# 第五讲 主程序与例程

之陳印斌

## 程序设计的“模块”化

- 所谓“模块化”就是把程序划分为若干个部分，每个部分独立存放、完成一个特定的功能。其目的是降低程序的复杂度，使设计出来的程序便于阅读、调试和维护。
- 一个“模块”可以是一条语句、一段程序、一个函数等
- “模块”的基本特征是其仅有一个入口和一个出口，“模块”相互独立，内聚性很强，一个“模块”完成一个功能

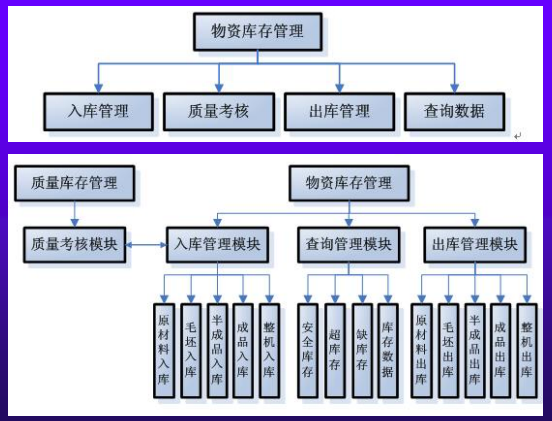
之陳印斌

## 结构物资库存管理信息系统

- 1.1 管理目的
  - 厂内通过库存数据，生产可能合理安排生产、了解资金占用情况，供应科能及时采购，财务科能进行成本分析，销售科能分析顾客需求和销售状况。
- 1.2 功能概述
  - 库房管理每天及时输入入库和出库数据。
- 1.3 具体功能要求
  - 1.3.1 库房信息管理；
  - 1.3.2 各库房管理员每天数据入库数据和出库数据；
  - 1.3.3 向生产设计部门提供库存数量信息；
  - 1.3.4 查询功能：安全库存、超库存、缺库存等信息；

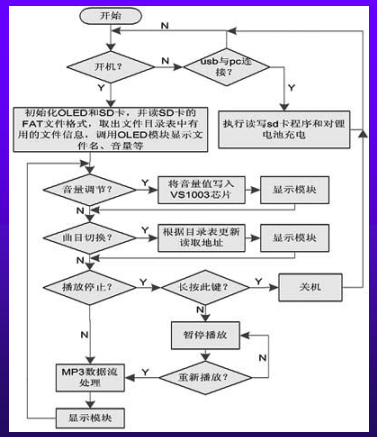
之陳印斌

## 模块化程序设计




之陳印斌

# 模块化程序设计－mp3播放器



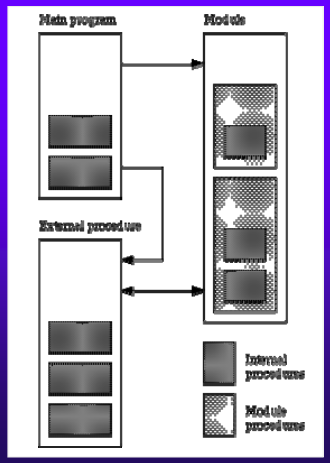

之陳
印斌

# 程序的构造单元

- 程序的构造单元包括：
  - 主程序（Main Program）
  - 模块（Module）：放置通用的例程，作为模块例程专供其它程序单元使用。
  - 外部例程
- 这三种程序单元都可含有其内部例程。
- 例程分为
  - 内部例程
    - 内部函数（function）、语句函数
    - 内部子程序（subroutine）
  - 外部例程（subroutine）

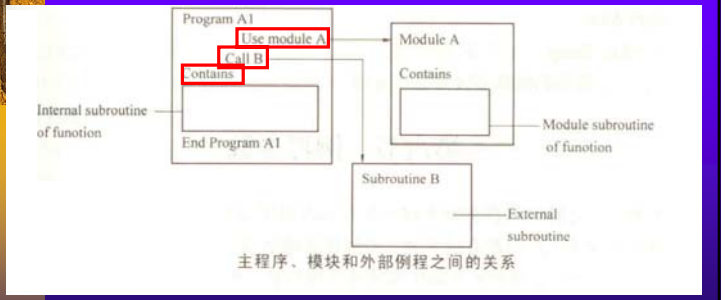
之陳
印斌

# 主程序/模块/例程之间的关系




之陳
印斌

# 主程序/模块/例程之间的关系




主程序、模块和外部例程之间的关系

之陳
印斌

## 主程序

- 一个完整的程序有且只有一个主程序

```
[PROGRAM   程序名]
        [USE MODULE 模块名]
        [声明语句]
        [执行语句]
        。。。
        [CALL 外部例程]
        。。。
[CONTAINS
        内部例程]
END [PROGRAM[程序名]]
```

之陳印斌

## 主程序

- 主程序使用时应注意以下几点：
  - 主程序只有END语句是必须的，其它都是可选的；
  - 若含有内部例程，则必须有CONTAINS关键字；
  - 可以有多个内部例程，但不允许内部例程的嵌套；
  - 主程序的END语句若出现程序名，其前面的PROGRAM关键字不能少。

之陳印斌

## 例程

- 内部例程
  - 内部函数 FUNCTION 函数名（参数列表）
  - 语句函数
  - 内部子程序 SUBROUTINE 子程序名（参数列表）
- 外部例程
  - 外部子程序 SUBROUTINE 子程序名（参数列表）

之陳印斌

## Example ball.f90

$$S = 4\pi R^2$$

$$V = \frac{4}{3}\pi R^3$$

之陳印斌

# Example ball.f90

```
PROGRAM Main
    IMPLICIT NONE
    REAL R, A, V
    REAL, PARAMETER PI = 3.1415926

    PRINT*,'本程序用来计算球体的表面积和体积'
    PRINT*,'请输入球体的半径...'
    READ*,R
    PRINT*, 'THE RADIUS OF SPHERE IS:', R

    A = 4.0 * PI * R**2
    PRINT*,'THE AREA   OF SPHERE IS:',A

    V = 4.0 / 3.0 * PI * R**3
    PRINT*,'THE VOLUME OF SPHERE IS:',V

END PROGRAM
```

之陳印斌

# 内部函数

虚参、形参、哑元

一 函数结构

[类型名] FUNCTION 函数名（[虚拟参数表]）

⋮

函数名=表达式

⋮

END FUNCTION 函数名

函数体

二 函数的调用

一般形式：函数名（[实际参数表]）

之陳印斌

# 内部函数—构造形式

- 内部函数位于主程序的CONTAINS关键字和END语句之间，构造形式为：

  FUNCTION 函数名（[参数列表 ]）

  [声明语句]

  [执行语句]

  END FUNCTION [函数名]

- 强制类型声明IMPLICIT NONE，作用域为整个程序单元，无须再重写。

之陳印斌

```
PROGRAM Main
    IMPLICIT NONE
    REAL R,A,V
    REAL， PARAMETER：：PI = 3.1415926

    PRINT*, '本程序用来计算球体的表面积和体积'
    PRINT*, '请输入球体的半径... '

    READ*,R
    PRINT*, 'THE RADIUS                IS:', R

    A = AREA(R)
    V = VOLUME(R)

    PRINT*,'THE AREA   OF SPHERE IS:',A
    PRINT*,'THE VOLUME OF SPHERE IS:',V

    CONTAINS
        FUNCTION AREA(Radius)
            REAL AREA, Radius
            AREA = 4.0 * PI * Radius**2
        END FUNCTION AREA

        FUNCTION VOLUME(Radius)
            REAL VOLUME, Radius
            VOLUME = 4.0 / 3.0 * PI * Radius**3
        END FUNCTION VOLUME

END PROGRAM
```

实参

虚参

之陳印斌

## 函数返回值

- 在Fortran中，通过将表达式的值赋给函数名来实现。此时，赋值号左边的函数名不能带参数表。

- 正确：AREA = PI * Radius **2
- 错误：AREA(Radius) = PI * Radius **2

- 在Fortran90中，函数值也可以通过RESULT子句来实现。

之陳印斌

## Example3.3： 3.3.F90

```
PROGRAM Main
        REAL    :: A = 2, B

        B = F(A)
        WRITE(*, 10), B
10      FORMAT('X**3 + X - 3 = ', E10.4E3)

        CONTAINS
                FUNCTION F(X) RESULT(R)
                        REAL R, X
                        R = X**3 + X - 3
                END FUNCTION F

END PROGRAM
```

之陳印斌

## 函数返回值

```
FUNCTION F(X) RESULT(R)
        REAL R, X
        R = X**3 + X - 3
END FUNCTION F

FUNCTION F(X)
        REAL F, X
        F = X**3 + X - 3
END FUNCTION F
```

- R代表函数结果，其数据类型代表函数类型；
- 赋值号左边不再是函数名，而是函数结果
- 相应的函数声明，也由声明函数改为声明函数结果

之陳印斌

## Example3.1： 3.1.F90

```
PROGRAM Newton
 IMPLICIT NONE
 INTEGER :: Its        =0        !迭代数
 INTEGER :: MaxIts     =0        !最大迭代数
 LOGICAL :: Converged  =.false.  !是否收敛
 REAL    :: Eps        =1e-6     !迭代精度
 REAL    :: X          =2        !根的初值
 DO WHILE (.NOT.Converged.AND.Its<MaxIts)
   X=X-F(X)/DF(X)
   PRINT*,X,F(X)
   Its=Its+1
   Converged=ABS(F(X))<=Eps
 END DO
 IF(Converged)THEN
   PRINT*,'Newton converged'
```

之陳印斌

## Example3.1（续）： 3.1.F90

```
ELSE
   PRINT*,'Newton diverged'
END IF
CONTAINS
   FUNCTION F(X)
        REAL F,X
        F=X**3+X-3
   END FUNCTION F
   FUNCTION DF(X)
        REAL DF,X
        DF=3*X**2+1
   END FUNCTION DF
END PROGRAM Newton
```

之陳印斌

## 语句函数

- 定义语句函数的形式为：
  - 函数名（参数1，参数2，…）=函数表达式
- 语句函数在使用时应注意：
  - 语句函数先定义后使用，且只能用一条语句来定义；
  - 定义语句应放在声明部分，且放在语句函数相关的类型声明之后
  - 参数列表可以为空（此时，函数实际为一常量表达式），但函数名后面的一对括号，无论在定义还是引用时都不能省略

之陳印斌

## Example3.4： 3.4.F90

```
PROGRAM Statement_Function
   IMPLICIT NONE
   REAL f,x,y        !函数名f、形参x、实参y均需声明类型
   f(x) = x**2 / SQRT(1.0 + 2.0*x + x**2)   !定义语句函数

   DO
        WRITE(*,'(A)',ADVANCE='NO') '输入x的值：'
        READ*,y
              IF(INT(y) == 0) EXIT        !0终止循环
        PRINT*,'f(',y,')=',f(y)
   END DO
END PROGRAM Statement_Function
```

之陳印斌

## 例程参数

- 例程在定义时，列表中的参数只是特定数据类型的占位符（称为形参或虚参），系统不会为它们分配存储单元；当例程被调用或引用时，列表中的参数才被分配一定的存储单元，并接收外部实参传递进来的值；例程执行完毕，例程中的参数所占用的存储空间被系统自动释放，所保存的参数值也随之消失。
- 例程中的局部变量也有同样的性质：例程执行时“生存”，例程不执行时“消亡”。除非被声明具有SAVE属性。

之陳印斌

## Example3.2： 3.2.F90 p23

```
PROGRAM Factorial
    IMPLICIT NONE
    INTEGER I
    DO I=1,10
        PRINT*,I,Fact(I)
    END DO
    CONTAINS
        FUNCTION Fact(N)
            INTEGER Fact,N,Temp
            Temp=1
            DO I=2,N
                Temp=I*Temp
            END DO
            Fact=Temp
        END FUNCTION Fact
END PROGRAM Factorial
```

之陳印斌

## 运行结果

```
 1           1
 3           6
 5         120
 7        5040
 9      362880
11    39916800
13  1932053504
15  2004310016
17  -288522240
19   109641728
Press any key to continue
```

之陳印斌

## 程序分析

- “同名覆盖”现象，若在内部函数中声明了和全局变量同名的局部变量，则全局变量被屏蔽，内部函数引用的是局部变量。
- 程序错误的原因
  - I是全局变量，当函数Fact第一次被引用时，I=1，该值被传递给虚参N，相同的I在函数循环体内被赋初值2，此时2>N,不执行DO循环（Fact=1），当函数Fact返回到主程序打印时，I的值为2；下次引用时，I在主程序DO循环中增至3，以此类推，程序决不会计算偶数阶乘。
- 正确的做法
  - 应该在内部函数中重新声明I，使之成为一局部变量。

之陳印斌

## 全局变量与局部变量

- 在内部例程中声明所有变量应成为一条编程规则，以消除全局变量带来的负面影响。
- 如果需要从程序单元向内部例程传递数据，最安全的方式是利用参数传递。
- 如果需要在多个内部例程中共享大量数据，最好的解决方案是在模块中统一声明全局变量，需要访问这些全局变量的内部例程引用该模块即可。

之陳印斌

# 子程序（例程）

一 子程序结构

```
SUBROUTINE 子程序名 [（虚拟参数表）]
    ⋮
    ⋮        } 子程序体
    ⋮
END SUBROUTINE 子程序名
```

二 子程序的调用

一般形式：CALL 子程序名 [（实际参数表）]

说明：函数子程序一般通过函数名将值带回调用程序；子程序是通过实参与虚参的联系，将值带回调用程序，即虚实结合。

之陳印斌

# 内部子程序

- 子程序和函数的主要差别:
  - 没有返回值和子程序名关联，因此无需声明子程序类型;
  - 通过CALL语句调用子程序;
  - 在例程原型（头）和END语句（尾）中，使用关键字SUBROUTINE;
  - 若子程序参数表为空，子程序名后的一对括号可以省略。
- 通常函数通过函数名返回一个值，而子程序通过参数可以返回多个值。

之陳印斌

# 内部子程序

- 内部子程序的构造形式:

```
SUBROUTINE 子程序名[(参数表)]
    [声明语句]
    [执行语句]
END SUBROUTINE [子程序名]
```

- Example： Area & Volume/Internal Function

之陳印斌

```
PROGRAM Main
        IMPLICIT NONE
        REAL R,A,V
        PARAMETER PI=3.14159265
        PRINT*,'本程序用来计算球体的表面积和体积'
        PRINT*,'请输入球体的半径... '
        READ*,R
        PRINT*,'THE RADIUS OF SPHERE IS:',R
        CALL CAL_SPHERE(R, A, V, A)
        !V = VOLUME(R)
        PRINT*,'THE AREA   OF SPHERE IS:',A
        PRINT*,'THE VOLUME OF SPHERE IS:',V
        CONTAINS
                SUBROUTINE CAL_SPHERE(RADIUS, AREA, VOLUME)
                        REAL RADIUS, AREA, VOLUME
                        AREA = 4.0 * PI * RADIUS**2
                        VOLUME = 4.0 / 3.0 * PI * R**3
                END SUBROUTINE CAL_AREA
        !       FUNCTION VOLUME(R)
        !               REAL VOLUME,R
        !               VOLUME = 4.0 / 3.0 * PI * R**3
        !       END FUNCTION VOLUME
END PROGRAM
```

之陳印斌

# 外部例程

- 通用的例程一般作为外部例程来实现，以便被多个调用程序调用。
- 通常，外部例程位于单独的文件中，除了头、尾部分外，外部例程和主程序在形式上是相同的。
- 外部例程和内部例程的主要差别：
  - 外部例程可以含有内部例程，而内部例程不能再含有内部例程；
  - END语句中的关键字FUNCTION/SUBROUTINE, 在外部例程中是可选的，但在内部例程中是必须的。
    WHY???

之陳印斌

# 外部例程的构造形式

```
SUBROUTINE  子程序名[(参数表)]
    [声明语句]
    [执行语句]
[CONTAINS
    内部例程]
END [SUBROUTINE[子程序名]]
```

之陳印斌

# Example3.6：　3.6.F90

```
PROGRAM Exchange
   IMPLICIT NONE
   EXTERNAL Swap       !声明例程Swap为外部的
REAL :: A=1,B=5
CALL Swap(A,B)
PRINT*,A,B
END PROGRAM Exchange

   SUBROUTINE Swap(X,Y)
       REAL Temp,X,Y
       Temp=X
       X=Y
       Y=Temp
   END [SUBROUTINE]
```

之陳印斌

# 外部例程Example——主程序

```
PROGRAM Main
   REAL R,A,V
   EXTERNAL CAL_AREA, CAL_VOLUME
   PRINT*,'THIS PROGRAM IS USED TO CALCULATE THE AREA
   AND VOLUME OF SPHERE'
   PRINT*,'PLEASE INPUT THE RADIUS OF SPHERE...'
   READ*,R
   PRINT*,'THE RADIUS OF SPHERE IS:',R
   CALL CAL_AREA(A, R)
   PRINT*,'THE AREA   OF SPHERE IS:',A
   CALL CAL_VOLUME(V,R)
   PRINT*,'THE VOLUME OF SPHERE IS:',V
END PROGRAM
```

之陳印斌

## 外部例程Example——外部例程

```
SUBROUTINE CAL_AREA(AREA, RADIUS)
    REAL AREA,RADIUS
    PARAMETER PI=3.1415926
    AREA = 4.0 * PI * RADIUS**2
END [SUBROUTINE CAL_AREA]

SUBROUTINE CAL_VOLUME(VOLUME, RADIUS)
    REAL VOLUME,RADIUS
    PARAMETER PI=3.1415926
    VOLUME = 4.0 / 3.0 * PI * RADIUS**3
END [SUBROUTINE CAL_VOLUME]
```

之陳印斌

## 注意

- 如果无意中用系统的标准例程名做了外部例程名，编译器会优先引用标准例程，而不是用户定义的外部例程。为避免这种情况，添加外部例程声明语句（EXTERNAL），这应该成为一个编程惯例。
- 假如一个文件包含多个外部例程，可考虑将外部例程移至模块中，从而将外部例程转换为模块例程。
- Example: area&volume/external 2 or 3

之陳印斌

## 参数传递

- 实参和虚参之间的数据传递有两种方式：
- 引用传递
  - 将实参的内存地址传递给虚参，使实参和虚参具有相同的内存地址，即虚参与实参共用一个存储单元，因此在被调过程中对虚参的任何操作都变成了对相应实参的操作，这样例程中虚参的变化就会传递到实参中。
- 值传递
  - 将实参值的一份拷贝传递给虚参，虚参的变化不会影响到实参。

之陳印斌

## 参数传递

- 在Fortran中，参数（包括数组参数）通常是以引用方式传递，即地址传递，但在实参为常量和表达式的情况下，参数以值方式传递。若将一变量实参括起来，该实参就转换为表达式，表达式以值方式传递。如：
  - CALL Sub((A),B)
  - 其中（A）是表达式，以值的方式传递
- 假如一个虚参没有规定INTENT属性，对应的实参可以是变量，也可以是常量或表达式。建议所有的虚参都规定INTENT属性，特别是函数参数要规定为INTENT(IN)属性。

之陳印斌

# 参数传递

```
SUBROUTINE  SUB(X,Y,Z)
      REAL,INTENT(IN)         ::X
      REAL,INTENT(OUT)        ::Y
      REAL,INTENT(INOUT)   ::Z
      ……
END SUBROUTINE  SUB
```

其中 INTENT(IN) 表示向例程传入数据，拥有该属性的虚参不允许改变

INTENT(OUT) 表示传出数据，对应的实参必须是一变量

INTENT(INOUT) 表示传进/传出数据，对应的实参也必须是一变量

之陳印斌

# Example3.8： 3.8.F90

```
PROGRAM Main
    IMPLICIT NONE
    EXTERNAL Sub
    INTEGER ::X=1
    CALL Sub(X)
    PRINT*,'X=',X
END PROGRAM

SUBROUTINE Sub(A)
    IMPLICIT NONE
    REAL,INTENT(INOUT) :: A
    A=A+1
END SUBROUTINE
```

参数类型必须匹配：实参和虚参类型不匹配会导致出现错误！ Why?

之陳印斌

# 参数的传递（例子）

```
PROGRAM MAIN
          IMPLICIT NONE
          REAL::X
          X=2.5
          CALL SUB(6.0,X,X)
          PRINT*, X
END

SUBROUTINE  SUB(X,Y,Z)
          IMPLICIT NONE
          REAL X, Y, Z
          Y=Y+X
          Z=Y+Z
END SUBROUTINE SUB
```

程序输出结果：（）

A) 6.000000　B) 17.00000　C) 2.500000　D) 5.000000

之陳印斌

# 结果分析

| | 子程序 | | | 主程序 |
|---|---|---|---|---|
| | X | Y | Z | X |
| 调用子程序前 | 0.0 | 0.0 | 0.0 | 2.5 |
| 进入子程序 | | | | |
| Y=Y+X | | | | |
| Z=Y+Z | | | | |

之陳印斌

# 接口块（Interface）

- 要正确地调用例程，编译器需要知道例程的有关信息：例程名、参数个数及各自的数据类型，这些信息的接口称为例程的接口
- 显式接口：标准例程、内部例程和模块例程（其接口对编译器透明）
- 隐式接口：对于外部例程，在调用程序中声明外部例程时，只提供了外部例程名，编译器无从知道其接口信息
- Fortran 90提供接口块（Interface）向调用程序明确外部例程的接口信息，以解决像可选参数这样的复杂情况

interface
　接口体（外部例程头、参数声明、外部例程尾）
end interface

之陳印斌

# Interface example (3.6)

```
Program exchange
    Implicit none

    Interface
          Subroutine swap(x, y)
                Real temp, x, y
          End subroutine swap
    End interface

    Real    :: A=1, B=5

    Call swap(A, B)
    Print*, A, B

End program exchange

Subroutine swap (x, y)
    real temp, x, y

    temp = x
    x = y
    y = temp
End subroutine swap
```

有了接口块，就不需要再用External语句声明外部例程了

之陳印斌

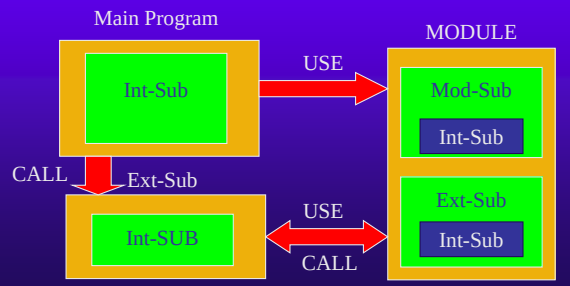
# 接口块（Interface）

- 必须使用接口块的场合：
  - 外部例程具有可选参数
  - 外部函数返回数组或变长字符串；
  - 外部例程具有假定形状数组、指针或目标参数；
  - 例程做参数；
  - 例程用来定义操作符重载；
  - 例程重载
- 除了重载，其余情况只需将外部例程转换为模块例程，就可避免接口块的麻烦。

之陳印斌

# 模块（Module）

MODULE 模块名
　[声明语句]
[CONTAINS
　[模块例程]
END [MODULE[模块名]]




之陳印斌

## 模块－Example Module.f90

- USE 模块名
- USE必须放在Implicit none语句之前
- 若模块和主程序放于同一个文件，模块应位于主程序之前
- 模块不仅可供主程序和外部例程调用，还可供其它模块使用。开发模块库时，应注意模块的层次关系。
- Example：Area&Volume/Module

之陳印斌

## Example3.9： 3.9.F90

参数可选！

```
MODULE  Mod
    IMPLICIT NONE
    CONTAINS
        REAL FUNCTION Func(X,A,B,C)
        !计算FUNC(X)=A*X^2+B*X+C
        !A,B,C不传入，值为0
                REAL,INTENT(IN)  :: X  !X的值一定要传入
                REAL,OPTIONAL,INTENT(IN) :: A,B,C !A,B,C可以不传入
                REAL RA,RB,RC
                RA=0.0;RB=0.0;RC=0.0   !几个简单的赋值语句放在一行
                IF(PRESENT(A))RA=A     !检查可选参数是否存在
                IF(PRESENT(B))RB=B
                IF(PRESENT(C))RC=C
                Func=RA*X**2+RB*X+RC
        END FUNCTION
END MODULE
PROGRAM  Main
    USE Mod
    IMPLICIT NONE
    PRINT*,Func(2.0,C=1.0)            !F(2)=0*2^2+0*2+1=1
    PRINT*,Func(2.0,B=1.0,A=2.0)      !F(2)=2*2^2+1*2+0=10
END PROGRAM
```

之陳印斌

## 一个项目中使用多个程序文件

- 一个项目中只能有一个主程序
- 程序员通常会把一些具备相关功能的函数，独立写作在不同的文件中，编译器可以分别编译这些程序文件，最后再把他们链接到同一个可执行文件中。
- INCLUDE指令用来在程序代码中，插入另一個程序文件中的内容，可以写在任何地方。
- 要确定所有程序文件都放在同一个目录下面才做编译
- 独立档案中的函数，可以编译成***.LIB**程序库来拿给别人使用。***.LIB**的内容经过编译，别人无法读到原始程序代码。
- 当只修改其中一个或者几个程序文件时，未修改之档案不需要重新编译，因此可以加快编译速度。

之陳印斌

## Example

- Area and Volume
  - External 2
  - External 3

之陳印斌