FORTRAN 90 程序设计

FORTRAN语言是最早的高级编程语言，它本身是一种结构化程序设计语言。

FORTRAN90的特点

1、新的数组功能

2、大量的内部例程

3、可以定义自己的数据结构类型

FORTRAN语言编程风格

1. 对每一个程序（包括子程序）功能做说明
2. 对每一个定义变量的含义做说明
3. 子程序安字母排列好，不同子程序之间要隔行
4. 用空格，空行使程序层次分明
5. 为了结构清楚，Format格式语句放在一起
6. 避免中途退出某个结构块

运行FORTRAN程序

根据公式编写程序

$$A(t) = 174.6(t - 1981.2)^3$$

```
PROGRAM AIDS
! Calculates number of accumulated AIDS cases in USA
INTEGER T           ! year
REAL  A             ! number of cases
READ(*,*) T
A = 174.6 * (T - 1981.2) ** 3
Write (*,*)  'Accumulated AIDS in US by year', T, ':', A
END PROGRAM AIDS
```

第一章 简介

1.1 FORTRAN介绍

1.2 运行FORTRAN程序

1.3.1 例程

总结

FORTRAN 介绍

- 程序语言——一套特殊的编码计算机指令的规则。
- FORTRAN全称是 FORmula TRANslation。
- 它用可以理解的形式写出公式。如 X=B/(2*A)

总结

- 计算机程序是一套解决特定问题的指令集
- FORTRAN利用 READ * 读取数据
- read(*,*)
- FORTRAN利用write * 输出显示结果
- write(*,*)

第二章　基本FORTRAN　I

1. 简单程序结构介绍
2. 数据类型
3. 常量
4. 名字和变量
5. 数学表达式和赋值
6. 简单的输入输出
7. 总结

简单程序结构介绍

本章和下一章介绍怎样编制FORTRAN程序解决简单问题

必须掌握的两点

1. 代码编写的准确规则
2. 解决实际问题的逻辑计划

FORTRAN程序结构：

```
PROGRAM program name  !告诉一个程序的开始
        [declaration statements]
        [executable statements]
END PROGRAM program name  !通知编译器程序结束
```

当运行一个F90程序的时候，发生两个过程

1. 源程序被编译连接 （编译器完成）

   编译期间， 被编译器生成的数据放在计算机内存中，

   内存定位根据程序的变量名配置

2. 编译后程序的执行

例程

```
      PROGRAM MONEY
! Calculates balance after interest compounde
     & REAL BALANCE, INTEREST, RATE          73 - 80
      BALANCE = 1000
      RATE = 0.09
      INTEREST = RATE * BALANCE
      BALANCE = BALANCE + INTEREST
      write(*,*) 'New balance:', BALANCE

      END PROGRAM MONEY
```

特点：除赋值语句外，每个语句开头处为某一关键字

早期的FORTRAN77有格式

1-5 标记行

6 续行

7-72 语句行

73-80 注释行

FORTRAN90没有格式问题，但为了清楚，可以采用fortran77的格式

注意问题：

1. 在进行赋值时必须先定义并指明数据类型，否则会丢失精度，

   如 double precision x

   x＝1.23d0=1.23_16

2. 关键字，名字，常量符号，标号内部不允许出现空格，其他地方忽略

3. 在字符前面加！或在语句开头用*表示注释语句

4. & 用于语句行续行，但不能用于注释续行

   如： BALANCE = BALANCE + &

   INTEREST

5. 建议一行写一个语句，写多个时用 分号 隔开。

   如： A=1; B=1; C=1

   B=2

数据类型

- 数据类型是F90的基础，它有5种内部数据类型，这5种类型又分归两类

数值型：

1） 整型 （short型， long型）

2） 实型（浮点型，双精度）

3） 复数型

非数值型

1） 字符型

2） 逻辑型

除外，用户可以定义自己的数据类型

变量 I

我们已经看到内存的定位由一个符号名字(即变量名字）定位。

注意问题：

（1) 变量名长度可为1-31个字符，且一般不区分大小写。

（2) 变量名必须以字母开头

（3) 在每一个程序单元中，变量名必须是唯一的

（4) 变量有隐含定义， I-N或以它们开头的变量默认为整型变量，其他的默认为实型变量。

（5) 可以利用IMPLICIT关键词来对变量进行类型隐含说明

(6) 最好用IMPLICIT NONE来取消隐含变量说明，对每一个使用的变量重新定义。

变量 II

- 变量的优先级：
  1. 复数变量
  2. 双精度变量
  3. 实型变量
  4. 整型变量

运算表：

	整型	实型	双精度	复型
整型	整型	实型	双精度	复型
实型	实型	实型	双精度	复型
双精度	双精度	双精度	双精度	复型（或×）
复型	复型	复型	复型（或×）	复型

变量 III

由于不同的计算机，对变量所占字节大小定义不同，因此一个程序最好是采用内部函数来获取变量的范围和精度大小

Result=selected_real_kind(precison,range);　　Result=selected_int_kind(range)

例子：

```
Program test_kinds
implicit none
integer, parameter :: long=selected_real_kind( 9, 199)
real(kind=long) :: a=2.8_long
write(*,*) KIND(a)
write(*,*) long
write(*,*) precision(a)
write(*,*) range(a)
write(*,*) huge(a)
end
```

结果

```
           8
           8
          15
         307
 1.7976931348623157E+308
```

变量 IV

变量的4种赋值方法：

1. DATA赋值

例如： REAL A,B

DATA A, B /1,2/

2.定义时赋值

例如: REAL :: A=1.0

3.直接赋值

REAL A

A=1.0

4.通过计算赋值

REAL A,B

B=2.0

A=1.2+B

例子：

重力场下的垂直运动：

```
PROGRAM Vertical
! Vertical motion under gravity
IMPLICIT NONE
REAL, PARAMETER :: G = 9.8 ! 有名常量，在计算中固定不变
REAL S                ! displacement (m)
REAL T                ! time
REAL U                ! initial speed(m/s)
Weite(*,*) ' Time    Displacement'
U = 60
T = 6
S = U * T - G / 2 * T ** 2
Write(*,*) T, S
END PROGRAM Vertical
```

操作符

- FORTRAN有以下几个内部运算操作符

按照优先级别分别为：

1. 括号 （）
2. 乘方 **
3. 乘除 * /
4. 加减 ＋ －

例子： REAL A

A=2.0*3.0-8.0/2.0*2**2+(1.0+1.0)/2.0

特殊情况由右向左

例如： A**B**C

注意的问题：

1. 整数除法 10/4=2, 而10.0/4.0=2.5
2. 做复合运算时，低级运算先转化为高级运算后再进行计算，结果要赋给高级变量，赋给低级变量时会砍掉一部分数据，导致结果会不准确

输入输出 Ⅰ

前面我们讲过计算机用READ读数据，用PRINT或WRITE输出数据。

例子：

```
READ (*,*)   A,B
READ  (*,*)  C
WRITE(*,*)   A
WRITE (*,*)  B,C
```

数据准备 2 4　6　　输出数据 2

5　　4 5

注意问题：

1. 输入输出的一行叫一个记录
2. 同一行数据用逗号，或空格隔开
3. 每一个读取输出语句对应一个记录
4. 一个记录中没有读到的数据作废
5. 如果一个记录数据不够READ可以到下一个记录读数据，直到 I/O结束
6. 没有足够的数据满足读时，程序将出现错误信息。

输入输出 II

从文件读或往文件中写

```
      program tt
      REAL A,B,C
      OPEN( 1, FILE = 'DATA1' )
      OPEN( 2, FILE = 'DATA2' )
      READ(1, 5) A, B, C
      WRITE(2,5) A, B, C
    5 FORMAT(1X,3F12.5)
       END
```

DATA1文件

```
  23.5,45.6,74.85
```

DATA2文件

```
   23.50000   45.60000   74.85000
```

输入输出 II

```
program main
implicit none
character(len=100):: fileDir,fileName,fileSuffix
real(kind=8):: a
  fileDir = 'C:\Users\air\Desktop\'！比如桌面
  fileName = '03013‘,
  fileSuffix = '.txt'
open(111,file = ADJUSTL(trim(fileDir))//&
&adjustl(trim(fileName))//adjustl(trim(fileSuffix)))
read(111,*)  a    !读桌面文件03013.dat的内容赋值给变量a
write(*,*) a
end
```

总结

1.成功的计算机程序需要好的编码规则和好的编程计划

2.编译器将程序语句翻译成机器码

3.所有语句除赋值语句外，开头都是关键字

4.用&作为续行标志，fortran77 在第六列做标记

5.变量名字可达32个字符，fortran77最多6个

总结

6.变量使用前最好先声明类型，防止出错

7.一群变量可以由DATA赋初值

8.最好用IMPLICIT NONE取消隐含变量类型说明

9.一个变量名是内存的一段地址。

10. 有5种内部数据类型。INTEGER REAL COMPLEX,LOGICAL,CHARACTER. 还有一种特殊的DOUBLE PRECISION

第三章 基本FORTRAN II

DO 循环

DO循环的结构

```
    INTEGER  I,J,K
   DO  I  =  J,  K
       block
   END DO
```

或

```
    DO 10  I=J,K
      BLOCK
10 CONTINUE
```

例子

用牛顿方法计算一个数的平方

```
    PROGRAM Newton
! Square rooting with Newton
    IMPLICIT NONE
    REAL    A               ! number to be square rooted
    INTEGER I               ! iteration counter
    REAL    X               ! approximate square root of A
    WRITE( *, 10, ADVANCE = 'NO' ) 'Enter number to be square rooted: '
10  FORMAT( A )
    READ(*,*)  A
    X = 1                   ! initial guess (why not?)
    DO I = 1, 6
       X = (X + A / X) / 2
       write(*,*) X
    ENDDO
    Write(*,*)
    Write(*,*) 'Fortran 90''s value:', SQRT( A )
    END
```

```
Enter number to be square rooted: 2

   1.5000000
   1.4166666
   1.4142157
   1.4142135
   1.4142135
   1.4142135
Fortran 90's value:    1.4142135
```

IF-THEN-ELSE条件语句

IF-THEN-ELSE 结构

```
IF condition THEN
        block1
[ELSE
        blockE]
END IF
```

IF 语句结构
IF (*condition*) *语句*

关系运算符

- > .GT.
- >= .GE.
- < .LT.
- <= .LE.
- == .EQ.
- /= .NE.

关系运算符		英语含义	所代表的数学符号
.GT.	>	Greater Than	> (大于)
.GE.	>=	Greater than or Equal to	≥ (大于或等于)
.LT.	<	Less Than	< (小于)
.LE.	<=	Less than or Equal to	≤ (小于或等于)
.EQ.	==	EQual to	= (等于)
.NE.	/=	Not Equal to	≠ (不等于)

IF-THEN-ELSE条件语句

逻辑运算符：

.NOT.　　.AND.　　.OR.　　.NEQV.　　.EQV.

五大逻辑运算符

逻辑运算符	含义	逻辑运算例	例子含义
.AND.	逻辑与	A.AND.B	A，B为真时，则A.AND.B为真
.OR.	逻辑或	A.OR.B	A，B之一为真，则A.OR.B为真
.NOT.	逻辑非	.NOT.A	A为真，则.NOT.A为假
.EQV.	逻辑等价	A.EQV.B	A和B值为同一逻辑常量时，A.EQV.B为真
.NEQV.	逻辑不等价	A.NEQV.B	A和B的值为不同的逻辑常量，则A.NEQV.B为真

例子

```
PROGRAM Final_Mark
! Final mark for course based on class record and exams
IMPLICIT NONE
REAL     CRM         ! Class record mark
REAL     ExmAvg      ! average of two exam papers
REAL     Final       ! final mark
REAL     P1          ! mark for first paper
REAL     P2          ! mark for second paper
INTEGER  Stu         ! student counter
! CHARACTER (Len = 15) Name    ! Name
OPEN( 1, FILE = 'MARKS.txt' )
 write(*,*) ' CRM          Exam Avg    Final Mark'
 write(*,*)
DO Stu = 1, 3
  READ( 1, * ) CRM, P1, P2 ! READ( 1, * ) Name, CRM, P1, P2
  ExmAvg = (P1 + P2) / 2.0
  IF (ExmAvg > CRM) THEN
    Final = ExmAvg
  ELSE
    Final = (P1 + P2 + CRM) / 3.0
  END IF
  IF (Final >= 50) THEN
     write(*,*)  CRM, ExmAvg, Final, 'PASS'
  ELSE
     write(*,*) CRM, ExmAvg, Final, 'FAIL'
  END IF
END DO
END
```

MARKS.txt

```
40 60 43
60 45 43
13 98 47
```

字符

到目前为止，我们只学了FORTRAN的两个内部数据类型，整型和实型，下面我们介绍字符型。

基本字符常量是由‘’引起来的一串字符

例如：'Jesus said, "Follow me"'

字符变量：定义形式为

```
CHARACTER (Len = 15) Name
CHARACTER Name*15
 Name = 'J. Soap'
```

例子：在前面的例子中加入如下语句

```
 CHARACTER (Len = 15) Name     ! Name
 PRINT*, 'Name           CRM       Exam Avg   Final Mark'
 READ( 1, * ) Name, CRM, P1, P2
  write(*,*) Name, CRM, ExmAvg, Final, 'PASS'
  write(*,*) Name, CRM, ExmAvg, Final, 'FAIL'
```

输入：

```
'Able, RJ'       40 60 43
'Nkosi, NX'      60 45 43
'October, FW'    13 98 47
```

则输出结果为

Name	CRM	Exam Avg	Final Mark
Able, RJ	40.0000000	51.5000000	51.5000000 PASS
Nkosi, NX	60.0000000	44.0000000	49.3333321FAIL
October,FW	13.0000000	72.5000000	72.5000000PASS

有名常量

有名常量是在程序运行中始终不变的量，下面是例子

```
REAL, PARAMETER :: Pi = 3.141593
INTEGER, PARAMETER :: Two = 2
REAL, PARAMETER :: OneOver2Pi = 1 / (2 * Pi)
REAL, PARAMETER :: PiSquared = Pi ** Two
```

If the named constant is of character type, its length may be declared with an asterisk. The actual length is then determined by the compiler, saving you the bother of counting all the characters. E.g.

```
CHARACTER (LEN = *), PARAMETER &
 :: Message = 'Press ENTER to continue'
```

类型说明语句 KIND

KIND是FORTRAN90的一个新特征，5个内部数据类型有缺省的类型长度，它们依赖于具体的计算机系统。函数SELECTED_INT_KIND(N)可以返回整型类型长度，对于实数等也是类似SELECTED_REAL-KIND(P,R), P是精度，R是范围。通过KIND可以人为定义数据类型定义类型长度。

例子：

```
INTEGER(KIND=2)  I
REAL(KIND=4)  A
CHARACTER (LEN = 10, KIND = GREEK) Greek_Word
CHARACTER (LEN = 10) English_Word ! default kind
CHARACTER (KIND = GREEK) Greek_Letter
                    ! default length of 1
CHARACTER (10, GREEK ) Greek_Word
```

复数类型

例子:

```
COMPLEX, PARAMETER :: i = (0, 1)    ! sqrt(-1)
COMPLEX X, Y
X = (1, 1)
Y = (1, -1)
PRINT*, CONJG(X), i * X * Y
```

Output:

```
(  1.0000000, -1.0000000) (  0.0000000E+00,  2.0000000)
```

When a complex constant is input with `READ*` it must be enclosed in parentheses.

Many of the intrinsic functions can take complex arguments.

内在函数

到目前为止，我们可以编写一些简单的程序了，但是，最有兴趣的问题是设计到一些特殊的数学函数，FORTRAN含有许多的内在函数。

例子：

```
PROGRAM Projectile
IMPLICIT NONE
REAL, PARAMETER :: g = 9.8      ! acceleration due to gravity
REAL, PARAMETER :: Pi = 3.1415927   ! a well-known constant
REAL A       ! launch angle in degrees
REAL T       ! time of flight
REAL Theta   ! direction at time T in degrees
REAL U       ! launch velocity
REAL V       ! resultant velocity
REAL Vx      ! horizontal velocity
REAL Vy      ! vertical velocity
REAL X       ! horizontal displacement
REAL Y       ! vertical displacement

READ*, A, T, U
A = A * Pi / 180        ! convert angle to radians
X = U * COS( A ) * T
Y = U * SIN( A ) * T - g * T * T / 2.
Vx = U * COS( A )
Vy = U * SIN( A ) - g * T
V = SQRT( Vx * Vx + Vy * Vy )
Theta = ATAN( Vy / Vx ) * 180 / Pi
PRINT*, 'x: ', X, 'y: ', Y
PRINT*, 'V: ', V, 'Theta: ', Theta
END
```

一些有用的内在函数

- `ACOS(X)`: ACOS arc cosine (inverse cosine) of `X`.
- `ASIN(X)`: ASIN arc sine of `X`.
- `ATAN(X)`: ATAN arc tangent of `X` in the range $-\pi/2$ to $\pi/2$.
- `ABS(X)`: ABS absolute value of integer, real or complex `X`.
- `ATAN2(Y, X)`: ATAN2 arc tangent of *y*/*x* in the range $-\pi$ to π.
- `COS(X)`: COS cosine of real or complex `X`.
- `COSH(X)`: COSH hyperbolic cosine of `X`.
- `COT(X)`: COT cotangent of `X`.
- `EXP(X)`: EXP value of the exponential function e_x, where `X` may be real or complex.
- `INT(X [,KIND])`:
- `LOG(X)`: LOG natural logarithm of real or complex `X`..
- `LOG10(X)`: LOG10 base 10 logarithm of `X`.
- `MAX(X1, X2[, X3, ...])`: MAX maximum of two or more integer or realarguments.
- `MIN(X1, X2[, X3, ...])`: MIN minimum of two or more integer or real arguments.
- `MOD(K, L)`: MOD remainder when `K` is divided by `L`. Arguments must be both integer or both real.
- `NINT(X [,KIND])`: NINT *nearest* integer to `X`, e.g. `NINT(3.9)` returns 4, while `NINT(-3.9)` returns -4.
- `REAL(X [,KIND])`: REAL function converts integer, real or complex `X` to real type, e.g. `REAL(2)/4` returns 0.5, whereas `REAL(2/4)` returns 0.0.
- `SIN(X)`: SIN sine of real or complex `X`.
- `SINH(X)`: SINH hyperbolic sine of `X`.
- `SQRT(X)`: SQRT square root of real or complex `X`.
- `TAN(X)`: TAN tangent of `X`.
- `TANH(X)`: TANH hyperbolic tangent of `X`.

内在子程序

FORTRAN90含有许多内在子程序

子程序和函数的区别是：

1. 子程序通过参数返回值
2. 函数通过函数名返回值

子程序例子：

```
! DATE_AND_TIME

CHARACTER*10 DATE, TIME, PRETTY_TIME
CALL DATE_AND_TIME( DATE, TIME )
PRINT*, DATE
PRETTY_TIME = TIME(1:2) // ':' // TIME(3:4) // ':' //
  TIME(5:10)
PRINT*, PRETTY_TIME
END
```

Output:

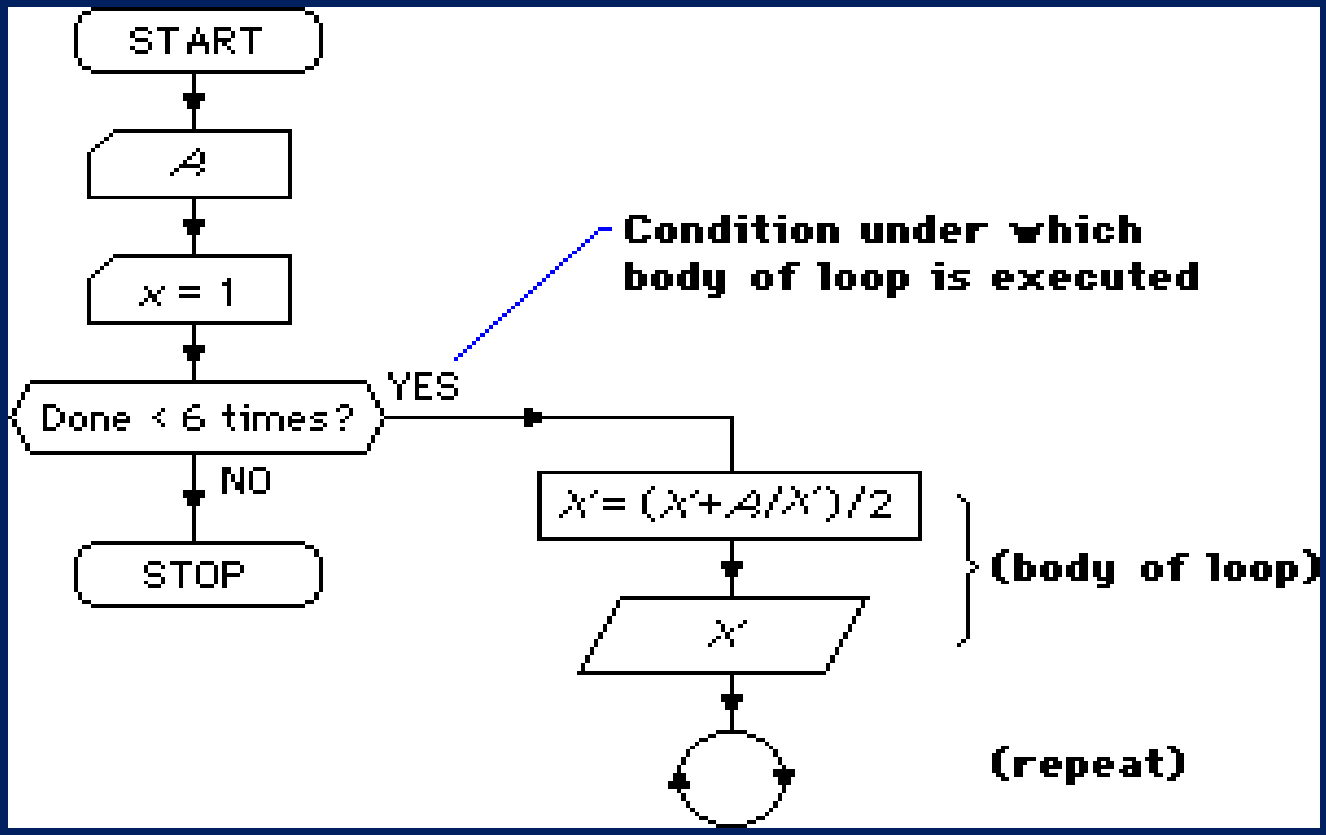
```
19930201
15:47:23.0
```

程序准备

前面我们简单的介绍了FORTRAN程序，但是实际问题是很复杂的，为了更好的解决问题，我们首先要根据问题设计流程图，以便是问题的解决更加明了。

我们可以使用伪代码来写出程序的执行流程，以用牛顿法求解方程的根值为例，我们写一个简单的流程图：

Newton method for square rooting



总结

1. 有名常数在程序中不能改变
2. DO循环是程序实现循环操作
3. IF-THEN-ELSE使程序实现条件判断
4. 字符常量在一对引号内部
5. 变量可以在声明时初始化
6. KIND参数依赖于处理系统
7. 字符有两个类型参数，一个是长度LEN，一个是类型KIND
8. 复数有内部类型定义COMPLEX支持
9. 复数常数在一对括号内
10. 内部函数可以直接被程序使用
11. 到目前为止，我们学习了5种内部数据类型中的4种，整型，实型，复数型，字符型。

第四章　选择结构

除了可以快速的进行算术运算，计算机的另一个主要特点是能够做出选择，象我们前面讲的IF-THEN-ELSE语句，它和顺序结构，循环结构一起构成计算机程序的框架，可以解决一切问题。

最常用的选择语句是 IF 语句。它的基本结构如下：

```
IF (条件1) THEN
        block1
ELSE IF (条件2) THEN
        block2
ELSE IF (条件3) THEN
        block3
            ...
ELSE
        block_E
END IF
```

可以有大量的ELSE IF，但一个选择块只能有一个ELSE

为了区分别的IF结构，也可以给IF结构加一个名字

例如：

```
GRADE: IF (Final >= 50) THEN
           PRINT*, 'Pass'
       ELSE [GRADE]
           PRINT*, 'Fail'
       END IF GRADE
```

例子：

```
READ（*,*) Final
IF (Final >= 75) THEN
    PRINT*, Name, CRM, ExmAvg, Final, '1'
     Firsts = Firsts + 1
ELSE IF (Final >= 70) THEN
     PRINT*, Name, CRM, ExmAvg, Final, '2+'
     UpSeconds = UpSeconds + 1
ELSE IF (Final >= 60) THEN
 PRINT*, Name, CRM, ExmAvg, Final, '2-'
 LowSeconds = LowSeconds + 1
ELSE IF (Final >= 50) THEN
 PRINT*, Name, CRM, ExmAvg, Final, '3'
 Thirds = Thirds + 1
ELSE
 PRINT*, Name, CRM, ExmAvg, Final, 'F'
 Fails = Fails + 1
END IF
```

嵌套选择

前面是基本的选择结构，可以在一个选择结构内部嵌套一个新的选择结构，例如：

```
Disc = B * B - 4 * A * C
Outer: IF (A /= 0) THEN
     Inner: IF (Disc < 0) THEN
                PRINT*, 'Complex roots'
            ELSE Inner
                X1 = (-B + SQRT( Disc )) / (2 * A)
                X2 = (-B - SQRT( Disc )) / (2 * A)
            END IF Inner
       END IF Outer
```

DO和IF

一个DO循环可以包含一个选择IF结构，相反也成立，但是不允许两个结构超出另一个结构。例如下面的程序是非法的。

```
DO I = 1, 10
   IF (I > 5) THEN
  ...
 END DO      ! Illegal: IF must end before DO
END IF
```

下面的语法也是错误的

```
IF ( ... ) THEN
 DO I = 1, 10
  ...
 END IF      ! Illegal: DO must end before IF
END DO
```

逻辑类型 I

下面我们讨论5种FORTRAN内部数据类型的最后一种，逻辑类型。

1.逻辑常量：缺省的逻辑常量有两个字常数，.TRUE. 和 .FALSE.

2.逻辑表达式：如前面所述，一共有有6个关系运算符

Relational Operator	Meaning	Example
.LT. or <	less than	A < 1e-5
.LE. or <=	less than or equal	B ** 2 .LE. 4 * A * C
.EQ. or ==	equal	B ** 2 == 4 * A * C
.NE. or /=	not equal	A /= 0
.GT. or >	greater than	B ** 2 - 4 * A * C > 0
.GE. or >=	greater than or equal	X >= 0

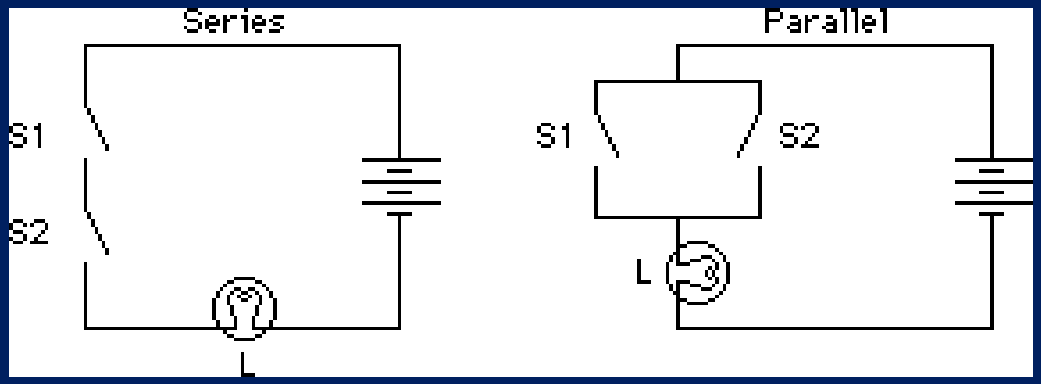
3.逻辑运算符： FORTRAN90有5个逻辑运算符

Logical Operator	Precedence	Meaning
.NOT.	1	logical negation
.AND.	2	logical intersection
.OR.	3	logical union
.EQV. and .NEQV.	4	logical equivalence and non-equivalence

逻辑类型 II

4. 逻辑变量：一个变量可以用LOGICAL定义为逻辑性变量

例子：




```
LOGICAL L1, L2, L3, L4, L5
REAL A, B, C
...
L1 = .TRUE.
L2 = B * B - 4 * A * C >= 0
L3 = A == 0
L4 = L1 .and. .not. L2 .or. L3
L5 = (L1 .and. (.not. L2)) .or. L3
```

5.位操作符: FORTRAN利用内部函数进行位操作。

例如：`IAND(I, J)`: logical `AND` on all corresponding bits of `I` and `J`.

GOTO语句

GOTO语句是一个无条件分支语句：

基本结构如下：

```
GO TO label
```

例子：

```
 IF (.NOT.L1) GOTO 10
   I = 1
   J = 2
   GOTO 30
10 IF (.NOT.L2) GOTO 20
   I = 2
   J = 3
   GOTO 30
20 I = 3
   J = 4
30 CONTINUE     ! Dummy statement - does nothing
```

注意：GOTO语句最好不要放在结构化模块里面，就是不要用GOTO跳出一个机构化模块，以至于破坏它的整体性。

总结

1. IF结构允许语句块的条件选择执行
2. IF语句允许单个语句的条件执行
3. IF结构可以有许多ELSE IF,但只能有一个ELSE语句
4. IF块可以命名
5. IF块可以嵌套
6. 逻辑常量有两个 .TRUE.　.FALSE.
7. 有6种关系运算符 > ,<, ==, >=, <=,/=
8. 有三个逻辑运算符，　.NOT. .AND. .OR.
9. Fortran可以用内部函数进行位操作
10. GOTO语句可以无条件分支。

第四章 循环结构

循环结构是另一种FORTRAN结构，它的基本机构如下：

[*name*:] DO *variable* = *expr1*, *expr2* [, *expr3*]

block

END DO [*name*]

Exp1:表达式1 循环起始点；exp2:表达式2循环结束点 ； exp3:循环步长

或

```
DO 100 I = 1, N
...
100   CONTINUE
```

例子：

```
DO I = 2, 7, 2
  WRITE( *, '(I3)', ADVANCE = 'NO' ) I
END DO
```

Output: 2 4 6

```
DO I = 5, 4
  WRITE( *, '(I3)', ADVANCE = 'NO' ) I
END DO
```

Output: (nothing)

```
DO I = 5, 1, -1
  WRITE( *, '(I3)', ADVANCE = 'NO' ) I
END DO
```

Output: 5 4 3 2 1

嵌套循环

嵌套循环是多层循环。例子：

```
IMPLICIT NONE
INTEGER     I                ! counter
INTEGER  :: N = 12           ! number of payments per year
INTEGER     K                ! repayment period (yrs)
INTEGER     TRIPS            ! iteration count
REAL     :: A = 1000         ! principal
REAL        P                ! payment
REAL        R                ! interest rate
REAL   R0, R1, RINC ! lowest, highest interest and increment
READ*, R0, R1, RINC
TRIPS = INT( (R1 - R0) / RINC + RINC/2 ) + 1
R = R0
PRINT*, "Rate      15 yrs    20 yrs    25 yrs"
PRINT*
DO I = 1, TRIPS
  WRITE( *, '(F5.2, "%")', ADVANCE = 'NO' ) 100 * R
  DO K = 15, 25, 5
    P = R/N * A*(1 + R/N) ** (N * K) / ((1 + R/N) ** (N * K) - 1)
    WRITE( *, '(F10.2)', ADVANCE = 'NO' ) P
  END DO
  PRINT*                     ! get a new line
  R = R + RINC
END DO
```

Some sample output (with input 0.1, 0.2, 0.01):

```
Rate      15 yrs    20 yrs    25 yrs

10.00%     10.75      9.65      9.09
11.00%     11.37     10.32      9.80
 ...
```

非确定性循环 I

前面讲的循环次数是固定的，下面介绍非确定性循环，即循环次数不确定

DO 条件退出：基本结构

```
DO
    IF (logical-expr) EXIT
    block
END DO
```

和

```
DO
      block
    IF (logical-expr) EXIT
END DO
```

例子：

```
DO
    WRITE( *, '("Your guess: ")', ADVANCE = 'NO' ) ! move up
    READ*, MyGuess                                 ! move up
    IF (MyGuess > FtnNum) THEN
      PRINT*, 'Too high.  Try again'
    ELSE IF (MyGuess < FtnNum) THEN      ! ELSE IF now
      PRINT*, 'Too low.  Try again'
    ELSE
      PRINT*, 'Well done!'               ! congrats here now
    END IF
  IF (MyGuess == FtnNum) EXIT                    ! move down
END DO
```

非确定性循环 II

DO WHILE是另一类法确定性循环。基本结构如下

```
DO  WHILE (logical-expr)
    block
END DO
```

例子：

```
REAL :: A, SUM
INTEGER :: N = 0
INTEGER :: IO = 0

OPEN( 1, FILE = 'DATA' )
SUM = 0

DO WHILE (IO == 0)
  READ (1, *, IOSTAT = IO) A
  IF (IO == 0) THEN
    SUM = SUM + A
    N = N + 1
    PRINT*, A
  END IF
END DO

PRINT*, "Mean:", SUM / N
```

另外CYCLE可以在循环体内作部分循环，但一般不提倡使用。

例子

下面我们给出一个完整的例子：

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

```
PROGRAM Taylor
! Computes sine(x) from Taylor series
REAL, PARAMETER :: Pi = 3.14159278
INTEGER    :: K         = 1      ! term counter
INTEGER    :: MaxTerms = 10      ! max number of terms
REAL       :: Err       = 1e-6   ! max error allowed
REAL          Sine                ! sum of series
REAL          Term                ! general term in series
REAL          X                   ! angle in radians
PRINT*, 'Angle in degrees?'
READ*, X
X = X * Pi / 180                  ! convert to radians
Term = X                          ! first term in series
Sine = Term
DO WHILE ((ABS( Term ) > Err) .and. (K <= MaxTerms))
  Term = - Term * X * X / (2 * K * (2 * K + 1))
  K = K + 1
  Sine = Sine + Term
END DO
IF (ABS( Term ) > Err) THEN                    ! why did DO end?
  PRINT*, 'Series did not converge'
ELSE
  PRINT*, 'After', K, 'terms Taylor series gives', Sine
  PRINT*, 'Fortran 90 intrinsic function: ', SIN( X )
END IF
END
```

总结

1.DO循环在程序中是确定次数的循环

2.DO循环的总次数参数N在执行中不可变

3.EXIT可以和IF,DO一起构成不定循环

4.DO WHILE可以形成不定循环

5.DO循环次数在循环体内不可变

6.IOSTAT可以在READ中确定文件的结束

7.可以进行嵌套循环

8.循环变量最好是整数

第五章 子程序和模块

一、引言：

子程序是指一段完整的程序单元，可以实现特定的任务，可以被其他的程序单元调用。FORTRAN90有两种重要的子程序：

1. 函数　　2. 子列行程序

它们和主程序是分离的。

子程序有内部和外部两种，有用的程序可以收集在一起形成一个库，这种收集叫做模块（**MODULES**).

程序单元：主程序，外部子程序，模块三者被成为程序单元

内部子程序和外部子程序的区别：

1. 内部子程序可以包含在另一个程序单元中一起编译，外部子程序不可如此。
2. 内部子程序可以使用程序单元声明的名字，而外部子程序不可包含在另一个程序单元。

内部子程序

有两种类型的内部子程序：函数和子列行程序

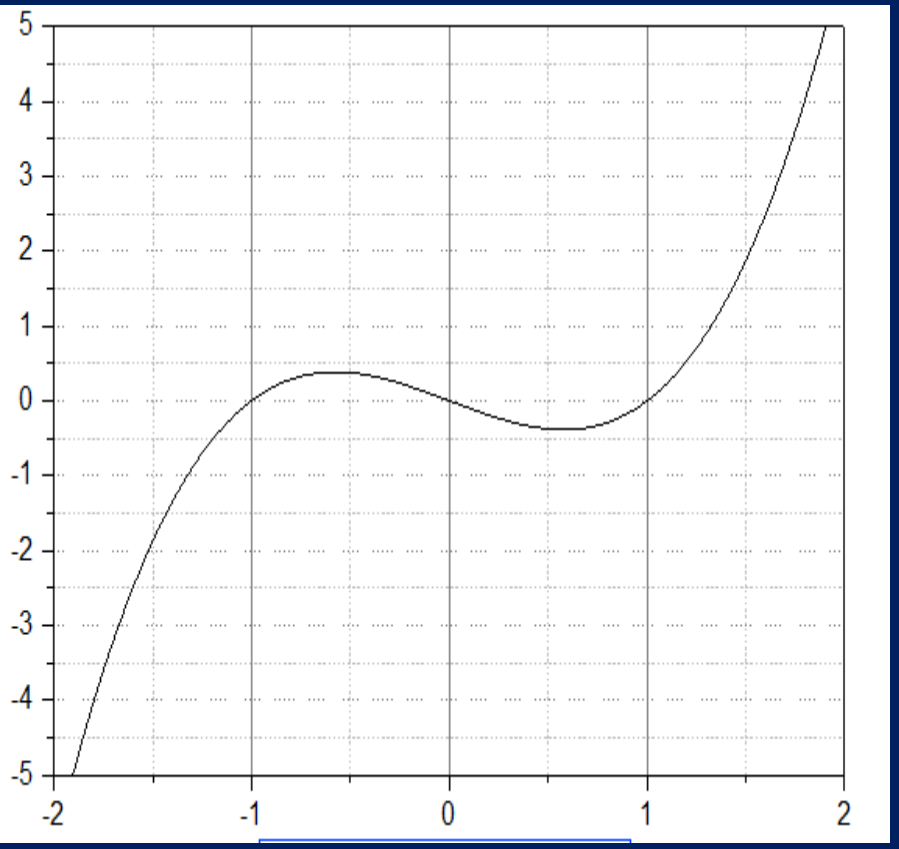
内部函数的基本语法结构如下：

```
FUNCTION Name( [argument  list] )
     [declaration statements]
     [executable statements]
     [RETURN]
END FUNCTION [Name]
```

内部函数被放在CONTAINS和主程序的END语句之间

函数可以通过END语句退出，也可以通过RETURN语句退出函数而返回到调用它的程序中。

Solves f(x) = 0 by Newton's method
$x^3-x=0$



2.牛顿法

2.1 定义

设已知方程$f(x)=0$有近似根x_k(假定$f'(x_k)\neq 0$),将函数$f(x)$在点x_k处展开,有

$$f(x)\approx f(x_k)+f'(x_k)(x-x_k),$$

于是方程$f(x)=0$可近似地表示为

$$f(x_k)+f'(x_k)(x-x_k)=0.$$

这是个线性方程,记其根为x_{k+1}，则x_{k+1}的计算公式为

$$x_{k+1}=x_k-\frac{f(x_k)}{f'(x_k)},\quad k=0,1,\ldots,$$

这就是牛顿法

例子

```
PROGRAM Newton
! Solves f(x) = 0 by Newton's method
IMPLICIT NONE
INTEGER  ::  Its       = 0              ! iteration counter
INTEGER  ::  MaxIts    = 20             ! maximum iterations
LOGICAL  ::  Converged = .false.        ! convergence flag
REAL     ::  Eps       = 1e-6           ! maximum error
REAL     ::  X         = 2              ! starting guess
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', its
ELSE
  PRINT*, 'Newton diverged',its
END IF
CONTAINS
  FUNCTION F(X)
  ! problem is to solve f(x) = 0
    REAL F, X
    F = X ** 3 + X
  END FUNCTION F
  FUNCTION DF(X)
  ! first derivative of f(x)
    REAL DF, X
    DF = 3 * X ** 2 + 1
  END FUNCTION DF
END PROGRAM Newton
```

局域和全局变量

先看一下下面一个例子：

```
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
END
```

此程序中，I是一个全局变量，它的值会被带到内部函数FACT中，而现在在函数内的I被重新赋值I, 使它变成局部变量。

内部子列行程序

子列行程序（SUBROUTINE)的语法结构：

```
SUBROUTINE Name[( argument list )]

           [declaration statements]
           [executable statements]

END SUBROUTINE [Name]
```

它的结构类似于函数，它们的区别在于：

1. 子列行程序不是通过名字返回函数值，所以不需要提前声明。
2. 子列行程序需要CALL语句进行调用。
3. 子列行程序可以没有任何虚参数。
4. 它是通过SUBROUTINE关键字来定义的。

例子

```
PROGRAM DAI
IMPLICIT NONE
REAL A, B

READ*, A, B
CALL SWOP( A, B )
PRINT*, A, B

CONTAINS
  SUBROUTINE SWOP( X, Y )
    REAL Temp, X, Y
    Temp = X
    X = Y
    Y = Temp
  END SUBROUTINE
END
```

主程序

每个完整的程序，必须包含一个且仅仅一个主程序

主程序的语法结构如下：

```
PROGRAM name
   [declaration statements]
   [executable statements]
[CONTAINS
   internal subprograms]
END [PROGRAM [name]]
```

外部子程序

外部子程序和主程序是分开的，可以单独编译它用于执行一个特定的任务，它的结构形式和内部子程序一样，只是不再包含在CONTIANS语句下了。

基本语法结构如下：

```
SUBROUTINE name[( arguments )]
  [declaration statements]
  [executable statements]
[CONTAINS
  internal subprograms]
END [SUBROUTINE [name]]
```

or

```
FUNCTION name([ arguments ])
  [declaration statements]
  [executable statements]
[CONTAINS
  internal subprograms]
END [FUNCTION [name]]
```

它与内部子程序的不同点：

外部子程序可以包含内部子程序，而内部子程序不可以包含一个内部子程序

例子

```
PROGRAM DAI
IMPLICIT NONE
EXTERNAL SWOP
   ! 此语句可以不要，但为了避免万一用重名的内部子程序，将
   ! 会引发错误，所以最好声明一下
REAL A, B

READ*, A, B
CALL SWOP( A, B )
PRINT*, A, B

END


SUBROUTINE SWOP( X, Y )
  REAL Temp, X, Y
  Temp = X
  X = Y
  Y = Temp
END SUBROUTINE
```

我们调用了一个外部子程序，如果有许多外部子程序，最好使用MODULE.

递归调用

许多数学问题需要自身调用，例如下面式子

$$n! = n * (n-1)!$$

实现它的程序如下：

program test

```
IMPLICIT NONE
INTEGER F, I

DO I = 1, 10
  CALL Factorial( F, I )
  PRINT*, I, F
END DO

CONTAINS
  RECURSIVE SUBROUTINE Factorial( F, N )
    INTEGER F, N
    IF (N == 1) THEN
      F = 1
    ELSE
      CALL Factorial( F, N-1 )
      F = N * F
    END IF
  END SUBROUTINE
END
```

第六章　数组

在一个程序中，经常会遇到一批具有同样性质的数据，为了避免定义过多的变量，引入了带标号的变量，即数组的概念，FORTRAN90的数组有许多FORTRAN77没有的性质。

数组的基本定义有如下几种如下：

1. DIMENSION X(10), Y(6,6), Z(1:10), H(1:6,1:6)

   REAL X,Y,Z,H

2. REAL X(10), Y(6,6), Z(1:10), H(1:6,1:6)

以上两种定义是FORTRAN77常用的方法，它们仍然被F90兼容。

在FORTRAN90中，数组的定义如下：

3.
```
REAL, DIMENSION(10) :: X
REAL, DIMENSION(0:100) :: A
REAL, DIMENSION(2,3) :: A
REAL, DIMENSION(5) :: A, B(2,3), C(10)
```

例子 1

```
IMPLICIT NONE
INTEGER                 :: I, N
REAL                    :: Std = 0
REAL, DIMENSION(100)    :: X
REAL                    :: XBar = 0
OPEN (1, FILE = 'DATA')
READ (1, *) N
DO I = 1, N
  READ (1, *) X(I)
  XBar = Xbar + X(I)
END DO
XBar = XBar / N
DO I = 1, N
  Std = Std + (X(I) - XBar) ** 2
END DO
Std = SQRT( Std / (N - 1) )
PRINT*, 'Mean: ', XBar
PRINT*, 'Std deviation: ', Std
```

文件数据如下

10 5.1 6.2 5.7 3.5 9.9 1.2 7.6 5.3 8.7 4.4

读完数据后，在内存的存储形式如下：

X(1)	*X*(2)	*X*(3)	...	*X*(10)
5.1	6.2	5.7	...	4.4

读数据可以使用下列方式替代循环读数

```
READ (1, *) ( X(I), I = 1, N )
```

例子 2

我们可以使用DO循环来读一批数据，加上在READ语句中加入IOST特性可以读一批未知数目的数据。但总数不应该超出数组的维数

例子：

```
 INTEGER, PARAMETER    :: MAX = 100
REAL, DIMENSION(MAX) :: X

OPEN (1, FILE = 'DATA')
READ( 1, *, IOSTAT = IO ) ( X(I), I = 1, MAX )

IF (IO < 0) THEN
  N = I - 1
ELSE
  N = MAX
END IF

PRINT*, ( X(I), I = 1, N )
...
```

数组作为子程序虚参

```
IMPLICIT NONE
INTEGER                :: I, N
REAL, DIMENSION(100)   :: X
REAL                   :: Std = 0
REAL                   :: XBar = 0
OPEN (1, FILE = 'DATA')
READ (1, *) N, (X(I), I = 1, N)
CALL Stats( X, N, XBar, Std )
PRINT*, 'Mean:          ', XBar
PRINT*, 'Std deviation: ', Std
CONTAINS
  SUBROUTINE Stats( Y, N, YBar, S )
    REAL, DIMENSION(:), INTENT(IN) :: Y
    REAL, INTENT(INOUT)            :: S, YBar
    INTEGER, INTENT(IN)            :: N
    INTEGER  I
    YBar = 0; S = 0
    DO I = 1, N
      YBar = Ybar + Y(I)
    END DO
    YBar = YBar / N
   DO I = 1, N
      S = S + (Y(I) - YBar) ** 2
    END DO

    S = SQRT( S / (N - 1) )
  END SUBROUTINE
END
```

分配动态数组

在F90中区别于F77的是它的动态数组分配

到目前为止，我们见到的变量类型都是静态的，即一旦声明了，就被分配了一定的内存地址，只要程序运行，它就一直存在，但被定义为动态后，则只有在运行的时候才被分配内存。

基本语法结构如下：

```
REAL, DIMENSION(:), ALLOCATABLE :: X
```

通过 ALLOCATE(X(N))语句来获取内存

而通过 DEALLOCATE(X) 来释放内存。

例子：

```
REAL, DIMENSION(:), ALLOCATABLE :: X, OldX
REAL      A ; INTEGER   IO, N

ALLOCATE( X(0) )                    ! size zero to start with?
N = 0
OPEN( 1, FILE = 'DATA' )
DO
  READ(1, *, IOSTAT = IO) A
  IF (IO < 0 ) EXIT
    N = N + 1
    ALLOCATE( OldX( SIZE(X) ) )
    OldX = X                        ! entire array can be assigned
    DEALLOCATE( X )
    ALLOCATE( X(N) )
    X = OldX
    X(N) = A
    DEALLOCATE( OldX )
END DO
PRINT*, ( X(I), I = 1, N )
```

实例

```
integer, parameter:: row = 5
integer err_mesg
integer ary1(:), ary11(:, :, :)
allocatable ary1, ary11          !第一种声明动态数组的方式
integer, allocatable:: ary2(:), ary22(:)      !第二种声明动态数组的方式
integer, dimension(:), allocatable::  ary3, ary33 !第三种声明动态数组的方式
!下面是测试部分
allocate(ary1(1:5), ary11(3, 3, 3), stat = err_mesg)
if(0 .eq. err_mesg) then
    print *, "ary1 initialize successed!"
else if(0 .ne. err_mesg) then
    print *, "ary1 initialize failed", err_mesg
end if !以上测试正常分配的情况，分配成功stat = 0
```

实例

```
    !查看动态数组是否分配空间
    if(.not. allocated(ary2))  print *, 'not allocated ary2'
    if(.false. .eqv. allocated(ary22)) print *, 'not allocated ary22'
    if(.true. .eqv. allocated(ary1))  print *, 'allocated ary1'
    !已被分配空间的数组不可重复分配
    allocate(ary11(2, 2, 2), stat = err_mesg)
    if(0 .ne. err_mesg)  print *, 'Ooops, ary11 has been allocated'
    !内存的释放，只有被allocate的内存才能用deallocate释放，否则报错
    deallocate(ary1, stat = err_mesg)
    if(0 .ne. err_mesg) then
       print *, 'ary1 is not allocated'
    else if(0 .eq. err_mesg) then
       print *, 'ary1 del successfully'
    end if
    deallocate(ary3, stat = err_mesg)
    if(0 .ne. err_mesg) then
       print *, 'ary3 is not allocated'
    else if(0 .eq. err_mesg) then
       print *, 'ary3 del successfully'
    end if
end
```

```
!上述程序的输出:
! ary1 initialize successed!
! not allocated ary2
! not allocated ary22
! allocated ary1
! Ooops, ary11 has been allocated
! ary1 del successfully
! ary3 is not allocated
```

数组特性 I

数组的选择

可以使用下列特殊的数组选择：

```
X(I:J)     ! rank one array of size J-I+1
Y(I, 1:J) ! rank one array of size J
X(2:5, 8:9) ! rank one array of size 4+2
A(J, :)       ! the whole of row J
REAL, DIMENSION(10) :: A = (/ (I, I = 2, 20, 2) /)
INTEGER, DIMENSION(5) :: B = (/ 5, 4, 3, 2, 1 /)
PRINT*, A(B)
```

数组特性 II

All or part of an array may be initialized in a `DATA` statement. There are a number of possibilities, e.g.

```
REAL, DIMENSION(10) :: A, B, C(3,3)
DATA A / 5*0, 5*1 /
     ! first 5 zero, last 5 one
DATA B(1:5) / 4, 0, 5, 2, -1 /
     ! section 1:5 only
DATA ((C(I,J), J= 1,3), I=1,3) / 3*0, 3*1, 3*2 /
       ! by rows
```

数组特性 III

在FORTRAN90中可以直接进行数组的运算和分配。

```
REAL, DIMENSION(10) :: X, Y

X + Y          ! result has elements X(I) + Y(I)
X * Y          ! result has elements X(I) * Y(I)
X * 3          ! result has elements X(I) * 3
X * SQRT( Y )  ! result has elements X(I) * SQRT( Y(I) )
X == Y         ! result has elements .TRUE. if X(I) == Y(I),
               ! .. and .FALSE. Otherwise
```

数组分配

```
REAL, DIMENSION(10)   :: X, Y
REAL, DIMENSION(5,5)  :: A, C
X = Y                  ! both rank one with same size
Y = 0                  ! Y full of zeroes
X = 1 / X              ! replace each element of X
                       !   with its reciprocal
X = COS( X )    ! replace each element of X with its cosine
X(1:5) = Y(4:8)        ! both rank one with size 5
A(I, 1:J) = C(K, 1:J)  ! row K of matrix C assigned to row I
                       !   Of matrix A
```

数组特性 IV

数组的WHERE结构

WHEREconstruct 可以对数组的特定元素进行运算

```
WHERE (A > 0)
  A = LOG( A )        ! log of all positive elements
ELSEWHERE
  A = 0          ! all non-positive elements set to zero
END WHERE
```

COMMON和BLOCK DATA

1. COMMON语句：定义可以全局数据存储机制，便于在程序单元之间共享数据.

   例子：

```
PROGRAM MyProg
 COMMON i, j, x, k(10)
 COMMON /mycom/ a(3)
  ...
END

SUBROUTINE MySub
  COMMON pe, mn, z, idum(10)
  COMMON /mycom/ a(3)
  ...
END
```

2. BLOCK DATA：定义一个数据块子程序，在那里，以COMMON数据块命名的变量和数组元素被初始化。

   例子：

```
 Program main
 CHARACTER(LEN=10) family
 REAL X(10), Y(4)
 integer a,b
 COMMON/Lakes/a,b,family/Blk2/x,y
Write(*,*) a,b,family
 end
 BLOCK DATA InitLakes
  CHARACTER(LEN=10) fname
  integer erie, huron
  COMMON /Lakes/ erie, huron, fname
  DATA erie, huron, fname /1,2,'GreatLakes'/
 END
```