Fortran

Chapter 7　Subroutine (副程式) and Function

7-1　subroution

主程式：程式碼在程式一開始就自動會去執行。

副程式：不會自動執行自己的程式碼，它需要別人來”呼叫”它後，才會執行屬於自己的程式碼。

The general form of a subroutine is

```
subroutine subroutine_name ( argument_list)
    ……
    (Declaration section)
    ……
    (Execution section)
    ……
retrun
end subroutine_name
```

To call a subroutine, the calling program places a CALL statement in it’s code. The form of a call statement is

```
call subroutine_name ( argument_list)
```

where the order and type of the actual arguments in the argument list must match the order and type of the dummy argunemts declared in the subroutine.

Remark：副程式獨立地擁有屬於自己的變數宣告，若主程式與副程式用了同樣的變數名稱，那它們仍然互不相關的，彼此之間不會有任何的關係。

Example：

```
Program ex0803
Implicit none
integer :: A=1, b=2
call sub1()
write(*,*) ‘In main program:’
write(*, ‘(2(A3,I3))’’ ‘A=’, A, ‘B=’, B
stop
end program ex0803
subroutine sub1()
implicit none
integer :: A=3, B=4
write(*,*) ‘In subroutine sub1:’
write(*, ‘(2(A3,I3))’’ ‘A=’, A, ‘B=’, B
return
end
```

執行結果：

```
In subroutine sub1:
A=3 b=4
In main program:
A=1 b=2
```

Example：Asimple subroutine to calculate the hypctenuse of a right triangle.

```
subtoutine calc_hypotenuse( side_1,    side_2,    hypotenuse )
inplicit none
real, intent(in) :: side_1, side_2
real, intent(out) :: hypotenuse
real :: temp
temp = side_1 ** 2 + side_2 ** 2
hypotenuse = sqrt(temp)
return
end subroutine
```

intent(in)　← 表示這個參數只供傳入使用，在副程式中不能改變它的數值

intent(out)　← 表示這個參數可以在副程式中改變數值

intent(inout)　← 表示這個參數可以在副程式中傳入或傳回數值

A test driver program fot subroutine calc_hyponteuse

```
Program test_hypotenuse
implicit none
real :: S1, S2
real :: hypot
write(*,*) ‘Program to test suubroutine calc_hypotenuse:’
write(*,*) ‘Enter the hength of side 1’
read(*,*) S1
write(*,*) ‘Enter the hength of side 2’
read(*,*) S2
call calc_hypotenuse(S1, S2, hypot)
write (*,10) hypot
10 Format(1X, ‘The length of the hypotenuse is “’, F10.4)
stop
end program test_hypotenuse
```

7-1-1 variable passing in Fortran : The pass-by-reference scheme

Fortran 在傳遞參數時，是傳遞這個變數的記憶體位址

```
Program test
real :: a, b(4)
integer :: next
……
call sub1(a, b, next)
……
end program test

subroutine sub1(x, y, i)
real, intent(out) :: x
real, dimension(4), intent(in) :: y
integer :: i
……
end subroutine
```

Memory address	Main program name	Subroutine name
001	a	x
002	b(1)	y(1)
003	b(2)	y(2)
004	b(3)	y(3)
005	b(4)	y(4)
006	nest	i
007		

Example：Illustrating the effects of a type mismatch when calling a subroutine.

```
Program bad_call
Implicit none
real :: x = 1.0
call bad_argument(x)
end program bad_call
subroutine bad_argument(I)
implicit none
integer ::i
write(*,*) 'I=', i
end subroutine
```

執行結果：I=106535321.6

7-1-2 Passing arrays to subroutines

There are two possible approaches to specify the length of a dummy array in a subroutine

(1) pass the bounds of each dimension of the array to the subroutine as arguments in the subroutine call and to declare the corresponding dummy array to be that length.

Example:

```
Subroutine process1(data1, data2, n, nvals)
integer, intent(in) :: n, nvals
real, intent(in), dimension(n) :: data1
real, intent(out), dimension(n) :: data2
do i = 1, nvals
   data2(i) = 3.0 * data1(i)
end do
return
end subroutine process1
```

(2) Declare the length of each dummy array with an asterisk as an assumed-size dummy array.

Example:

```
Subroutine process2(data1, data2, nvals)
real, intent(in), dimension(*) :: data1
real, intent(out), dimension(*) :: data2
integer, intent(in) :: nvals
do i = 1, nvals
   data2(i) = 3.0 * data1(i)
end do
return
end subroutine process2
```

Not Good. Complier 無法偵測運算時，array 的大小是否超過實際 size.

7-2 save

The values of all local variables and arrays in a procedure become indefined when we exist the procedure.

SAVE: guarantee the local variables and arrays to be saved unchanged between calls to a procedure.

Example:

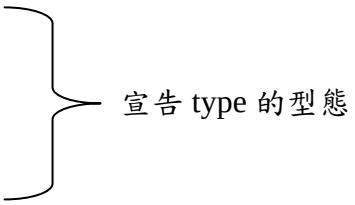
```
Subroutine running_average(x, ave, nvals, reset)
Implicit none
real, intent(in) :: x
real, intent(out) :: ave
integer, intent(out) :: nvals
logical, intent(in) :: reset
! List of local variables:
integer, save :: n
real, save :: sum_x
if (reset) then
   n = 0; sum_x = 0.0; ave = 0.0; nvals = 0
else
   n = n+1
   sum_x = sum_x + x
   ave = sum_x / real(n)
   nvals = n
end if
return
end subroutine running_average
```

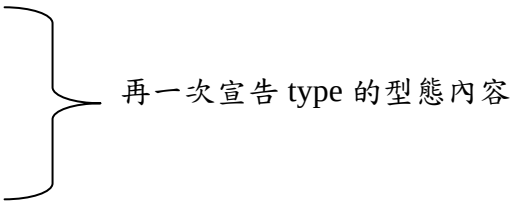
7-3 Sharing data rusing modules

Example:

```
Program main
Implicit none
type :: mytype
……
……
end type mytype
……
……
stop
end program main

subroutine sub1()
Implicit none
type :: mytype
……
……
end type mytype
……
return
end subroutine sub1()
```





主程式與 subroutine 皆需使用 mytype 的資料型態，上述方法較為繁雜，可以使用 module 來簡化之：

```
module typedef
  Implicit none
  type :: mytype
  ……
  end type mytype
end module typefef
program main
use type def
…
stop
end program main
subroutine sub1()
use type def
…
return
end subroutine sub1
```

以 module 來儲存”全域變數”

Example:

```
Module vars
   implicit none
   real, save :: a, b, c
end module vars
```

在程式中，使用上面這個模組的主、副程式，都可以使用到一樣的變數 a, b, c

Example:

```
 Module constants
    implicit none
    real, parameter :: pi=3.14159
    real, parameter :: g=9.81
end module constants
program main
use constants
……
stop
end program main
subroutine sub1()
use constants
……
return
end subroutine sub1
```

7-4 Fortran Functions

Two different types of functions : intrinsic functions and
User_defined functions

Intrinsic functions are built into the Fortran language

e.q. sin(x) and log(x)

The general form of a user_defined Fortran function is

```
Function name (argument_list)
……
(Declaration section must declare type of name)
……
(Execution section)
……
name = expr
return
end function [name]
```


在函數結束之前，記得要把“函數名稱”設定一個數值，這個數值會傳回呼叫處

The type declaration of a user_defined Fortran function can take one of two equivalent forms:

```
integer function my_function (i, j)
```

or

```
function my_function (i, j)
integer :: my_function
```

Example:

A function to evaluate a quadratic polynomial of the form $quad(x) = ax^2 + bx + c$

```
real function quadf(x, a, b, c)
implicit none
real, intent(in) :: x, a, b, c
quadf = a * x ** 2 + b * x + c
return
end function

program test_quadf
implicit none
real :: quadf
real :: a, b, c, x
write(*,*) 'Enter quadratic coefficients a, b and c :'
read(*,*) a, b, c
write(*,*) 'Enter location at which to evaluate equation :'
real(*,*) x
write(*,100) 'quadf(', x, ')=', quadf(x, a, b, c)
100 format(A, F10.4, A, F12.4)
stop
end program test_quadf
```

The function should never modify its own imput arguments.

7-5 Passing user_definited functions as arguments.

Example:

```
program test
real, external :: fun_1, fun_2
real :: x, y, output
……
call evaluate(fun_1, x, y, output)
call evaluate(fun_2, x, y, output)
……
end program test

subroutine evaluate(fun, a, b, result)
real, external :: fun
real, intent(in) :: a, b
real, intent(out) :: result
result = b * fun(a)
return
end subroutine evaluate
```

7-6 Procedure interfaces and interface blocks

Interface between the function/subroutine and a calling program unit

The general form of an interface is

```
interface
    interface_body_1
    interface_body_2
    ……
end interface
```

Each interface_body consists of the initial subroutine or function statement of the corresponding external procedure, the type specification statemts associated with its arguments, and an end subroutine or end function statement.

Example:

```
    Program ex0815
    Implicit none
    real :: angle, speed
    interface
        function get_distance(angle, speed)
            implicit none
            real :: get_distance
            real, intent(in) :: angle, speed
        end function get_distance
    end interface
    write(*,*) 'Input shoot angle:'
    read(*,*) angle
    write(*,*) 'Input shoot speed:'
    read(*,*) speed:
    write(*, '(T2, A4, F7.2, 1A)') 'Fly', get_distance(angle, speed), 'm'
    stop
    end program ex0815

function get_distance(angle, speed)
implicit none
real :: get_distance
real, intent(in) :: speed , angle
real :: rad
real, parameter :: G=9.81

interface
    function angle_to_rad(angle)
        implicit none
        real :: angle_to_rad
        real, intent(in) :: angle
    end function angle_to_rad(angle)
end interface

rad = angle_to_rad(angle)
get_distance = (speed * cos(rad)) * (2.0 * speed * sin(rad) / G)
return
end function get_distance
```

```
function angle_to_rad(angle)
implicit none
real :: angle_to_rad
real, intent(in) :: angle
real, parameter :: pi=3.14159
angle_to_rad = angle * pi / 180.0
return
end function angle_to_rad
```

Fortran 90 的標準並沒有嚴格限制一定要寫作 interface，但是在下面的情況之下，寫作 interface 是必要的：

(i) 指定參數位置來傳遞參數時
(ii) 所呼叫的函式參數數目不固定時
(iii) 傳入指標參數時
(iv) 陣列參數沒有設定大小時
(v) 函數傳回值為陣列時
(vi) 函數傳回值為指標時

7-7 不定個數的參數傳遞

Fortran 90 中，我們可以用 optional 這個敘述來表示某些參數是"可以忽略的"

Example

```
Program ex0817
implicit none
imteger :: a=10, b=>0
interface
  subroutine sub(a, b)
  implicit none
  integer, intent(in) :: a
  integer, intent(in), optional :: b
  end subroutine sub
end interface
write(*,*) 'Call sub with arg a'
call sub(a)
write(*,*) 'Call sub with arg a, b'
call sub(a, b)
stop
end program ex0817

subroutine sub(a, b)
      implicit none
      integer, intent(in) :: a
      integer, intent(in), optional :: b
      write(*,*) a
      if (present(b)) write(*,*) b
      return
end subroutine sub
```

使用 optional 這個敘述來表示後面所宣告的參數可以不一定要傳入

使用函數 present 來檢查參數 b 是否有傳入

Output:

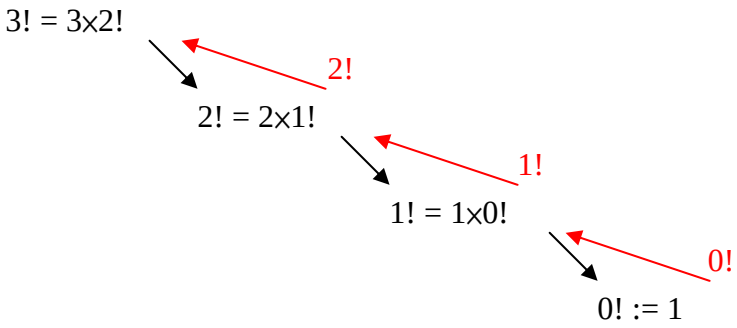
```
Call sub with arg a
      10
Call sub with arg a, b
      10
      20
```

函數 present 可以查看宣告成 optional 的參數是否有傳入，函數 present 的傳回值是邏輯值，如果有傳入查看的參數，就會傳回 .true.，沒有則傳回 .false.

※ 要呼叫這一類不定數目參數的函式時，一定要先宣告出函式的 interface

7-8 Recursive procedures

副程式或是函數自己呼叫自己來執行，叫做”遞迴”



Example:

```
program ex0818
implicit none
integer :: n, ans
interface
  subroutine fact(n, and)            ← function fact(n) result(ans)
  implicit none
  integer, intent(in) :: n
  integer, intent(inout) :: ans
  end subroutine fact
end interface
write(*,*) 'Input N:'
read(*,*) n
call fact(n, ans)                    ← 省略 for function fact
write(*, '(t2, i2, a3, i10)') n, '!=', ans   ← fact(n)
stop
end program ex0818
```

```
recursive subroutine fact(n, ans)
  implicit none
  integer, intent(in) :: n
  integer, intent(inout) :: ans
  integer :: temp
  if (n<0) then
    ans>0
    return
  end if
  if (n>=1) then
    call fact(n-1, temp)
    ans = n * temp
  else
    ans = 1
  end if
  return
  end subroutine fact
```

副程式 fact 的一開頭就以 recursive 來起頭，表示這個副程式可以遞迴地來被自己呼叫

上述副程式可改用以下函數來寫作：

```
recursive function fact(n) result(ans)
implicit none
integer, intent(in) :: n
integer :: ans          ←宣告"ans"變數的型態也就等於宣告函數傳回值的型態
select case(n)
    case(0)
      ans = 1
    case(1)
      ans = n * fact(n-1)   ←改用 ans，而非 fact 來設定函數的傳回值
    case default
      ans = 0
    end select
    return
 end function fact
```

Result 是用來指定一個變數來當成傳回函數值的"替身變數"，e.g. 改成使用"ans"來傳回函數的結果

7-9 Contains statement

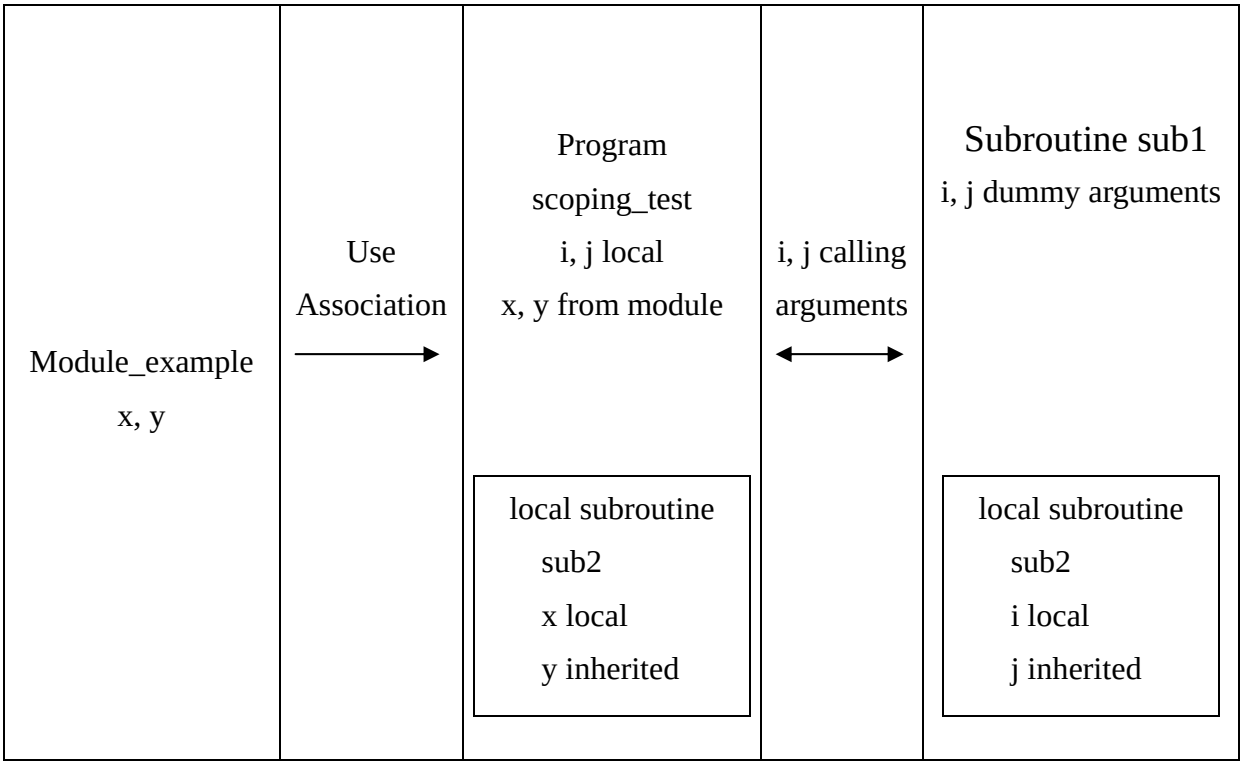
定義某些函式或副程式只能被某個特定的函式(或副程式)、或是只能在主程式中被呼叫。

Example:

```
module module_example
implicit none
real :: x = 100.0
real :: y = 200.0
end module
program scoping_test
use module_example
implicit none
integer :: i = 1, j = 2
write(*, ’(A25, 2I7, 2f7.1)’) ‘Beginning:’, i, j, x, y
call sub1(i, j)
write(*, ’(A25, 2I7, 2f7.1)’) ‘After sub1:’, i, j, x, y
call sub2(i, j)
write(*, ’(A25, 2I7, 2f7.1)’) ‘After sub2:’, i, j, x, y
contains
    subroutine sub2
    real :: x
    x = 1000.0
    y = 2000.0
    write(*, ‘(A25, 2F7.1)’) ‘In sub2:’, x, y
    end subroutine sub2
end program scoping_test
```

← Appears after the last excutable statement in program scoping_test. Only program scoping_test can use this subroutine sub2.

```
subroutine sub1(i, j)
implicit none
integer, intent(inout) :: i, j
integer, dimension(5) :: array
write(*, '(A25, 2I7)') 'In sub1 before sub2 :', i, j
call sub2
write(*, '(A25, 2I7)') 'In sub1 after sub2 :', i, j
array = (/(1000*i, i = 1, 5)/)
write(*, '(A25, 2I7)') 'After array def in sub2 :', i, j, array
contains
    subroutine sub2
    integer :: i
    i = 1000
    j = 2000
   write(*, '(A25, 2I7)') 'In sub1 in sub2 :', i, j
    end subroutine sub2
end subroutine sub1
```



執行結果

```
                Beginning           1           2       100.0       200.0
     In sub1 before sub2:           1           2
        In sub1 in sub2:        1000        2000
     In sub1 after sub2:           1        2000
After array def in sub2:           1        2000        1000        2000        3000        4000        5000
            After sub1:           1        2000       100.0       200.0
               In sub2:      1000.0      2000.0
            After sub2:           1        2000       100.0      2000.0
```

module 中還可以容納副程式，函數的存在，結構如下：

```
module module_name          ← 建立一個新的 module
   use prher_module_name    ← module 中也可以使用別的 module
   implicit none
    integer :: i                ← 宣告告屬於 module 的變數，這些變數可以被
    ……                             module 中的副程式使用
    ……
    type :: type_name           ← 宣告自訂型態，這個型態可以直接被 module
    ……                             中的副程式來使用
    end type :: type_name
contains                        ← 要先加上 contains，再開始寫 module 中的副
   subroutine sub1(a)               程式式或函數
   ……
   end subroutine sub1
   function fun1(b)
   ……
   end function fun1
end module module_name
```

Example:

```
module constants
    implicit none
    real, parameter :: pi = 3.14159
    real, parameter :: g = 9.81
end module constants
module calculate_distance
    use constants
contains
    function argle_to_rad(angle)
        implicit none
        real :: angle_to_rad
        real, intent(in) :: angle
        angle_to_rad = angle * pi / 180.0
        return
    end function argle_to_rad
    function get_distance(speed, angle)
        implicit none
        real :: get_distance
        real, intent(in) :: speed, angle
        real :: rad
        rad = angle_to_rad(angle)
        get_distance = (speed * cos(rad)) * (2.0 * speed * sin(rad) / g)
        return
    end function get_distance
end module calculate_distance
program ex0820
    use calculate_distance
    implicit none
    write(*,*) 'Input shoot angle:'
    read(*,*) angle
    write(*,*) 'Input shoot speed:'
    read(*,*) speed
    write(*, '(T2, A4, F7.2, 1A)') 'Fly', get_distance(angle, speed), 'm'
    stop
end program ex0820
```

7-10 Intrinsic, external

Datatype, external :: Func1, Func2

宣告 Func1 及 Func2 是程式中的函式名稱，而不是變數。

Intrinsic 則是用來宣告某個名詞所指的是庫存的函式。

real, intrinsic :: sin, cos

在實際寫作程式時，這兩個宣告可以省略，不過當我們要把函式名稱當成參數來傳遞到其它函式中時，external 及 intrinsic 就不能省略

Example:

```
program ex0821
implicit none
real :: A = 30.0
real, intrinsic :: sin, cos
real, external :: trig_func
write(*,*) trig_func(sin, A)
write(*,*) trig_func(cos, A)
stop
end program ex0821

function trig_func(func, x)
implicit none
real :: trig_func
real, external :: func
real, intent(in) :: x
trig_func = func(x * 3.14159 / 180.0)
return
end function trig_func
```