1種ベッセル関数 (次数はN1～N2)	BESSEL_JN(N 1,N2,X)	－	3	i	i	r			r	変形関数
第2種ベッセル関数 (次数は0)	BESSEL_Y0 (X)	－	1	r					r	要素別処理関数

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
第2種ベッセル関数 (次数は1)	BESSEL_Y1 (X)	－	1	r					r	要素別処理関数
第2種ベッセル関数 (次数はN)	BESSEL_YN(N ,X)	－	2	i	r				r	要素別処理関数
第2種ベッセル関数 (次数はN1～N2)	BESSEL_YN(N 1,N2,X)	－	3	i	i	r			r	変形関数

r　：半精度実数型、単精度実数型、倍精度実数型、および4倍精度実数型。
c　：半精度複素数型、単精度複素数型、倍精度複素数型、4倍精度複素数型。
R　：単精度実数型、および総称名の半精度実数型。
D　：倍精度実数型。
Q　：4倍精度実数型。
C　：単精度複素数型、および総称名の半精度複素数型。
DC　：倍精度複素数型。
QC　：4倍精度複素数型。
備考1.　x、x1、x2、. . . は、実引数を表し、xj はj 番目の実引数であることを示します。
　　xr は実引数の実部、xi は実引数の虚部を表します。
備考2.　y は関数値を表します。

表A.3 文字関数

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
文字変換 文字 x の大小順序が関数値 y	ICHAR (C[,KIND])	ICHAR	1	ch1					I4	要素別処理関数
			2	ch1	i				i (*1)	
大小順序が x である文字が関数値 y	CHAR (I[,KIND])	－ CHAR CHAR －	1 or 2	I1 I2 I4 I8	i i i i				ch1 (*1) ch1 (*1) ch1 (*1) ch1 (*1)	
部分列の位置 (*2)	INDEX (STRING, SUBSTRING [,BACK,KIND])	INDEX	2	ch	a1				I4	要素別処理関数
			3	ch	a1	l			I4	
				ch	a1		i		i (*1)	
			4	ch	a1	l	i		i (*1)	
大小比較 (*3)	ifn (STRING_A, STRING_B) LGE LGT LLE LLT	 LGE LGT LLE LLT	2	 CH1 CH1 CH1 CH1	 CH1 CH1 CH1 CH1				 L4 L4 L4 L4	要素別処理関数
ASCII 文字	ACHAR (I[,KIND])	－	1	i					CH1	要素別処理関数
			2	i	i				ch1 (*1)	

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
ASCII 文字の位置	IACHAR (C[,KIND])	－	1	ch1					I4	要素別処理関数
			2	ch1	i				i (*1)	
後ろの空白を除く長さ	LEN_TRIM (STRING[,KIND])	－	1	ch					I4	要素別処理関数
			2	ch	i				i (*1)	
先頭の空白移動	ADJUSTL (STRING)	－	1	ch					a1	要素別処理関数
最後の空白移動	ADJUSTR (STRING)	－	1	ch					a1	要素別処理関数
共通文字の位置	SCAN (STRING, SET [,BACK,KIND])	－	2	ch	a1				I4	要素別処理関数
			3	ch	a1	l			I4	
				ch	a1		i		i (*1)	
			4	ch	a1	l	i		i (*1)	
文字列の検証	VERIFY (STRING, SET [,BACK,KIND])	－	2	ch	a1				I4	要素別処理関数
			3	ch	a1	l			I4	
				ch	a1		i		i (*1)	
			4	ch	a1	l	i		i (*1)	

－ ：対応する総称名、または個別名が存在しないことを示します。
I1 ：1バイトの整数型。
I2 ：2バイトの整数型。
I4 ：4バイトの整数型。
I8 ：8バイトの整数型。
i ：1バイトの整数型、2バイトの整数型、4バイトの整数型、および8バイトの整数型。
CH ：基本文字型。
CH1：1文字の基本文字型。
ch ：文字型。
ch1：1文字の文字型。
L4 ：4バイトの論理型。
l ：1バイトの論理型、2バイトの論理型、4バイトの論理型、および8バイトの論理型。
a1 ：第1引数と同じ型。
*1) 引数でKIND を指定したとき、結果の種別型パラメタはKIND の指示に従います。
*2) 文字列x1 を左から走査して、文字列x2 で識別される部分列があった場合の、その部分列の開始位置が関数値y です。x2 がx1 の部分列として存在しなければ y=0 です。
*3) ASCII コードによる文字列の大小比較です。LGE の関数値y は、x1 >= x2 ならば真、そうでなければ偽です。LGT の関数値y は、x1 > x2 ならば真、そうでなければ偽です。LLEの関数値y は、x1 <= x2 ならば真、そうでなければ偽です。LLT の関数値y は、x1 < x2 ならば真、そうでなければ偽です。
備考1. x、x1、x2、... は、実引数を表し、xj はj 番目の実引数であることを示します。
備考2. y は関数値を表します。
備考3. 個別名CHAR の組込み関数が手続引用の実引数として指定された場合に、それと結合される仮引数（仮手続）の引用における実引数は、4バイトの整数型だけが許されます。

表A.4 ビット関数

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
論理否定 x のビット単位の論理否定	NOT (I)	― ― INOT NOT JNOT ―	1	I1 I2 I2 I4 I4 I8					I1 I2 I2 I4 I4 I8	要素別処理関数
論理積 x1 と x2 の ビット単位の論理積	IAND (I,J)	― ― IIAND IAND JIAND ― ― ―	2	I1 I2 I2 I4 I4 I8 i nd	I1 I2 I2 I4 I4 I8 nd i				I1 I2 I2 I4 I4 I8 i i	要素別処理関数
	AND (I,J)	― ― AND ―	2	I1 I2 I4 I8	I1 I2 I4 I8				I1 I2 I4 I8	
論理和 x1 と x2 の ビット単位の論理和	IOR (I,J)	― ― IIOR IOR JIOR ― ― ―	2	I1 I2 I2 I4 I4 I8 i nd	I1 I2 I2 I4 I4 I8 nd i				I1 I2 I2 I4 I4 I8 i i	要素別処理関数
	OR (I,J)	― ― OR ―	2	I1 I2 I4 I8	I1 I2 I4 I8				I1 I2 I4 I8	
排他的論理和 x1 と x2 のビット単位の 排他的論理和	IEOR (I,J)	― ― IIEOR IEOR JIEOR ― ― ―	2	I1 I2 I2 I4 I4 I8 i nd	I1 I2 I2 I4 I4 I8 nd i				I1 I2 I2 I4 I4 I8 i i	要素別処理関数
	XOR (I,J)	― ― XOR ―	2	I1 I2 I4 I8	I1 I2 I4 I8				I1 I2 I4 I8	
論理シフト (*1) |x2| <= BIT_SIZE(x1)	ISHFT (I,SHIFT)	― ― IISHFT ISHFT JISHFT ―	2	I1 I2 I2 I4 I4 I8	i i i i i i				I1 I2 I2 I4 I4 I8	要素別処理関数

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
ビット並びの循環移動 (*2) |x2| <= BIT_SIZE(x1) 0 <= |x2| <= x3 <= BIT_SIZE(x1) x3 > 0	ISHFTC (I,SHIFT [,SIZE])	― ― IISHFTC ― JISHFTC ―	2	I1 I2 I2 I4 I4 I8	i i i i i i				I1 I2 I2 I4 I4 I8	要素別処理関数
		― ― IISHFTC ― JISHFTC ―	3	I1 I2 I2 I4 I4 I8	i i i i i i	i i i i i i			I1 I2 I2 I4 I4 I8	
左方向論理シフト (*3)	LSHIFT (I,SHIFT)	― ― LSHIFT ―	2	I1 I2 I4 I8	i i i i				I1 I2 I4 I8	要素別処理関数
	SHIFTL (I,SHIFT)	― ― ― ―	2	I1 I2 I4 I8	i i i i				I1 I2 I4 I8	
右方向算術シフト (*4)	RSHIFT (I,SHIFT)	― ― RSHIFT ―	2	I1 I2 I4 I8	i i i i				I1 I2 I4 I8	要素別処理関数
	SHIFTA (I,SHIFT)	― ― ― ―	2	I1 I2 I4 I8	i i i i				I1 I2 I4 I8	
右方向論理シフト (*5)	LRSHFT (I,SHIFT)	― ― LRSHFT ―	2	I1 I2 I4 I8	i i i i				I1 I2 I4 I8	要素別処理関数
	SHIFTR (I,SHIFT)	― ― ― ―	2	I1 I2 I4 I8	i i i i				I1 I2 I4 I8	
結合左シフト	DSHIFTL (I,J,SHIFT)	― ― ― ― ― ― ― ― ― ― ― ―	3	I1 I2 I4 I8 I1 I2 I4 I8 nd nd nd nd	I1 I2 I4 I8 nd nd nd nd I1 I2 I4 I8	i i i i i i i i i i i i			I1 I2 I4 I8 I1 I2 I4 I8 I1 I2 I4 I8	要素別処理関数

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
結合右シフト	DSHIFTR (I,J,SHIFT)	－ － － － － － － － － － －	3	I1 I2 I4 I8 I1 I2 I4 I8 nd nd nd	I1 I2 I4 I8 nd nd nd nd I1 I2 I4 I8	i i i i i i i i i i i i			I1 I2 I4 I8 I1 I2 I4 I8 I1 I2 I4 I8	要素別処理関数
算術シフト	ISHA (I,SHIFT)	－ － － －	2	I1 I2 I4 I8	i i i i				I1 I2 I4 I8	要素別処理関数
循環シフト	ISHC (I,SHIFT)	－ － － －	2	I1 I2 I4 I8	i i i i				I1 I2 I4 I8	要素別処理関数
論理シフト	ISHL (I,SHIFT)	－ － － －	2	I1 I2 I4 I8	i i i i				I1 I2 I4 I8	要素別処理関数
ビットの反転	IBCHNG (I,POS)	－ － － －	2	I1 I2 I4 I8	i i i i				I1 I2 I4 I8	要素別処理関数
ビットセット (*6) 0 <= x2 < BIT_SIZE(x1)	IBSET (I,POS)	－ － IIBSET IBSET JIBSET －	2	I1 I2 I2 I4 I4 I8	i i i i i i				I1 I2 I2 I4 I4 I8	要素別処理関数
ビットクリア (*7) 0 <= x2 < BIT_SIZE(x1)	IBCLR (I,POS)	－ － IIBCLR IBCLR JIBCLR －	2	I1 I2 I2 I4 I4 I8	i i i i i i				I1 I2 I2 I4 I4 I8	要素別処理関数
ビットテスト (*8) 0 <= x2 < BIT_SIZE(x1)	BTEST (I,POS)	－ － BITEST BJTEST BTEST －	2	I1 I2 I2 I4 I4 I8	i i i i i i				L4 L4 L4 L4 L4 L4	要素別処理関数
ビットの取出し 0 <= x2 + x3 <= BIT_SIZE(x1)	IBITS (I,POS,LEN)	－ － IIBITS －	3	I1 I2 I2 I4	i i i i	i i i i			I1 I2 I2 I4	要素別処理関数

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
x2 > 0 , x3 > 0 (*9)		JIBITS —		I4 I8	i i	i i			I4 I8	
ゼロ拡張	IZEXT (A)	—	1	L1					I2	要素別処理関数
	IZEXT2 (A)	—	1	I2					I2	
	JZEXT (A)	—	1	L4					I4	
	JZEXT2 (A)	—	1	I2					I4	
	JZEXT4 (A)	—	1	I4					I4	
ビット大小比較 I <= J	BLE (I,J)	— — — —	2	i i nd nd	i nd i nd				L4 L4 L4 L4	要素別処理関数
ビット大小比較 I < J	BLT (I,J)	— — — —	2	i i nd nd	i nd i nd				L4 L4 L4 L4	要素別処理関数
ビット大小比較 I >= J	BGE (I,J)	— — — —	2	i i nd nd	i nd i nd				L4 L4 L4 L4	要素別処理関数
ビット大小比較 I > J	BGT (I,J)	— — — —	2	i i nd nd	i nd i nd				L4 L4 L4 L4	要素別処理関数
左からI ビットまで 1をセット	MASKL (I[,KIND])	— —	1 2	i i	 i				I4 i (*10)	要素別処理関数
右からI ビットまで 1をセット	MASKR (I[,KIND])	— —	1 2	i i	 i				I4 i (*10)	要素別処理関数
先行ビット 0を数える	LEADZ (I)	—	1	i					I4	要素別処理関数
後続ビット 0を数える	TRAILZ (I)	—	1	i					I4	要素別処理関数
ビット1を数える	POPCNT (I)	—	1	i					I4	要素別処理関数
0と1でのパリティ	POPPAR (I)	—	1	i					I4	要素別処理関数
MASKのビット値に従って どちらか一方のビット選択	MERGE_BITS (I,J,MASK)	— — — — — —	3	I1 I2 I4 I8 I1 I2	I1 I2 I4 I8 nd nd	I1 I2 I4 I8 I1 I2			I1 I2 I4 I8 I1 I2	要素別処理関数

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
		—		I4	nd	I4			I4	
		—		I8	nd	I8			I8	
		—		nd	I1	I1			I1	
		—		nd	I2	I2			I2	
		—		nd	I4	I4			I4	
		—		nd	I8	I8			I8	
		—		I1	I1	nd			I1	
		—		I2	I2	nd			I2	
		—		I4	I4	nd			I4	
		—		I8	I8	nd			I8	
		—		i	nd	nd			i	
		—		nd	i	nd			i	

ー ：対応する総称名、または個別名が存在しないことを示します。
I1 ：1バイトの整数型。
I2 ：2バイトの整数型。
I4 ：4バイトの整数型。
I8 ：8バイトの整数型。
i ：1バイトの整数型、2バイトの整数型、4バイトの整数型、および8バイトの整数型。
L4 ：4バイトの論理型。
nd ：非10進定数表現。
*1) x2 < 0ならば、x1 を右に|x2|ビット論理シフトします。x2 > 0ならば、x1 を
左に|x2|ビット論理シフトします。x2=0 ならば、シフトしません。
*2) 引数が2個の場合、x2 < 0ならば、x1 を右に |x2|ビット循環シフトし、
x2 > 0ならば、x1を左に |x2|ビット循環シフトします。引数が3個の場合、
x2 < 0ならば、x1 の右端のx3 個のビットを右に |x2|ビット循環シフトし、
x2 > 0ならば、x1 の右端のx3 個のビットを左に |x2|ビット循環シフトします。
いずれの場合も、x2 = 0 ならば、シフトしません。
*3) x1 を左にx2 ビット論理シフトします。
*4) x1 を右にx2 ビット算術シフトします。
*5) x1 を右にx2 ビット論理シフトします。
*6) x1 のx2 番目のビットを1にします。x2 番目とは、最右端（最下位）ビットを0として、
右から数えた位置です。
*7) x1 のx2 番目のビットを0にします。x2 番目とは、最右端（最下位）ビットを0として、
右から数えた位置です。
*8) x1 のx2 番目のビットを調べ、1ならば真、0ならば偽の値です。x2 番目とは、
最右端（最下位）ビットを0として、右から数えた位置です。
*9) x1 のx2 番目のビット位置から左にx3 個のビットを取り出し、右詰めにして、
残りのビットを0とした値です。x2 番目とは、最右端（最下位）ビットを0として、
右から数えた位置です。
*10) 引数でKIND を指定した時、結果の種別型パラメタはKIND の指示に従います。
備考1. x、x1、x2、... は、実引数を表し、xj はj 番目の実引数であることを示します。
備考2. y は関数値を表します。
備考3. 以下の個別名をもつ組込み関数が手続引用の実引数として指定された場合に、
それと結合される仮引数（仮手続）の引用における実引数は、
4バイトの整数型だけが許されます。
ISHFT、IBSET、IBCLR、BTEST、LSHIFT、RSHIFT、LRSHFT

表A.5 ビット複写サブルーチン

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	分類
ビット列の複写 (*1)	MVBITS (FROM,FROMPOS,LEN,TO,TOPOS)	－ － － －	5	I1 I2 I4 I8	i i i i	i i i i	I1 I2 I4 I8	i i i i	要素別処理サブルーチン

－ ：対応する総称名、または個別名が存在しないことを示します。
I1 ：1バイトの整数型。
I2 ：2バイトの整数型。
I4 ：4バイトの整数型。
I8 ：8バイトの整数型。
i ：1バイトの整数型、2バイトの整数型、4バイトの整数型、および8バイトの整数型。
*1) x1 のx2 番目のビット位置から左にx3 個のビットをx4 のx5 番目のビットを
開始位置とする場所に複写します。x2 番目およびx5 番目とは、
最右端（最下位）ビットを0として、右から数えた位置です。
実引数x4 には、x1 と同じ変数を指定してもかまいません。
実引数は、0 <= x2 + x3 <= BIT_SIZE(x1) 、 0 <= x5 + x3 <= BIT_SIZE(x4) 、
x2 >= 0 、x3 >= 0 、および x5 >= 0 の関係を満たさなければなりません。
備考. x、x1、x2、... は、実引数を表し、xj は j 番目の実引数であることを示します。

表A.6 問合せ組込み関数

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
文字長 文字型のデータxの長さが関数値y	LEN (STRING[,KIND])	LEN	1	ch					I4	問合せ関数
			2	ch	i				i (*3)	
割付け状態 (*1)	ALLOCATED (ALLOCATABLE)	－	1	ay					L4	問合せ関数
ビット数 (*2)	BIT_SIZE (I)	－ － － －	1	I1 I2 I4 I8					I1 I2 I4 I8	問合せ関数
有効数字のビット数	DIGITS (X)	－ －	1	i r					I4 I4	問合せ関数
十分小さい正のモデル数	EPSILON (X)	－ － －	1	R D Q					R D Q	問合せ関数
モデル表現数の底	RADIX (X)	－ －	1	i r					I4 I4	問合せ関数
モデル表現数の最小正値	TINY (X)	－ － －	1	R D Q					R D Q	問合せ関数
モデル表現数の最大数	HUGE (X)	－ － － － －	1	I1 I2 I4 I8 R					I1 I2 I4 I8 R	問合せ関数

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
		ー ー		D Q					D Q	
モデル表現の最大指数	MAXEXPONENT (X)	ー	1	r					I4	問合せ関数
モデル表現の最小指数	MINEXPONENT (X)	ー	1	r					I4	問合せ関数
種別型パラメタ	KIND (X)	ー	1	it					I4	問合せ関数
任意引数判定	PRESENT (A)	ー	1	ay					L4	問合せ関数
10進精度	PRECISION (X)	ー ー ー ー ー ー	1	R D Q C DC QC					I4 I4 I4 I4 I4 I4	問合せ関数
10進指数範囲	RANGE (X)	ー ー ー	1	i r cm					I4 I4 I4	問合せ関数
ポインタ結合状態	ASSOCIATED (POINTER [,TARGET])	ー	1	ay					L4	問合せ関数
		ー	2	ay	ay				L4	
配列の下限	LBOUND (ARRAY [,DIM,KIND])	ー	1	ay					I4	問合せ関数
			2	ay	i				I4	
				ay		i			i (*3)	
			3	ay	i	i			i (*3)	
配列の上限	UBOUND (ARRAY [,DIM,KIND])	ー	1	ay					I4	問合せ関数
			2	ay	i				I4	
				ay		i			i (*3)	
			3	ay	i	i			i (*3)	
配列の形状	SHAPE (SOURCE [,KIND])	ー	1	ay					I4	問合せ関数
			2	ay	i				i (*3)	
宣言要素の数	SIZE (ARRAY [,DIM,KIND])	ー	1	ay					I4	問合せ関数
			2	ay	i				I4	
				ay		i			i (*3)	
			3	ay	i	i			i (*3)	
次元数	RANK (A)	ー	1	ay					I4	問合せ関数
コマンド引数の個数	COMMAND_ ARGUMENT_ COUNT ()	ー	0	ー					I4	問合せ関数
改行文字	NEW_LINE (A)	ー	1	ch					ch1	問合せ関数

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
バイト単位のサイズ	SIZEOF (X)	ー	1	ay					I8	問合せ関数
ビット単位のサイズ	STORAGE_SIZE (A[,KIND])	ー ー	1 2	ay ay	 i				I4 i (*3)	問合せ関数
Aの動的な型がMOLDの動的な型の型拡張か	EXTENDS_TYPE_OF (A,MOLD)	ー	2	TY	TY				L4	問合せ関数
Aの動的な型がBの動的な型と同じか	SAME_TYPE_AS (A,B)	ー	2	TY	TY				L4	問合せ関数
配列が連続した記憶域を持つか	IS_CONTIGUOUS (ARRAY)	ー	1	ay					L4	問合せ関数

ー　：対応する総称名、または個別名が存在しないことを示します。
I1　：1バイトの整数型。
I2　：2バイトの整数型。
I4　：4バイトの整数型。
I8　：8バイトの整数型。
i　：1バイトの整数型、2バイトの整数型、4バイトの整数型、および8バイトの整数型。
R　：半精度実数型、および単精度実数型。
D　：倍精度実数型。
Q　：4倍精度実数型。
r　：半精度実数型、単精度実数型、倍精度実数型、および4倍精度実数型。
C　：半精度複素数型、および単精度複素数型。
DC　：倍精度複素数型。
QC　：4倍精度複素数型。
cm　：半精度複素数型、単精度複素数型、倍精度複素数型、および4倍精度複素数型。
CH　：基本文字型。
CH1　：1文字の基本文字型。
ch　：文字型。
ch1　：1文字の文字型。
L4　：4バイトの論理型。
it　：派生型を除くすべての型。
ay　：すべての型。
TY　：拡張可能型。
*1)　関数値y は、割付け変数ALLOCATABLE が現在割り付けられているならば真、
　　割り付けられていないのならば偽の値です。変数ALLOCATABLE は、
　　ALLOCATABLE 属性をもつ割付け変数でなければなりません。
*2)　x のビット数が関数値y です。
*3)　引数でKIND を指定したとき、結果の種別型パラメタはKIND の指示に従います。
備考1.　x は、実引数を表します。
備考2.　y は、関数値を表します。

表A.7 日付および時刻サブルーチン

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	分類
プロセッサ時間	CPU_TIME (TIME)	ー	1	R D Q					サブルーチン
実時間時計のデータおよび日付 (*1)	DATE_AND_TIME ([DATE] [,TIME]	ー	1	CH8					サブルーチン
		ー	2	CH8	CHA				
		ー	3	CH8	CHA	CH5			

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	分類
	[,ZONE] [,VALUES])	ー	4	CH8	CHA	CH5	I4		
実時間の時計からの整数 (*2)	SYSTEM_CLOCK ([COUNT] [,COUNT_RATE] [,COUNT_MAX])	ー	1	i					サブルーチン
		ー	2	i	i				
		ー		i	R D Q				
		ー	3	i	i	i			
		ー		i	R D Q	i			

ー ：対応する総称名、または個別名が存在しないことを示します。
I4 ：基本整数型。
i ：1バイトの整数型、2バイトの整数型、4バイトの整数型、および8バイトの整数型。
R ：単精度実数型。
D ：倍精度実数型。
Q ：4倍精度実数型。
CH5：長さ5以上の基本文字型。
CH8：長さ8以上の基本文字型。
CHA：長さ10以上の基本文字型。
*1) x1 には、左詰めに'ccyymmdd' が設定されます。ここで、cc は世紀、yy は西暦年、
mm は月、dd は日を表す数字列です。x2 には、左詰めに'hhmmss.sss' が設定されます。
ここで、hh は時、mm は分、ss.sss は秒およびミリ秒を表す数字列です。
x3 には、左詰めに'shhmm'が設定されます。ここで、s は符号であり、hh および mm は
時および端数の分を表す数字列です。配列x4 の第1要素には西暦年が設定され、
第2要素には月が設定され、第3要素には日が設定され、第4要素には分で表した
協定世界時(UTC) からの時差が設定され、第5要素には0～23の範囲の時が設定され、
第6要素には0～59の範囲の分が設定され、第7要素には0～60の範囲の秒が設定され、
第8要素には0～999 の範囲のミリ秒が設定されます。実引数x1、x2、およびx3 は、
スカラでなければなりません。実引数x4 は、大きさ8以上の次元数1の
配列でなければなりません。
*2) x1が指定されている場合、x1の型に合わせた以下の値がx2とx3に設定されます。
設定できない場合、x1には-HUGE(COUNT)、x2およびx3には0が設定されます。

	x1の型			
	1バイト整数型	2バイト整数型	4バイト整数型	8バイト整数型
x2の値	1	1000	1000	1000000
x3の値	127	32767	2147483647	9223372036854775807

x1が指定されていない場合、x2とx3の型に従った値が設定されます。
備考. x、x1、x2、... は、実引数を表し、xj はj 番目の実引数であることを示します。

表A.8 アドレス取得関数

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
アドレス取出し (*1)	LOC (X)	ー	1	ay					I8	アドレス取得関数

－ ：対応する総称名、または個別名が存在しないことを示します。
I8 ： 8バイトの整数型。
ay ：すべての型。
*1) 実引数x で指定される変数のアドレス値が関数値です。
備考1. x は、実引数を表します。
備考2. y は、関数値を表します。

表A.9 共配列組込み手続

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
アトミックな定義	ATOMIC_DEFINE(ATOM, VALUE [, STAT])	－	2 or 3	I4(*1)	i	i			SU	アトミックサブルーチン
				L4(*1)	l	i				
アトミックな参照	ATOMIC_REF(VALUE, ATOM [, STAT])	－	2 or 3	i	I4(*1)	i			SU	アトミックサブルーチン
				l	L4(*1)	i				
各像の最大値	CO_MAX(A [, RESULT_IMAGE, STAT, ERRMSG])	－	1 or 2 or 3 or 4	i(*2)	i	i	CH		SU	集団サブルーチン
				r(*2)	i	i	CH			
				ch(*2)	i	i	CH			
各像の最小値	CO_MIN(A [, RESULT_IMAGE, STAT, ERRMSG])	－	1 or 2 or 3 or 4	i(*2)	i	i	CH		SU	集団サブルーチン
				r(*2)	i	i	CH			
				ch(*2)	i	i	CH			
各像の合計値	CO_SUM(A [, RESULT_IMAGE, STAT, ERRMSG])	－	1 or 2 or 3 or 4	i(*2)	i	i	CH		SU	集団サブルーチン
				r(*2)	i	i	CH			
				cm(*2)	i	i	CH			
共添字から像番号への変換	IMAGE_INDEX(COARRAY, SUB)	－	2	ay(*2)	i				I4	問合せ関数
共下限	LCOBOUND(COARRAY [, DIM, KIND])	－	1	ay(*2)					I4	問合せ関数
			2	ay(*2)	i				I4	
				ay(*2)		i			i(*3)	
			3	ay(*2)	i	i			i(*3)	
像の数	NUM_IMAGES()	－	0						I4	変形関数

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	関数結果の型	分類
呼び出した像の共添字	THIS_IMAGE()	－	0						I4	変形関数
	THIS_IMAGE(COARRAY [, DIM])		1 or 2	ay(*2)	I4				I4	
共上限	UCOBOUND(COARRAY [, DIM, KIND])	－	1	ay(*2)					I4	問合せ関数
			2	ay(*2)	i				I4	
				ay(*2)		i			i(*3)	
			3	ay(*2)	i	i			i(*3)	

－　：対応する総称名、または個別名が存在しないことを示します。
I4　：4バイトの整数型。
i　：1バイトの整数型、2バイトの整数型、4バイトの整数型、および8バイトの整数型。
l　：1バイトの論理型、2バイトの論理型、4バイトの論理型、および8バイトの論理型。
r　：半精度実数型、単精度実数型、倍精度実数型、および4倍精度実数型。
cm　：半精度複素数型、単精度複素数型、倍精度複素数型、および4倍精度複素数型。
CH　：基本文字型。
ch　：文字型。
L4　：4バイトの論理型。
ay　：すべての型。
SU　：組込みサブルーチンであり、関数ではありません。
*1)　共配列、または共添字付き実体でなければなりません。
*2)　共配列でなければなりません。
*3)　引数でKIND を指定したとき、結果の種別型パラメタはKIND の指示に従います。

表A.10 その他の手続

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	第6引数の型	関数結果の型	分類
空状態のポインタ、または未割付けの割付け	NULL ([MOLD])	－	0 or 1	ay						a1 (*2)	変形関数
文字複写連結	REPEAT (STRING, NCOPIES)	－	2	ch	i					ch	変形関数
後ろの空白削除	TRIM (STRING)	－	1	ch						ch	変形関数
整数型の種別型パラメタ値	SELECTED_ INT_KIND (R)	－	1	i						I4	変形関数
文字型の種別型パラメタ値	SELECTED_ CHAR_KIND (NAME)	－	1	CH 1						I4	変形関数
実数型の種別型パラメタ値	SELECTED_ REAL_KIND ([P][,R][,RADIX])	－	1 or 2 or 3	i	i	i				I4	変形関数

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	第6引数の型	関数結果の型	分類
コマンドライン	EXECUTE_COMMAND_ LINE (COMMAND [,WAIT] [,EXITSTAT] [,CMDSTAT] [,CMDMSG])	－	1 or 2 or 3 or 4 or 5	CH	L4	I4	I4	CH		SU	サブルーチン
指定した値の要素の位置	FINDLOC (ARRAY, VALUE, DIM, [,MASK] [,KIND] [,BACK])	－	3 or 4 or 5 or 6	I1 I2 I4 I8 R D Q C DC QC CH	I1 I2 I4 I8 R D Q C DC QC CH	i	l	i	l	i (*1)	変形関数
配列内の論理積	IALL (ARRAY [,MASK])	－	1 or 2	i	l					i	変形関数
	IALL (ARRAY, DIM [,MASK])	－	2 or 3	i	i	l				i	変形関数
配列内の論理和	IANY (ARRAY [,MASK])	－	1 or 2	i	l					i	変形関数
	IANY (ARRAY, DIM [,MASK])	－	2 or 3	i	i	l				i	変形関数
配列内の排他的論理和	IPARITY (ARRAY [,MASK])	－	1 or 2	i	l					i	変形関数
	IPARITY (ARRAY, DIM [,MASK])	－	2 or 3	i	i	l				i	変形関数
配列のL2 norm	NORM2 (X [,DIM])	－	1 or 2	r	i					r	変形関数
配列の論理非等価	PARITY (MASK [,DIM])	－	1 or 2	l	i					l	変形関数

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	第6引数の型	関数結果の型	分類
第2引数の形式に変換	TRANSFER (SOURCE,MOLD [,SIZE])	―	2	ay	ay					a2	変形関数
		―	3	ay	ay	i				a2	
2つのベクトルの内積	DOT_PRODUCT (VECTOR_A, VECTOR_B)	― ―	2	ar	ar					cu	変形関数
				l	l					cu	
行列の積	MATMUL (MATRIX_A, MATRIX_B)	― ―	2	ar	ar					cu	変形関数
				l	l					cu	
すべてが真かの判定	ALL (MASK [,DIM])	―	1	l						a1	変形関数
		―	2	l	i					a1	
一部が真かの判定	ANY(MASK [,DIM])	―	1	l						a1	変形関数
		―	2	l	i					a1	
真を数える	COUNT (MASK [,DIM,KIND])	―	1	l						I4	変形関数
			2	l	i					I4	
				l		i				i (*1)	
			3	l	i	i				i (*1)	
配列内の最大値	MAXVAL (ARRAY [,MASK])	― ― ― ― ― ― ― ―	1 or 2	I1 I2 I4 I8 R D Q ch	l l l l l l l l					I1 I2 I4 I8 R D Q ch	変形関数
	MAXVAL (ARRAY, DIM[,MASK])	― ― ― ― ― ― ― ―	2 or 3	I1 I2 I4 I8 R D Q ch	i i i i i i i i	l l l l l l l l				I1 I2 I4 I8 R D Q ch	
配列内の最小値	MINVAL (ARRAY [,MASK])	― ― ― ― ― ― ― ―	1 or 2	I1 I2 I4 I8 R D Q ch	l l l l l l l l					I1 I2 I4 I8 R D Q ch	変形関数
	MINVAL (ARRAY, DIM[,MASK])	― ― ― ― ― ―	2 or 3	I1 I2 I4 I8 R	i i i i i	l l l l l				I1 I2 I4 I8 R	

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	第6引数の型	関数結果の型	分類
		― ―		D Q ch	i i i	l l l				D Q ch	
配列内の積	PRODUCT (ARRAY [,MASK])	― ― ― ― ― ― ― ― ― ―	1 or 2	I1 I2 I4 I8 R D Q C DC QC	l l l l l l l l l l					I1 I2 I4 I8 R D Q C DC QC	変形関数
	PRODUCT (ARRAY, DIM[,MASK])	― ― ― ― ― ― ― ― ― ―	2 or 3	I1 I2 I4 I8 R D Q C DC QC	i i i i i i i i i i	l l l l l l l l l l				I1 I2 I4 I8 R D Q C DC QC	
配列内の和	SUM (ARRAY [,MASK])	― ― ― ― ― ― ― ― ― ―	1 or 2	I1 I2 I4 I8 R D Q C DC QC	l l l l l l l l l l					I1 I2 I4 I8 R D Q C DC QC	変形関数
	SUM (ARRAY, DIM[,MASK])	― ― ― ― ― ― ― ― ― ―	2 or 3	I1 I2 I4 I8 R D Q C DC QC	i i i i i i i i i i	l l l l l l l l l l				I1 I2 I4 I8 R D Q C DC QC	
1次元の配列化	PACK (ARRAY, MASK[,VECTOR])	―	2	ay	l					a1	変形関数
		―	3	ay	l	a1				a1	
配列の形状変換	RESHAPE (SOURCE, SHAPE [,PAD] [,ORDER])	―	2 or 3 or 4	ay	i	a1	i			a1	変形関数

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	第6引数の型	関数結果の型	分類
次元の追加配列	SPREAD (SOURCE, DIM,NCOPIES)	－	3	ay	i	i				a1	変形関数
アンパック	UNPACK (VECTOR, MASK,FIELD)	－	3	ay	l	a1				a1	変形関数
配列値循環シフト	CSHIFT (ARRAY, SHIFT[,DIM])	－	2 or 3	ay	i	i				a1	変形関数
配列値終端シフト	EOSHIFT (ARRAY, SHIFT, [,BOUNDARY] [,DIM])	－	2 or 3 or 4	ay	i	a1	i			a1	変形関数
次元数2配列置換	TRANSPOSE (MATRIX)	－	1	ay						a1	変形関数
最大値の要素位置	MAXLOC (ARRAY [,MASK,KIND,BACK])	－ － －	1 or 2 or 3	i r ch	l l l		l l l			I4 I4 I4	変形関数
		－ － －	2 or 3 or 4	i r ch	l l l	i i i	l l l			i(*1) i(*1) i(*1)	
	MAXLOC (ARRAY, DIM[,MASK, KIND,BACK])	－ － －	2 or 3 or 4	i r ch	i i i	l l l		l l l		I4 I4 I4	
		－ － －	3 or 4 or 5	i r ch	i i i	l l l	i i i	l l l		i(*1) i(*1) i(*1)	
最小値の要素位置	MINLOC (ARRAY [,MASK,KIND,BACK])	－ － －	1 or 2 or 3	i r ch	l l l		l l l			I4 I4 I4	変形関数
		－ － －	2 or 3 or 4	i r ch	l l l	i i i	l l l			i(*1) i(*1) i(*1)	

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	第6引数の型	関数結果の型	分類
	MINLOC (ARRAY, DIM[,MASK, KIND,BACK])	ー ー ー	2 or 3 or 4	i r ch	i i i	l l l		l l l		I4 I4 I4	
		ー ー ー	3 or 4 or 5	i r ch	i i i	l l l	i i i	l l l		i(*1) i(*1) i(*1)	
擬似乱数	RANDOM_NUMBER (HARVEST)	ー	1	r						SU	サブルーチン
擬似乱数の始動 / 問合せ	RANDOM_SEED ([SIZE] [,PUT] [,GET])	ー	0	ー						SU	サブルーチン
		ー	1	I4						SU	
		ー	1		I4					SU	
		ー	1			I4				SU	
コマンド全体	GET_COMMAND ([COMMAND] [,LENGTH] [,STATUS])	ー	0 or 1 or 2 or 3	CH	I4	I4				SU	サブルーチン
コマンド引数	GET_COMMAND_ARG UMENT (NUMBER [,VALUE] [,LENGTH] [,STATUS])	ー	1 or 2 or 3 or 4	I4	CH	I4	I4			SU	サブルーチン
環境変数の値	GET_ENVIRONMENT_V ARIABLE (NAME [,VALUE] [,LENGTH] [,STATUS] [,TRIM_NAME])	ー	1 or 2 or 3 or 4 or 5	CH	CH	I4	I4	l		SU	サブルーチン
割付けの移動	MOVE_ALLOC (FROM,TO)	ー	2	ay	a1					SU	純粋サブルーチン
引数の値渡し	VAL (X)	ー	1	I1 I2 I4 I8						I1 I2 I4 I8	引数値渡し関数

機能	総称名 (引数 キーワード)	個別名	引数の数	第1引数の型	第2引数の型	第3引数の型	第4引数の型	第5引数の型	第6引数の型	関数結果の型	分類
				L1 L2 L4 L8 R D						L1 L2 L4 L8 R D	
領域確保	MALLOC (SIZE)	ー ー ー ー	1	I1 I2 I4 I8						I8 I8 I8 I8	領域獲得関数
領域解放	FREE (ADDR)	ー	1	I8						SU	サブルーチン

ー　：対応する総称名、または個別名が存在しないことを示します。
I1　：1バイトの整数型。
I2　：2バイトの整数型。
I4　：4バイトの整数型。
I8　：8バイトの整数型。
i　：1バイトの整数型、2バイトの整数型、4バイトの整数型、および8バイトの整数型。
R　：半精度実数型、および単精度実数型。
D　：倍精度実数型。
Q　：4倍精度実数型。
r　：半精度実数型、単精度実数型、倍精度実数型、および4倍精度実数型。
C　：半精度複素数型、および単精度複素数型。
DC　：倍精度複素数型。
QC　：4倍精度複素数型。
CH　：基本文字型。
CH1　：1文字の基本文字型。
ch　：文字型。
ch1　：1文字の文字型。
L4　：4バイトの論理型。
l　：1バイトの論理型、2バイトの論理型、4バイトの論理型、および8バイトの論理型。
a1　：第1引数と同じ型。
a2　：第2引数と同じ型。
cu　：第1引数と第2引数の演算結果の型。
ar　：1バイトの整数型、2バイトの整数型、4バイトの整数型、8バイトの整数型、
　　半精度実数型、単精度実数型、倍精度実数型、4倍精度実数型、
　　半精度複素数型、単精度複素数型、倍精度複素数型、および4倍精度複素数型。
ay　：すべての型。
SU　：組込みサブルーチンであり、関数ではありません。
*1)　引数でKIND を指定したとき、結果の種別型パラメタはKIND の指示に従います。
*2)　関数結果は空状態、または未割付けです。
備考1.　x は、実引数を表します。
備考2.　y は、関数値を表します。

付録B　サービスルーチン一覧

サービスルーチンは、外部手続として提供されるサブルーチン群および関数群です。また、サービスルーチンの引用仕様を定義したモジュール‘SERVICE_ROUTINES’も合わせて提供されます。

サービスサブルーチンは、CALL 文にサービスサブルーチン名を指定することにより引用することができます。サービスサブルーチンは純粋サブルーチンではありません。

サービス関数は、関数引用の形式で引用することができます。サービス関数は純粋関数ではありません。

サービスルーチンは、手続引用として指定するだけでも使用できますが、各サービスルーチンの引用仕様を定義したモジュール‘SERVICE_ROUTINES’を引用することにより、引数の数、引数の型などの正当性の検査が実施されます。また、引数キーワードも使用可能になります。モジュール‘SERVICE_ROUTINES’には、すべてのサービスルーチンおよびサービス関数の引用仕様宣言が含まれます。引用する場合はUSE 文にONLY 句を指定し、実際に引用するサービスルーチンの引用仕様だけを参照結合することをお勧めします。モジュール‘SERVICE_ROUTINES’を引用しない場合、サービスルーチン引用の正当性の検査は実施されません。

各サービスルーチンの詳細については、“第2章 文および手続の詳細”を参照してください。

表B.1 サービスサブルーチン

機能	サービスサブルーチン名 (引数キーワード) (注)	引数の数	引数
強制異常終了	ABORT	0	なし
ビットクリア	BIC (POS, I)	2	POS:基本整数型スカラ I:基本整数型スカラ
ビットセット	BIS (POS, I)	2	POS:基本整数型スカラ I:基本整数型スカラ
CPU時間の取得	CLOCK (G, I1, I2)	3	G:4バイトの整数型、8バイトの整数型、単精度実数型、倍精度実数型、または4倍精度実数型のスカラ変数 I1:基本整数型スカラ I2:基本整数型スカラ
CPU時間の取得	CLOCKM (I)	1	I:4バイトの整数型のスカラ変数
CPU時間の取得	CLOCKV (G1, G2, I1, I2)	4	G1:4バイトの整数型、8バイトの整数型、単精度実数型、倍精度実数型、または4倍精度実数型のスカラ変数 G2:4バイトの整数型、8バイトの整数型、単精度実数型、倍精度実数型、または4倍精度実数型のスカラ変数 I1:基本整数型スカラ I2:基本整数型スカラ
実行年月日の取得	DATE (CH)	1	CH:長さ8の基本文字型スカラ
標準出力ファイルへの文字列出力	ERROR (STRING)	1	STRING:基本文字型スカラ
エラー項目の退避	ERRSAV (ERRNO, DAREA)	2	ERRNO:基本整数型スカラ DAREA:長さ16の基本文字型スカラ
エラー制御表の情報変更	ERRSET (ERRNO, ESTOP, MPRINT, TRACE, UEXIT, R)	6	ERRNO:基本整数型スカラ ESTOP:基本整数型スカラ MPRINT:基本整数型スカラ TRACE:基本整数型スカラ UEXIT:基本整数型スカラ R:基本整数型スカラ
エラー項目の格納	ERRSTR (ERRNO, DAREA)	2	ERRNO:基本整数型スカラ DAREA:長さ16の基本文字型スカラ
トレースバックマップ出力	ERRTRA	0	なし
実行中止	EXIT	0	なし

機能	サービスサブルーチン名 (引数キーワード) (注)	引数の数	引数
現在日付および現在時刻の取得	FDATE (STRING)	1	STRING:基本文字型スカラ
バッファ上のデータ出力	FLUSH (UNIT)	1	UNIT:基本整数型スカラ
領域の解放	FREE (ADDR)	1	ADDR: 8バイトの整数型スカラ
オプション文字列の取得	GETARG (ARGNO, ARGST)	2	ARGNO:基本整数型スカラ ARGST:基本文字型スカラ
引数文字列の取得	GETCL (STRING)	1	STRING:基本文字型スカラ
現在日付の取得	GETDAT (YEAR, MONTH, DAY)	3	YEAR:2バイトの整数型スカラ MONTH:2バイトの整数型スカラ DAY:2バイトの整数型スカラ
環境変数値の取得	GETENV (ENV , STRING)	2	ENV:基本文字型スカラ STRING:基本文字型スカラ
ログイン名の取得	GETLOG (UNAME)	1	UNAME:基本文字型スカラ
オプションの文字長および文字列の取得	GETPARM (LEN, PARM)	2	LEN:基本整数型スカラ PARM:基本文字型スカラ
現在時間の取得	GETTIM (HOUR, MINUTE, SECOND, SECOND1_100)	4	HOUR:2バイトの整数型スカラ MINUTE:2バイトの整数型スカラ SECOND:2バイトの整数型スカラ SECOND1_100:2バイトの整数型スカラ
ある時点からの経過時間の取得	GETTOD (G)	1	G:倍精度実数型スカラ
システム時間(グリニッジ標準時間)の取得	GMTIME (TIME, T)	2	TIME:基本整数型スカラ T:基本整数型配列
絶対値の文字への変換	IBTOD (I, J)	2	I:基本整数型スカラ J:基本整数型スカラ
現在日付の取得	IDATE (IA)	1	IA:基本整数型配列
実行時の診断メッセージの取得	IOSTAT_MSG (ERRNUM, MESSAGE)	2	ERRNUM:基本整数型スカラ MESSAGE:基本文字型スカラ
現在時刻の取得	ITIME (IA)	1	IA:基本整数型配列
領域の転送	IVALUE (I, J, K)	3	I:基本整数型スカラまたは基本整数型配列 J:基本整数型スカラ K:基本整数型スカラ
システム時間(ローカル時間)の取得	LTIME (TIME, T)	2	TIME:基本整数型スカラ T:基本整数型配列
標準エラー出力ファイルへの文字列出力	PERROR (STRING)	1	STRING:基本文字型スカラ
有効けた数を超えた部分に出力される文字の設定	PRECFILL (LETTER)	1	LETTER:長さ1の基本文字型スカラ
精度縮小数の変更	PRNSET (I)	1	I:基本整数型スカラ
促進メッセージの設定	PROMPT (STRING)	1	STRING:基本文字型スカラ
1次元配列のクイックソート	QSORT (ARRAY, NEL, WIDTH, COMPARE)	4	ARRAY:任意の型の1次元配列 NEL:基本整数型スカラ WIDTH:基本整数型スカラ COMPARE:2バイトの整数型を復帰値とする外部関数名

機能	サービスサブルーチン名 (引数キーワード) (注)	引数の数	引数
入力されたFortran記録の長さの獲得	REDLEN (I, RETCD)	2	I:基本整数型スカラ RETCD:基本整数型スカラ
ビットセット	SETBIT (POS, I, STATUS)	3	POS:基本整数型スカラ I:基本整数型スカラ STATUS:基本整数型スカラ
復帰コードの設定	SETRCD (I)	1	I:基本整数型スカラ
プロセスの中断	SLEEP (I)	1	I:基本整数型スカラ
センスライトの点灯、消灯	SLITE (I)	1	I:基本整数型スカラ
センスライトの状態確認	SLITET (I , J)	2	I:基本整数型スカラ J:基本整数型スカラ
午前0時からの通算1/100秒の取得	TIMER (IX)	1	IX:基本整数型スカラ

注)引数キーワードは、モジュール‘SERVICE_ROUTINES’を引用することによって明示的な引用仕様宣言をもつ場合にだけ指定できます。

表B.2 サービス関数

機能	サービス関数名 (引数キーワード)	引数の数	引数	関数結果の型
ファイルの存在およびアクセスモードの検査	ACCESS (FNAME , MODE)	2	FNAME:基本文字型スカラ MODE:基本文字型スカラ	基本整数型スカラ
サブルーチンの実行待機	ALARM (TIME, SUB)	2	TIME:基本整数型スカラ SUB:サブルーチン名	基本整数型スカラ
ビット検査	BIT (POS, I)	2	POS:基本整数型スカラ I:基本整数型スカラ	基本論理型スカラ
デフォルトディレクトリの変更	CHDIR (DIRNAME)	1	DIRNAME:基本文字型スカラ	基本整数型スカラ
パーミッションモードの変更	CHMOD (FNAME, MODE)	2	FNAME:基本文字型スカラ MODE:基本文字型スカラ	基本整数型スカラ
システム時間の取得	CTIME (TIME)	1	TIME:基本整数型スカラ	基本文字型スカラ
乱数の取得	DRAND (I)	1	I:基本整数型スカラ	倍精度実数型スカラ
CPU時間の取得	DTIME (TM)	1	TM:基本実数型配列	基本実数型スカラ
CPU時間の取得	ETIME (TM)	1	TM:基本実数型配列	基本実数型スカラ
ファイルから1文字読込み	FGETC (UNIT, CH)	2	UNIT:基本整数型スカラ CH:基本文字型スカラ	基本整数型スカラ
プロセスの複写	FORK	0	なし	基本整数型スカラ
ファイルへの1文字書出し	FPUTC (UNIT, CH)	2	UNIT:基本整数型スカラ CH:長さ1の基本文字型スカラ	基本整数型スカラ

機能	サービス関数名 (引数キーワード)	引数の数	引数	関数結果の型
ファイルのオフセットの再配置	FSEEK (UNIT, OFFSET, FROM)	3	UNIT:基本整数型スカラ OFFSET:基本整数型スカラ FROM:基本整数型スカラ	基本整数型スカラ
ファイルのオフセットの再配置	FSEEKO64 (UNIT, OFFSET, FROM)	3	UNIT:基本整数型スカラ OFFSET:8バイトの整数型スカラ FROM:基本整数型スカラ	基本整数型スカラ
ファイル状態の取得	FSTAT (IX, STATUS)	2	IX:基本整数型スカラ STATUS:基本整数型配列	基本整数型スカラ
ファイル状態の取得	FSTAT64 (IX, STATUS)	2	IX:基本整数型スカラ STATUS:8バイトの整数型配列	基本整数型スカラ
ファイルの現在のオフセット取得	FTELL (UNIT)	1	UNIT:基本整数型スカラ	基本整数型スカラ
ファイルの現在のオフセット取得	FTELLO64 (UNIT)	1	UNIT:基本整数型スカラ	8バイトの整数型スカラ
標準入力ファイルから1文字読込み	GETC (CH)	1	CH:基本文字型スカラ	基本整数型スカラ
作業ディレクトリ名の取得	GETCWD (CH)	1	CH:基本文字型スカラ	基本整数型スカラ
ファイル記述子の取得	GETFD (UNIT)	1	UNIT:基本整数型スカラ	基本整数型スカラ
グループIDの取得	GETGID	0	なし	基本整数型スカラ
プロセスIDの取得	GETPID	0	なし	基本整数型スカラ
ユーザIDの取得	GETUID	0	なし	基本整数型スカラ
ホスト名の取得	HOSTNM (NAME)	1	NAME:基本文字型スカラ	基本整数型スカラ
オプション文字列の個数取得	IARGC	0	なし	基本整数型スカラ
システムエラー番号の取得	IERRNO	0	なし	基本整数型スカラ
整数型の最大値の取得	INMAX	0	なし	基本整数型スカラ
整数型の乱数発生	IRAND (I)	1	I:基本整数型スカラ	基本整数型スカラ
端末かどうかの判定	ISATTY (UNIT)	1	UNIT:基本整数型スカラ	基本論理型スカラ
実行年月日の取得	JDATE	0	なし	長さ8の基本文字型スカラ
プロセスの中断	KILL (PID, SIG)	2	PID:基本整数型スカラ SIG:基本整数型スカラ	基本整数型スカラ
既存ファイルに関するリンクの作成	LINK (PATH1, PATH2)	2	PATH1:基本文字型スカラ PATH2:基本文字型スカラ	基本整数型スカラ
文字列末尾の空白を除いた文字長	LNBLNK (STRING)	1	STRING:基本文字型スカラ	基本整数型スカラ

機能	サービス関数名 (引数キーワード)	引数の数	引数	関数結果の型
基本整数型スカラへの変換	LONG (IX)	1	IX:2バイトの整数型スカラ	基本整数型スカラ
ファイル状態の取得	LSTAT (NAME, STATUS)	2	NAME:基本文字型スカラ STATUS:基本整数型配列	基本整数型スカラ
ファイル状態の取得	LSTAT64 (NAME, STATUS)	2	NAME:基本文字型スカラ STATUS:8バイトの整数型配列	基本整数型スカラ
領域の確保	MALLOC (SIZE)	1	SIZE: 8バイトの整数型スカラ	8バイトの整数型スカラ
オプション文字列の個数の取得	NARGS	0	なし	基本整数型スカラ
標準出力ファイルへの1文字書出し	PUTC (CH)	1	CH:長さ1の基本文字型スカラ	基本整数型スカラ
乱数	RAN (SEED)	1	SEED:基本整数型スカラ	実数型スカラ
乱数の発生	RAND (I)	1	I:基本整数型スカラ	実数型スカラ
ファイル名の変更	RENAME (OLD, NEW)	2	OLD:基本文字型スカラ NEW:基本文字型スカラ	基本整数型スカラ
文字列中の最も右にある他の文字列位置の取得	RINDEX (STRING, SUBSTRING)	2	STRING:基本文字型スカラ SUBSTRING:基本文字型スカラ	基本整数型スカラ
UTC通算秒の取得	RTC	0	なし	倍精度実数型スカラ
システム時間の経過秒数の取得	SECNDS (SEC)	1	SEC:実数型スカラ	実数型スカラ
ユーザ時間の取得	SECOND	0	なし	実数型スカラ
bourneシェルコマンドの発行	SH (COMMAND)	1	COMMAND:基本文字型スカラ	基本整数型スカラ
2バイトの整数型への変換	SHORT (IX)	1	IX:基本整数型スカラ	2バイトの整数型スカラ
シグナル発生時の動作指定	SIGNAL (I, FUNC, FLAG)	3	I:基本整数型スカラ FUNC:関数名 FLAG:基本整数型スカラ	基本整数型スカラ
ファイル状態の取得	STAT (NAME, STATUS)	2	NAME:基本文字型スカラ STATUS:基本整数型配列	基本整数型スカラ
ファイル状態の取得	STAT64 (NAME, STATUS)	2	NAME:基本文字型スカラ STATUS:8バイトの整数型配列	基本整数型スカラ
シンボリックリンクの作成	SYMLNK (PATH1, PATH2)	2	PATH1:基本文字型スカラ PATH2:基本文字型スカラ	基本整数型スカラ
コマンドの実行	SYSTEM (CH)	1	CH:基本文字型スカラ	基本整数型スカラ
00:00:00 GMT(1970年1月1日)からの経過秒数の取得	TIME	0	なし	基本整数型スカラ
経過時間の取得	TIMEF	0	なし	倍精度実数型スカラ
端末装置のパス名の取得	TTYNAM (UNIT)	1	UNIT:基本整数型スカラ	基本文字型スカラ

機能	サービス関数名 (引数キーワード)	引数の数	引数	関数結果の型
リンク総数の削減	UNLINK (CH)	1	CH : 基本文字型スカラ	基本整数型スカラ
現プロセスの中断	WAIT (STATUS)	1	STATUS : 基本整数型スカラ	基本整数型スカラ

注) 引数キーワードは、モジュール‘SERVICE_ROUTINES’を引用することによって明示的な引用仕様宣言をもつ場合にだけ指定できます。

付録C　拡張仕様一覧

本処理系がサポートしているFortran 2008 規格からの拡張仕様を下表に示します。

表C.1 Fortran 2008 規格からの拡張仕様一覧

[Fortran用の文字]
・ 英字としての通貨記号($)
・ 特殊文字としてのバックスラッシュ(¥)
[原始プログラムの記述]
・ 固定形式の原始プログラム
継続行の無制限
・ 自由形式の原始プログラム
空白が区切りでない
継続行の無制限
[COARRAY]
・ 共配列は、成分にALLOCATABLE属性またはPOINTER属性を指定できません。
・ 共配列は、パラメタ付き派生型、またはパラメタ付き派生型を成分としてもつ型であってはなりません。
[文の分類]
・ 実行文
DO UNTIL 文
・ 非実行文
STRUCTURE 文
END STRUCTURE 文
UNION 文
END UNION 文
MAP 文
END MAP 文
RECORD 文
POINTER 文(CRAY 仕様)
AUTOMATIC 文
STATIC 文
BYTE 文
CHANGEENTRY 文
[属性]
・ AUTOMATIC属性
・ CHANGEENTRY属性
・ STATIC属性
[データの型]
・ STRUCTURE 文による派生型の定義
STRUCTURE 文、END STRUCTURE 文

・ UNION 文による共用体の宣言

UNION 文、END UNION 文、MAP 文、END MAP 文

[定数]

・ ホレリス定数

・ 8進定数

'*digits*'O、"*digits*"O

・ 16進定数

X'*digits*'、X"*digits*"、'*digits*'X、"*digits*"X

・ 2進、8進、16進定数は以下にも指定可能

PARAMETER 文

型宣言文の初期値指定

[変数]

・ 構造体成分(.)

[データの宣言および指定]

・ 型宣言文

文字型以外の型宣言文における長さ指定

初期値設定(/.../)

・ RECORD 文

・ BYTE 文

[派生型パラメタ]

・ 長さ型パラメタをもつ派生型の仮引数は、INTENT(OUT)属性を指定できません。

・ 長さ型パラメタをもつ派生型の定義、実体宣言、または成分宣言をBLOCK構文の有効域で指定できません。

・ ゼロでない次元数の部分参照より右にある構造体成分は、配列上下限または文字長の指定で長さ型パラメタを指定できません。

・ 長さ型パラメタが指定された末端成分をもつ派生型の実体または式は、入出力項目またはNAMELIST文に指定することはできません。

・ 引継ぎ長さ型パラメタをもつ仮引数に対応する実引数として、結果がパラメタ付き派生型である組込み手続は指定できません。

・ ALLOCATE文においてMOLD指定子を指定する場合、割付け実体の型はパラメタ付き派生型であってはなりません。

・ パラメタ付き派生型となる式は、選択子に指定できません。

・ 長さ型パラメタをもつ派生型には、成分初期値を指定できません。

・ パラメタ付き派生型、またはパラメタ付き派生型を成分としてもつ型は、共配列の型指定子に指定できません。

[構造体構成子]

・ 派生型指定子の型パラメタ指定は、その派生型がパラメタをもつときは省略できません。

[定数式]

・ 定数式には、演算結果の型が半精度実数型または半精度複素数型のべき乗(**)演算子は指定できません。

・ 定数式では、組込み関数MAXVAL、MINVAL、NORM2、またはSPACINGの引数は、半精度実数型であってはなりません。

・ 定数式では、組込み関数ABS、DOT_PRODUCT、またはMATMULの引数は、半精度複素数型であってはなりません。

・ 定数式では、組込み関数CSHIFT、EOSHIFT、PACK、PRODUCT、SPREAD、SUM、TRANSFER、またはUNPACKの引数は、半精度実数型または半精度複素数型であってはなりません。

[仮データ実体]

・ 共添字付き実体の実引数は、ポインタ末端成分または割付け末端成分をもってはなりません。

・ 実引数が共添字付き実体なら、仮引数は、INTENT(IN) 属性をもたなければなりません。

・ 仮引数が、CONTIGUOUS 属性をもつ配列共配列、または形状引継ぎの配列共配列でない場合、対応する実引数は単純 CONTIGUOUS でなければなりません。

[IMPLICIT 文]

・ IMPLICIT UNDEFINED

・ 文字型以外の長さ指定

・ 型代表文字としての通貨記号

[POINTER 文（CRAY 仕様）]

・ POINTER 文（CRAY 仕様）

[書式なし入出力文]

・ NUM 指定子

[OPEN 文]

・ ACTION 指定子にBOTH が指定可能

・ FORM 指定子にBINARY が指定可能

・ BLOCKSIZE 指定子

・ TOTALREC 指定子

・ CONVERT 指定子

・ STATUS 指定子のSHR

・ ACCESS 指定子のAPPEND

[INQUIRE 文]

・ BINARY 指定子

・ BLOCKSIZE 指定子

・ CONVERT 指定子

・ FLEN 指定子

[CLOSE 文]

・ STATUS 指定子のFSYNC

[ALLOCATE文]

・ MOLD 指定子の初期値指定式は、型パラメタをもつ派生型であってはなりません。

[NAMELIST文]

・ 変数群要素は、末端成分に長さ型パラメタをもつ派生型であってはなりません。

[書式]

・ 編集記述子

G*w*の形式の編集記述子

$ 編集記述子

Q 形編集記述子

Q 残余文字編集記述子

R 形編集記述子

¥ 編集記述子

D,E,F,G,I,L,B,O およびZ 形編集記述子の*w,d* および*e* の省略

- 変数群書式

  変数群記録 &*name*...&end

[プログラム単位]

- 関数副プログラム

  文字型以外の型に対する長さ指定

[組込み手続]

- 総称組込み関数

  DCMPLX
  CBRT、EXP2、EXP10、LOG2、COTAN、LGAMMA
  SIND、COSD、TAND、ASIND、ACOSD、ATAND、ATAN2D
  SINQ、COSQ、TANQ、COTAND、COTANQ、ASINQ、ACOSQ、ATANQ、ATAN2Q
  AND、OR、XOR、LSHIFT、RSHIFT、LRSHFT
  LOC
  JFIX、INT1、INT2、INT4、I2NINT、IBCHNG、ISHA、ISHC、ISHL、IZEXT、JZEXT、
  IZEXT2、JZEXT2、JZEXT4、VAL
  SIZEOF

- 組込み関数の個別名

  HFIX
  DFLOAT、DBLE、DREAL、CMPLX、DCMPLX、CDABS、MAX、MIN、IMAG、
  DIMAG、DCONJG
  CDSQRT、CDEXP、CDLOG、CDSIN、SIND、DSIND、COSD、DCOSD、TAND、
  DTAND、ASIND、DASIND、ACOSD、DACOSD、ATAND、DATAND、ATAN2D、
  DATAN2D
  CBRT、DCBRT、EXP2、DEXP2、EXP10、DEXP10、LOG、LOG10、LOG2、ALOG2、
  DLOG2、SINQ、DSINQ、COSQ、DCOSQ、TANQ、DTANQ、COTAN、DCOTAN、
  COTAND、DCOTAND、COTANQ、DCOTANQ、ASINQ、DASINQ、ACOSQ、DACOSQ、
  ATANQ、DATANQ、ATAN2Q、DATAN2Q、ERF、DERF、ERFC、DERFC、GAMMA、
  DGAMMA、LGAMMA、ALGAMA、DLGAMA
  NOT、IAND、IOR、IEOR、ISHFT、IBSET、IBCLR、BTEST、AND、OR、XOR
  LSHIFT、RSHIFT、LRSHFT
  AIMAX0、AJMAX0、I2MAX0、IMAX0、JMAX0、IMAX1、JMAX1、AIMIN0、AJIMIN0、
  I2MIN0、IMIN0、JMIN0、IMIN1、JMIN1、FLOATI、FLOATJ、DFLOTI、DFLOTJ、IIABS、
  JIABS、I2ABS、IIDIM、JIDIM、I2DIM、IIFIX、JIFIX、IINT、JINT、ININT、JNINT、
  IIDNNT、JIDNNT、IIDINT、JIDINT、IMOD、JMOD、I2MOD、IISIGN、JISIGN、I2SIGN、
  BITEST、BJTEST、IIBCLR、JIBCLR、IIBITS、JIBITS、IIBSET、JIBSET、IIAND、
  JIAND、IIEOR、JIEOR、IIOR、JIOR、INOT、JNOT、IISHFT、JISHFT、IISHFTC、
  JISHFTC

[構造構文]

- DO 構文のDO 変数およびそのスカラ整数式に基本実数型および倍精度実数型が指定可能

[BLOCK構文]

- BLOCK 構文には、以下を含めてはなりません。
  - 長さ型パラメタをもつ派生型（“1.5.11.2 派生型パラメタ”参照）の定義、実体宣言、または成分宣言
  - 共配列の宣言（“1.17 共配列”参照）
  - USE 文（“2.496 USE文”参照）

付録D　Fortran 2018サポート仕様一覧

本処理系では、Fortran 2018規格の仕様の一部を実現しています。本処理系が実現しているFortran 2018規格の仕様を下表に示します。

表D.1 実現済みFortran 2018サポート仕様一覧

[データの宣言]
・ 型引継ぎ
・ 次元引継ぎ
[整数型指定子]
・ DATA文
・ 配列構成子
[純粋手続]
・ 純粋手続は、多相的な言語要素の解放を引き起こす可能性のある文を含んではなりません。
[組込み手続]
・ CO_MAX、CO_MIN、CO_SUM、RANK

付録E　用語集

MODULE PROCEDURE 副プログラム

MODULE PROCEDURE副プログラム文により定義する手続です。分離引用仕様によって、その手続の宣言部が宣言され、手続の特性や仮引数名の指定を省略できます。

TKR 適合

型が適合し、種別型パラメタが同一であり、更に次元数が一致していることです。

後始末

実体を解体抹消する直前に利用者定義後始末サブルーチンが呼び出されることです。

後始末サブルーチン

後始末の間に自動的に呼び出されるサブルーチンです。

暗黙形状配列

宣言時の定数式と同じ形状になる名前付き定数の配列です。

暗黙的引用仕様

ある手続が自分自身以外の有効域内で引用されているとき、その手続が引用仕様宣言をもたない外部手続、引用仕様宣言をもたない仮手続、または文関数であれば、その手続は、暗黙的引用仕様をもつといいます。

暗黙的初期値指定

型定義中で初期値指定を指定すると、その型の実体は、自動的に初期化されます。非ポインタ成分は暗黙的に値を与えて初期化することができ、ポインタ成分は初期状態を暗黙的に空状態にすることができます。暗黙的初期値指定は、組込み型の実体にはありません。

引用

実行中のその時点での値を要求する形でデータ実体の名前を書くこと、その時点での手続の実行を要求する形でその手続の名前、その演算子記号もしくは利用者定義代入文を書くこと、またはUSE 文にモジュール名を書くことです。変数を確定する動作および手続の名前を実引数として書くことは、引用とはみなしません。

引用仕様

手続引用仕様の項を参照してください。

引用仕様宣言

INTERFACE 文から対応するEND INTERFACE 文までの文の列です。

引用仕様本体

引用仕様宣言内の、FUNCTION 文またはSUBROUTINE文から対応するEND文までの文の列です。

上書き

明示的初期値指定または暗黙的初期値指定が、暗黙的初期値指定に優先するとき、優先する初期値指定だけが指定されたかのように扱われます。手続が直接拡張型に束縛されているとき、それが、親型から継承されるはずの手続に優先します。

演算

1つまたは2つの演算対象に作用する計算処理です。

演算子

演算を指定する構文素です。

演算対象

演算子の左または右にある式です。

大きさ

配列の、要素の総数です。

大きさ引継ぎ配列

結合される実引数から大きさを引き継ぐ仮配列です。右端の次元の上限を星印‘*’で指定します。

親型

直接拡張型の派生元の拡張可能型です。

親子結合

内部副プログラム、モジュール副プログラムまたは派生型定義が、その親プログラムの言語要素を参照するときの作用、および、引用仕様本体において、その引用仕様本体のIMPORT 文によって親有効域の名前付き要素を参照するときの作用です。

親成分

直接拡張型の継承された部分に対応する、その型の要素の成分です。

親プログラム

内部副プログラムを包む主プログラムまたは副プログラムを、その内部副プログラムの親プログラムと呼びます。モジュール副プログラムを包むモジュールを、そのモジュール副プログラムの親プログラムと呼びます。

親有効域

他の有効域を、直接に取り囲んでいる有効域です。

外部結合

C 言語の言語要素がプログラムに対して大域的であることを記述する特性です。

外部装置

外部ファイルを参照するために用いる機構です。外部装置は、非負整数によって識別されます。

外部手続

外部副プログラムまたはFortran 以外の手段によって定義される手続です。

外部ファイル

プログラムの外部の媒体にある記録の列です。

外部副プログラム

副プログラムであって、主プログラムにもモジュールにも他の副プログラムにも包まれていないものです。

拡張型

直接拡張型は、それ自体およびその親型が拡張型となっている型の拡張型です。

拡張可能型

EXTENDS 属性を用いて新しい型を派生しうる型。BIND 属性をもたず、連続型でない型です。

確定

データ実体について、有効な値をもっているまたは与えられた状態です。

確定可能

変数は、代入文の左辺にその名前または特定子を書いてその値を変更してよい場合、確定可能であるといいます。割り付けられていない割付け変数は、確定可能でないデータ実体の例です。確定可能でない部分実体の例としては、c を定数の配列とし i を整変数としたときのc(i) があります。

型

データ型のことです。

型が適合する

すべてのデータ要素は、同じ型の他のデータ要素と型が適合します。無制限多相的データ要素は、すべてのデータ要素と型が適合します。無制限多相的でない多相的データ要素は、実行時の型がその多相的データ要素の宣言時の型の拡張型であるデータ要素と型が適合します。

型宣言文

INTEGER 文、REAL 文、DOUBLE PRECISION 文、COMPLEX 文、CHARACTER 文、LOGICAL 文、TYPE (*type-name*) 文、またはBYTE 文です。

型束縛手続

型定義中の手続束縛です。その手続は、その実行時の型をもつ何らかの実体を通じて束縛名によってか、利用者定義演算としてか、利用者定義代入によってか、または後始末処理の一部として、引用することができます。

型パラメタ

組込みデータ型のパラメタです。型パラメタには、種別(KIND)と文字長(LEN)とがあります。派生型の型パラメタは、派生型定義の中で定義します。

型引継ぎ

型指定子TYPE(*)で宣言します。無制限多相的の仮引数です。

合致

プログラムは、それが規格に定められた形と関係だけを用いて書かれ、かつ規格に従った解釈をもつならば、規格に合致しています。プログラム単位は、それをあるプログラムに含めてそのプログラムが規格合致するようにできるならば、規格に合致しています。

共形状明示配列

共配列の共次元数および共上下限を宣言します。割付けでない共配列です。

共次元数

共次元の数です。

共添字

像選択子で指定するスカラ整数式です。共添字の値は、共次元の共上下限の範囲です。

共添字付き実体

像選択子の指定をもつ実体です。

共配列

COARRAY仕様において、いかなる像からも、直接参照または定義できるデータ要素です。

共配列末端成分

共配列をもつ末端成分です。

虚実部特定子

特定子%REまたは特定子%IMです。特定子の実部、特定子の虚部です。

仮データ実体

データ実体である仮引数です。

仮手続

手続として宣言または引用されている仮引数です。

仮配列

配列である仮引数です。

仮引数

FUNCTION 文、SUBROUTINE 文、ENTRY 文、または文関数定義文の中で、手続名の後の括弧付き並びに名前が書いてある言語要素です。

仮ポインタ

ポインタである仮引数です。

関数

式中で呼び出し、値を計算し、その値をその式の評価に用いる手続です。

関数結果

関数の値を返すデータ実体です。

関数副プログラム

引用仕様宣言内にないFUNCTION 文から対応するEND 文までの文の列です。

キーワード

文の構文の一部を成す単語または並び中の項目を識別するのに用いる名前です。

記憶単位

文字記憶単位、数値記憶単位またはファイル記憶単位です。

記憶列

連続する記憶単位の列です。

記憶列結合

一方の記憶列のいずれかの記憶単位が、他方の記憶列のいずれかの記憶単位と同一であるような、2つの記憶列の間の関係です。

刻み幅

添字三つ組に指定する増分値です。

行

Fortran の文、注釈、またはINCLUDE 行からなる、0 ～ 255 個の文字の列です。

共通ブロック

プログラム中のどの有効域からも参照できる、物理的な記憶場所の区画です。

局所変数

特定の有効域に対して局所的な変数です。参照結合または親子結合によって取り込まれたのでもなく、仮引数でもなく、共通ブロック中の変数でもない変数です。

局所要素

1つの有効域を有効範囲とする構文素によって識別される言語要素です。

記録

ファイル中で、ひとまとまりとして扱われる値または文字の列です。

空状態

空状態のポインタは、どの指示先とも結合していません。ポインタは、DEALLOCATE 文またはNULLIFY 文の実行後、または空状態のポインタとのポインタ結合後、空状態になります。

組込み

規格で定義されていて、特に定義したり宣言したりせずにどの有効域中でも用いることのできるデータ型、演算、代入、手続、およびモジュールに関する修飾語です。

クラス

拡張型のことです。

継承

親から引き継ぎ、獲得すること。直接拡張型の中で明示的に宣言せずに親型から自動的に獲得される型パラメタ、成分、または手続束縛は、継承されるといいます。

形状

配列について、次元数および各寸法です。形状は、各次元の寸法を要素とする1次元の配列で表現することができます。

継承結合

直接拡張型における、継承された親成分の結合です。

形状適合

2つの配列は、同じ形状をもつとき、形状適合しているといいます。スカラは、いかなる配列とも形状適合します。

形状引継ぎ配列

結合される実引数から形状を引き継ぐ、ポインタでない仮配列です。

形状明示配列

明示的な上下限を伴って宣言された名前付き配列です。

結果変数

関数の値を返す変数です。

結合

ポインタ結合、記憶列結合、または継承結合です。

結合名

SELECT TYPE 構文またはASSOCIATE 構文の中で、選択子が結合する構文内要素の名前です。

言語要素

プログラム単位、手続、抽象引用仕様、演算子、総称引用仕様、共通ブロック、外部装置、文関数、型、データ要素、文番号、構造構文、および変数群を指すのに用いる用語です。

構造構文

ASSOCIATE 文、SELECT CASE 文、DO 文、構造FORALL 文、IF 文、SELECT TYPE 文、または構造WHERE 文から対応する端末文までの文の列です。

構造体

派生型のスカラデータ実体です。

構造体構成子

派生型の値を構成するための構文上の機構です。

構造体成分

派生型の実体の一部分です。

構文結合

ASSOCIATE 構文またはSELECT TYPE 構文における選択子と結合名との間の結合です。

構文素

指定された解釈をもつ、1文字以上の文字の列です。

構文内要素

1つの構造構文を有効範囲とする構文素によって識別される言語要素です。

サブモジュール

モジュール、または他のサブモジュールを拡張するプログラム単位です。

サブルーチン

CALL 文または利用者定義代入文によって呼び出す手続です。

サブルーチン副プログラム

引用仕様宣言内にないSUBROUTINE 文から対応するEND 文までの文の列です。

参照結合

USE 文によって指定される、異なる有効域間での名前の結合です。

式

演算対象、演算子、および括弧の列です。変数、定数、関数引用、または計算の表現の場合もあります。

次元数

配列の次元の個数です。スカラについてはゼロとします。

次元引継ぎ

有効実引数の次元数、形状、大きさを引継ぐ仮引数です。

指示先

TARGET 文またはTARGET 属性をもつ型宣言文で指定された名前付きデータ実体、ポインタに対するALLOCATE 文によって生成されたデータ実体、またはそれらの実体の部分実体です。

指示状態

ポインタ代入の後またはALLOCATE 文の有効な実行の後の、ポインタと指示先との関係です。

事前接続

プログラムの実行の始めに外部ファイルに接続しているという、装置の性質です。そのような装置は、その装置に対するOPEN 文を実行しなくても入出力文に指定することができます。

実行構文

ASSOCIATE 構文、CASE 構文、DO 構文、FORALL 構文、IF 構文、SELECT TYPE 構文、WHERE 構文、または単純実行文です。

実行時の型

データ要素が実行時にもつ型のことです。多相的でないデータ要素の実行時の型は、宣言時の型と同じです。

実行文

1つ以上の計算処理動作を行うまたは制御する命令です。

実在する

ある副プログラムの分身において、仮引数は、それが実引数に結合されていて、かつその実引数が、呼出し側の副プログラム内で実在する仮引数であるときまたは呼出し側の副プログラムの仮引数でないとき、その副プログラムの分身内で実在するといいます。

実体

データ実体のことです。

実引数

手続引用に指定する式、変数、手続、または選択戻り指定子です。

自動割付け変数

副プログラムの局所要素であって、仮引数でなく、その文字長またはその配列の上下限が定数でない変数です。

授受特性

手続でもポインタでもない仮引数の、属性の1つです。授受特性は、その仮引数が、データを手続の中へ転送するために用いられるのか、手続の外へ転送するために用いられるのか、その両方のために用いられるのかを示します。

主プログラム

モジュールでも外部副プログラムでも初期値設定プログラム単位でもないプログラム単位です。Fortran 以外の手段で定義してもかまいません。

種別型パラメタ

1つの組込み型に用意されている種別の識別番号を値とするパラメタ、またはKIND 属性をもつと宣言された派生型パラメタです。

純粋手続

純粋組込み手続、純粋副プログラムによって定義された手続、または純粋関数だけを引用した文関数のことです。

上下限

名前付き配列について、配列要素の添字の値が収まっていなければならない限界です。

小数点記号

ファイル中の実数の10進表現において、整数部と小数部とを分ける文字です。暗黙の小数点記号は'.'(ピリオド)です。現在の小数点記号は、現在の小数点編集モードによって決まります。

初期指示先手続

手続ポインタの初期設定による指示先です。

初期指示先データ

データポインタの初期設定による指示先です。

初期値設定プログラム単位

名前付き共通ブロック内のデータ実体に初期値を与えるためのプログラム単位です。

処理系

計算機システムと、プログラムをその計算機システムで使用できるように変形する機構とを組み合わせたものです。

処理系依存

規格で完全には規定していない機能の呼称です。そのような機能に対しては、処理系が手段または意味を定めます。

数値型

整数型、実数型、または複素数型です。

数値記憶単位

ポインタでなく型が基本実数型、基本整数型、または基本論理型であるスカラを、保持するための記憶場所の単位です。

スカラ

配列でない単一のデータ、または配列であるという性質をもたないことです。

寸法

配列の1つの次元の大きさです。

制御配列

WHERE 文またはWHERE 構文において、各WHERE 代入文でどの配列要素が確定されるかを論理値によって決定する論理型配列のことです。

制限式

長さ型パラメタ、配列上下限の指定に使えるように制限を課したスカラ整数式です。

成分

派生型の構成要素です。

成分順序

組込み書式付き入出力および構造体構成子に用いられる派生型の成分の順序付けです。

接続されている

外部装置について、外部ファイルを参照していること、または外部ファイルについて、それを参照している外部装置があることです。

宣言式

長さ型パラメタ、配列上下限の指定に使えるように制限を課したスカラ整数式です。

宣言時の型

データ要素が宣言時にもつ型のことです。多相的データ要素の場合、実行時にもつ型(実行時の型)と異なることができます。

全体配列

名前付き配列のことです。

選択子

以下のものを特定する構文上の機構です。

- データ実体の部分。部分列、配列要素、部分配列、または構造体成分を特定することができる。
- あるCASE ブロックを実行する場合値の組です。
- SELECT TYPE 構文のどの分岐が実行されるかを型によって決める実体です。
- ASSOCIATE 構文において結合名に結合される実体です。

相互利用可能

Fortran の言語要素について、それと同等なC 言語の言語要素が定義しうるということを保証する性質です。

像選択子

共添字付き実体の像番号を決定します。

像番号

像を識別する1から像数の範囲の整数値です。

総称引用仕様

総称手続束縛または総称引用仕様宣言によって指定される引用仕様です。

総称引用仕様宣言

総称指定をもつ引用仕様宣言のことです。

総称識別子

INTERFACE 文中に書いて、その引用仕様宣言内のすべての手続に関連付けられている構文素、または、GENERIC 文中に書いて、そのすべての個別型束縛手続に関連付けられている構文素です。

添字

配列要素選択子中のスカラ整数式の並びの1項目です。

添字三つ組

部分配列選択子中の並びの項目であって、コロンを含むものです。添字三つ組は、整数値の規則的な列を指定します。

属する

EXIT文またはCYCLE文にDO構文名が書いてあるとき、その文は、その名前をもつDO構文に属するといいます。DO 構文名が書いてないとき、その文は、最も内側を包んでいるDO 構文に属するといいます。

属性

型宣言文で指定できる、データ実体の性質です。

束縛ラベル

C プログラムとの結合において変数、共通ブロック、サブルーチン、または関数を一意的に識別する基本文字型の値です。束縛ラベルについては、“Fortran 使用手引書”も参照してください。

大域要素

1つのプログラムを有効範囲とする構文素によって識別される言語要素です。プログラム単位、共通ブロック、または外部手続の場合があります。

大小順序

個々の種別型パラメタに関する、すべての異なる文字の順序付けです。

代入文

“変数 = 式”の形の文です。

多相的

プログラムの実行中に異なる型をもつことができることです。キーワードCLASS を指定して宣言した実体は、多相的になります。

単純実行文

計算処理動作を指定するまたは制御する単一の文です。

遅延束縛

DEFERRED 属性をもつ束縛です。遅延束縛は、抽象型定義の中にだけ書くことができます。

抽象型

ABSTRACT 属性をもつ型です。多相的でない実体は、抽象型として宣言してはなりません。多相的な実体は、動的な抽象型をもつように構成したり割り付けたりしてはなりません。

直接拡張型

他の型の拡張である拡張可能型です。EXTENDS 属性をもって宣言された型です。

定数

プログラムを実行している間、値を変えることのできないデータ実体です。定数は、名前付き定数または定数表現の場合があります。

定数式

翻訳時に評価可能な式です。

定数表現

名前のない定数です。

データ

任意の型の任意の値の集合です。

データ型

値の集合、それらの値の表現方法およびそれらの値を解釈し操作する演算の集合によって特徴付けられる、データの名前付き分類区分です。組込み型の場合、データ値の集合は、型パラメタの値に依存します。

データ実体

定数、変数、または部分定数(定数の部分実体)のいずれかであるデータ要素です。

データ要素

データ実体、式の評価結果、または関数引用の実行結果(関数結果と呼びます)です。データ要素は、データ型(組込み型または派生型)をもち、不定である変数の場合を除いて、データ値をもちます。すべてのデータ要素は、次元数をもち、したがってスカラまたは配列となります。

データポインタ初期化適合

ポインタ変数または成分が、ポインタと結合先で同じ型、同じ次元数、ポインタの無指定でない型パラメタが対応する結合先と同じ値および、ポインタがCONTIGUOUS属性をもつなら結合先がCONTIGUOUSであるなら、データポインタ初期化適合です。

手続

プログラムの実行中に呼び出すことのできる計算処理です。手続は、関数またはサブルーチンの場合があります。手続は、組込み手続、外部手続、モジュール手続、内部手続、仮手続、または文関数の場合があります。1つの副プログラムは、その中にENTRY 文を含んでいれば、2つ以上の手続を定義することができます。

手続引用仕様

手続の特性、手続名、仮引数名、および総称識別子による引用仕様のことです。

手続特定子

手続の特定子です。

問合せ関数

組込み関数または組込みモジュール関数の結果が、1つ以上の引数の値ではなく、性質に依存してきまるものです。

当該実体仮引数

型束縛手続の仮引数または手続ポインタ成分の仮引数であり、その手続を呼び出すのに使った実体と結合されるものです。

特性

手続の特性、仮引数の特性、仮データ実体の特性、仮手続の特性、関数結果の特性があります。

内部手続

内部副プログラムによって定義される手続です。

内部ファイル

データを内部記憶から内部記憶へ転送し変換するために用いる文字変数です。

内部副プログラム

副プログラムであって、主プログラムまたは他の副プログラムに包まれているものです。

名前

英字または‘$’の後ろに239文字以内の英数字下線(英字、数字、および下線)および‘$’を書いた構文素です。

名前付き

名前をもっていることです。すなわち、“名前付き変数”のような用法における“名前付き”は、その変数名が添字並び、部分列指定などによって修飾されていないことを意味します。

名前付き定数

名前をもつ定数です。

廃止事項

以前のFortran 規格の機能のうち、冗長になっていてほとんど使用されなくなっているとみなされるものです。

廃止予定事項

冗長にはなっているがまだ頻繁に使用されているとみなされる機能です。

配列

同じ型、同じ型パラメタをもつスカラデータの集合であって、要素が四角い形に配置されたものです。配列は、名前付き配列、部分配列、構造体成分、関数値、または式の場合があります。配列の次元数は、1以上です。

配列値

配列であるという性質をもつことです。

配列ポインタ

配列へのポインタです。

配列要素

名前付き配列または構造体成分である配列を構成する、個々のスカラデータです。

派生型

成分をもつデータの型です。それぞれの成分は、組込み型または別の派生型とします。

引数

実引数または仮引数です。

引数キーワード

仮引数の名前です。手続が明示的引用仕様をもつ場合に、手続引用中で等号‘=’を後ろに付けて用いることができます。

引数結合

手続引用を実行している間の、実引数と仮引数との関係です。

非実行文

計算処理動作が行われるときのプログラム環境を指定するために用いる文です。

非数

IEEE 算術の非数の値です。定義されない値または不正な演算で生成されます。

ファイル

内部ファイルまたは外部ファイルです。

ファイル記憶単位

書式なしファイルまたは流れファイルの記憶単位です。

副プログラム

関数副プログラムまたはサブルーチン副プログラムです。

副プログラムの分身

ある副プログラムで定義されている手続を、呼び出す時に生成されるその副プログラムの複製です。

符号なし

C 言語の数値型の属性です。その型が非負の値だけから構成されることを示すものです。Fortranには同様のものはありません。

不定

データ実体について、確定した値をもっていない状態です。

部分実体

名前付きデータ実体の一部分であって、他の部分とは独立に引用したり確定したりできる部分です。部分実体は、配列要素、部分配列、構造体成分、部分列、複素数型実体の実部、または複素数型実体の虚部です。

部分配列

配列の部分実体であって、構造体成分でないものです。

部分配列添字

部分配列選択子中の添字、ベクトル添字、または添字三つ組です。

部分列

スカラ文字列の連続的な一部分です。

プログラム

ちょうど1つの主プログラムを含むプログラム単位の一組です。

プログラム単位

プログラムの基本的な構成要素です。文、注釈、およびINCLUDE 行の列です。プログラム単位は、主プログラム、モジュール、外部副プログラム、または初期値設定プログラム単位です。

ブロック

ある実行構文に埋め込まれ、その構文特有の文で両端を区切られた実行構文の列です。列全体を1つのものとして扱います。

文

構文素の列です。文は、通常は単一の行で構成するが、ある行から次の行へ文を継続することができ、セミコロンを用いて1つの行の中に複数の文を書くことができます。

文関数

代入文と同様の形をした単一の文によって定義される手続です。

文内要素

1つの文または1つの文の一部分を有効範囲とする構文素によって識別される言語要素です。

文番号

文の左に先行して置き、その文を参照するために用いることのできる、5文字以内の数字からなる構文素です。

分離引用仕様

分離モジュール手続の引用仕様を指定します。

分離モジュール手続

モジュール副プログラム部において手続接頭辞MODULEを指定した関数、手続接頭辞MODULEを指定したサブルーチン、またはMODULE PROCEDURE副プログラム文で定義した手続です。分離モジュール手続は、分離引用仕様があるモジュールまたはサブモジュール、またはその子孫サブモジュールで実装することができます。

ベクトル添字

次元数1の整数式である部分配列添字です。

変形関数

要素別処理関数でも問合せ関数でもない組込み関数です。多くの変形関数は、配列の引数および配列の結果をもち、結果の配列の要素の値が引数の複数の要素の値に依存して決まります。

変数

プログラムを実行している間に、値を確定したり再確定したりできるデータ実体です。変数は、名前付きデータ実体、配列要素、部分配列、構造体成分、または部分列です。

ポインタ

POINTER 属性をもつ変数です。ポインタは、指示先に対して指示状態でなければ、引用も確定もしてはなりません。指示先が配列である場合、指示状態でなければ形状をもたないが、次元数はもちます。

ポインタ結合

ポインタを指示先に対して指示状態にする作用です。

ポインタ代入

ポインタ代入文の実行、またはポインタを部分実体としてもつ派生型のデータ実体への代入文の実行による、ポインタと指示先とのポインタ結合です。

ポインタ代入文

“ポインタ実体 => 指示先”の形の文です。

ポインタ末端成分

POINTER属性をもつ末端成分です。

末端成分

派生型または構造体について、以下のいずれかです。

- 組込み型である成分
- ALLOCATABLE 属性またはPOINTER 属性をもつ成分
- 派生型であってPOINTER 属性およびALLOCATABLE 属性をもたない成分の末端成分

丸めモード

厳密に表現できない演算の結果を選択する方法です。IEEE 算術には、直近丸め、(ゼロへの)切捨て、(∞ への)切上げ、および(- ∞への)切下げの4つのモードがあります。入出力には更に、COMPATIBLE およびPROCESSOR_DEFINED の2つのモードがあります。

無指定型パラメタ

実体の宣言時には値を指定せず、その実体の割付け時またはポインタ代入時に値を指定する長さ型パラメタです。

無制限多相的

型指定子CLASS(*)またはTYPE(*)で宣言します。型をもつ実体として宣言されません。

明示的引用仕様

ある有効域内で引用されている手続について、内部手続、モジュール手続、組込み手続、引用仕様宣言をもつ外部手続、自分自身の有効域内で引用されている再帰手続、または引用仕様宣言をもつ仮手続のいずれかであるという性質です。

明示的初期値指定

明示的初期値指定は、型宣言文中またはDATA 文中での、組込み型または派生型の実体に対して指定できます。暗黙的初期値指定をもつ派生型の実体は、DATA 文中に書かれなくてもかまいません。

文字

英字、数字、およびその他の記号です。

文字記憶単位

ポインタでなく型が基本文字型であり文字長が1であるスカラを、保持するための記憶場所の単位です。

文字長

文字列の長さのことです。

文字長パラメタ

文字型のデータ要素の、文字の個数を指定する型パラメタです。

モジュール

プログラム単位であって、他のプログラム単位から参照される定義を、内部にもっているまたは参照しているものです。

モジュール手続

モジュール副プログラムによって定義される手続です。

モジュール副プログラム

モジュールに包まれていて内部副プログラムではない副プログラムです。

文字列

左から右へ順に1、2、3、... と番号付けられた文字の列です。

文字列の長さ

文字列中の文字の個数です。

有効域

有効域には以下に示すものがあります。

- 派生型定義の部分。
- 引用仕様本体から、それに包まれている派生型定義および引用仕様本体をすべて除いた部分。
- プログラム単位または副プログラムから、それに包まれている派生型定義、引用仕様本体、および副プログラムをすべて除いた部分。

有効項目

入出力並びを展開して得られるスカラ実体のことです。

有効範囲

ある構文素がある単一の解釈をもつようなプログラムの一部分です。有効範囲は、1つのプログラム、1つの有効域、1つの構造構文、1つの文、または1つの文の一部分の場合があります。

要素別処理

1つの配列の要素ごとに、または一組の形状適合する配列およびスカラの対応要素ごとに、独立に適用される演算、手続または代入のことです。

呼び出す

サブルーチンを、CALL 文または利用者定義代入文によって呼ぶこと、式の評価中に関数を、その名前または演算子を用いた引用によって呼ぶこと、または後始末によって，後始末サブルーチンを呼ぶことです。

利用者定義演算

演算であって組込み演算でないものです。利用者定義演算は、総称識別子に関連付けた関数によって定義します。

利用者定義代入文

代入文であって組込み代入文でないものです。利用者定義代入は、サブルーチンおよびASSIGNMENT(=) の指定のある引用仕様宣言によって定義します。

利用者定義派生型入出力

派生型の並び項目についての実際のデータ転送処理が、利用者定義派生型入出力手続によって制御できます。

利用者定義派生型入出力手続

派生型入出力総称指定によって参照可能な手続です。

連携処理系

大域的なデータおよび手続を引用、確定、または定義できる機構のことです。そのような言語要素をFortran 以外の手段でも引用、確定、または定義することができます。

割付け共配列

ALLOCATABLE 属性をもつ共配列です。

割付け末端成分

ALLOCATABLE属性をもつ末端成分です。

割付け変数

ALLOCATABLE 属性をもつ変数です。割付け変数は、割り付けられているときにだけ、引用したり確定したりしてかまいません。配列である場合、割り付けられているときにだけ、形状をもちます。割付け変数は、名前付き変数または構造体成分です。

付録F　ASCII コード表

文字型データの内部表現は、ASCII コード系で表され、1個の文字は、パリティビットなしの8ビットの2進数で表現されます。

文字の大小順序は、文字のコード系における10進順序によって定まります。

ASCII コード系の文字の大小順序を下表に示します。

表F.1 ASCII コード系の文字の大小順序

文字	大小順序 (10進)	大小順序 (16進)	文字	大小順序 (10進)	大小順序 (16進)	文字	大小順序 (10進)	大小順序 (16進)
NUL	0	00	+	43	2B	V	86	56
SOH	1	01	,	44	2C	W	87	57
STX	2	02	-	45	2D	X	88	58
FTX	3	03	.	46	2E	Y	89	59
EOT	4	04	/	47	2F	Z	90	5A
ENQ	5	05	0	48	30	[	91	5B
ACK	6	06	1	49	31	¥	92	5C
BEL	7	07	2	50	32	]	93	5D
BS	8	08	3	51	33	^	94	5E
HT	9	09	4	52	34	_	95	5F
LF	10	0A	5	53	35	`	96	60
VT	11	0B	6	54	36	a	97	61
FF	12	0C	7	55	37	b	98	62
CR	13	0D	8	56	38	c	99	63
SO	14	0E	9	57	39	d	100	64
SI	15	0F	:	58	3A	e	101	65
DLE	16	10	;	59	3B	f	102	66
DC1	17	11	<	60	3C	g	103	67
DC2	18	12	=	61	3D	h	104	68
DC3	19	13	>	62	3E	i	105	69
DC4	20	14	?	63	3F	j	106	6A
NAK	21	15	@	64	40	k	107	6B
SYN	22	16	A	65	41	l	108	6C
ETB	23	17	B	66	42	m	109	6D
CAN	24	18	C	67	43	n	110	6E
EM	25	19	D	68	44	o	111	6F
SUB	26	1A	E	69	45	p	112	70
ESC	27	1B	F	70	46	q	113	71
FS	28	1C	G	71	47	r	114	72
GS	29	1D	H	72	48	s	115	73
RS	30	1E	I	73	49	t	116	74
US	31	1F	J	74	4A	u	117	75

文字	大小順序 (10進)	大小順序 (16進)	文字	大小順序 (10進)	大小順序 (16進)	文字	大小順序 (10進)	大小順序 (16進)
SP	32	20	K	75	4B	v	118	76
!	33	21	L	76	4C	w	119	77
"	34	22	M	77	4D	x	120	78
#	35	23	N	78	4E	y	121	79
$	36	24	O	79	4F	z	122	7A
%	37	25	P	80	50	{	123	7B
&	38	26	Q	81	51	|	124	7C
'	39	27	R	82	52	}	125	7D
(	40	28	S	83	53	~	126	7E
)	41	29	T	84	54	DEL	127	7F
*	42	2A	U	85	55			

索　引

[D]

[E]

[F]

[G]

[H]

[I]

[R]

[S]

[さ]

[た]

[な]

[は]

[ま]

[や]

[ら]

[わ]