Chapter Six
Derived Types and Modules IN FORTRAN 90

الأنواع المشتقة Derived Types

لقد مر معنا أنواع المعطيات في لغة FORTRAN (صحيحة ،حقيقية ، منطقية ، مركبة ورمزية) وتجيز لغة FORTRAN 90 أضافة أنواع جديدة يجري تعريفها من قبل المبرمج وهذه الأنواع يجب أن تكون من نوع تركيبي أو بنيوي structured وبالمقابل فأن التركيب يجب أن يكون من ضمن أنواع جديدة أشتقها المبرمج .

تجري عملية تعريف نوع جديد بأستخدام كلمة Type في البداية وكلمة End Type في النهاية وما بين هاتين الكلمتين ينحصر تعريف مكونات التركيب التي تجري بطريقة التعريف العادية .

والشكل العام للتعريف هو

```
TYPE [ :: ] type-name
component-definitions
END TYPE [ type-name ]
```

وللتصريح بمتغيرات تحمل نفس النوع نستخدم التعبير **TYPE (*type-name*) :: *var1, var2, ...***

لنفترض أننا نريد تعريف تركيب بأسمLine ومكونات هذا التركيب هي Line_Number و Text المكون الأول من النوع الصحيح والمكون الثاني من نوع رمزي تجري عملية تعريف هذا التركيب كما يلي :

```
TYPE LINE
INTEGER :: LINE_NUMBER
CHARACTER (LEN= LINE_LENGTH) :: TEXT
END TYPE LINE
```

ولتعريف متغير من نوع تركيبي من نوع LINE وليكن أسم هذا المتغير NEW_LINE نكتب :

```
TYPE (LINE)::NEW_LINE
```

ولتعريف متغيرBASIC_PROGRAMكمصفوفة عناصرها من نوع التركيبLINE ، يجري التعريف كما يلي :

```
TYPE (LINE) , DIMENSION (MAX_LINES) :: BASIC_PROGRAM
```

أن الطريق البديل لهذا التعريف ، هو أن تعرفBASIC_PROGRAMكمصفوفتين أحداهما من نوع صحيح لتمثيل رقم السطر والأخرى من نوع رمزي يمثل جمل البرنامج .

لو أستخدمنا الطريقة البديلة ، لأحتجنا عند رغبتنا بطباعة سطر من البرنامج لاستخدام جملة كهذه :

```
PRINT "(I5,X,A)" , LINE_NUMBER(L) , TEXT(L)
```

أما أذا أستخدمنا التركيب فإن طريقة الطباعة تكون : **PRINT "(I5,X,A)" , LINE(L)**

وهي طريقة أنسب من الأولى .

وكمثال آخر لتعريف أي شخص يشمل التعريف (الأسم والعمر ورقم الهوية) بالشكل التالي :

```
TYPE  PERSON
CHARACTER ( LEN = 10 ) :: NAME
REAL :: AGE
INTEGER :: ID
END TYPE PERSON
```

ولتعريف المتغيرين ALI , AHMED نكتب : TYPE (Person) :: ALI , AHMED

ولتعريف مصفوفة تسمى people تحوي على 10 أشخاص بنفس التعريف السابق نكتب :

```
TYPE ( Person ), DIMENSION ( 10 ) :: people
```

ولتعريف سجل لأشخاص يحوي أرقام تلفوناتهم ، بحيث يحتوي الرقم على رمز المدينة ورقم التلفون ، وكذلك يحتوي سجل كل شخص على أسم الدولة والمدينة وكذلك الرمز البريدي .

أن أفضل طريقة لهذا التعريف أن نأخذ تعريف رقم التلفون وتعريف العنوان كلاً على حدة وأن نعرف كلاً منها كتركيب لوحده كما يلي :

```
TYPE PHONE_TYPE
INTEGER :: AREA_CODE , NUMBER
END TYPE PHONE_TYPE

TYPE ADDRESS_TYPE
INTEGER :: NUMBER
CHARACTER (LEN= 30) :: STREET , CITY
CHARACTER (LEN= 20) :: STATE
END TYPE ADDRESS _TYPE
```

ومن ثم دمج هذين التعريفين في تعريف واحد بالإضافة الى بعض المكونات الأخرى كما يلي :

```
TYPE PERSON_TYPE
CHARACTER (LEN= 40) :: NAME
    TYPE (ADDRESS_TYPE) :: ADDRESS
    TYPE (PHONE_TYPE) :: PHONE
CHARACTER (LEN= 100) :: REMARKS
END TYPE PERSON_TYPE
```

وأخيراً يمكن تعريف المتغير AHMED كما يلي :

```
TYPE (PERSON_TYPE) :: AHMED
```

المقطوعات MODULES

المقطوعة هي مجموعة من المتغيرات والبرامج الفرعية ، ويستطيع أي برنامج رئيسي أو فرعي أن يرجع اليها حسب حاجته ، وبذلك تكون هذه المتغيرات والبرامج الفرعية عامة لأي برنامج فرعي ، والشكل العام للمقطوعة هو :

```
MODULE module-name
IMPLICIT NONE
[specification part]
CONTAINS
[internal functions/subroutines]
END MODULE module-name
```

main program (one instance only)	module	external subprogram
[PROGRAM *program_name*] [*specification_statements*] [*executable_statements*] [CONTAINS *internal-subprogram(s)*] END [PROGRAM [*program_name*]]	MODULE *module_name* [*specification_statements*] [CONTAINS *module subprogram(s)*] END [MODULE [*module_name*]]	(external) procedure(s)

ملاحظات :

١- أن المقطوعة MODULE تشبه البرنامج الرئيسي الا أنه لا تحتوي على عبارات تنفيذية .

٢- أن التعريف بالمتغيرات داخل المقطوعة ووجود البرامج الفرعية الداخلية هو أختياري.

٣- يمكنللمقطوعاتMODULESأن تحتوي في داخلها على برامج فرعية داخلية أو لا تحتوي .

٤- لاستعمال المقطوعة داخل البرنامج الرئيسي نستخدم الأمر USE بعد العبارة الأولى في البرنامج الرئيسي .

ومثال على MODULE لا يحتوي على برامج فرعية هو MODULE CONSTANTS:

```
MODULE CONSTANTS
IMPLICIT NONE
REAL, PARAMETER :: PI=3.1415926
REAL, PARAMETER :: g = 980
INTEGER :: Counter
END MODULE CONSTANTS
```

مثال على MODULE لا يحتوي على جزء التعريف بالمتغيرات :

```
MODULE SUMAVERAGE
CONTAINS
REAL FUNCTION Sum(a, b,c)
IMPLICIT NONE
REAL, INTENT(IN) :: a, b, c
Sum = a + b + c
END FUNCTION Sum
REAL FUNCTION Average(a, b, c)
IMPLICIT NONE
REAL, INTENT(IN) :: a, b, c
Average = Sum(a,b,c)/3.0
END FUNCTION Average
END MODULE SumAverage
```

وكمثال على MODULE يحتوي على جزء التعريف بالمتغيرات وبرامج فرعية داخلية وكتابة برنامج رئيسي :

```
PROGRAM MAIN
USE MyUtils
IMPLICIT NONE
REAL :: A, B
READ*, A
B = Pi
CALL SWAP( A, B )
PRINT*, A, B
END PROGRAM MAIN
```

Whilethe module(again saved in a separatefile)is

```
MODULE MyUtils
IMPLICIT NONE
REAL, PARAMETER :: Pi = 3.1415927
CONTAINS
SUBROUTINE SWAP( X, Y )
REAL Temp, X, Y
Temp = X
X = Y
Y = Temp
END SUBROUTINE SWAP
END MODULE MyUtils
```

لنفترض أن لدينا قائمة بأرقام بطاقات ضائعة LOSTCARD وهناك متغير يحمل عدد هذه البطاقات NUMBER_OF_LOST_CARDS، بإمكاننا الآن تعريف المقطوعة التالية :

```
MODULE LOST_CARD_MODULE
   INTEGER , DIMENSION (:) , ALLOCATABLE :: LOST_CARD
   INTEGER :: NUMBER_OF_LOST_CARDS
END MODULE LOST_CARD_MODULE
```

من الملاحظ أن المقطوعة محصورة بين الثنائية MODULE و END MODULE وأن هذه المقطوعة تحمل الاسم LOST_CARD_MODULE.

ويمكن لروتين مثل SEARCH أن يستعمل هذه المتغيرات وذلك بأستخدام جملة التعريف USE كما يلي :

```
SUBROUTINE  SEARCH (CARD_NUMBER , FOUND)
  USE LOST_CARD_MODULE
   …..
  DO I = 1 , NUMBER_OF_LOST_CARDS
   IF (CARD_NUMBER == LOST_CARD(I) ) THEN
   …..
```

وأذا استخدم أي روتين آخر نفس المقطوعة ، فإنه يشترك مع هذا الروتين باستخدام المتغيرات الموجودة في هذه المقطوعة ، وبذلك يمكن أحد الروتينات أن يعطي قيماً لهذه المتغيرات ويقوم روتين آخر بالاستفادة من هذه القيم .

ملاحظة :هناك شكلين لأستخدام العبارة use في البرنامج الرئيسي هي :

```
USE module-name
USE module-name, ONLY: name_1, name_2, ..., name_n
```

مثال ذلك :

```
MODULE SomeConstants
  IMPLICIT NONE
  REAL, PARAMETER :: PI = 3.1415926
  REAL, PARAMETER :: g = 980
  INTEGER         :: Counter
END MODULE SomeConstants
```

```
PROGRAM Main
  USE SomeConstants
  IMPLICIT NONE
    ..........
END PROGRAM Main
```

```
MODULE DoSomething
  USE SomeConstants, ONLY : g, Counter
  IMPLICIT NONE
CONTAINS
  SUBROUTINE Something(…)
    ……
  END SUBROUTINE Something
END MODULE DoSomething
```

PI is not available

ملاحظة :

يمكن كتابة MODULE بنفس الملف للبرنامج الرئيسي وذلك بكتابة ال MODULE في البداية ثم كتابة البرنامج الرئيسي كما في المثال التالي :

```
MODULE CONSTANTS
IMPLICIT NONE
REAL, PARAMETER :: PI=3.1415926 , e = 2.7182818
END MODULE CONSTANTS

PROGRAM USECONSTANT
USECONSTANTS
IMPLICIT NONE
real :: radius , area
radius = 10.5
AREA = pi * radius **2
print *, "area = " , area
ENDPROGRAM USECONSTANT
```

كما يمكن كتابة كلاً من البرنامج الرئيسي والمقطوعة MODULE في ملف منفصل كما في المثال التالي :

- أطبع module بأسم constmod.f90
- والبرنامج الرئيسي بأسم const.f90
- ثم ترجم MODULE ثم ترجم البرنامج الرئيسي وبعدها أعمل الربط (LINKING) ثم التنفيذ .

```
module constants
implicit none
real, parameter :: pi = 3.1415926536
real, parameter :: e = 2.7182818285
        contains
subroutine show_consts
print*, "pi = ", pi
print*," e = ", e
        end subroutine show_consts
end module constants
```

```
program module_example
use constants
implicit none
real :: twopi
twopi = 2 * pi
call show_consts()
print*, “twopi = “, twopi
end program module_example
```