はじめての Fortran90

浅岡 香枝 *　平野 彰雄 **

1　はじめに

プログラミング言語の FORTRAN の歴史は非常に古いです。1956 年に IBM 社によって IBM704 計算機用に FORTRAN(FORmula TRANslation: 数式翻訳) の名称で発表されたのが最初です。その後、機能追加や改良が行われ、科学技術計算分野でのプログラミング言語として最もポピュラーな言語として発展してきました。

これまでの規格は、

- 1965 年 FORTRAN66
- 1978 年 FORTRAN77
- 1991 年 Fortran90

という経過をたどり、最新は Fortran90 です。

センターで利用されるプログラミング言語は、Fortran が圧倒的に多く、Fortran90 コンパイラも一昨年の 6 月から運用しています。

しかし、Fortran90っていっても今までの 77 のプログラムがちゃんと動くわけだし、わざわざきちんと動くプログラムを 90 に書き換える必要はないし、そんなのはめんどうだ、と思っておられる人も多いのではないでしょうか。

そんな人のために、具体的にどんなことが 90 ではできるようになったのかという事を中心に Fortran90 のプログラミングについて解説していきたいと考えています。

なお、**ここに載せたプログラム例は、プログラム中のキーワードはすべて大文字で表し、[ ] で囲った部分は省略可能であることを示しています。**

また、この解説記事「はじめての Fortran90」は連載する予定でガンバリたいなと思っています。

今回は、まず Fortran90 の新しい機能について述べ、プログラムの形式、宣言、DO ループ、手続き、配列演算についてプログラム例も示しながら解説します。

2　Fortran90 の新しい機能

Fortran90 の主な追加機能には次のようなものがあります。

1) 自由形式のプログラム

カラムに依存しない自由なプログラムが書けます。

2) プログラムの移植性の向上

プログラムが扱う数値の精度をパラメータ化することにより、計算機ハードに依存しないプログラムが書けます。

3) プログラムの安全性の向上

引数の型や属性を定義する仕様が追加され、これらを用いてコーディングされたプログラムの誤りは、実行時では無くコンパイル時にエラーとして検出してくれます。

4) 強力な配列演算機能

同じ形状の配列で並列実行可能な配列演算が DO ループを用いずに書く事ができます。また多くの配列処理の組込み関数が新たに追加されています。

5) 動的なメモリ管理機能

配列をプログラム実行中に割付けることができます。これにより任意の大きさの配列を扱うことができます。またサブルーチンや関数では自動的に割付けられる配列も宣言できます。

6) 手続きの拡張

これまでの外部手続きに加えて、内部手続きが使えます。また、再帰手続きも書けます。さらに、モジュールという新しいプログラム単位が追加されています。

7) その他

ポインターが使えます。また、基本データ型に加えて利用者定義の構造型が定義できます。

* あさおか かえ ,** ひらの あきお (京都大学大型計算機センター)

3　廃止予定事項

Fortran90 では今後の改訂で廃止することを予告されているものがあります。これらはあまり使われていない文や他の文で書く事ができるものです。

- 算術 IF 文
- IF 構文のブロックの外からその END IF 文への飛び越し
- 実数型, 倍精度実数型の DO 変数および DO 制御式
- 共有 DO 端末文、および CONTINUE 文または END DO 文ではない DO 端末文
- ASSIGN 文と割当て GO TO 文
- 選択 RETURN 文
- PAUSE 文
- 割当て形 FORMAT 仕様
- H 型編集記述子

```
PROGRAM free_source_form
  ! このような長い変数名をつけられます
  INTEGER :: &
   long_name_max31_with_underscore
  REAL :: ra,rb,rc ! 以降行末まで注釈です
  ra=1.0; rb=2.0; rc=ra * rb
  ! 一行に複数の文も書けます
  ! 行の継続は、 & で
  WRITE(*,*) &
        ra,rb,rc
END PROGRAM free_source_form
```

4　自由形式のプログラム

- 一行は 132 カラム以内で、どのカラムからでも自由に書くことができます。
- 英数字と特殊文字を使ってプログラムを書きます。英小文字も使えますが、英大文字と英小文字の区別はされません。
- 変数名や手続き名などのプログラムで使用される名前は_(アンダーライン) を含めて 31 文字まで使えます。
- ;(セミコロン) で区切ることにより複数のステートメントを一行に書くことができます。
- !(エクスクラメーション) 以降から行末までは注釈となります。
- &(アンパサンド) を行末につけることによって次の行に継続します。
- 空白を識別します。
- 表 1に示す新たな関係演算子が使えます。

表 1. 関係演算子

演算子	演算子	意味
a.lt.b	a<b	a は b より小さい
a.gt.b	a>b	a は b より大きい
a.le.b	a>=b	a は b 以上
a.ge.b	a<=b	a は b 以下
a.eq.b	a==b	a と b は等しい
a.ne.b	a/=b	a と b は等しくない

5　データ型と宣言文

プログラムで扱うデータは定数と変数があります。また、扱うデータの種類には整数、実数などの種類がありこれをデータ型といいます。

ここでは、 Fortran で扱えるデータ型と宣言について説明します。

5.1　組込みデータ型

Fortran で扱えるデータは数値データ型として整数型、実数型、複素数型の三つがあり、また、非数値データには論理演算のための論理型と文字型があります。これら五つを組込みデータ型といいます。Fortran90 ではこれらの組込みデータ型に加え構造型があります。これは利用者が自分のデータ処理に応じて定義できるものです。

表 2に組込みデータ型の種類と宣言で用いるキーワードを示します。

表 2. 組込みデータ型の種類

データ型	キーワード
整数型	INTEGER
実数型	REAL
複素数型	COMPLEX
論理型	LOGICAL
文字型	CHARACTER

5.2　種別パラメータ

新機能の紹介で プログラムが扱う数値の精度をパラメータ化することで計算機ハードに依存しないプログラムを書くことができるといいましたが、こ

れが種別パラメータ、 KIND パラメータと呼ばれるものです。

規格では、種別パラメータの値 (種別値) は正の整数で表される値であり、具体的なパラメータ値は各処理系で定義され、整数型で一つ以上、実数型では二つ以上持つことになっています。

センターで指定可能なデータ型と種別値の対応を表 3に示します。

表 3. データ型と種別値の対応

データ型	種別値
整数型 (1 バイト)	1
整数型 (2 バイト)	2
整数型 (4 バイト)	4
整数型 (8 バイト)	8
実数型 (単精度)	4
整数型 (倍精度)	8
複素数型 (単精度)	4
複素数型 (倍精度)	8
論理型 (1 バイト)	1
論理型 (4 バイト)	4
文字型 (1 バイト)	1

また種別パラメータは省略可能であり、省略すると各データ型の基本データ型として宣言されます。

表 4にそれぞれの基本データ型の種別値を示します。

表 4. 基本データ型の種別値

データ型	種別値
INTEGER	4
REAL	4
COMPLEX	4
LOGICAL	4
CHARACTER	1

5.3 変数の宣言

プログラムで使用する変数は、全て型宣言文で宣言する必要があります。

型宣言文は、次のような形式です。

type [, **attribute** ,...] :: **name**[, ...]

type には変数のデータ型を種別パラメータと合わせて指定します。

また、 **attribute** は変数に付加する属性で、次のものがあります。

- PARAMETER　名前付き定数
- DIMENSION　配列
- ALLOCATABLE　割付け可能
- SAVE　状態保存
- INTENT　授受特性
- OPTIONAL　省略可能
- POINTER　ポインタ
- TARGET　ターゲット
- INTRISTIC　組込み手続き名の宣言
- EXTERNAL　外部手続き名の宣言

name には変数名を指定します。,(コンマ) で区切って同じデータ型のものを複数宣言できます。

例えば、整数の基本データ型の変数integer_varの宣言は、次のように書きます。

```
INTEGER(KIND=4) :: integer_var ! (1)
INTEGER(4)      :: integer_var ! (2)
INTEGER         :: integer_var ! (3)
```

(1) の形式は、もっとも基本的な形式です。

(2) のように **KIND=** は省略しても構いません。

(3) の形式は種別パラメータを省略した場合です。この場合は基本データ型で宣言されます。

倍精度実数型の変数double_a,double_b の二つ宣言するには次のように,(コンマ) で区切って並べます。

```
REAL(KIND=8)  ::  double_a,double_b  !(1)
REAL(8)       ::  double_a,double_b  !(2)
```

実数型の基本データ型は、単精度ですから倍精度実数型の変数を宣言する場合には種別パラメータを省略できません。

5.4 定数の宣言

変数名は型宣言文でデータ型と種別パラメータを指定して宣言しましたが、定数も同じように種別パラメータの組合わせで宣言することができます。

定数の宣言では、定数の数値の後に_(アンダーライン) をつけて種別値を指定します。

例えば、整数の定数の宣言は、次のように書きます。

```
1000                ! (1)
12345_4             ! (2)
5000000000_8        ! (3)
```

(1),(2) は、基本データ型の整数で 4 バイト領域を占有します。

(3) は、 8 バイトで表され 5000000000 の値を持っています。 5000000000 は基本データ型では表せないので_8 をつけて 8 バイト整数型の定数として宣言しています。

また、実数の定数の宣言は、次のように書きます。

```
1.0                 ! (1)
1.0_4               ! (2)
1.0e0               ! (3)
1.0_8               ! (4)
1.0d0               ! (5)
```

(1)、 (2)、 (3) が単精度実数、 **(4)、 (5)** が倍精度実数の宣言です。 **(3)、 (5)** は、これまでの実数定数の書き方です。

5.5 型宣言文での初期値設定

型宣言文で変数に初期値を設定できます。指定は変数名に=(イコール) で定数を代入します。また、=(イコール) の右辺には演算式も書けます。

```
INTEGER(KIND=4) :: integer_var=12345_4
INTEGER(8)  :: double_int=5000000000_8
REAL(8)  :: double_a=2.0_8*3.141592654_8
```

また、 PARAMETER 属性を指定して名前つき定数も簡潔に定義できます。

```
REAL,PARAMETER :: pi=3.14
INTEGER,PARAMETER :: cols=100,row=cols+1
```

5.6 利用者が定義する種別パラメータ

自分自身のプログラムが扱うデータがどういう範囲の値を取り、また、必要な指数桁、有効桁を考えることは容易ですが、それを表すのに必要な種別パラメータを考えるのはめんどうです。

このために Fortran90 には有効桁数を 10 進数で指定し、必要な種別値を求めてくれる組込み関数が用意されています。

- **SELECTED_INT_KIND(p)**

これは整数型の種別値を求める関数で、引数には必要な桁数を 10 進で指定します。引数を p とすると関数は処理系が持つ 10^{-p} 〜 10^{p} の範囲表せる整数の最小の種別値を教えてくれます。

指定された p が処理系で扱えないなら -1 を返します。

- **SELECTED_REAL_KIND(r,e)**

これは実数型の種別値を求める関数で、引数には 10 進精度 r と 10 進指数範囲 e を指定します。関数は処理系が持つ精度が r 以上で指数範囲が e 以上の実数型の最小の種別値を返します。

その処理系で 10 進精度 r が表せないとき -1、指数範囲 e が表せないとき -2、両方扱えないとき -3 を返します。

これらの組込み関数を用いて利用者が独自の種別パラメータを定義しプログラムが書けます。

```
INTEGER,PARAMETER :: &
  ip2=SELECTED_INT_KIND(2)       ! (1)
INTEGER,PARAMETER :: &
  rp10=SELECTED_REAL_KIND(10,70) ! (2)
INTEGER(KIND=ip2) :: n           ! (3)
REAL(KIND=rp10)   :: a,b         ! (4)
COMPLEX(KIND=rp10):: c           ! (5)
    :
n=10_ip2                         ! (6)
a=100.0_rp10; b=33.33_rp10       ! (7)
c=CMPLX(a,b,KIND=rp10)           ! (8)
    :
```

(8) の CMPLX は指定された引数をそれぞれ実部、虚部とする複素数型のデータを生成する組込み関数です。第三引数として種別パラメータ (KIND=) を指定し、結果の精度を指定しています。

表 5. 数値問合わせ組込み関数一覧

組込み関数	機　　能
KIND(x)	引数の種別値を返す
HUGE(x)	実数型、整数型の引数の数体系の最大値を返す
TINY(x)	実数型の引数の数体系の正の数の最小値を返す
RANGE(x)	実数型、整数型、複素数型の引数の数体系の 10 進指数範囲を返す
EPSILON(x)	実数型の引数の数体系の 1 に対してほとんど無視できる正の数を返す
PRICISION(x)	実数型、複素数型の引数の数体系の 10 進精度を返す
BIT_SIZE(x)	整数型の引数の数体系の整数のビット数を返す

5.7　処理系が扱う数値精度を調べる関数

Fortran90 では処理系が持つ数値データの精度などを知るために多くの組込み関数が用意されており、これらを使えば効率的に数値計算プログラムが書けます。

表 5に主な数値問合わせ関数と機能を示し、また、これらの組込み関数を用いたプログラムと実行結果を図 1に示します。

```
PROGRAM ex_inq_num
  IMPLICIT NONE
  INTEGER,PARAMETER :: &
       ip=SELECTED_INT_KIND(9), &
       rp=SELECTED_REAL_KIND(10,10)
  INTEGER(KIND=ip) :: ivar
  REAL(KIND=rp)    :: rvar
  WRITE(*,*) '<< INTEGER >>'
  WRITE(*,*) 'KIND       :',KIND(ivar)
  WRITE(*,*) 'HUGE       :',HUGE(ivar)
  WRITE(*,*) 'RANGE      :',RANGE(ivar)
  WRITE(*,*) 'BIT_SIZE   :',BIT_SIZE(ivar)
  WRITE(*,*) '<< REAL  >>'
  WRITE(*,*) 'KIND       :',KIND(rvar)
  WRITE(*,*) 'HUGE       :',HUGE(rvar)
  WRITE(*,*) 'TINY       :',TINY(rvar)
  WRITE(*,*) 'RANGE      :',RANGE(rvar)
  WRITE(*,*) 'PRICISION  :',PRICISION(rvar)
  WRITE(*,*) 'EPSILON    :',EPSILON(rvar)
END PROGRAM ex_inq_num
```

```
[ 実行結果 ]
<< INTEGER >>
KIND       : 4
HUGE       : 2147483647
RANGE      : 9
BIT_SIZE   : 32
<< REAL >>
KIND       : 8
HUGE       : 1.797693134862316+308
TINY       : 2.225073858507201-308
RANGE      : 307
PRICISION  : 15
EPSILON    : 2.220446049250313E-16
```

図 1. 数値問合せ関数を用いた例と実行結果

5.8　暗黙の型宣言の禁止

型宣言文で定義ない変数は、変数名の先頭の英字が I,J,K,L,M,N 場合と基本整数データ型、これ以外の英字のものは全て基本実数データ型の変数と見なすという暗黙の型宣言という概念があります。

これはプログラム内で変数を宣言せずに使えるので一見便利ように思えますが、プログラミングにとっては、とっても危険です。

次の例は FRTRAN77 の DO ループの制御式ですが、初期値と終了値の間を,(コンマ) と.(ピリオド) を間違えています。

```
DO 10 I=1.5
```

なのに、暗黙の型宣言により DO10I という基本実数型の変数に 1.5 を代入する式として解釈されしまい、 DO ループとして機能しない事が発生します。暗黙の型宣言が無ければ、コンパイル時に DO10I の変数が宣言されていないとコンパイラが教えてくれるはずです。

Fortran90 では各プログラム単位の宣言文の前に、

```
IMPLICIT NONE
```

と書けば、暗黙の型宣言を無効にすることができます。プログラムを書くときは必ず暗黙の型宣言を無効にし、使う変数は必ず型宣言文で定義すれば、一寸した誤りで実行時のデバックに時間を取られる事は避けられます。

6　プログラム単位と手続き

Fortran プログラムは一つのメインと複数の手続き (サブルーチン、関数) から構成されます。 For-

tran90 ではこの手続きに関して次のような機能が追加されています。

1) プログラムが読みやすいように、 END 文にプログラム単位を示すキーワード (PROGRAM、SUBROUTINE、 FUNCTION) と共に手続き名を指定できます。
2) これまで外部手続きしかありませんでしたが、新たに内部手続きが書けます。また、再帰手続きも書けるようになっています。
3) 仮引数に引数の扱い (授受特性という) を指定できます。これにより引数の誤りはコンパイルにチェックされます。
4) 手続きの引数では、キーワード引数やオプション引数が指定可能になり、より柔軟な手続きか書けます。

6.1 内部手続き

内部手続きとは、宣言された手続きの内だけで有効なものでサブルーチン、関数が書けます。手続きの最後に **CONTAINS 文**を書きその後に宣言します。複数の手続きを並べて書くこともできますが、内部手続き内に内部手続きを書くことはできません。

内部手続きを呼出す側を親手続きといい、ここで宣言された変数は、内部手続き内でも使えます。ただし、内部手続きで宣言された変数はその手続き内だけで有効です。

あまり意味のないプログラムですが、内部手続きの使用例を図 2に示します。これは`sub_init` というサブルーチンと`func_add` という内部手続きを持つメイン`main_program_name` から構成されています。

メインでは基本実数型の変数 a,b,c を宣言し、初期値として c に 100.0 を代入しています。また、サブルーチン`sub_init` は`b=b*c` の式で引数 (b) で与えられた値と変数 (c) の値を掛けたものを返します。ここで変数 (c) はサブルーチン内では宣言していないので、メイン (親手続き) で宣言された変数 c を使い結果として 100.0 倍されたものが返されます。

```
PROGRAM main_prog_name
 IMPLICIT NONE
 REAL :: a,b,c=100.0
 b=1.0
 CALL sub_init(b)
 a=func_add(b,c)
 CONTAINS
   SUBROUTINE sub_init(b)
     REAL,INTENT(INOUT) :: b
     b=b*c
   END SUBROUTINE sub_init
   REAL FUNCTION func_add(x,y)
     REAL,INTENT(IN) :: x,y
     func_add=x+y
   END FUNCTION func_add
END PROGRAM main_prog_name
```

図 2. 内部手続きを使ったプログラム例

6.2 手続きと引数の属性

6.2.1 授受属性 (INTENT)

手続きの引数の宣言には、キーワード INTENT を用いて引数の授受特性が指定できるようになりました (図 2参照)。

引数の授受特性とは引数の値の受渡し扱いを示すもので、手続きに値が渡されるもの (入力)、手続きから値を返すもの (出力)、値が渡され、結果を返すもの (入出力) に分類し、これを表 6のような形式で指定します。

表 6. 授受特性と INTENT の指定

授受特性	INTENT の指定
入力	INTENT(IN)
出力	INTENT(OUT)
入出力	INTENT(INOUT)

例えば、図 2のプログラムで引数の INTENT 指定を書かないでサブルーチンの呼び出しを

```
CALL sub_init(1.0)
```

のように引数に定数を指定しても正常にコンパイルできてしまいます。しかし、実行させると思った通りの結果が得られないことになります。

これは Fortran の引数の渡しは、アドレスが渡されるのでコンパイラが用意した定数 1.0 の格納領域のアドレスがサブルーチンに渡され、サブルーチン内で c 倍 (100.0) されたものが、元の定数 1.0 の格納領域にセットされてしまうからです。

サブルーチン呼出しの引数を誤っただけなのにプログラム内の定数の値が変わってしまう何て事は、かなり熟練したプログラマでないと原因を発見できません。

INTENT を仮引数宣言に指定すれば、このような誤りもコンパイル時にエラーになります。引数宣言には、必ず INTENT 属性を正しく指定しましょう。

6.2.2 キーワード引数

手続きの引数の機能拡張では、キーワード引数という概念もあります。これは引数の宣言に特別な指定をするのでは無く、宣言された仮引数名を使って、呼出し側で、

仮引数名 = 実引数

形式で書けるようになっています。

手続きの呼出し時、実引数は仮引数で指定された順に,(コンマ) で区切って指定する必要がありますが、キーワード引数で指定すると順序と関係なく引数を書くことができ、また、仮引数と実引数の対応が明確なプログラムになります。

例えば、次のように宣言された、

```
INTEGER FUNCTION scale(start,finish,point)
```

関数の呼出しは、以下のように

```
inc=scale(1,20,10)                      !(1)
inc=scale(point=2,start=1,finish=20)    !(2)
inc=scale(1,finish=20,point=2)          !(3)
```

指定できます。

もちろん、実引数の並びで仮引数の定義順に書けばキーワードは要りません **(1)** が、一旦、キーワード引数で指定すると後の並びはキーワードで指定する必要がありますのでご注意下さい。

6.2.3 省略可能な引数

手続き呼出し時に省略可能な引数を定義できます。省略可能な引数の指定は、仮引数の宣言に **OPTIONAL** 属性を指定します。また、手続き内でその引数が指定されたか否かを判定するには、組込み関数 **PRESENT** が使えます。

PRESENT は、指定された仮引数が指定されていると**.TRUE.**、省略された場合は**.FALSE.** を返します。

図 3はキーワード引数と省略可能引数を使った例です。

```
PROGRAM ex_key_optional
  IMPLICIT NONE
  INTEGER  :: inc
      :
  inc=scale(start=1,finish=20)
      :
  CONTAINS
    INTEGER FUNCTION  &
      scale(start,finish,point)
    INTEGER,INTENT(IN) :: start,finish
    INTEGER,INTENT(IN),OPTIONAL:: &
       point
    INTEGER :: lpoint=10
! Check optional argment
    IF (PRESENT(point)) THEN
      lpoint=point
    ENDIF
      scale=(finish-start)/lpoint
    END FUNCTION scale
END PROGRAM ex_key_optional
```

図 3. キーワードと省略可能引数

6.3 外部手続きの引数宣言と引用仕様

これまで内部手続きの例を元に、キーワード引数や省略可能引数の指定方法および引数の宣言に INTENT で授受特性と指定すると誤ったプログラムは、コンパイルにチェックされエラーになると説明しました。

しかし、外部手続きは予めコンパイルしライブラリ化しておき、別にコンパイルするプログラムと結合することもできます。

したがって、外部手続きを結合するプログラムのコンパイル時には、外部手続きの仮引数名や属性をコンパイラが知る方法がありません。

引用仕様宣言は、これを補い外部手続での仮引数の宣言の情報を定義し、コンパイラに教えるものです。

引用仕様宣言 **INTERFACE ブロック**の形式は、外部手続きの仮引数の宣言部を **INTERFACE** 文と **END INTERFACE** 文の間に並べて書きます。例えば、図 3の関数 scale が外部手続とした場合の **INTERFACE ブロック**は次のようになります。

```
INTERFACE
  INTEGER FUNCTION  &
      scale(start,finish,point)
  INTEGER,INTENT(IN) :: start,finish
  INTEGER,INTENT(IN),OPTIONAL:: &
       point
  END FUNCTION scale
END INTERFACE
```

7　DO ループと配列演算

配列演算機能とは、 FORTRAN77 から Fortran90 に進歩した中で最も大きな改良点の一つです。これまで全ての配列演算は DO ループを用いて書いていましたが、 Fortran90 からは同じ形状の配列で並列実行可能な配列であれば DO ループを使わずに簡潔に表すことができます。

また、 DO ループの記述も改良され、より読みやすいプログラムが書けるようになっています。

ここでは、まず、新しい DO ループの書き方について説明し、そして、配列の宣言、配列演算について説明します。

7.1　DO ループの改良と機能強化

FORTRAN77 で DO ループの記述はループ毎に文番号 (5 桁以内の数値) を割り当て、対応する端末文として文番号を付けた CONTINUE 文を置くという形式でした。

```
    DO 100 i=1,100,5
      :
      実行文
      :
100 CONTINUE
```

これを Fortran90 では、次のように書きます。

```
loop:  DO  i=1,100,5
       :
       実行文
       :
       END DO loop
```

改良された点は、 1) 端末文として CONTINUE 文の代わりに **END DO** 文を指定し、 2) DO ループには、文番号を付ける必要がなくなり、 3) 文番号の代わりに**ラベル (loop)** を DO、 END DO 文に指定することでループの範囲が明確で、分かりやすい記述ができるようになっています。

また、 FORTRAN77 では無限ループは、 GOTO 文と IF 文の組合わせでしか書けませんが、 Fortran90 では DO ループの制御式 (i=1,100,5) を省略すると無限ループになります。さらに新たに追加されたループの制御文 CYCLIC(繰返し)、 EXIT(抜出し) を使って制御できます。

これまで文番号は 5 桁の数字だけで識別するのでコーディング時の管理が大変で、また、 Fortran のプログラムが読みづらい原因の一つでした。

Fortran90 では 31 文字以内の識別名をラベルとして書けるので、適切なラベルを付けることで読みやすいプログラムが書けます。なお、 Fortran90 では、 GOTO 文を止め入出力文の書式指定も文字定数や文字変数を使えば文番号を一切使わないプログラムが書けます。

図 4は、ゼロが入力されるまでの値を読み、入力された数 (total) と正の値の数を数え上げるプログラムを DO ループと CYCLE 文と EXIT 文を使用して書いた例です。

```
PROGRAM ex_do
  IMPLICIT NONE
  INTEGER :: num,total,count
  total=0; count=0
  data_input:  DO
     WRITE(*,*) 'Input number '
     READ(*,*) num
     IF ( num == 0 ) EXIT data_input
     total = total + 1
     IF ( num < 0 ) CYCLE data_input
     count=count+1
  END DO input_data
  WRITE(*,*) total,count
END PROGRAM ex_do
```

図 4. 無限ループの繰返しと抜出し例

7.2　配列の宣言

配列を宣言するには次のように、

type,DIMENSION(extent,...) :: name,...

型宣言文に **DIMENSION** 属性を指定します。 **DIMENSION** 属性には、各次元の寸法 (extent) を,(コンマ) で区切って括弧で囲んで指定します。また、次元寸法は、:(コロン) で区切ることで

[下限:] 上限

の形式で添字の下限、上限を指定することもできます。**下限:** が省略された時は、下限は 1 です。

```
INTEGER,DIMENSION(100) :: array100
INTEGER,DIMENSION(0:9) :: array10
REAL,DIMENSION(10,5)   :: array50
REAL,DIMENSION(-3:1,8) :: array32
```

7.2.1　配列の形状と用語

配列宣言の形式を説明したので、次は配列演算を理解する上で必要な用語紹介します。

キーワードは、**形状 (shape)** と**形状適合 (conformable)** です。

- **形状 (shape)** とは

配列の次元数 (rank) と各寸法 (extent) を一次元配列で表したものです。

- **形状適合 (conformable)** とは

二つの配列が同じ形状の場合に、この二つの配列は形状適合しているといい、一つの式で配列同士の演算が表せます。なお、スカラはどのような形状の配列とも形状適合します。

次のような配列が宣言されている場合に

```
INTEGER,DIMENSION(2:4,3:10) :: a
INTEGER,DIMENSION(3,8)      :: b
INTEGER                     :: c
INTEGER,DIMENSION(2,5)      :: d
```

配列 a は

- 次元数 (rank)：2
- 寸法 (extent)：3 と 8
- 大きさ (size)：24(3×8)
- 形状 (shape)：(/3,8/)

であるといいます。

また、配列 a に形状適合する配列は、配列 b とスカラ変数 c です。

7.2.2　配列構成子と初期値設定

配列構成子とは、配列定数の記述で (/ と /) の中に各要素を, (コンマ) 区切って一次元の配列として表現します。

例えば、次のものは、

```
(/ 1,2,3,4,5,6 /)
```

1,2,...,6 という整数を要素とする配列構成子です。

また、連続した値の場合には、 DO の制御式を括弧で囲んで表せ、また、演算式も書けます。

```
(/ (i,i=1,6) /)        !(1)
(/ ((i/10),i=2,10,2) /)   !(2)
(/ 10,20,(i,i=2,10,2) /)   !(2)
```

ここで、

(1) は (/ 1,2,3,4,5,6 /)、

(2) は (/ 2.0,4.0,6.0,8.0,10.0 /)、

(3) は (/ 10,20,2,4,6,8,10 /)

を表しています。

一次元配列の初期値は配列構成子を用いれば、型宣言文で次のように与えることができます。

```
INTEGER,DIMENSION(6) :: &
    line=(/ (i,i=1,6) /)
```

また、多次元配列は組込み関数 **RESHAPE** を使えば初期値が与えられます。

```
INTEGER,DIMENSION(2,4) :: box = &
  RESHAPE( (/ (i,i=1,8) /), (/ 2, 4 /) )
```

この宣言文で二次元配列 box には、

```
1 3 5 7
2 4 6 8
```

という値が設定されます。

7.3　配列演算式

7.3.1　全体配列と部分配列

配列演算式は DO ループを書く代わりに、配列の各次元に添字式を次のような形式で指定します。

[初期値]:[限界値][: 増分値]

これは DO ループの制御式を:(コロン) で区切ったのと同じで、指定された次元の添字を初期値から限界値まで増分値を足しながら繰り返すことを指定しています。

増分値は省略可能で、省略すると増分値 1 と見なされます。また、初期値、限界値も省略して:(コロン) だけを指定するとその次元の全体を指定ことになります。

また、配列全体を表すときは、コロンも省略することもできます。

全体配列とは、各次元をコロンで表すか、コロンも省略したものを指し、一方、部分配列とは、添字式が指定された配列を指します。

1) 全体配列式の例

例えば a(10,10) という二次元配列の全体を 0.0 で初期化するには、これまでは DO ループを使って (1) のように記述していましたが、これを配列演算式では (2) のように簡単に書けます。全体配列の演算式は、(3) のように書くことも可能ですが、このような書き方では、a が配列あるいは変数かを知るには、型宣言文を見る必要があり、曖昧な式となるので (2) のように書く方が良いと思います。

```
DO i=1,10
  DO j=1,10
    a(j,i)=0.0       !(1)
  END DO
END DO

a(:,:)=0.0           !(2)

a=0.0                !(3)
```

2) 部分配列式の例

同じ配列 a の一部に DO ループと部分配列式で値 1.0 を代入する例です。

```
DO i=2,10,2
   DO j=1,10
      a(j,i)=1.0     !(1)
   END DO
END DO

a(:,2:10:2)=1.0      !(2)
```

7.3.2 配列演算式と組込み関数

簡単な組込み関数を使った配列演算式の例を以下に示します。

例 1

二次元配列の a と b を値を足し、c に代入します。

```
c(:,:)=a(:,:)+b(:,:)
```

例 2

二次元配列の a の平方根を求め、1000 倍します。

```
a(:,:)=SQRT(a(:,:))*1000.0
```

ここで、SQRT は平方根を求める組込み関数です。

例 3

配列 a の平均を求める式

```
mean=SUM(a(:,:))/SIZE(a(:,:))
```

ここで、SUM は指定された引数 a の総和を求める関数で、配列の大きさを返す関数です。

7.3.3 配列演算式の注意

配列演算式で表せるのは、並列実行可能な演算のみです。

例えば、

```
DO i=2,n
  a(i)=a(i-1)+a(i)      !(1)
END DO
```

の DO ループを単純に

```
a(2:n)=a(2:n)+a(1:n-1) !(2)
```

と配列演算式に直しても正しい結果は得られません。

なぜなら

- (1) 式は
  a(i)=a(i)+a(i-1)+a(i-2)+...+a(1)
- (2) 式は
  a(i)=a(i-1)+a(i)

の計算をしているからです。注意してください。

なお、(1) 式は、組込み関数 **SUM** と配列構成子を使えば配列演算式で表せます。

```
a(2:n)=(/ (SUM(a(1:i)),i=2,n) /)
```

7.3.4 大きさゼロの配列配列と演算

大きさゼロの配列とは、a(4:3) のように添字の下限が上限を越えているものをいい、配列演算式では、大きさゼロの配列に対しては何も実行しないことが保証されています。

次のプログラムは、連立一次方程式の下三角形を解く式です。

```
DO i=1,n
 x(i)=b(i)/a(i,i)                    !(1)
 b(i+1:n)=b(i+1:n)-a(i+1:n,i)*x(i)   !(2)
END DO
```

このプログラムの式 (2) は、i=n の時配列 a,b の添字式は n+1:n となって大きさゼロの配列して扱われ式 (2) は実行されなくなります。

7.3.5　配列選別代入文

配列演算式において配列要素の値により異なる演算を行う場合、**単純 WHERE 文**、**WHERE 構文**を用います。

単純 WHERE 文例

```
WHERE( MOD(odd(:),2) /= 0)  &
        odd(:)=odd(:)*2
```

これは配列 odd の中身が偶数か否かを組込み関数 **MOD** で調べ、奇数の要素は 2 倍して偶数に直しています。

WHERE 構文例

```
WHERE( a(:) > 0.0 )
  a(:) = SQRT(a(:))
ELSEWHERE
  a(:) = 0.0
END WHERE
```

これは配列の値を調べ正であれば平方根を求め、それ以外は 0.0 を設定する式です。

7.3.6　割付け配列

Fortran90 では、実行時に任意の多きさの配列を動的に割付けて使えるようになり、この配列を割付け配列と呼びます。

割付け配列の宣言は、型宣言文で **ALLOCATABLE** 属性を指定し、また、**DIMENSION** 各次元の寸法を:(コロン) で表し、次元数だけを指定します。

配列の割付けおよび解放は **ALLOCATE** 文、**DEALLOCATE** 文で行います。

次は、実行時に **READ**　文で読んだ n の値で二次元配列 a を確保する割付け配列の宣言と割付け、解放するプログラム例です。

```
PROGRAM allocatable_array
  REAL,DIMENSION(:,:),ALLOCATABLE :: a
  INTEGER  n
  READ(*,*) n
  ALLOCATE(a(n,n+1)))
          :
  DEALLOCATE(a)
END PROGRAM allocatable_array
```

7.3.7　手続きの引数としての配列

1) 形状引継ぎ配列

形状引継ぎ配列とは、手続きに配列を渡す場合に仮引数の宣言文で **DIMENSION** に次元の数だけ:(コロン) を,(コンマ) で区切って宣言しておくと、実引数で指定された配列の各次元の寸法が呼出しに解決され、同じ形状の配列として手続き内で使用できるものです。

2) 自動割付け配列

自動割付け配列とは、仮引数配列の大きさによって必要な配列を手続き内部で宣言するもので、実際の割付け、解放は処理系が動的に行います。

図 5に示すサブルーチン swap で、仮引数配列 a,b は形状引継ぎ配列であり、 work が自動割付け配列の宣言です。 **SIZE** は配列の大きさを調べる組込み関数です。

```
SUBROUTINE swap(a,b)
  REAL,DIMENSION(:) :: a,b
  REAL,DIMENSION(SIZE(a)) :: work
  work=a; a=b; b=work;
END SUBROUTINE swap
```

図 5. 形状引継ぎ配列と自動割付け配列

8　プログラム例

これまでに解説した Fortra90 の機能を使って簡単なプログラム例を示します。

1) 得点計算

オリンピックの体操競技の採点のように、 5 人の審判員がつけた点の最大値と最小値を除いた残りの平均で得点を計算します。最大値と最小値は組込み関数 **MAXVAL**,**MINVAL** で求めています。

```
PROGRAM ex_mean
  IMPLICIT NONE
  REAL,DIMENSION(5) :: vec
  REAL :: mean
  READ(*,*) vec
  mean = (SUM(vec)-MAXVAL(vec)- &
     MINVAL(vec))/REAL(SIZE(vec)-2)
  WRITE(*,*) " 得点    := ", mean
END PROGRAM ex_mean
```

2) 平均、分散、標準偏差

まず、最初にデータの数を読み、次に指定されたデータ数分の配列を確保し、配列に読み込んだデータの平均、分散、標準偏差値を求めるプログラムです。

```
PROGRAM ex_std
 IMPLICIT NONE
 INTEGER ::  n
 REAL,DIMENSION(:),ALLOCATABLE :: vec
 READ(*,*) n
 ALLOCATE(vec(n))
 READ(*,*) vec
 WRITE(*,*) " 平均      := ", mean(vec)
 WRITE(*,*) " 分散      := ", disp(vec)
 WRITE(*,*) " 標準偏差 := ", std_vec(vec)
 DEALLOCATE(VEC)
 CONTAINS
   REAL FUNCTION mean(vec)
     REAL,DIMENSION(:) :: vec
     mean = SUM(vec)/REAL(SIZE(vec))
   END FUNCTION mean
   REAL FUNCTION disp(vec)
     REAL,DIMENSION(:) :: vec
      disp = &
      SUM((vec-mean(vec))**2) &
                /REAL(SIZE(vec))
     END FUNCTION disp
     REAL FUNCTION std_vec(vec)
       REAL,DIMENSION(:) :: vec
       std_vec = SQRT(disp(vec))
     END FUNCTION std_vec
 END PROGRAM ex_std
```

3) 連立一次方程式の解

教科書に良く出ているガウスの消去法による連立一次方程式の解を求めるプログラムを Fortran90 で書いたものです。

このプログラムで使っている MAXLOC は、指定された配列の最大値要素を持つ要素位置を一次元配列で返す組込み関数です。また、DOT_PRODUCT は引数で与えられた配列の内積を結果として返す組込み関数です。

9　おわりに

Fortran90 の基礎と配列演算を中心に解説しました。一人でも多く人が 90 の良さに気がついて頂けたらうれしいです。

次回は、構造体、ポインタ、モジュールを取り上げたいと考えています。それではまた。

```
PROGRAM ex_gauss
  IMPLICIT NONE
  INTERFACE
     SUBROUTINE gauss(a,order)
       REAL,DIMENSION(:,:), &
           INTENT(inout) :: a
       INTEGER,DIMENSION(:),&
           INTENT(inout) :: order
     END SUBROUTINE gauss
  END INTERFACE
  INTEGER :: n,i
  REAL,DIMENSION(:,:), &
       ALLOCATABLE :: a
  INTEGER,DIMENSION(:),&
       ALLOCATABLE :: order
  READ(*,*) n
  ALLOCATE(a(n+1,n),order(n))
  DO i=1,n
     READ(*,*) a(:,i)
  END DO
  order=(/(i,i=1,n)/)
  CALL gauss(a,order)
  WRITE(*,*) a(n+1,:)
  DEALLOCATE(a)
END PROGRAM ex_gauss

SUBROUTINE gauss(a,order)
 IMPLICIT NONE
 INTEGER :: i,j ,n
 REAL,DIMENSION(:,:) :: a
 INTEGER,DIMENSION(:) :: order
 INTEGER,DIMENSION(1) :: line
 ! 消去部
 n=SIZE(a,2)
 DO j=1,n
    line=MAXLOC(ABS(a(j,j:n)))+j-1
    IF (line(1) /= j) &
      CALL swap(a(:,line(1)),a(:,j), &
        order(line(1)),order(j))
    DO i=j+1,n
      a(j,i)=a(j,i)/a(j,j)
       a(j+1:n+1,i)=a(j+1:n+1,i)- &
            a(j,i)*a(j+1:n+1,j)
    END DO
 END DO
 ! 後退代入部
 DO i=n,1,-1
 a(n+1,i)=a(n+1,i)- &
   DOT_PRODUCT(a(n+1,i+1:n),a(i+1:n,i))
 a(n+1,i)=a(n+1,i)/a(i,i)
 END DO
 a(n+1,order(:))=a(n+1,:)
CONTAINS
   SUBROUTINE swap(a_i,a_j,o_i,o_j)
     REAL,DIMENSION(:) :: &
       a_i,a_j
     INTEGER o_i,o_j
     REAL,DIMENSION(SIZE(a_i)) :: work
     INTEGER tmp
     work=a_i
     a_i=a_j
     a_j=work
     tmp=o_i
     o_i=o_j
     o_j=tmp
   END SUBROUTINE swap
END SUBROUTINE gauss
```