Fortran

Chapter 1 Introduction

1-1 Computer Languages

(1) 低階語言：語言的思考方式和電腦完全相同

機械語言

組合語言

(2) 高階語言：思維方式較進乎於人腦

FORTRAN, COBOL, BASIC, C, C++, PASCAL, LISP, JAVA

1-2 The History of Fortran Language

ForTran : Formula Translate

應用於理、工方面的計算

1953：開始發表.

1957：第一個 Fortran compiler 出廠.

1966：美國國家標準局制訂 Fortran 語言的官法標準 ~ Fortran 66.

1979：新的 Fortran 語言標準形成 ~ Fortran 77，較"結構化".

1991：~ Fortran 90，物件導向的觀念及工具，提供指標，加強陣列的功能.

1997：~ Fortran 95，minor update of Fortran 90.

2003：~ Fortran 2003，object-oriented and generic programming.

2010：~ Fortran 2008，was approved in September 2010. As with Fortran 95, this is a minor upgrade, incorporating clarifications and corrections to Fortran 2003, as well as introducing a select few new capabilities.

1-3 Properties of Fortran Language

Paradigm multi-paradigm: imperative (procedural), structured, object-oriented, generic

Stable release Fortran 2008 (ISO/IEC 1539-1:2010) (2010)

Typing discipline strong, static, manifest

Major implementations Absoft, Cray, GFortran, G95, IBM, Intel, Lahey/Fujitsu, Open Watcom, Pathscale, PGI, Silverfrost, Oracle, XL Fortran, Visual Fortran, others

Influenced by Speedcoding

Influenced ALGOL 58, BASIC, C, PL/I, PACT I, MUMPS, Ratfor

Usual file extensions .f, .for, .f90, .f95, .f03

1-4 G95

Free Fortran 95 compliant compiler.

Stable version 0.92, June 2009
http://www.g95.org/downloads.shtml#V0.92

g95 -o hello h1.f90 h2.f90 h3.f90
Compiles multiple source files and links them together to an executable file named hello on unix, or hello.exe on MS Windows systems.

Runtime Error Codes
Running a g95-compiled program with the --g95 option will dump this list of error codes to standard output.

-2 End of record
-1 End of file
0 Successful return
Operating system error codes (1 - 199)
200 Conflicting statement options
201 Bad statement option
202 Missing statement option
203 File already opened in another unit
204 Unattached unit
205 FORMAT error
206 Incorrect ACTION specified
207 Read past ENDFILE record
208 Bad value during read
209 Numeric overflow on read
210 Out of memory
211 Array already allocated
212 Deallocated a bad pointer
214 Corrupt record in unformatted sequential-access file
215 Reading more data than the record size (RECL)
216 Writing more data than the record size (RECL)

Chapter 2 Basic Elements of Fortran

Fortran 90：

(1) “!”後的字元當為註解

(2) 每一行的行尾出現”&”符號時表示下一行和本行是連續的，在行頭出現”&”時，如果上一行的行尾符號也是”&”，表示本行程式碼和上一行接續

(3) 字元集：寫作程式時，可以使用的基本字元及符號

(a) 文字字元：A~Z or a~z (不區分字母的大小寫)

(b) 數字字元：0~9

(c) 特殊字元：+ - * / = ( ) , . “ ‘ & : _

(4) 使用變數名稱之原則：

(a) 變數的名稱以英文字母為原則，可以內含下底線及數字，但不能以數字來起頭。

(b) 變數名字的長度，只有前 31 個字元有效。

(c) 變數的名稱不能和 Fortran 的執行指令同名。

(d) 程式中辨認變數時，不會區分它的大小寫。

(5) data type：

bit：最小儲存單位

一個 bit 只能儲存一個 0 或是 1

1 byte = 8 bit : 位元組

通常電腦資料是以位元組來做為最小的儲存單位

(a) integer 整數

(i) integer(kind=2)

需 16 bits (2 bytes) 儲存，整數介於±32767 之間

(i.e. $2^{15}-1$及$-2^{15}+1$之間)

(ii) integer(kind=4)

需 32 bits (4 bytes) 儲存，整數介於±2147483647 之間

(i.e. $2^{31}-1$及$-2^{31}+1$之間)

(b) real 浮點數

(i) real(kind=4)單精確度

佔 32 bits 的長度，有效位數為 6~8 位

max. value = 3.4×10^{38} min. value = 1.18×10^{-38}

(ii) real(kind=8)單精確度

佔 64 bits 的長度，有效位數為 15~61 位

max. value = 1.79×10^{308} min. value = 2.23×10^{-308}

(c) complex

a+bi 表示之；a，b 為浮點數

(d) character

(e) logical

“True” and “False”

(5) 數學運算式

＋	加法	－	減法
＊	乘法	／	除法
＊＊	次方	（）	括號

優先順序：（）→＊＊→＊／→＋－

Chapter 3　輸出入及宣告

3-1 WRITE, PRINT

Example:

```
program main
write(*,*)  “hello, world!”
stop
end program main
```

程式架構：

程式開始→　program main　←自訂的名稱

主程式碼→　……

……

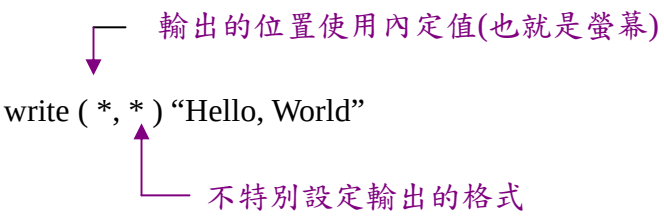
……

程式結束→　stop　←這一行可省略

主程式碼結束→　end

end program

end program main



完整寫法：

```
write (unit = *, FMT = *) “Hello, World”
```

unit = *　與　unit = 6　相同，皆輸出至螢幕

上一行程式亦可改成

```
print *, “Hello, World”
```

3-2 宣告(Declaration)

在程式當中，程式設計師要問電腦的作業系統，要求電腦的記憶體中，預留一個存放程式進行所需要資料的空間。

Example：

```
Program ex0405
implicit none
integer :: A
real :: B
character(len=1) :: C
logical :: D
```

```
A = 1
B = 1.0
C = 'c'
D = .TRUE.
write(*,*) "A=",A, "B=",B, "C=",C, "D=",D
stop
end program ex0405
```

(1) 整數型態：

integer :: A　　宣告一個叫做 A 的整數變數

Example：

```
Program ex0406
implicit none
integer :: A
A = 2+2*4-3
write(*,*) "2+2*4-3=",A
stop
end program ex0406
```

以 2+2*4-3 計算結果來設定變數 A 的數值

Note:

A = 3/2 → A = 1

A = 1/2 → A = 0

} 小數點的部份無條件忽略

integer :: A,B,C → A,B,C 皆為整數變數

integer (kind = 2) :: a → 使用兩個位元組來記錄一個整數

integer (kind = 4) :: b → 使用四個位元組來記錄一個整數

省略(kind)敘述時，會以(kind = 4)為 integer 型態的內定值

使用變數名稱之原則：

(1) 變數的名稱以英文字母為原則，可以內含下底線及數字，但不能以數字來起頭。

(2) 變數名字的長度，只有前 31 個字元有效。

(3) 變數的名稱不能和 Fortran 的執行指令同名。

(4) 程式中辨認變數時，不會區分它的大小寫。

(2) real (浮點數)

real :: a　　a 為單精確度浮點數

real (kind = 4) :: a　　a 為單精確度浮點數

real (kind = 8) :: a　　a 為雙精確度浮點數

電腦在儲存浮點數時，都會先把它轉成以指數來表示的科學符號型式。

e.q.

12345 → 0.123450×10^{5}

12345678 → 0.123457×10^{8}

小數點以後的部分可以儲存 6 位小數

Example:

```
Program ex0409
implicit none
real :: a, b
a = 100000.0
b = 0.0001
write (*,*)   a, ”+”, b, ”=”, a + b
stop
end program ex0409
```

(3) complex

```
complex :: a
complex (kind = 4) :: a
complex (kind = 8) :: a
```

A = (x,y)，其中 x 為實部，y 為虛部

e.q. A = (3.2, 2.5) 表示 A = 3.2 + 2.5i

(4) character

```
character (len = 1) :: a
character (1) :: a
character*1 :: a
```

} a 為一個字元變數

```
character (len = 80) :: a
character (80) :: a
character*80 :: a
```

} a 為一個字元變數，字串長度為 80 個字元

a = “ hello, world ” or a = ‘ hello world ’

(5) logical

```
logical :: A
```

A = .TRUE. 真值

A = .FALSE. 假值

3-3 Read 輸入敘述

Example

```
Program ex0418
integer :: A
real :: B
complex :: C
character (len = 40) :: D
logical :: E
write (*,*) “Please input a (integer) number:”
read (*,*) A
write (*,*) “please input a (real) number:”
read (*,*) B
write (*,*) A, B
stop
end program ex0418
```

輸入的位置使用內定的位置(ie. keyboard)

```
read ( *,   * ) A
```

不規定輸入的格式

嚴謹使用法：

```
read (unit = 5, FMT = *) A
```

unit = * 與 unit = 5 相同，皆指由 keyboard 輸入

Example

```
program ex0419
integer :: A,B,C
read (*,*) A, B, C
write (*,*) A, B, C
stop
end program ex0419
```

同時輸入三個變數

e.q. 1, 2, 3 or 1 2 3

格式化輸出入 (Format) :

Example

```
program ex0420
integer :: A
A = 100
write(*, 100) A
```

≡ write(*,’(I4)’) A

```
100     format( I4 )
        stop
        end program ex0420
```

格式控制敘述

重要格式控制敘述：

Aw ：以 w 個字元寬來輸出字串

Dw.d ：以 w 個字元的寬度來輸出浮點數，小數部分佔 d 個字元寬

Ew.d ：以 w 個字元的寬度來輸出指數型態的浮點數，小數部分佔 d 個字元寬

nX ：把輸出的位置向下跳過 n 個位置

e.q.

write(*,'(1X,A10)') 'hello'

→ □□□□□hello

write(*,'(E(15.7)') 123.45

→ □□0.1234500E+03 w≧d+7

7 位

15 位

Fw.d ：以 w 個字元的寬度來輸出浮點數，小數部分佔 d 個字元寬

write(*, '(1X,F9.3)') 123.45

→ □□123.450

Iw ：以w個字元的寬度來輸出整數

write(*,'(1X,I5)') 100

→ □□100

Lw ：以w個字元的寬度來輸出 T 或 F 的真假值

write(*,'(1X,L5)') .TRUE.

→ □□□□T

Notes:

(1) write(*,'(1X,I3)') 10000

→ ＊＊＊

10000 需 5 個欄位，但只設定 3 個欄焦欄位不足，會輸出＊

(2)

```
        A=10
        write(*,100) A
100     format(1X, 'Ans = ' , I3 )
```

→ Ans = □10

直接放入要寫出的提示字串

(2X,F5.2)重複輸出 3 三次

```
(3)  write(*, '( 3 (2X,F5.2) )' ) A, B, C
```

≡ write(*,’(2X, F5.2, 2X, F5.2, 2X, F5.2)’) A, B, C

(4) write(*, ”(1X, ‘Yor are ’, I4, ‘ years old.’)”) AGE

≡ write(*,100) AGE

100 format(1X,’You are ’, I4 ,’ years old.’)

3-4 宣告的其它補述

(1) implicit

第一個字母為 I,J,K,L,M,N 的變數內定成整數型態，其它的變數則被當成浮點數來使用 →

易發生人為錯誤

implicit integer (A-F, I, K)

把 A 到 F 開頭及 I, K 開頭的變數都當成整數

implicit real (F-K)

把 F 到 K 開頭的變數都當成浮點數

implicit none

把”內定型態”的功能關閉

(2) parameter

Example

```
program ex0428
implicit none
real, parameter :: pi = 3.14159
write(*,”(1X,A10,F5.2)”) ‘sin(pi/6)=’, sin(pi/6.0)
stop
end program ex0428
```

real, parameter :: pi = 3.14159

↑ 不能省略

pi 為浮點常數，其值為 3.14159

pi 不是變數，所以不佔記憶體(直接放在暫存器)，程式執行中不能改變其值

(3) 設定變數的初值

Example

```
Program ex0430
implicit none
integer :: a = 1
real :: b = 1.0
complex :: c = (1.0, 1.0)
character(len=1) :: d = ’A’
```

```
logical :: e = .TRUE.
write (*,'(I3,TR1,F4.2,TR1,(F4.2,TR1,F4.2),A2,L3)') a,b,c,d,e
stop
end program ex0430
```

(4) 宣告在程式中的結構

```
program main
implicit none        ← implicit 要放在宣告的最前端
integer :: A         }
real :: B            }  ← 開始變數宣告
……                   ← 主程式敘述區
stop                 ← 程式結束
end program main
```

Example:

```
Program ex0431
implicit none
integer :: A = 2, B = 1
real :: C
C = B / A            → C = real(B) / real(A)
write(*,*) C         → 0.00000
stop
end program ex0431
```

把整數型態轉換成浮點數型態 ↓

```
A + int(B) / C   ← A, C 為 integer, B 為 real 時
```

↑ 把 real 轉換成 integer

(5) 自訂資料型態

```
type :: person   ← 開始宣告一個叫做 "person" 的資料型態
    character(len=30) :: name
    integer :: age
    integer :: length
    integer :: weight
    character(len=80) :: address
end type person      ← 自訂資料型態結束
```

Example:

```
Program ex0433
implicit none
```

```
type :: person
    character(len=30) :: name
    integer :: age
    integer :: length
    integer :: weight
    character(len=80) :: address
end type person
type(person) :: a
write(*,*) 'Input his(her) name:'
read(*,*) a% name
write(*,*) 'Input his(her) age:'
read(*,*) a% age
write(*,*) 'Input his(her) body length:'
read(*,*) a% length
write(*,*) 'His(her) name is ', a% name
write(*,*) 'His(her) age is ', a% age
stop
end program ex0433
```

```
a = person("Tom",    15,     170,      60,      "Taipei")
    type   a%name   a%age   a%lenth   a%weight   a%address
```

(6) module

Example:

```
module typedef          ← 要自訂一個 module 區塊的名稱
    implicit none
    type :: person
        character(len=30) :: name
        integer :: age
        real :: length
        real :: weight
    end type person
end module typedef
program ex0434
use typedef             ←  要搶在宣告之前先宣告要使用那一個 module 的名字
```

```
implicit none
type(person) :: a          <-------- 使用 typedef 這個 module 中的 person
write(*,*) 'Input his name:'
read(*,'(a30)') a%name
write(*,*) 'Input hes age:'
read(*,*) a%age
      :
write(*,'(1x, a12, a20)') 'Name :' a%name
write(*,'(1x, a12, I6)') 'age :' a%age
      :
stop
end program ex0434
```

Example:

```
module constants
    implicit none
    real, parameter :: pi= 3.14159
    real, parameter :: g = 9.81
end module constants
module vars
    implicit none
    real :: pi_2
end module vars
program ex0435
use constants
use vars          ——
implicit none
write(*,*) 'PI = ', pi
write(*,*) 'G = ', g
pi_2 = 2.0* pi
write(*,*) '2 PI = ', pi_2
stop
end program ex0435
```

(7) kind 的使用 (On PC)

integer(kind=1)　　-127~127

integer(kind=2)　　-32767~32767

integer(kind=4)　　-2147483647~2147483647

real(kind=4)　　1.18×10^{-38}~3.40×10^{38}

real(kind=8)　　2.23×10^{-308} ~ 1.79×10^{308}

增加程式碼”跨平台”能力

selected_int_kind(r)：

　　傳回記錄 r 位整數時，所應宣告的 kind 值。

selected_real_kind(p,r)：

　　傳回記錄 p 位有效位數，指數達到 r 的浮點數所需的 kind 值。

Example

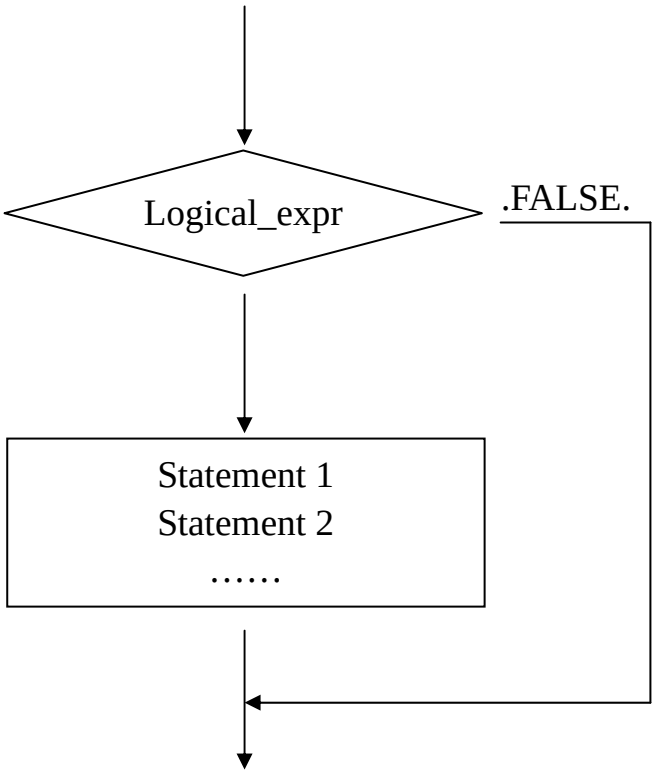
```
 module kind_var
   implicit none
      integer, parameter :: long_int=selected_int_kind(9)
      integer, parameter :: short_int=selected_int_kind(3)
      integer, parameter :: long_real=selected_real_kind(10,50)
      integer, parameter :: short_real=selected_real_kind(3,3)
 end module kind_var
program ex0436
use kind_var
implicit none
                              =4
integer(kind=long_int) :: a=12345678
                              =2
integer(kind=short_int) :: b=12
                           =8
real(kind=long_real) :: c=1.23456789D45
                           =4
real(kind=short_real) :: d=1230
write(*,'(I10)') a
write(*,'(I10)') b
write(*,'(E15.5)') c
write(*,'(E15.5)') d
stop
end program ex0436
```

```
□□12345678
          12
0.12346E+46
0.12300E+04
```

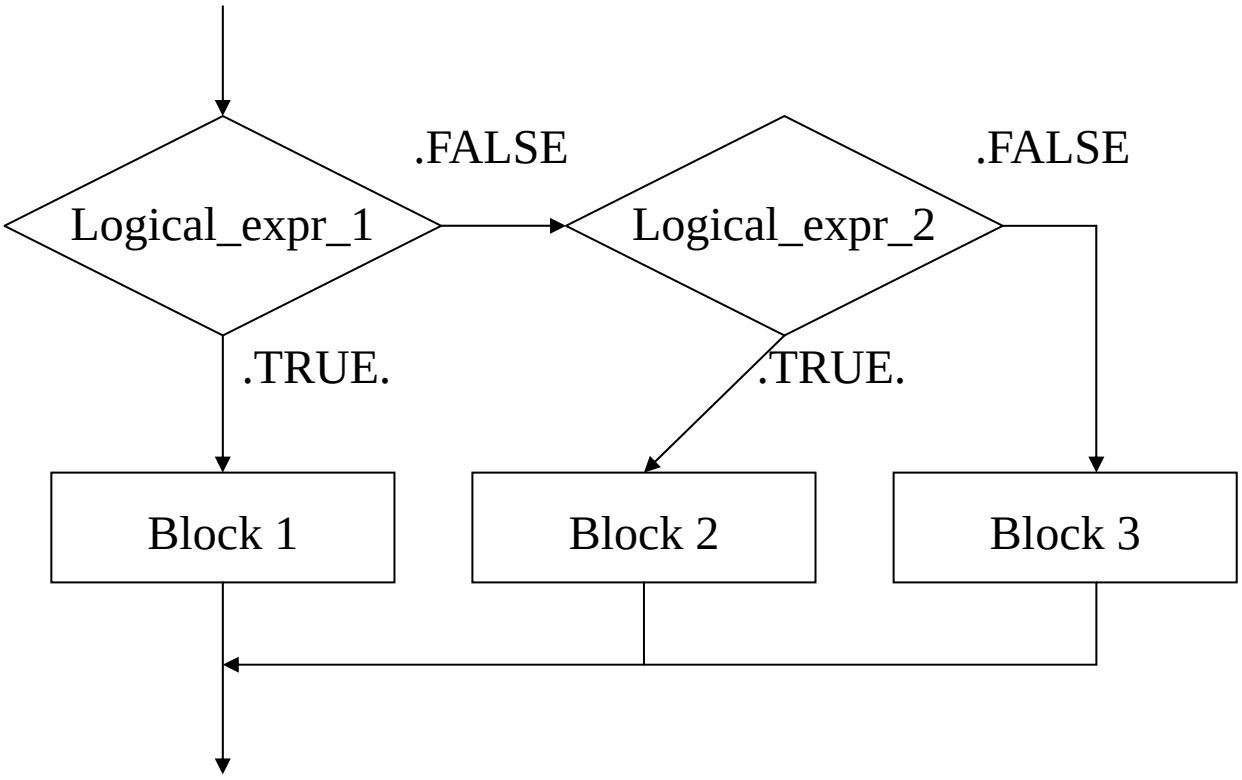
Chapter 4　流程控制

4-1 IF

```
if (logical_expr) then
    statement 1
    statement 2          } block
    ……
end if
```



```
if     (logical_expr) then
       statement 1
       statement 2          } block 1
       ……
else if (logical_expr_2) then
       statement 1
       statement 2          } block 2
       ……
else
       statement 1
       statement 2          } block 3
       ……
end if
```



Example:

```
Program ex0501
Implicit none
Real :: Height, Weight
Real :: Standard_Weight
Write(*,*) ‘Please input your height :’
Read(*,*) Weight
Standard_Weight = Height – 100.0
If (Weight .GT. Stand_Weight) then
      Write(*,*) ‘You are overweighted !’
Else
      Write(*,*) ‘Your weight is under control !’
End if
Stop
End program ex0501
```

邏輯判斷運算

.EQ.	or	= =	判斷是否	”等於”
.NE.	or	/=	判斷是否	”不等於”
.GT.	or	>	判斷是否	”大於’”
.GE.	or	>=	判斷是否	”大於或等於”
.LT.	or	<	判斷是否	”小於”
.LE.	or	<=	判斷是否	”小於或等於”

判斷集合的運算

.AND.
.OR.
.NOT.

Example:

某同學這一次微積分小考拿了 85 分，如果把成績分成 A，B，C，D，E 這 5 個等級，其中 90~100 分為 A 級、80~89 分為 B 級、70~79 分為 C 級、60~69 分為 D 級、60 分以下為 E 級，請寫一個程式來判斷此同學這次成績的等級。

```
Program ex0502
Implicit none
Integer :: Grades
Character(len=1) :: Level = ‘?’
Write(*,*) ‘Please input your Grades:’
Read(*,*) Grades
```

```
If ((Grades .LE. 100) .AND. (Grades .GE. 90)) then
      Level = ‘A’
else If ((Grades .LE. 89) .AND. (Grades .GE. 80)) then
      Level = ‘B’
else If ((Grades .LE. 79) .AND. (Grades .GE. 70)) then
      Level = ‘C’
else If ((Grades .LE. 69) .AND. (Grades .GE. 60)) then
      Level = ‘D’
else if (Grades .LT. 60) then
      Level = ‘E’
else
      write(*,*) ‘Input error’
end if
write(*,*) ‘You get : [‘ ,Level, ’]’
stop
end program ex0502
```

4-2 Select -- case

```
Select csae (變數)
case (數值 1)
      ……          ←變數等於數值 1 時，會執行此區段
case (數值 2)
      ……
case default
      ……          ←變數不等於任何數值時，會執行此區段
end select
```

Example: 上例

```
Program ex0507
Implicit none
Integer :: Grades
Character (len = 1) :: Level
Write (*,*) Grades
Select case (Grades)
Case (90:100)
      Level = ‘A’
Case (80:89)          ←  80 <= Grades <= 89
      Level = ‘B’
Case (70:79)
      Level = ‘C’
Case (60:69)
      Level = ‘D’
Case (:59)            ←  Grades <= 59
```

```
        Level = ‘E’
Case default
        Level = ‘?’
End select
```

Chapter 5　迴圈

5-1 Do

使用時機：連續重複執行某一段程式碼時。

Example:

```
Program ex0601
implicit none
integer :: I
integer, parameter :: N=10
do 10, I = 1, N, 1
    write(*,*) 'Do – Loop Demo'
10  continue
stop
end program ex0601
```

迴圈每次的增量，若增量為 1，則可省略

```
(1) do 10, I = 1, N, 1
```

變數 I 的終止數值

設定變數的初值

指定一個行代碼，從本行起到此行代碼間的程式會變成一個

```
10 continue
```

```
(2) do 10, I = 2, N, 2
```

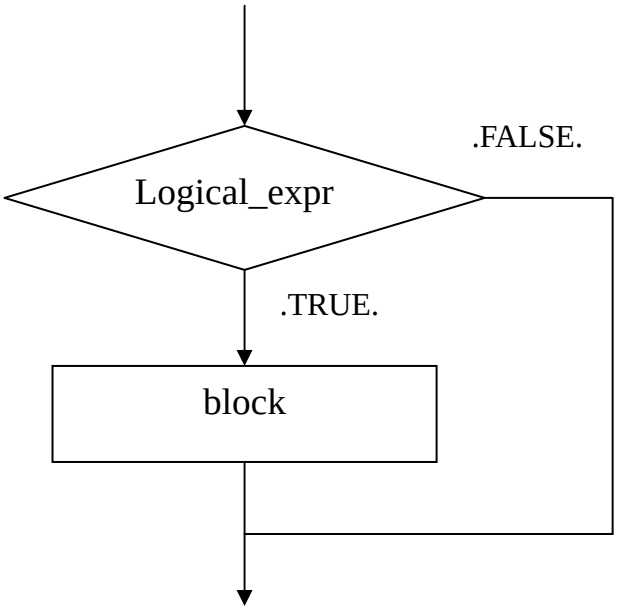
增值設為 2，起始值為 2，=> I = 2, 4, 6, 8, 10

```
(3) do I = 1, N, 1
        write(*,*) 'Do-Loop Demo'
    end do
```

(1) 與(3)相同

5-2 do while

```
do while (logical_expr)
    statement 1
    statement 2
    ……
end do
```



使用時機：不能事先預知會執行幾次的迴圈時。

Example:

```
Program ex0604
    implicit none
    integer, parameter :: Weight=45
    integer :: input = 0
    do while (input /= Weight)
        write(*,*) 'Weight ='
        read(*,*) input
    end do
    write(*,*) 'You are right!'
    stop
end program ex0604
```

5-3 Cycle

cycle：略過迴圈中之後的敘述，直接跳回迴圈的開頭來執行下一次迴圈。

Example:

```
Program ex0605
implicit none
integer :: i
integer, parameter :: n=10
do i = 1, n
    if (i = = 4) cycle
```

```
        write(*,*, “(I3)”) I
    end do
    stop
    end program ex0605
```

5-4 Exit

exit：直接跳出一個在運作中的迴圈

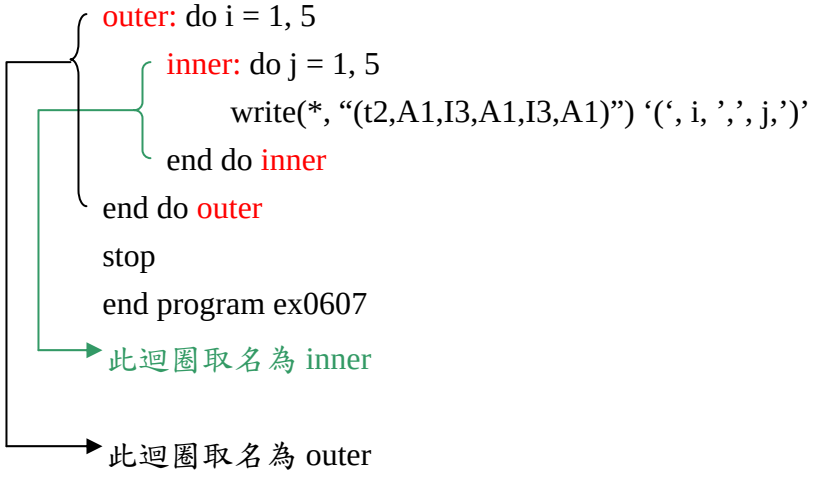
Example:

```
    Program ex0606
    implicit none
    integer, parameter :: Weight=45
    integer :: input
    do while (.true.)        ← 永久迴圈
        write(*,*) “Weight =”
        read(*,*) input
        if (input = = Weight) exit   ← 等式成立，則跳離此迴圈
    end do
    write(*,*) “You are right!”
    stop
    end program ex0606
```

5-5 具名的迴圈

Example:

```
    Program ex0607
    implicit none
    integer :: i, j
    outer: do i = 1, 5
        inner: do j = 1, 5
            write(*, “(t2,A1,I3,A1,I3,A1)”) ‘(‘, i, ’,’, j,’)’
        end do inner
    end do outer
    stop
    end program ex0607
```



5.6 具名迴圈與 cycle，exit 配合使用

Example:

```
    Program ex0608
    implicit none
```

```
integer :: i, j
loop1: do i = 1,3
      loop2:do j = 1,3
            if(j == 2) cycle loop2
            if(i == 2) exit loop1
            write(*,”(t2,A1,I3,A1,I3,A1)”) ‘(‘,i,’,’,j,’)’
      end do loop2
end do loop1
stop
end program ex0608
```

跳到 loop2 的下一次迴圈

跳出整個 loop1 迴圈(也會同時跳出 loop2 迴圈

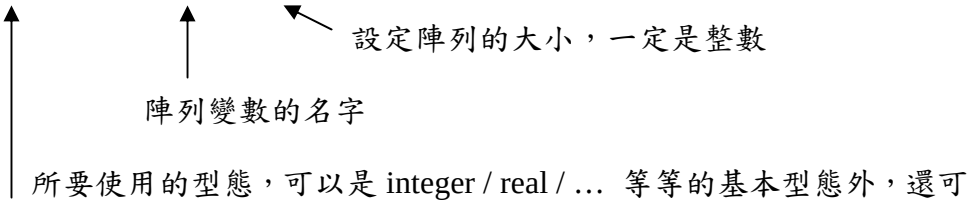
output: (1,1),(1,3)

Chapter 6　陣列

6-1　矩陣的宣告與使用

陣列的宣告法：

DataType :: name (Size)



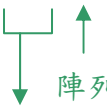
or　DataType, dimension (Size) :: name

Example:

(1) integer :: A(10)　　宣告 10 個元素的整數陣列 A，其元素的取用範圍為 1~10，i.e. A(1),A(2),…,A(10)

(2) integer :: A(-1:8)　　宣告 10 個元素的整數陣列 A，其元素為 A(-1),A(0),A(1),A(2),…,A(10)

(3) integer :: A(0:10: 2)　　宣告 6 個整數元素陣列，其元素為 A(0),A(2),A(4),A(6),A(8),A(10)



(4) integer :: A(10:0:-2)　　宣告 6 個整數元素陣列，其元素為 A(10),A(8),A(6),A(4),A(2),A(0)

(5) integer :: A(3,3)　　宣告一個 3x3 的二維陣列

(6) integer :: A(3,3,3)　　宣告一個 3x3x3 的三維陣列

(7) integer :: A(0:3,0:3)　　宣告一個 4x4 的矩陣

6-2　設定陣列的初值

(1) integer :: a(5) = (/ 1, 2, 3, 4, 5 /)

a(1) = 1, a(2) = 2, a(3) = 3, a(4) = 4, a(5) = 5

※括號和斜號之間不能有空白

(2) integer :: a(5) = (/ 3, i = 1, 5 /)

a(1) = a(2) = a(3) = a(4) = a(5) = 3

integer :: a(5) = (/ i, i = 1, 5 /)

a(1) = 1, a(2) = 2, a(3) = 3, a(4) = 4, a(5) = 5

(3) integer :: a(5) = (/ 0, (i, i = 2, 4), 5 /)

a(1) = 0, a(2) = 2, a(3) = 3, a(4) = 4, a(5) = 5

6-3 陣列在電腦記憶體中的儲存方法

(1) one-dimension

integer :: A(5)

元素在記憶體的連續區塊中的排列情況為

A(1)→A(2)→A(3)→A(4)→A(5)

integer :: A(0:10:2)

元素在記憶體的連續區塊中的排列情況為

A(0)→A(2)→A(4)→A(6)→A(8)→A(10)

(2) two- dimension

integer :: A(3,3)

A(1,1)	A(1,2)	A(1,3)
A(2,1)	A(2,2)	A(2,3)
A(3,1)	A(3,2)	A(3,3)

The first row

The first column

陣列 A 在記憶體的排列情況為

A(1,1) → A(1,2) → A(1,3) ← 先放 first column

→A(2,1) → A(2,2) → A(2,3) ← 再放 second column

→A(3,1) → A(3,2) → A(3,3) ← 其次 third column

~column major 排列法

Example: compute sum = $\sum_{i=1}^{3}\sum_{j=1}^{3}A(i,j)$

(i)
```
sum=0
do i = 1, 3
   do j = 1, 3
        sum = sum + A(i,j)
   end do
end do
```

not good,因為 cpu 要不斷的在記憶中跳躍式的來讀取資料，無法使用到快取記憶體（Cache）的便利（Cache 會自動抓取鄰近近的記憶體資料到 Cache 中）

(ii)
```
sum=0
do i = 1, 3
   do j = 1, 3
```

```
            sum = sum + A(j,i)
          end do
        end do
```

better than (i)，充份利用 cache，程式執行的速度會較快

6-4 Fortran90 新增有關矩陣的功能

6-4-1 整個矩陣的運作

(1) a = b

```
do i = 1, k
   a(i) = b(i)
end do
```

(2) a = b + c

```
do i = 1, k
     a(i) = b(i) + c(i)
end do
```

(3) a = b - c

```
do i = 1, k
      a(i) = b(i) - c(i)
end do
```

(4) a(1:3) = b(4:6)

a(1) = b(4), a(2) = b(5), a(3) = b(6)

(5) a(1:5:2) = 3

a(1) = 3, a(3) = 3, a(5) = 5

(6) a(1:10) = a(10:1:-1)

把 a(1~10)的內容給反轉

(7) a = b * c

```
do i = 1, k
    a(i) = b(i) * c(i)
end do
```

(8) a = b / c

```
do i = 1, k
    a(i) = b(i) / c(i)
end do
```

6-4-2 where, forall

```
where  (矩陣元素值的邏輯運算)
      矩陣運作          ←邏輯為真時，執行這一段程式
else where
      矩陣運作          ←邏輯為假時，執行這一段程式
end where
```

Example:

```
    Program ex0707
  implicit none
    integer :: I
    real :: a(5) = (/ ( I, I = 5, 1, -1) /)    ←a(1)=5,a(2)=4,a(3)=3,a(4)=2,a(5)=1
    real :: b(5) = (/ (I, I = 1, 5) /)         ←b(1)=1,b(2)=2,b(3)=3,b(4)=4,b(5)=5
    real :: c(5)
(*) where   ( a > b )
        c = b
    else where
        c = a
    end where
    write (*,100)   'Array C = ',   ( c(i),   I = 1, 5 )
    100  format ( A10,   5F6.2 )
    stop
    end program ex0707
```

```
(*)≡ do I = 1, k
        if   ( a(i) > b(i) ) then
            c(i) = b(i)
        else
            c(i) = a(i)
        end if
     end do
```

forall ~ Fortran95 所新增的功能

```
(1) forall ( i = 1:2, j = 1:2 )   a(i,j) = i + j
  do j = 1, 2
    do i = 1, 2
      a(i,j) = i + j
    end do
  end do
(2) forall ( i = 1:2,   j = 1:2 )
      a(i,j) = i + j
      a(j,i) = i - j
    end forall
```

如果 forall 中的敘述超過一行，就要使用區塊的型式，最後還要加上 end forall 來做結束

```
(3) forall ( i = 1:10,   j = 1:10,   a(i,j) > 0 )
        where   ( a < 10 )   b = 10
```

```
end forall

do j = 1, 10
    do i = 1, 10
        if   ( a(i,j) > 0 ) then
            if   ( a(i,j) < 10 ) then
                b(i,j) = 10
            end if
        end if
    end do
end do
```

6-4-3 可變大小的陣列

Fortran 90 中的陣列可以等到程式執行中再來決定它所要使用的長度

Example:

```
Program ex0709
implicit none
integer :: students, i
integer :: allocatable :: grades( : )
write(*,*) 'How many strdents in this class?'
read(*,*) students
allocate(grades(students))
do i = 1, students
   write(*, '(1X, A25, I2, A2)') 'Input grades of number', I, ':'
   read(*,*) grades(i)
end do
do i = 1, students
   if (grades(i) < 60) then
      write(*, '(1X, A10, I2, A8)') number', I, 'fail!'
   end if
end do
stop
end program ex0709
```

表示矩陣"可變大小"

說明：

(a) integer, allocatable :: grades(:)

≡ integer, allocatable, dimension(:) :: grades

只有一個冒號表示矩陣只有一個維度，要宣告兩個維度的矩陣則為 grades(:,:)

(b) allocate(grades(students))

設定矩陣大小為 students

```
allocate(grades(m:n:inc))
```

自定矩陣的上下限範圍及增值量(inc)

宣告完成後，要先經由 allocate 這個敘述來設定出陣列的大小後才能使用陣列。

allocate：問電腦的記憶體要求空間來儲存資料

deallocate：將 allocate 所得到的陣列記憶體空間釋放

Example:

```
Program ex0710
implicit none
integer :: students, i
integer :: allocatable :: grades( : )
write(*,*) ‘How many strdents in this class A ?’
read(*,*) students
allocate(grades(students))
do i = 1, students
   write(*, ‘(1X, A25, I2, A2)’) ‘Input grades of number’, I, ‘:’
   read(*,*) grades(i)
end do
do i = 1, students
   if (grades(i) < 60) then
      write(*, ‘(1X, A10, I2, A8)’) number’, I, ‘fail!’
   end if
end do
deallocate(grades)
write(*,*) ‘How many strdents in this class B ?’
read(*,*) students
do i = 1, students
   write(*, ‘(1X, A25, I2, A2)’) ‘Input grades of number’, I, ‘:’
   read(*,*) grades(i)
end do
do i = 1, students
   if (grades(i) < 60) then
      write(*, ‘(1X, A10, I2, A8)’) number’, I, ‘fail!’
   end if
```

```
    end do
    stop
    end program ex0710
```

```
allocate(grades(students), stat = error)
```

Error 是事先宣告好的整數變數，做 allocate 這個動作時會經由 stat 這個敘述傳給 error 一個數值。If error = 0，then allocate 陣列成功。Otherwise，allocate 失敗。電腦的記憶不足

```
deallocate(grades(students), stat = error)
```

6-4-4 陣列的庫存函數

1. dot_product(vector_a, vector_b)

   Calculates the dot product of two equal-sized vectors.

2. matmul(matrix_A, matrix_B)

   Performs matrix muttiplication on to conformable matrices.

3. maxloc(array, mask) ⇔ minloc(array, mask)

   找出陣列中最大值的所在位置，傳回一個整數。Mask 的運算方法和陣列在 where 中所使用的邏輯運算一樣。

   e.q. b = maxloc(a, a<50)

   找出陣列中小於 50 的最大數值的所在位置

4. maxval(array, mask) ⇔ minval(array, mask)

   returns the maximum value in array among those elements for which mask was true.

5. reshape(data_source, shape)

   把資料”整型”好後，再傳給一個陣列，常用在設定陣列初值時使用。

6. sum(array, mask)

   calculates the sum of the elements in array for which the mask is true. If mask is not present, it calculates the sum of all of the elements in the array.

   e.q. integer :: a(3,3) = reshape((/1,2,3,4,5,6,7,8,9/), (/3,3/))

$$a = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

e.q. integer :: b(2,3) = reshape((/1,2,3,4,5,6/), (/2,3/))

$$b = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

7. transpose(matrix)

returns the transpose of matri

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
```

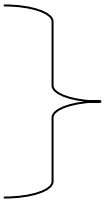
```
    ave = sum_x / real(n)
    nvals = n
  end if
  return
  end subroutine running_average
```

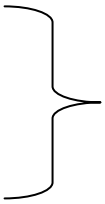
7-3 Sharing data rusing modules

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_defined 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:

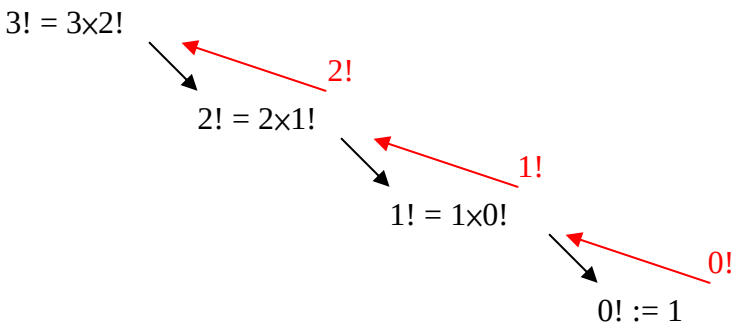
```
Call sub with arg a
      10
Call sub with arg a, b
      10
      20
```

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)  ← 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
  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
```

上述副程式可改用以下函數來寫作：

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

```
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
```

7-9 Contains statement

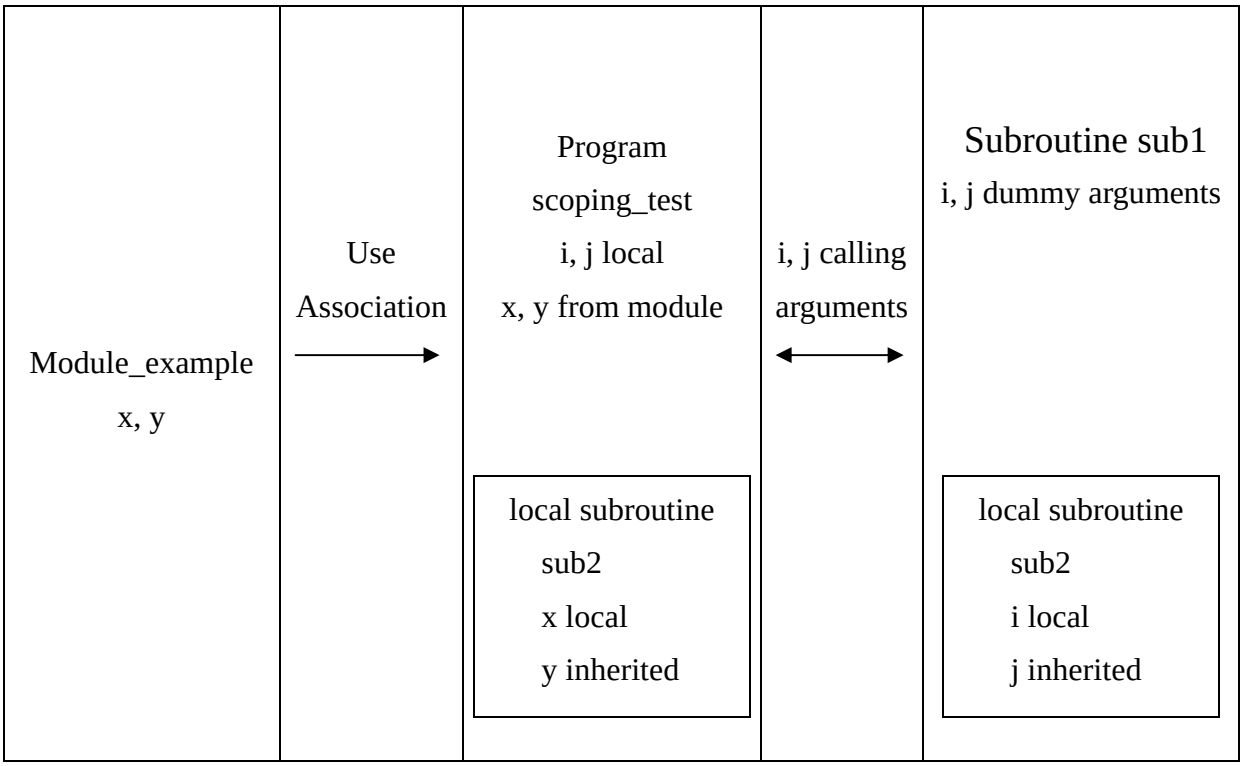
定義某些函式或副程式只能被某個特定的函式(或副程式)、或是只能在主程式中被呼叫。

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                                    ←  Appears after the last excutable statement in
    subroutine sub2                            program scoping_test. Only program
    real :: x                                  scoping_test can use this subroutine sub2.
    x = 1000.0
    y = 2000.0
    write(*, ‘(A25, 2F7.1)’) ‘In sub2:’, x, y
    end subroutine sub2
end program scoping_test

 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
```

Chapter 8 檔案

檔案讀取可分為"循序讀取"及"直接讀取"兩種情形：

(1) 循序讀取：對一個檔案在讀入或者是寫出時，我們只能從頭開始，一步步地向下來一筆一筆地讀取資料。
(2) 直接讀取：對一個檔案在讀入或者是寫出資料時，我們可以任意、直接地跳躍到檔案的任何一個位置上來從事讀取的工作。

檔案儲存模式分為"文字檔"及"二進位檔"

(1) 文字檔：把所有的資料都用我們人眼可以明白理解的字元符號來做儲存。優點：易懂，可直接修改。
(2) 二進位檔：直接把資料在電腦記憶中的儲存情形(也就是二進位碼)直接寫入檔案中。優點：讀取較快速、省空間。

8-1-1 The open statement

OPEN(unit = int_expr, file = char_expr, status = char_expr, action = char_expr, iostat = int_var)

(1) unit = int_expr：開啟一個檔案時要給定這個檔案一個讀取的編號，以後使用 write, read 時使用這個編號就可以對這個檔案來讀寫了。

unit = int_expr 的值最好避開 1, 2, 5, 6。2, 6 是指內定的輸出位置，也就是螢幕。1, 5 則是捏內定的輸入位置，也就是鍵盤。

(2) file = char_expr：用來指定開啟檔案的名稱。

(3) status = char_expr：'New', 'OLD', "Scratch' or 'unknown'用來標示是要開啟一個新檔或是已經存在的舊檔

states = 'New'：這個檔案原本不存在，是第一次開啟
states = 'OLD'：這個檔案原本就已經存在

(4) Action = char_expr : 'read', 'write', 'readwrite'
Action = 'readwrite'： 表示所開啟的檔案可以用來讀取及寫入，這是內定值
Action = 'read'： 表示所開啟的檔案只能用來讀取資料
Action = 'write'： 表示所開啟的檔案只能用來寫入資料

(5) Iostate = int_var： 表示檔案開啟的狀態
int_var > 0 表示讀取動作發生錯誤

int_var = 0 表示讀取動作正常

int_var < 0 檔案終了

(6) Access = ‘sequential’ or ‘direct’

Access = ‘sequential’ 讀取檔案的動作會以”循序”的方法來做讀取

Access = ‘direct’ 讀取檔案的動作可以壬意指定位置

(7) Position = ‘asis’ or ‘rewind’ or ‘append’

Position = ‘asis’ 表示檔案開啟時的讀取位置，不特別指定。(內定值)

Position = ‘rewind’ 檔案開啟時的讀取位置移到檔案的開頭處。

Position = ‘append’ 檔案開啟時的讀取位置移到檔案的結尾處。

Case 1: opening a file for input

```
integer :: ierror
 open (unit = 8, file = ‘EXAMPLE.DAT’, status = ‘OLD’, action = ‘read’, iostat = ierror)
```

Case 2: opening a file for output

```
integer ::n_unit, ierror
character(len = 6) :: filename
n_unit = 25
filename = ‘outdat’
 open (unit = n_unit, file = filename, status = ‘new’, action = ‘write’, iostat= ierror)
```

Case 3: opening a scratch file

```
 open (unit = 12, status = ‘scratch’, iostat= ierror)
```

8-1-2 The close statement

close (close_list)

8-1-3 reads and writes to disk files

(1) open(unit = 8, file = ‘input.dat’, status = ‘old’, iostat = ierror)

read(8, *) x, y, z

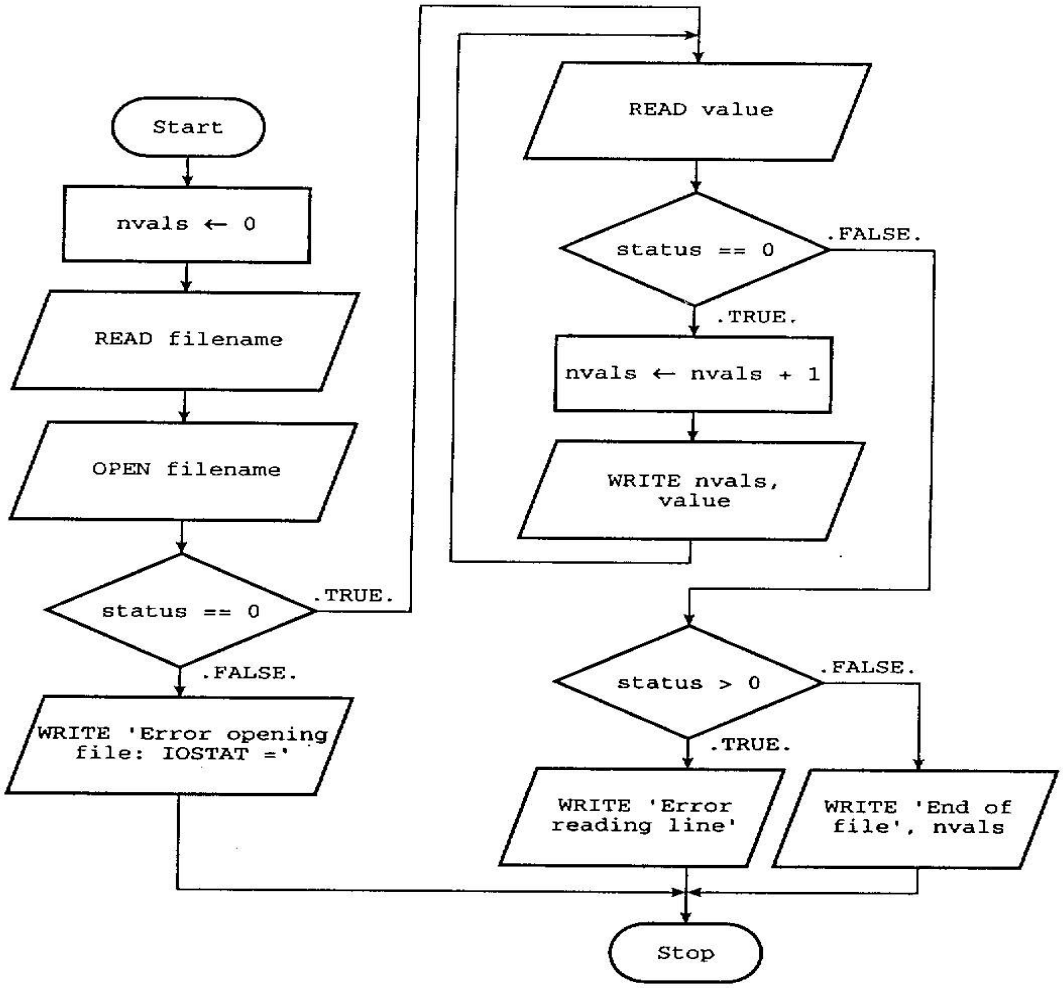
~read the values of variables x, y and z from the file “input.dat”.

(2) open(unit = 9, file = ‘output.dat’, status = ‘new’, iostat = ierror)

```
    read(9, 100) x, y, z
100   Format(‘X = ‘, F10.2, ‘Y=’, F10.2, ‘Z=’, F10.2)
```

~write the values of variables x, y and z to the file output.dat.

Example: Reading data from a file:



Flowchart for a program to read an unknown number of values from an input data file.

```
Program read
      implicit none
      character (len =20) :: filename
      integer :: nvals = 0
      integer :: status
      real :: value

      ! Get the file name and echo it back to the user.
      write(*,*) 'Please enter input file name:'
      read(*,*) filename
      write(*,10) filename
10    format(1x, 'The input file name is : ', A)

      ! Open the file and check for errors on open
      open(3, file = filename, status = 'old', action = 'read', iostat = status)
      openif: If(status == 0) then
        ! open was OK. Read values
        readloop: do
```

```
        read(3, *, iostat = status) value
        if(status /= 0) Exit
        nvals = nvals + 1
        write(*, 20) nvals, value
20      format(1x, 'Line', I6, ': Value = ', F10.4)
    end do readloop

    ! The while loop has terminated. Was it because of a read error or because
    ! of the end of the input file?
    readif: if(status > 0) then
        write(*, 30) nvals + 1
30      format(1x, 'An error occurred reading line', I6)
    else
        write(*, 40) nvals
40      format(1x, 'End of file reached. There were ', I6, 'values in the file.')
    end if readif
    else openif
    write(*, 50) status
50  format(1x, 'Error opening file: IOSTAT = ',I6)
    end if openif
    close(3)
    end program read
    read(3, *, iostat = status) value
```

status > 0 表示讀取動作發生錯誤

status = 0 表示讀取動作正常

status < 0 表示檔案終了

直接存取檔的操作

把檔案的空間、內容事先加以分割成一個個同樣大小的小區塊，並且把這些區塊按順序加以編號。而讀寫檔案時，要先指定檔案讀寫位置要在那一個區塊上，才能進行讀寫的工作。

直接存取檔可以任意在檔案的任何一個地方來進行讀寫的工作。

Example:

"兄弟象"在一場棒球比賽中的打擊者打擊率依棒次順序列表在檔案 List 中如下：

3.12

2.98

3.34

2.86

2.54

2.78

2.23

2.56

請寫一個可以由棒次來查尋打者打擊率的程式。

```
Program ex0909
   implicit none
   character(len = 20), parameter :: input = 'List'
   integer, parameter :: players = 9
   integer :: player
   integer, parameter :: rec_length = 6
   real :: hit_rate
   open(10, file = input, form = 'formatted', access = 'direct', & recl = rec_length)
   do while (.true.)
     write(*,*) 'Number:'
     read(*,*) player
     if(player < 1 .or. player > players) exit
     read(10, fmt = '(F4.2)', rec = player) hit_rate
      write(*, 100) 'Number ', player, 'hit_rate = ', hit_rate
100   format(1X, A8, I2, A10, F5.2)
   end do
   stop
   end program ex0909
```

讀取檔案位置

(1) 開啟直接讀取檔時，open 敘述中的 access = 'direct'及 recl 後的數值不能省略。這個數值是用來切分出檔案區塊大小使用的。

(2) 在 DOS 作業系統中，文件檔中每一行的行尾都有兩個看不見的符號用來代表一行文字的結束。所以真正一行的長度就是"一行文字字元的數量再加上 2"

e.g. 在 List 檔中每行長度 = 4 + 2 = 6

在 unix 中，每一行的行尾只需一個結束符號，所以一行的長度就是"一行文字字元的數量再加 1"

Example : 依選手的背號順序，輸入選手的打擊率

```
Program ex0910
   implicit none
   character(len = 20), parameter :: input = 'newList'
```

```
integer, parameter :: players = 9, rec_length = 6
integer :: player
real :: hit_rate
open(10, file = input, form = 'formatted', access = 'direct', & recl = rec_length)
do while (.true.)
  write(*,*) 'Hit Number:'
  read(*,*) player
  if(player < 1 .or. player > players) exit
  read(10, fmt = '(F4.2)', rec = player) hit_rate
  write(*,*) 'Input hit rate:'
  read(*,*) hit_rate
  write(10, fmt = '(F4.2)', rec = player) hit_rate
end do
stop
end program ex0910
```

執行結果:

```
Hit Number : 3
Input hit rate : 2.54
Hit Number : 5
Input hit rate : 3.46
Hit Number : 2
Input hit rate : 3.44
Hit Number : 0
```

Newlist 檔案

□ □ □ □ □ □3・4 4□ □2・5 4□ □ □ □ □ □ □ □3・4 6□ □

二進位檔的操作

二進位檔：肉眼無法明白了解之亂碼檔

Example: 把輸入棒球選手打擊率的程式，改成使用二進位檔來運作：

```
Program ex0911
Implicit none
character(len = 20), parameter :: output = 'List.bin'
integer, parameter :: players = 9, rec_length = 4
integer :: player
real :: hit_rate
```

hit_rate 是單精度

```
 open(10, file = output, form = 'unformatted', access = 'direct', & recl = rec_length)
do while (.true.)
   write(*,*) 'Hit Number :'
   read(*,*) player
   if (player < 1 .or. player > players) exit
   read(*,*) hit_rate
   write(10, rec = player) hit_rate
end do
stop
end program ex0911
```

可節省儲存的空間，儲存單精度的浮點數只需 4 個 bytes。若以文字檔來儲存同樣精準度的浮點數，其所需之 byte 遠超過 4 個 bytes。因此，如果要存放"精確"及"大量"的資料時，使用二進位檔案是比較好的選擇。