はじめての Fortran90(その三)

浅岡 香枝 [*]　平野 彰雄 [**]

1　はじめに

新しい年を迎え、皆さま、いかがお過ごしでしょうか？前号には三回目の連載を掲載する予定だったのですが、できませんでした。このはじめてのシリーズがはじまるときに、連載のお約束をしたはずなのですが、三回目にして早くも破ってしまいました。ごめんなさい。

「祝 !!、京大 100 周年」。大型計算機センターのサテライト開場に、とっても多くの方々にきて頂いて、苦労して準備した者として感激しています。なーんて訳の判らないことを言っていないで、話を進めたいと思います。

今回は、 Frotran90 で新しく導入されたモジュールについて紹介します。

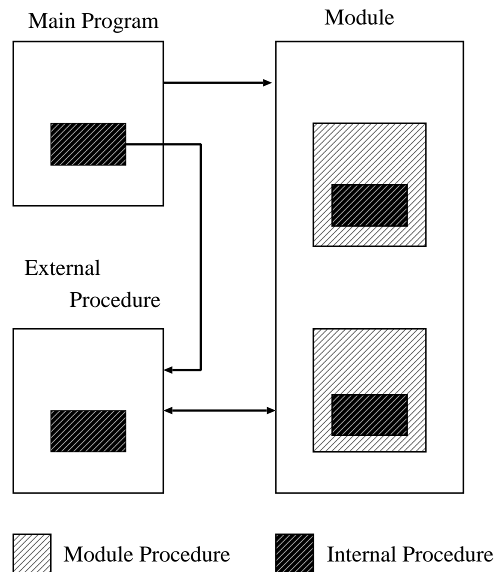

図 1. プログラム単位

2　モジュール手続き

2.1　プログラム単位

モジュール手続きの個々の機能を紹介する前に、Fortran90 のプログラム単位について整理して、紹介します。

プログラム単位とは、別々にコンパイルできるものです。 FORTRAN77 では、メインプログラムと外部手続き (サブルーチン、関数) の二つしかありませんでしたが、 90 では新たにモジュール手続きというものが追加されました。

図 1は、 90 のプログラム単位の関係を示してます。メインプログラムは、 Fortran の実行単位に一つ必要なもので、一番最初に実行されるモジュールです。また、外部手続きには、自分でプログラムの処理を整理して、分割し作ったサーブルーチンや関数あるいは Fortran の組込み関数および科学技術計算ライブラリ (SSL) といったものが含まれます。

基本的に外部手続きは、ソースプログラムを一つのファイルにまとめて書こうが、別々のファイルに分けようがそれぞれをサブルーチン、関数ごとにコンパイルして、目的モジュールを作成し、結合編集という手続きをへて実行モジュールが作られます。

モジュールも他のプログラム単位と同様に独立してコンパイルされ、他のプログラムから呼び出して利用できるものです。ただし、モジュールの結合処理はコンパイラが処理します。結合編集という手続きではありません。

また図 1は、各プログラム単位が内部手続きを持つことも表しています。モジュールにも手続き (サブルーチン、関数) を置くことができ、これをモジュール手続きといいます。

2.2　モジュールの宣言

モジュールは、メインプログラムやサブルーチンと同じように、先頭に **MODULE** とモジュールの名前を書き、最後は **END MODULE** とモジュールの名前を書きます。

[*] 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)

すなわち、モジュールの宣言は、

```
MODULE module_name
      ⋮
END MODULE module_name
```

のようになります。
また、モジュールを引用するには、

```
USE module_name
```

のように **USE 文**でモジュール名を指定します。

モジュールから他のモジュールを引用することもできます。基本的にモジュールで定義されているものを使用するプログラム単位には **USE 文**が必須です。当然、モジュール手続きの呼び出しは、外部手続きと同じで **CALL 文**や**関数名**を指定します。

2.3 文の順序

モジュールを引用するための **USE 文**は、プログラム内で書ける位置が決まっており、**PROGRAM 文**、**FUNCTION 文**、**SUBROUTINE 文**、**MODULE 文**の後で、**IMPLICIT NONE** より前に書く必要があります。

プログラムの文を何処に書けるのかは、それがどの範囲で有効かを考える上でとっても重要なことです。これを**文の順序**といい Fortran90 では表 1のように決められています。

表 1. 文の順序

<table>
<tr><td colspan="3">program 文,function 文,subroutine 文,module 文</td></tr>
<tr><td colspan="3">use 文</td></tr>
<tr><td rowspan="4">format 文</td><td colspan="2">implicit none 文</td></tr>
<tr><td>parameter 文</td><td>implicit 文</td></tr>
<tr><td>parameter 文
data 文</td><td>構造型定義
引用仕様宣言
型宣言文
単純宣言文</td></tr>
<tr><td colspan="2">実行文</td></tr>
<tr><td colspan="3">contains 文</td></tr>
<tr><td colspan="3">内部手続き
モジュール手続き</td></tr>
<tr><td colspan="3">end 文</td></tr>
</table>

2.4 名前の有効域

メインと外部手続きしかなかった FORTRAN77 と違い、色々と機能が拡張された 90 では、文の順序と同じように重要な概念として**有効域 (Scoping Unit)** があります。

有効域とは、宣言された外部手続き名やモジュール名さらに変数名などの名前がどの範囲で有効かについての定義です。

基本的に外部手続き名やモジュール名は、プログラム全体で有効なものとして扱われます。逆にいうと、一つのプログラムでは、同じ名前を持つ異なる外部手続きやモジュールを定義したり、引用できないことを意味します。

しかし、変数名に関しては、同じ名前であってもそれが何処で定義されたかにより有効域が決まり、参照できる名前についての規則が決められています。規則といっても複雑なものでは無く、簡単なものです。

具体的には、次のものに書かれた変数名は、独立な有効域を持ち、他に影響を与えません。

- 構造型の宣言
- 引用仕様の宣言

次に、 90 で導入された手続きとして内部手続きというものがあります。内部手続きも独立した有効域を持ちます。ただし、個々の変数についての有効域は少し複雑で、何処で宣言されたかで決まります。

```
PROGRAM internal
 IMPLICIT NONE
 REAL(4) ::  x,y,z
 x = 0.0; y = 0.0; z=0.0
 call sub(x)
 write(*,*) " x=",x," y=",y," z=",z
 CONTAINS
  SUBROUTINE sub(x)
  REAL(4) :: x
  REAL(4) :: z=1.0
  y = y + z
  x = y + z
  END SUBROUTINE sub
END PROGRAM internal
```

図 2. 内部手続きと変数の親子結合

図 2に示すプログラムは、この関係を明らかにするためのもので、あまり意味のあるプログラムではありませんが、プログラムの構造と変数の宣言に着目してみてください。

このプログラムは、 **sub** という内部サブルーチンを含んでおり、メインで x、 y、 z という変数を宣言し、それぞれに 0.0 という値を代入後、変数 x を引数として **sub** を呼出しています。

一方、内部サブルーチン **sub** は、引数 x と変数 z を宣言し、 z には 1.0 を初期値として代入しています。そして、二つの実行文を実行しています。この実行文で変数 y を用いていますが、これはサブルーチン **sub** では宣言してないのでメインで宣言された y がそのまま使われます。このような変数の扱いを**親子結合**といいます。**親**とは内部手続きを含む手続きで、**子**とは含まれる内部手続きを指す用語です。引数に現れる x や内部で宣言される z は、宣言された内部だけで有効になります。

このプログラムの実行結果は、 x= 2.0, y= 1.0, z=0.0 となるはずです。試してください。

図 3は、プログラム内での有効域の入れ子構造を概念的に示したものです。

```
PROGRAM scope1                  !(scope1)
      :                         !(scope1)
  CONTAINS                      !(scope1)
  +-------------------------------------+
  | SUBROUTINE scope2           !(scope2) |
  | +----------------------------------+|
  | | TYPE scope3               !(scope3)||
  | |     :                     !(scope3)||
  | | END TYPE scope3           !(scope3)||
  | +----------------------------------+|
  |   INTERFACE                 !(scope2) |
  | +----------------------------------+|
  | |       :                   !(scope4)||
  | +----------------------------------+|
  |   END INTERFACE             !(scope2) |
  |         :                   !(scope2) |
  |         :                   !(scope2) |
  |   CONTAINS                  !(scope2) |
  | +----------------------------------+|
  | |   FUNCTION scope5(...)    !(scope5)||
  | |           :               !(scope5)||
  | |   END FUNCTION scope5     !(scope5)||
  | +----------------------------------+|
  | END SUBROUTINE scope2       !(scope2) |
  +-------------------------------------+
END PROGRAM scope1              !(scope1)
```

図 3. 有効域の例

モジュールもその内部では図 3のメインプログラムと同じ有効域の構造を持ちます。しかし、他のプログラム単位から **USE 文**で引用される時、モジュール内部で宣言した変数などは、少し異なった扱いを受けます。詳しくは後で解説します。

2.5　モジュールの機能と用途

モジュール手続きの解説といいながらプログラム単位とか有効域とかの説明を長々とやってしまいましたが、モジュールを理解し、活用する上では重要な概念です。

基本的にモジュールの機能と用途は、次のようなものがあります。

1) グローバルな変数の宣言

FORTRAN77 の COMMON ブロックと同じようなグローバルな変数が宣言できます。

COMMON 宣言した変数に初期値を持たせる場合、 BLOCK DATA にまとめる必要があるなど複雑な機能になっていました。

しかし、モジュールを使えばより整理された形で、グローバルな変数の宣言が可能になります。

2) INCLUDE 文の代わり

INCLUDE 文は、 PARAMETER 文などでプログラムの一部の変更して使うような機能として便利な機能として使われてきました。

ただし、これは FORTRAN77 の規格にはなくメーカの独自の拡張仕様として提供されてきたもので、今回 Fortran90 の規格として新たに採入れられました。

しかし、同じ機能がモジュールにより整理された形で利用できるので、 INCLUDE 文を使わない方が良いといわれています。

3) モジュールライブラリの定義

プログラム単位の説明で、モジュール手続きがあることを述べましたが、 Fortran90 では、このモジュール手続きを使って言語の機能を個々の利用者が拡張できるようなことも可能になっています。

2.6　グローバルな変数の宣言

図 4に示すプログラムは、モジュール **global** で変数 a,b を定義し、 b には初期値を与えています。COMMON とは異なり初期値を設定することに制約はありません。 **main** と **sub01** は **USE 文**で **global** を引用し、それぞれ使っています。

```
MODULE global
  IMPLICIT NONE
  REAL(4) :: a,b=1.0
END MODULE global

PROGRAM main
  USE global
  IMPLICIT NONE
  REAL(4) :: c=1.0
  a=1.0;
  CALL sub01
  CALL sub02(c)
  WRITE(*,*) a,b,c
END PROGRAM main

SUBROUTINE sub01
  USE global
  IMPLICIT NONE
  a=a+b
END SUBROUTINE sub01

SUBROUTINE sub02(b)
  IMPLICIT NONE
  REAL(4), INTENT(INOUT) :: b
  b=b*100.0
END SUBROUTINE sub02
```

図 4. グローバルな変数

sub02 は引数としてデータを渡しています。変数 b は、モジュール **global** を引用していないので、**sub02** のローカルな変数として扱われます。

2.7 INCLUDE 文の代わり

図 5に示すプログラムは、INCLUDE 文の代わりにモジュール **parameter_module** を使った簡単な例です。

このプログラムは、モジュール内で **PARAMETER** 属性を使って、定数 pi および配列の大きさを規定しています。

```
MODULE parameter_module
  IMPLICIT NONE
  REAL(4),PARAMETER    :: pi=3.1415927
  INTEGER(4),PARAMETER :: array_size=100
END MODULE parameter_module

PROGRAM main
  USE parameter_module
  IMPLICIT NONE
  REAL(4),DIMENSION(array_size) :: array
  array=pi
  write(*,*) array(array_size:array_size)
END PROGRAM main
```

図 5. パラメータモジュールの例

2.8 モジュール手続

図 6に示すプログラムは、構造型を使ったモジュール手続きの例です。モジュール **point_module** において **CONTAINS 文**の後に書いている関数 **addpoints** をモジュール手続きといいます。

複数のモジュール手続きを **CONTAINS 文**の後に置くこともできます。また、モジュール手続き内に **CONTAINS 文**を置き内部手続きを書くことも可能です。

この例は **point** という構造型を宣言し、その演算のための手続きをモジュール手続きとしてまとめて定義しておき、 **main** で引用して使う例です。

構造型を使う場合、構造型に対する演算をこのプログラム例のようにモジュール手続きとして同じモジュール内に定義することで、非常に管理しやすいものとなります。

```
MODULE point_module
  IMPLICIT NONE
  TYPE point
    REAL :: x,y
  END TYPE point

  CONTAINS
    FUNCTION addpoints(p,q)
      TYPE(point),INTENT(IN) :: p,q
      TYPE(point) :: addpoints
      addpoints%x=p%x+q%x
      addpoints%y=p%y+q%y
    END FUNCTION addpoints
END MODULE point_module

PROGRAM main
  USE point_module
  IMPLICIT NONE
  TYPE(point) :: pp,pq,pr
  pp=point(1.0,1.0)
  pq=point(2.0,3.0)
  pr=addpoints(pp,pq)
  WRITE(*,*) pr
END PROGRAM main
```

図 6. モジュール手続き

2.9 総称名手続きの定義

FORTRAN77 では、組込み関数において総称名で使えばコンパイル時に引数のデータ型にあわせた個別名の組込み関数に置き換えてくれる総称名呼出し機能が提供されています。

すなわち、平方根を求める式をプログラム上で

sqrt(1.0d0) と書いた場合に、コンパイラが倍精度の **dsqrt(1.0d0)** に置き換えるといった話です。

Fortran90 からは、これがより一般的な機能として利用者が総称名手続きを定義できるようになっています。

図 7に示すプログラムが総称名手続き **swap** の定義例です。

```
MODULE generic_swap
  IMPLICIT NONE
  INTERFACE swap
    MODULE PROCEDURE integer_swap, &
     real_swap
  END INTERFACE

  CONTAINS
    SUBROUTINE integer_swap(a,b)
      IMPLICIT NONE
      INTEGER,INTENT(INOUT) :: a,b
      INTEGER :: tmp
      tmp=a; a=b; b=tmp
    END SUBROUTINE integer_swap
    SUBROUTINE real_swap(a,b)
      IMPLICIT NONE
      REAL,INTENT(INOUT) :: a,b
      REAL :: tmp
      tmp=a; a=b; b=tmp
    END SUBROUTINE real_swap
END MODULE generic_swap

PROGRAM main
  USE generic_swap
  IMPLICIT NONE
  INTEGER :: a_int,b_int
  REAL :: a_real,b_real
            :
  CALL swap(a_int, b_int)
  CALL swap(a_real, b_real)
            :
END PROGRAM main
```

図 7. 総称名関数の定義

モジュール内の次の部分は総称名引用仕様と呼ばれるものです。

```
  INTERFACE swap
    MODULE PROCEDURE integer_swap, &
     real_swap
  END INTERFACE
```

この中で、総称名 **swap** で呼ばれる個々の手続きは、モジュール手続きとして定義され、それぞれ **integer_swap** と **real_swap** という個別名を持つことを宣言しています。

一方、プログラム **main** では、 **call swap** という総称名で呼出すことでコンパイラが引数のデータ型にあったモジュール手続きの呼出しに置き換えてくれます。

2.10 利用者定義演算子

総称名手続きの定義と同じような形式ですが、四則演算子 (+,-,/,*) を用いて構造型データの演算を利用者が定義することができます。

図 8は、図 6のプログラムを利用者定義演算子を用いて書き直したものです。

```
MODULE point_module
  IMPLICIT NONE
  TYPE point
    REAL :: x,y
  END TYPE point
  INTERFACE OPERATOR(+)
    MODULE PROCEDURE addpoints
  END INTERFACE

  CONTAINS
    FUNCTION addpoints(p,q)
      TYPE(point),INTENT(IN) :: p,q
      TYPE(point) :: addpoints
      addpoints%x=p%x+q%x
      addpoints%y=p%y+q%y
    END FUNCTION addpoints
END MODULE point_module

PROGRAM main
  USE point_module
  IMPLICIT NONE
  TYPE(point) :: pp,pq,pr
  pp=point(1.0,1.0)
  pq=point(2.0,3.0)
  pr=pp+pq
  WRITE(*,*) pr
END PROGRAM main
```

図 8. 利用者演算子定義 (その一)

利用者定義演算子のための引用仕様定義の一般形は、次のような形式です。

```
INTERFACE OPERATOR(operator_symbol)
  interface body
END INTERFACE
```

ここで、 **operator_symbol** には四則演算子 (+,−,/,∗) および (**.sum.**) のようにアンダーラインを含まない英字をピリオドではさんだものが指定できます。

プログラム **main** の **pp+pq** の演算子 (+) は、コンパイラによりモジュール手続き **addpoints**

に置き換えられて処理されます。

図 6よりも図 8の方がより洗練されたプログラムとなっていることは、ご理解頂けることと思います。

図 9は、文字型データについて文字の連結を演算子（＋）で置き換えたプログラムの例です。

```
MODULE chara_module
  IMPLICIT NONE
  INTERFACE OPERATOR(+)
    MODULE PROCEDURE add_chara
  END INTERFACE

  CONTAINS
    FUNCTION add_chara(p,q)
      CHARACTER(*),INTENT(IN) :: p,q
      CHARACTER(LEN=LEN(p)+LEN(q)) &
                  :: add_chara
      add_chara=p//q
    END FUNCTION add_chara
END MODULE chara_module

PROGRAM main
  USE chara_module
  IMPLICIT NONE
  CHARACTER(LEN=100) :: cr
  cr="aa"+"bbb"
  WRITE(*,*) cr
END PROGRAM main
```

図 9. 利用者演算子定義 (その二)

なお、組込みデータ型について利用者定義演算子を用いて演算子の意味を変えてしまうようなことは、当然制限されていますので、あしからず。

2.11 利用者定義代入

これも利用者定義演算子と同じもので、自分で定義した構造型データに対する代入文を（＝）で表すためのものです。

引用仕様定義は、次のように表します。

```
INTERFACE ASSIGNMENT(=)
  interface body
END INTERFACE
```

図 10は、利用者定義代入のプログラム例です。

このプログラム例には構造型の 2 つの成分 **(x,y)** に同じ値を代入する **point_assign_real** という手続きを定義しています。

プログラム中の **p_point=3.0** の（＝）は、コンパイラによりモジュール手続き **point_assign_real**

```
MODULE point_assign_module
  IMPLICIT NONE
  TYPE point
     REAL :: x,y
  END TYPE point
  INTERFACE ASSIGNMENT(=)
     MODULE PROCEDURE point_assign_real
  END INTERFACE

  CONTAINS
    SUBROUTINE point_assign_real(p,a)
      TYPE(point),INTENT(OUT) :: p
      REAL,INTENT(IN) :: a
      p%x=a;  p%y=a
    END SUBROUTINE point_assign_real
END MODULE point_assign_module

PROGRAM point_assign
  USE point_assign_module
  IMPLICIT NONE
  TYPE(point) :: p_point
  p_point=3.0
  WRITE(*,*) p_point
END PROGRAM point_assign
```

図 10. 利用者定義代入

とに置き換えられて処理されます。

3 モジュールの引用と参照属性

3.1 PUBLIC と PRIVATE 属性

これまでの説明でお気づきのように、モジュール内で宣言された変数やモジュール手続きは、引用した手続き内で有効域を持ちます。

グローバルな変数宣言のためのモジュールであれば、これで良いのですが、例えば、総称名手続き宣言のためのモジュールで、総称名以外に個別名が見えることは、あまり良いことではありません。

すなわち、モジュールライブラリとして作成した場合、必要な情報だけを公開し、他を隠してしまった方が保守などを考えると大変便利です。

これを管理するものとして、次の二つの属性があります。

- **PUBLIC 属性**　参照可能
- **PRIVATE 属性**　参照不可能

モジュール内で宣言されたものは、省略値として **PUBLIC 属性**を持ちます。

これらの属性の具体的な使い方として、図 7のプログラムに対し **PUBLIC,PRIVATE 属性**を明示したものを図 11に示します。

```
MODULE generic_swap
  IMPLICIT NONE
  PRIVATE
  PUBLIC swap
  INTERFACE swap
     MODULE PROCEDURE integer_swap, &
     real_swap
  END INTERFACE

  CONTAINS
    SUBROUTINE integer_swap(a,b)
      IMPLICIT NONE
      INTEGER,INTENT(INOUT) :: a,b
      INTEGER :: tmp
      tmp=a; a=b; b=tmp
    END SUBROUTINE integer_swap
    SUBROUTINE real_swap(a,b)
      IMPLICIT NONE
      REAL,INTENT(INOUT) :: a,b
      REAL :: tmp
      tmp=a; a=b; b=tmp
    END SUBROUTINE real_swap
END MODULE generic_swap

PROGRAM main
  USE generic_swap
  IMPLICIT NONE
  INTEGER :: a_int,b_int
  REAL :: a_real,b_real
            :
  CALL swap(a_int, b_int)
  CALL swap(a_real, b_real)
            :
END PROGRAM main
```

図 11. PUBLIC と PRIVATE 属性

モジュール **generic_swap** の何も指定していない **PRIVATE 文**がモジュール内で定義されるものすべてに対して参照不可能属性を与えます。次の **PUBLIC 文**で指定された **swap** だけが参照可能属性を持ちます。

3.2 USE 文での指定

モジュール内で **PUBLIC**、 **PRIVATE** 属性を指定する代わりに **USE 文**で特定の定義だけを参照したり、名前の衝突を避けるために別名をつけて参照するといった指定が可能です。

具体的には、次のような形式で指定します。

- モジュール (**module_name**) 内で定義されているもののうち、ある要素 (**access_name**) だけを参照する。

```
USE module_name,ONLY: access_name
```

- モジュール (**module_name**) 内で定義されているもののうち、ある要素 (**access_name**) を別名 (**local_name**) で参照する。

```
USE module_name,  &
      local_name => access_name
```

図 12に **USE 文**の使用例を示します。

```
MODULE ex_module
  IMPLICIT NONE
  REAL,PRIVATE :: a
  REAL,PUBLIC :: b
  INTEGER :: c
END MODULE ex_module

SUBROUTINE sub_1
  USE ex_module
  REAL :: a
      :
END SUBROUTINE sub_1

SUBROUTINE sub_2
  USE ex_module, ONLY : c
  REAL :: a,b
      :
END SUBROUTINE sub_2

SUBROUTINE sub_3
  USE ex_module, local_c=>c
  REAL :: a,c
      :
END SUBROUTINE sub_3
```

図 12. USE 文の使用例

ここで、 a は参照不可能、 b は 参照可能、 c は **PRIVATE,PUBLIC 属性**を指定していないので **PUBLIC 属性**を持ち参照可能となっています。

sub_1 内の **USE 文**では、 b と c を参照することができます。 **sub_2** 内の **ONLY 句**を使った **USE 文**では、 c だけを参照することを指定します。 **sub_3** 内の **USE 文**では、 b と c を参照することができますが、 **c は local_c** という別名で参照することを指定します。

4 おわりに

今回はモジュールについて紹介しました。プログラムの例題を中心に説明したので解かりにくいかも知れませんが、今回あげた例題を実際にコンパイル、実行して頂くことでモジュールに対して少しでも理解を深めて頂ければとても幸いです。

次回の予定はFortran90の文字列処理、入出力について紹介します。

この連載の読者から参考文献に関する問い合わせをメールで頂き、とっても感激すると共にこれまでに参考文献について触れなかったことを深く反省しております。

筆者の回りにある文献すべてをISBNも明記して整理しましたので、活用くださいませ。

参考文献

- 財団法人 日本規格協会:JIS 規格 X3001 -1994 プログラム言語 Fortran(1994)
- M.Metcalf, J.Reid 著 西村 他訳: bit 別冊 詳解 Fortran90, 共立出版 (1993)

  雑誌:07608-12
- 竹澤 照 著: FortranI 基礎, 共立出版 (1995)

  ISBN:4-320-02731-0
- 竹澤 照 著: FortranII 数値計算, 共立出版 (1997)

  ISBN:4-320-02868-6
- 東田 他 著: 入門 Fortran90 実践プログラミング, ソフトバンク (1994)

  ISBN:4-89052-493-2
- Jim Kerrigan: Migrating to Fortran90,O'Reilly & Associates(1993)

  ISBN:1-56592-049-x
- T.M.R.Ellis,Ivor R.Philips,Thomas M.Lahey: Fortran90 Programing,Addison-Wesley(1994)

  ISBN:0-201-54446-6
- Walter S.Brainerd,Charles H.Goldberg,Jeanne C.Adams: Programmer's Guide to Fortran90,Intertext(1996)

  ISBN:0387945709