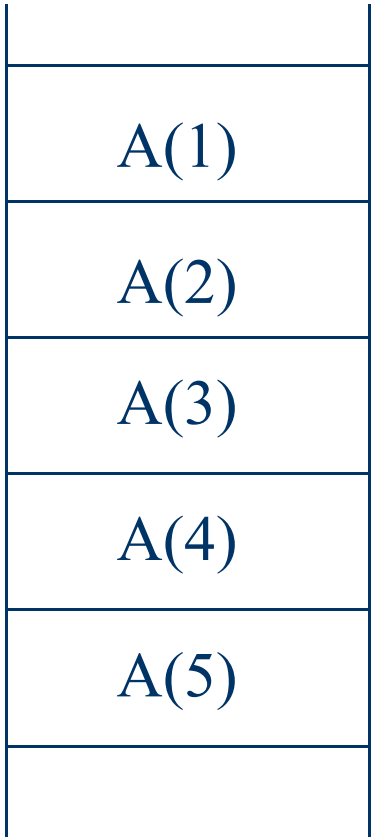
Fortran95 程序设计

彭国伦 编著

第7章 数组

7-0 概述

- 数组是一组类型全相同、且用单个名字来引用的变量或常量，这组数值占用计算机内存中的连续若干个位置，数组中的单个数值称为数组元素，数组元素用数组名和指向特定数组位置的下标来标识。
- 如图：第一个变量用a(1)引用，第五个用a(5)引用。



7-1 基本使用

数组（ARRAY）是一种使用数据的方法。它可以配合循环等的功能，用很精简的程序代码来处理大量的数据

7-1-1 一维数组

- 数组可一次声明出一长串同样数据类型的变量
- 数组也是一种变量，使用前要声明

Datatype　name(size)

数组的大小，必须使用整型常数

数组变量的名字

Datatype指数组的类型，除了4种基本类型(integer,real,complex,logical)之外，也可用type自订出来的类型

记录成绩并按座号查询成绩

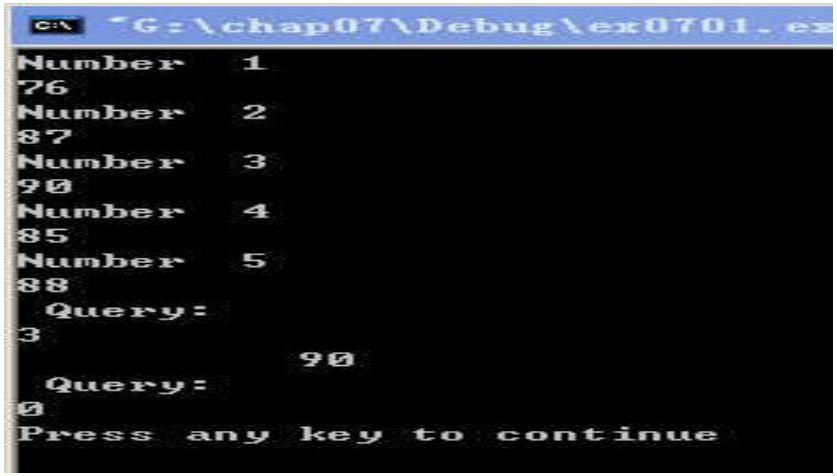
[ex0701.f90]

```
program ex0701
implicit none


 integer, parameter :: students = 5
  integer :: student(students)
  integer i


do i=1, students
    write(*,"('Number ',I2)") i
    read(*,*) student(i)
end do
```

```
do while( .true. )
    write(*,*) "Query:"
    read(*,*) i
    if ( i<=0 .or. i>students ) exit
    write(*,*) student(i)
  end do
 stop
End
```



在声明数组时，要说明数组的大小。在声明时，只能使用常数来赋值数组的大小，常数包括直接填入数字或是使用声明为 parameter 的常数。

使用数组时，配合括号，再加上索引值就可以使用其中的数组元素。数组的索引值不仅可以使用常数，还可以使用一般的变量。

使用数组时超出范围是很危险的，绝对要避免发生这种情况。在编译过程中，通常不会检查数组使用是否超出范围，而且很多情况下根本也无法在编译过程中就发现这种错误。这个责任都是由程序员自行负担。

记录成绩并按座号查询成绩

[ex0702.f90]

```
program ex0702
implicit none
  integer :: student1
  integer:: student2
  integer:: student3
  integer:: student4
  integer:: student5
  integer :: i
write(*,*) "Number 1"
 read(*,*) student1
 write(*,*) "Number 2"
 read(*,*) student2
 write(*,*) "Number 3"
 read(*,*) student3
 write(*,*) "Number 4"
 read(*,*) student4
 write(*,*) "Number 5"
 read(*,*) student5
```



```
do while(.true.)
  write(*,*) "Query:"
          read(*,*) i
          select case(i)
          case(1)
            write(*,*) student1
          case(2)
            write(*,*) student2
          case(3)
            write(*,*) student3
          case(4)
            write(*,*) student4
          case(5)
            write(*,*) student5
          case default
            exit
          end select
  end do
  stop
end
```

如果不用数组，程序很长

7-1-1 一维数组

比较 EX0701.f90　（“聪明的写法”）
　　EX0702.f90　（“笨的写法”）

假如程序中要处理100，1000，10000个学生，
这个程序就不知道怎么写了！！！



Fortran中声明数组的语法：

Datatype name(size) !最简单的方法
Datatype, dimension(size) :: name !另一种方法

Datatype name !这是Fortran 77的做法，先声明name类型
dimension name(size) !再声明name是大小为size的数组

Integer a(10) → 最简单的方法

Integer, dimension(10):: a → 另外一种写法

Integer a

Dimension a(10) → Fortran 77的写法

数组除了可以使用基本的4种类型外，还可以使用自定义类型。

```
Type :: person
    real :: height, weight
End type
Type(person) :: a(10) !用person这个新类型来声明数组
……
……
!同样在变量后面加上“%”来使用person类型中的元素
a(2)%height = 180.0
a(2)%weight = 70.0
```



7-1-2 二维数组

- 声明数组大小时，如使用两个数字，它就变成二维数组。使用二维数组时要给两个坐标索引值。

```
program ex0703
implicit none
  integer, parameter :: classes  = 5
  integer, parameter :: students = 5
  integer :: student(students, classes)
  integer s ! 用来指定学生号码
  integer c ! 用来指定班级号码

  do c=1, classes
    do s=1, students
      write(*,"('Number ',I2,' of class ',I2)") s,c
      read(*,*) student(s,c) ! 第c班的第s位学生
    end do
  end do
```

双循环给二维数组赋值

```
do while( .true. )
    write(*,*) "class:"
    read(*,*) c
    if ( c<=0 .or. c>classes ) exit
              write(*,*) "student:"
              read(*,*) s
    if ( s<=0 .or. s>students ) exit
    write(*,"('score:',I3)") student(s,c)
 ! 第c班的第s位学生
  end do

  stop
End
```

“死循环”进行查询输出，退出条件为班级或成绩超出范围。

输入2×2矩阵，然后相加

```
program ex0704
implicit none
  integer, parameter :: row = 2
  integer, parameter :: col = 2
  integer :: matrixA(row,col)
  integer :: matrixB(row,col)
  integer :: matrixC(row,col)
  integer r ! 用来指定row
  integer c ! 用来指定column
  ! 读入矩阵A的内容
  write(*,*) "Matrix A"
  do r=1, row
    do c=1, col
      write(*,"('A(',I1,',',I1,')=')") r,c
                read(*,*) matrixA(r,c)
              end do
  end do
 ! 读入矩阵B的内容
write(*,*) "Matrix B"
  do r=1, row
    do c=1, col
      write(*,"('B(',I1,',',I1,')=')") r,c
                read(*,*) matrixB(r,c)
              end do
  end do
  ! 把矩阵A,B相加并输出结果
  write(*,*) "Matrix A+B="
  do r=1, row
    do c=1, col
                matrixC(r,c) =
matrixB(r,c)+matrixA(r,c) ! 矩阵相加
                write(*,"('(',I1,',',I1,')=',I3)")
r,c,matrixC(r,c)
              end do
  end do
  stop
end
```

7-1-3 多维数组

- Fortran最多可以声明高达七维的数组
- 使用多维数组，头脑一定要清醒，因为会很容易把坐标位置搞混。

```
Integer a(D1, D2,……, Dn) !n维数组
a(I1, I2,……, In)

!使用n维数组时要给n个坐标值
```



使用三维数组改写矩阵相加程序

[ex0705.f90]

```
program ex0705
implicit none
  integer, parameter :: row = 2
  integer, parameter :: col = 2
  integer :: matrix(row, col, 3)
  integer m ! 用来指定第几个矩阵
  integer r ! 用来指定row
  integer c ! 用来指定column

  ! 读入矩阵的内容
  do m=1, 2
  write(*,"('Matrix ',I1)") m
    do r=1, row
      do c=1, col
        write(*,"('(',I1,',',I1,')=')") r,c
                read(*,*) matrix(r,c,m)
              end do
    end do
  end do
  ! 把第1,2个矩阵相加
  write(*,*) "Matrix 1 + Matrix 2 = "
  do r=1, row
    do c=1, col
                matrix(r,c,3) =
matrix(r,c,1)+matrix(r,c,2) ! 矩阵相加
                write(*,"('(',I1,',',I1,')=',I3)")&
r,c,matrix(r,c,3)
              end do
  end do

  stop
end
```

7-1-4 另类的数组声明

- 在没有特别赋值的情况下，数组的索引值都是由1开始。
  - Integer a(5) !a(1), a(2), a(3), a(4), a(5)
- 在声明时可以特别赋值数组的坐标值使用范围
  - Integer a(0:5) ! a(0), a(1), a(2), a(3), a(4), a(5)
  - Integer a(-3:3) ! a(-3), a(-2), a(-1),a(0), a(1), a(2), a(3)
  - integer a(5, 0:5) !a(1~5, 0~5)是可以使用的元素
  - integer b(2:3, -1:3) ! b(2~3, -1~3)是可以使用的元素



7-2 数组内容的设置

- 与普通变量相同，使用数组之前，必须初始化数组。

```
INTEGER, DIMENSION(10):: J

WRITE(*,*)  ‘J(1)=‘, J(1)
```

没有初始化之前，数组元素的值都是不确定的，随计算机的不同而不同。



7-2-1 DATA语句赋初值

使用**DATA**命令设置数组的初值：

```
INTEGER A(5)
DATA  A  /1,2,3,4,5/
```

结果:

A(1)=1、 A(2)=2、 A(3)=3、 A(4)=4、 A(5)=5

DATA的数据区还可以使用星号“*”表示数据重复。

```
INTEGER A(5)
DATA  A  /5*3/
```

结果:

A(1)=3、 A(2)=3、 A(3)=3、 A(4)=3、 A(5)=3



“隐含式”循环设置数组初值

```
INTEGER A(5)
INTEGER  I
DATA (A(I),I=2,4)/2,3,4/
```

这就是一个“隐含式”循环，I会从2增加到4，依照顺序到后面取数字

结果:

A(2)=2、 A(3)=3、 A(4)=4、 A(1)和A(5)没有设定

“隐含式”循环还可应用于输出数组内容

```
write(*, *)(a(i), I=2, 4)
!显示a(2)、a(3)、a(4)的值
```

“隐含式”循环只要在最后面再多加一个数字，同样可以改变计数器的累加数值。

(A(I), I=2, 10, 2)

!循环执行5次，I分别为2，4，6，8，10

“隐含式”循环也可以是多层嵌套的，也可以应用在多维数组上。

```
INTEGER A(2, 2)
INTEGER I, J
DATA((A(I, J), I=1, 2), J=1, 2)/1, 2, 3, 4/
```

里面括号的循环会先执行

结果为A(1, 1)=1、 A(2, 1)=2、 A(1, 2)=3、 A(2, 2)=4

Fortran 90中，可以省略掉DATA描述，直接设置初值。

```
INTEGER :: A(5)=(/1, 2, 3, 4, 5/)
```

注意括号跟除号之间不能有空格

结果: A(1) =1 、 A(2)=2、 A(3)=3、 A(4)=4、A(5)=5

省略DATA直接赋初值时使用隐含式循环：

```
INTEGER :: I
INTEGER :: A(5)=(/1, (2, I=2, 4), 5/)
```

“隐含式”循环， A(2)=2、A(3)=2、 A(4)=2

结果： A(1) =1 、 A(2)=2、 A(3)=2、 A(4)=2、A(5)=5

省略DATA直接把初值写在声明后面时，不能只对数组的部分元素设置初值，每个元素都必须给定初值。使用隐含式循环时也要给出全部元素的初值。

```
integer :: I
integer ;; a(5)=(/(2, I=2, 4)/)
! (2, I=2, 4)是一个隐含式循环，表示a(2)=2, a(3)=2, a(4)=2，这里没有给出a(1)和a(5)，会发生错误
```

```
integer :: I
integer ;; a(5)=(/1, (2, I=2, 4), 5/)
!a(1~5)都给定初值，a(1)=1, a(2)=2, a(3)=2, a(4)=2, a(5)=5，这种声明方法是正确的。
```

成绩直接记录在程序代码中的查询成绩程序

[ex0706.f90]

```
program ex0706
implicit none
  integer, parameter :: students = 5
  integer :: student(students) = (/ 80, 90, 85, 75, 95 /)
  integer i

  do while( .true. )
    write(*,*) "Query:"
    read(*,*) i
    if ( i<=0 .or. i>students ) exit
    write(*,*) student(i)
  end do

  stop
end
```



设置二维矩阵的内容，再显示在屏幕上

[ex0707.f90]

```
program ex0707
implicit none
  integer, parameter :: row = 2
  integer, parameter :: col = 2
  integer :: m(row, col)
  integer r ! 用来指定row
  integer c ! 用来指定column
  data ((m(r,c), r=1, 2), c=1,2) /1,2,3,4/

  ! 依次输出m(1,1), m(1,2), m(2,1), m(2,2)这4个数字,
  write(*,"(I3,I3,/,I3,I3)") (( m(r,c), c=1,2 ), r=1,2)

  stop
end
```

注意赋值和输出的区别



- 也可以通过赋值语句初始化数组：

```
REAL, DIMENSION(10):: ARRAY1
DO I=1, 10
   ARRAY(I)=REAL(I)
END DO
```



7-2-2 对整个数组的操作

可以通过简单的命令来操作数组

【a=5】 数组a的所有元素赋值为5

【a=(/1, 2, 3/)】 a(1)=1、a(2)=2、a(3)=3

等号右边所提供的数字数目，必须跟数组a的大小一样。

【a=b】

把数组a同样位置元素的内容设置成和数组b一样。要求a和b是同样维数及大小的数组。



【a=b+c】

【a=b-c】

【a=b*c】

【a=b/c】

把数组b及c中同样位置的数值相加、减、乘、除，得到的数值再放回数组a同样的位置中。

要求a,b,c维数和大小相同。

【a=sin(b)】

数组a的每一个元素为数组b对应元素的sin值。数组b必须是浮点数类型。内部函数都可以这样使用。

【a=b>c】

a, b, c是三个同样维数及大小的数组，数组a为逻辑类型数组，数组b, c为同类型的数值变量。如果数组b的元素大于数组c中同样位置的元素值，把.true.放回到数组a同样的位置中，否则把.false.放回到数组a同样的位置中。



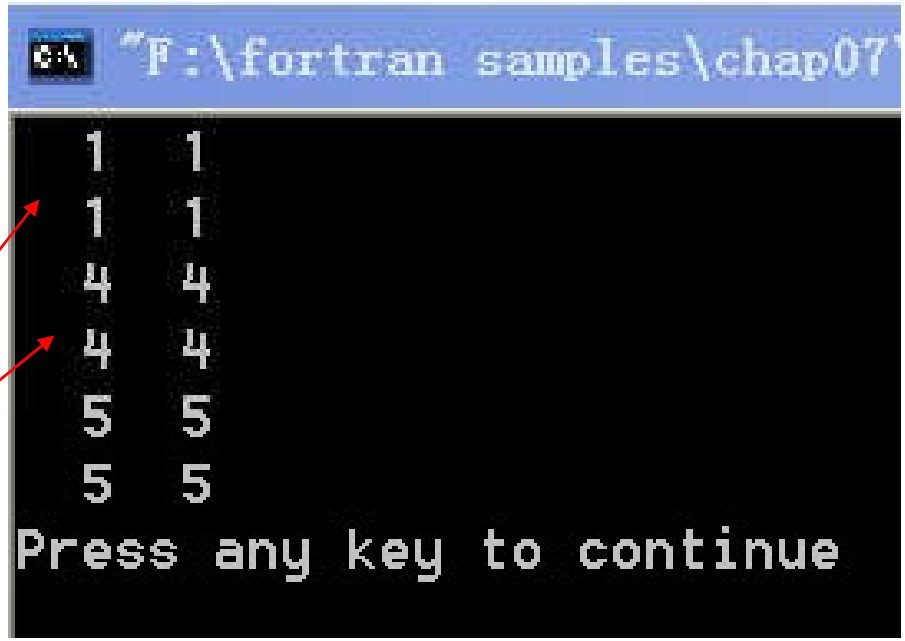
矩阵相加

[ex0708.f90]

```
program ex0708
implicit none
  integer, parameter :: row = 2
  integer, parameter :: col = 2
  integer :: ma(row, col) = 1
  integer :: mb(row, col) = 4
  integer :: mc(row, col)
  integer :: i,j

  mc = ma + mb ！一行程序码就可以做矩阵相加
  write(*,"(I3,I3,/,I3,I3)") ((mc(i,j), j=1,2),i=1,2)

  stop
end
```



声明时直接初始化数组，赋值的一种形式

7-2-3 对部分数组的操作

F90、F95还提供一次只挑出部分数组来操作的功能。

【a(3:5)=5】

！把a(3)、a(4)、a(5)的内容设置为5，其他值不变

【a(3:)=5】

！把a(3)之后的元素内容都设置为5， a(1)、 a(2)则不变

【a(3:5)=(/3, 4, 5/)】

！执行a(3)=3、a(4)=4、a(5)=5的设置，其他值不变。

！等号左边所赋值的元素数目必须跟等号右边所提供的数字数量一样多。

【a(1: 3)=b(4: 6)】

！设置a(1)=b(4)、 a(2)=b(5)、 a(3)=b(6)，等号两边的数组元素数量必须一样多。

【a(1:5:2)=3】
!设置a(1)=3、a(3)=3、a(5)=3

隐含式循环，从1到5，步长为2

【a(1: 10)=a(10:1:-1)】
!把a(1~10)的内容翻转

【a(:)=b(:, 2)】
!要求等号两边的元素数目一样多。
!执行结果为a(i)=b(i, 2)

把二维数组的部分元素赋值给一维数组

【a(:, :)=b(:, :, 1)】
!要求等号两边的元素数目一样多。
!执行结果为a(i, j)=b(i, j, 1)

把三维数组的部分元素赋值给二维数组

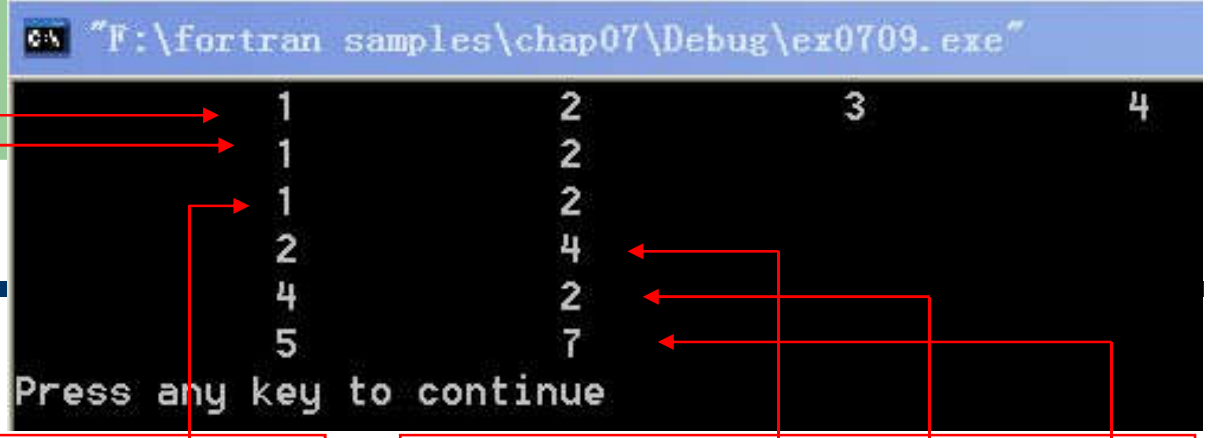


拿数组中一部分内容来使用时，需要把握两个原则：

- 等号两边所使用的数组元素数目要一样多
- 同时使用多个隐含式循环时，较低维的循环可以想像成是内层循环。

实例：

```
integer :: a(2, 2), b(2, 2)
b=a(2:1:-1, 2:1:-1)
!b的下标没特别赋值时，等于b(1:2:1, 1:2:1)
!低微的是内层循环，会先执行。所以这个命令的结果为
!b(1, 1)=a(2, 2)
!b(2, 1)=a(1, 2)
!b(1, 2)=a(2, 1)
!b(2, 2)=a(1, 1)
```



[ex0709.f90]

```
program ex0709
implicit none
  integer, parameter :: row = 2
  integer, parameter :: col = 2
  integer :: a(2,2)=(/ 1,2,3,4 /)
  ! a(1,1)=1, a(2,1)=2,
  ! a(1,2)=3, a(2,2)=4
  integer :: b(4)=(/ 5,6,7,8 /)
  integer :: c(2)

  write(*,*) a
   ! 写出a(1,1), a(2,1), a(1,2), a(2,2)
  write(*,*) a(:,1)
   ! 写出a(1,1), a(2,1)

   c = a(:,1)
  ! c(1)=a(1,1), c(2)=a(2,1)
   write(*,*) c ! 写出c(1), c(2)

   c = a(2,:)
   ! c(1)=a(2,1), c(2)=a(2,2)
   write(*,*) c ! 写出c(1), c(2)
   write(*,*) c(2:1:-1)
   ! 写出c(2), c(1)

   c = b(1:4:2) ! c(1)=b(1), c(2)=b(3)
   write(*,*) c ! 写出c(1), c(2)

   stop
end
```

7-2-4 WHERE

WHERE是F95添加的功能，可以经过逻辑判断来取出部分数组内容来设置。

[ex0710.f90]

```
program ex0710
implicit none
  integer :: i
  integer :: a(5)=(/ (i,i=1,5) /)
  integer :: b(5)=0
  ! 把a(1～5)中小于3的元素值赋值给b
  where( a<3 )
    b = a
  end where

  write(*,"(5(I3,1X))") b
  stop
end
```

和使用DO循环比较，使用WHERE命令的程序代码比较精简，执行起来也会比较快。而且可以拿来做并行处理，而DO循环则不能拿来做并行处理。

程序模块中只有一行命令时，可以把这一行命令写在WHERE后面，并且省略END WHERE。

WHERE是用来设置数组的，所以它的程序模块中只能出现与设置数组相关的命令。

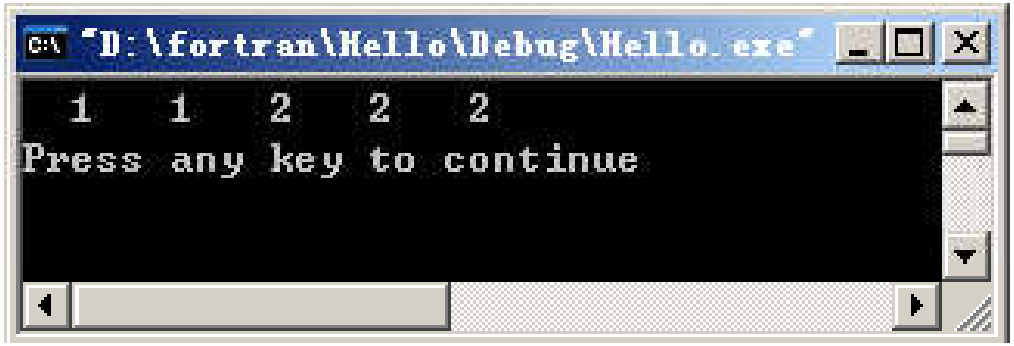
在整个程序模块中所使用的数组变量，都必须是同样维数及大小的数组。



WHERE除了可以处理逻辑成立的情况外，还可以配合ELSEWHERE来处理逻辑不成立的情况。

[ex0711.f90]

```
program ex0711
implicit none
  integer :: i
  integer :: a(5)=(/ (i,i=1,5) /)
  integer :: b(5)=0
  where( a<3 )
    b = 1
  elsewhere
    b = 2
  end where
  write(*,"(5(I3,1X))") b
  stop
end
```



相当于

```
do i=1, 5
   if (a(i)<3) then
      b(i)=1
   else
      b(i)=2
   end if
end do
```

WHERE描述还可以做多重判断。只要在ELSEWHERE后面接上逻辑判断就行了。

```
where (a<2)
   b=1
elsewhere(a>5)
   b=2
elsewhere !剩下2<=a(i)<=5的部分
   b=3
end where
```

相当于

```
do i=1, n
   if(a(i)<2) then
      b(i)=1
   else if(a(i)>5) then
      b(i)=2
   else
      b(i)=3
   end if
end do
```



WHERE也可以是嵌套的，并且可以取名字。不过取名的WHERE描述在结束时END WHERE后面一定要接上名字，用来明确所要结束的是哪一个WHERE模块。

```
where (a<5)
    where (a/=2)
        b=3
    elsewhere
        b=1
    end where
elsewhere
    b=0
end where

name: where(a<5) !where模块可以取名字
   b=a
end where name     ！有取名字的where结束时也要赋值名字
```



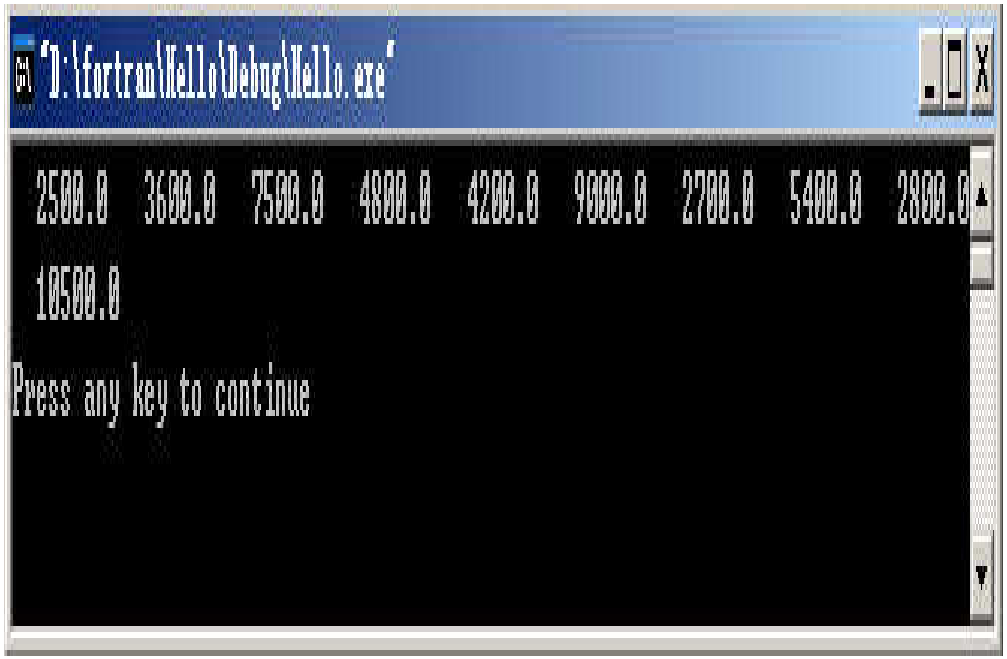
所得税分级计算程序

```
program ex0712
implicit none
  integer :: i
  real :: income(10)=(/ 25000, 30000, 50000, 40000, 35000, &
                        60000, 27000, 45000, 28000, 70000 /)
  real :: tex(10)=0

  where( income < 30000.0 )
    tex = income*0.10
  elsewhere( income < 50000.0 )
    tex = income*0.12
  elsewhere
    tex = income*0.15
  end where

  write(*,"(10(F8.1,1X))") tex

  stop
end
```



7-2-5 FORALL

FORALL是F95添加的功能，是一种使用隐含式循环来使用数组的方法。

[ex0713.f90]


```
program ex0713
implicit none
  integer i
  integer :: a(5)
  forall(i=1:5)
    a(i)=5
  end forall
  ! a(1)=a(2)=a(3)=a(4)=a(5)=5
  write(*,*) a
  forall(i=1:5)
    a(i)=i
  end forall
  ! a(1)=1, a(2)=2, a(3)=3, a(4)=4, a(5)=5
  write(*,*) a

  stop
end
```

FORALL的详细语法为：

FORALL(triplet1 [, triplet2 [,triplet3……]], mask)

$triplet_n$是使用隐含式循环来赋值数组坐标范围的值。

FORALL中可以赋值好几个triplet，数组最多有几维就可以赋值多少个。

mask用来做条件判断。可以用来限定FORALL程序模块中，值作用于数组中符合条件的元素，还可以做其他的条件限制。

```
integer :: a(10, 5)
forall (I=2: 10: 2, J=1: 5) !二维数组可以用两个数字
    a(I, J)=I+J                    !可以使用算式
end forall
```



```
integer :: a(5, 5)
integer :: I, J
……
……
forall(I=1: 5, j=1: 5, a(I, J)<10) !只处理数组a中小于10的元素
   a(I, J)=1
end forall
```

```
forall( I=1: 5, J=1: 5, I==J) !只做I==J的情况，也就是只处理
!a(1, 1), a(2, 2), a(3, 3), a(4, 4), a(5, 5)这5个元素
   a(I, J)=1
end forall
```

```
forall (I=1: 5, J=1: 5, ((I>J) .and. a(I, J)>0) !还可以赋值好几个条件
!这个条件可以想像成只处理a(5, 5)这个二维矩阵的上半部三角形部分
!而且a(I, J)大于0的元素
   a(I, J)=1/a(I, J)
end forall
```

FORALL描述中的程序模块如果只有一行程序代码时，可以省略END FORALL，把程序模块跟FORALL写在同一行。

```
forall(I=1: 5, J=1:5, a(I, J)/=0) a(I, J)=1/a(I, J)
!模块中只有一行时可以省略end forall。
```

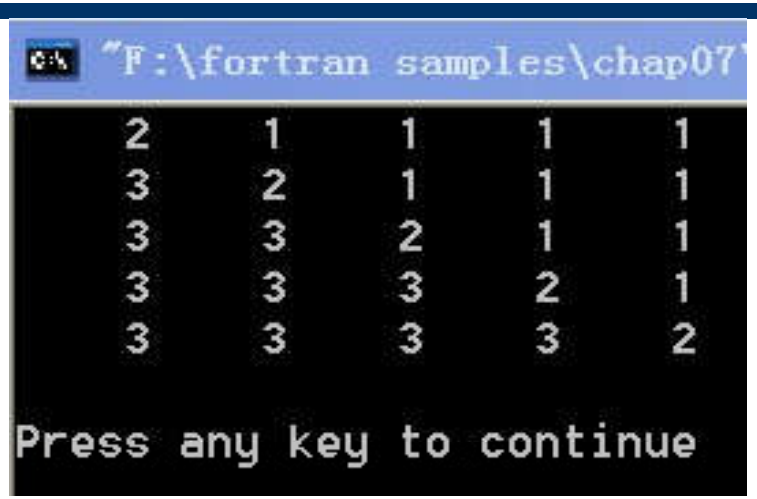


使用forall设置二维矩阵程序

```
program ex0714
implicit none
  integer I,J
  integer, parameter :: size = 5
  integer :: a(size,size)

  forall ( I=1:size, J=1:size, I>J )  a(I,J)=1 ！上半部分
  forall ( I=1:size, J=1:size, I==J ) a(I,J)=2 ！对角线部分
  forall ( I=1:size, J=1:size, I<J )  a(I,J)=3 ！下半部分

  write(*,"(5(5I5,/))") a
  stop
end
```



FORALL可以写成多层的嵌套结构，但里面只能出现跟设置数组数值相关的程序命令。还可以在FORALL中使用WHERE。不过WHERE中不能使用FORALL。

```
forall(I=1: 5) !嵌套的forall
    forall(J=1: 5)
        a(I, J)=2
    end forall
    forall(J=6: 10)
        a(I, J)=4
    end forall
end forall
```

```
forall(I=1: 5)
    where(a(:, I)/=0)
     !forall中可以使用where
        a(:, I)=1/ a(:, I)
    end where
end forall
```

7-3 数组的保存规则

一个数组不管是声明成什么“形状”（指维数和大小），它所有元素都是分布在计算机内存的同一个连续模块当中。了解这些，在程序优化时可编写出效率较高的代码。

一维数组的元素在内存中依照元素的顺序排列。

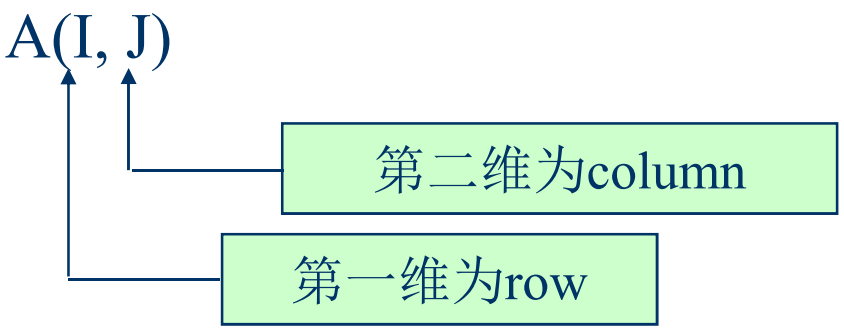
INTEGER A(5)
元素在内存连续模块中的排列情况为
A(1)=> A(2)=> A(3)=> A(4)=> A(5)

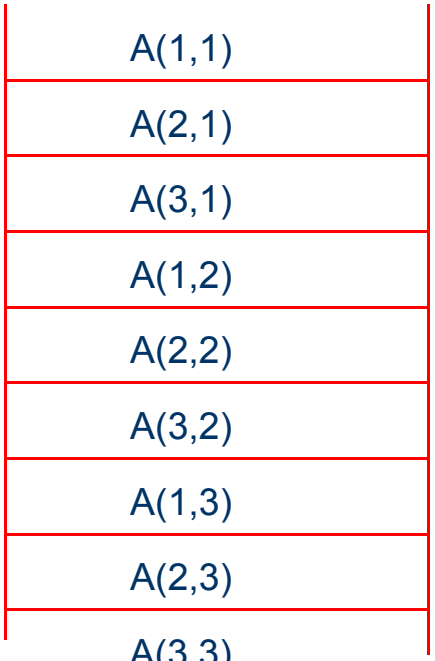
INTEGER A(-1:3)
元素在内存连续模块中的排列情况为
A(-1)=> A(0)=> A(1)=> A(2)=> A(3)



7-3 数组的保存规则

多维数组的元素，在内存的连续模块中是以一种“Column major”（列优先）的方法来排列。





INTEGER A(3,3)

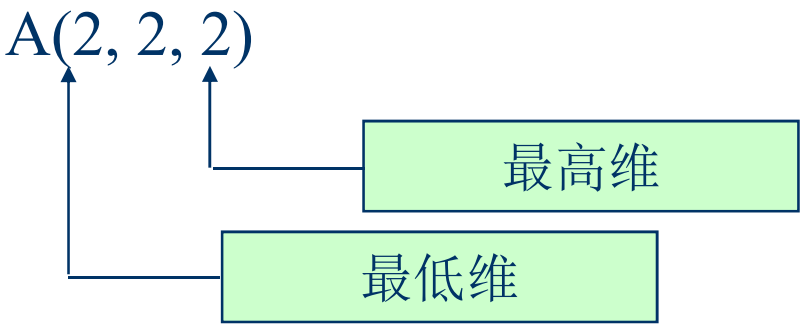
二维数组，9个元素

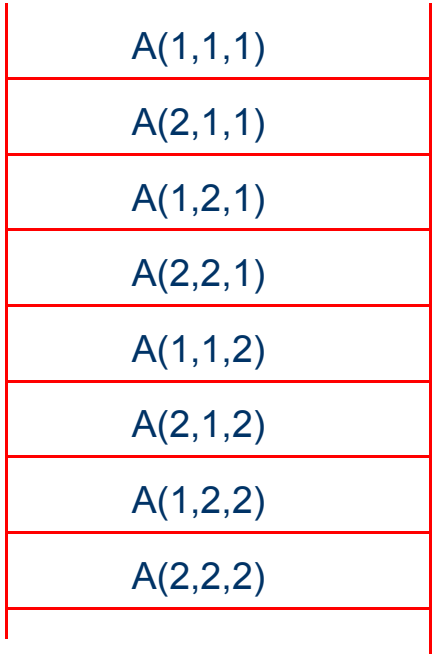
元素在内存连续模块中的排列情况为

A(1,1)=> A(2,1)=> A(3,1)　先放第1列的元素

=> A(1,2)=> A(2,2)=> A(3,2)　再放第2列的元素

=> A(1,3)=> A(2,3)=> A(3,3)　最后放第3列的元素





元素在内存中的排列情况为

A(1, 1, 1)=> A(2, 1, 1)
=> A(1, 2, 1)=> A(2, 2, 1)
=> A(1, 1, 2)=> A(2, 1, 2)
=> A(1, 2, 2)=> A(2, 2, 2)

先放第1维的元素
再放第2维的元素
再放第3维的元素

数组对计算机来说只是一大块内存，实际使用数组时，会先根据它的索引值计算出现在所要使用的是内存中的第几个数字。

越高维的数组，所需要计算的式子就越长，所以高维数组的读取速度会比较慢。

在安排数组时（尤其是多维数组），最好能把同一组经常一起使用的数据，放在内存的邻近模块中。

比较不好的写法：

```
do I=1, N
   do J=1, M
      a(I, J)=……
   end do
end do
```

较好的写法：

```
do I=1, N
   do J=1, M
      a(J, I)=……
   end do
end do
```

因为a(I, J)跟a(I, J+1)在内存中的位置不是连续的，而a(J, I)跟a(J+1, I)在内存中的位置是连续的，所以只要很简单地把I、J的使用位置交换，就可以得到较好的效率。



7-4 可变大小的数组

记录人数不固定班级的学生成绩的程序（假定数组大小不能改变）

[ex0715.f90]

```
program ex0715
implicit none
  integer, parameter :: max = 1000
  integer :: a(max) ! 先声明一个超大的数组
  integer :: students
  integer :: i
  write(*,*) "How many students:"
  read(*,*) students
  ! 输入成绩
  do i=1,students
    write(*,"('Number ',I3)") i
    read(*,*) a(i)
  end do
  stop
end
```



记录人数不固定班级的学生成绩的程序（使用可变大小数组）

[ex0716.f90]

```
program ex0716
implicit none
  integer :: students
  integer, allocatable :: a(:) ! 声明一个可变大小的一维数组
  integer :: i
  write(*,*) "How many students:"
  read(*,*) students
  allocate( a(students) ) ! 配置内存空间
  ! 输入成绩
  do i=1,students
    write(*,"('Number ',I3)") i
    read(*,*) a(i)
  end do
  stop
end
```

使用可变大小数组要经过两个步骤：
使用形容词**ALLOCATABLE**声明可变大小数组
使用**ALLOCATE**命令为可变大小数组配置内存空间

判断内存分配是否成功的方法

Allocate(a(100), stat=error)

error是事先声明好的整型变量，做allocate这个动作时会经由stat这个叙述传给error一个数值，如error等于0则表示allocate数值成功，否则失败。

测试计算机能承受多大数组的程序

[ex0717.f90]

```
program  ex0717
implicit none
  integer :: size, error=0
  integer, parameter :: one_mb=1024*1024 ! 1MB
  character, allocatable ::  a(:)
do while( .true. )
    size=size+one_mb ! 一次增加1MB个字符,
                          也就是1MB的内存空间
    allocate( a(size), stat=error )
    if ( error/=0 ) exit
    write(*,"('Allocate ',I10, ' bytes')") size
    write(*,"(F10.2,' MB used')") real(size)/real(one_mb)
    deallocate( a )
end do
 stop
end
```

```
Allocate 2071986176 bytes
   1976.00 MB used
Allocate 2073034752 bytes
   1977.00 MB used
Allocate 2074083328 bytes
   1978.00 MB used
Allocate 2075131904 bytes
   1979.00 MB used
Allocate 2076180480 bytes
   1980.00 MB used
Allocate 2077229056 bytes
   1981.00 MB used
Allocate 2078277632 bytes
   1982.00 MB used
Allocate 2079326208 bytes
   1983.00 MB used
Allocate 2080374784 bytes
   1984.00 MB used
Allocate 2081423360 bytes
   1985.00 MB used
Allocate 2082471936 bytes
   1986.00 MB used
Allocate 2083520512 bytes
   1987.00 MB used
Press any key to continue
```

可变大小的数组维度的声明方法：

```
integer, allocatable :: a1(:)       !使用1个冒号，代表一维数组
integer, allocatable :: a2(:, :)    !使用2个冒号，代表二维数组
integer, allocatable :: a1(:, :, :) !使用3个冒号，代表三维数组
allocate(a1(5))                     !给定一维数组的大小
allocate(a2(5, 5))                  !给定二维数组的大小
allocate(a1(5, 5, 5))               !给定三维数组的大小
```

在allocate中可以特别赋值数组索引坐标的起始及终止范围

```
allocate(a1(-5: 5))
allocate(a2(-3: 3, -3: 3))
```

DEALLOCATE命令用来把用ALLOCATE命令所得到的内存空间释放掉。使用DEALLOCATE命令和ALLOCATE命令可以重新设置数组大小。

ALLOCATED函数
用以检查一个可变大小的数组是否已经配置内存来使用，返回值为逻辑值。

使用举例：

```
if(.not. allocated(a)) then
  allocate(a(5))
end if
```

!检查数组a是否有配置内存，若还没配置就去要求5个元素的内存空间。



7-5 数组的应用

最简单的排序方法：选择排序法

[ex0718.f90]

```
program  ex0718
implicit none
  integer, parameter :: size=10
  integer :: a(size) = (/ 5,3,6,&
                    4,8,7,1,9,2,10 /)
  integer :: i,j
  integer :: t
  do i=1, size-1
    do j=i+1, size
      if ( a(i) > a(j) ) then
      ! a(i)跟a(j)交换
        t=a(i)
        a(i)=a(j)
        a(j)=t
      end if
    end do
  end do
 write(*,"(10I4)") a
stop
end
```

- 选择排序算法是很容易理解的算法，但是它的计算效率不高，对于实际的大量数据（据说多于**1000**个元素）进行排序时很慢，千万别用该算法。

矩阵相乘

[ex0719.f90]

```
program  ex0719
implicit none
  integer, parameter :: L=3, M=4, N=2
  real :: A(L,M) = (/ 1,2,3,4,5,6,7,8,9,10,11,12 /)
  real :: B(M,N) = (/ 1,2,3,4,5,6,7,8 /)
  real :: C(L,N)
  integer :: i,j,k

  do i=1,L
    do j=1,N
          C(i,j) = 0.0
```



```
            do k=1,M
                C(i,j) = C(i,j)+A(i,k)*B(k,j)
            end do
        end do
    end do

    do i=1,L
        write(*,*) C(i,:)
    end do

    stop
end
```



7-5 什么时候用数组

- 在典型的Fortran课程中，刚刚学完数组的人，不管需要与否，都会被诱惑着使用数组解决问题，其原因也很简单，就是因为学会了数组的使用。
- 如果必须要很多或全部数据同时存储在计算机内存中，为了有效解决问题，那么对于解决问题来说，用数组存储数据时很合理的。反之，就不需要数组。



7-6 不必要的数组的缺点

- **浪费了内存**。不必要的数组能吃掉大量内存，使程序使用的内存比实际需要大。大型程序需要大量的内存来运行，使得能运行这类程序的计算机更昂贵。在某些情况下，额外的内存消耗会导致程序在一些特定的计算机上根本不能运行。
- **不必要的数组限制了程序的能力**。假如设计的程序使用了1000个元素的静态数组，那么工作时的数据元素仅仅高达1000个，遇到多于1000个的数据元素，必须重新编译程序。

7-7 良好的数组编程习惯

- 分析问题，判断是否真的有必要使用数组；
- 任何数组的大小应该用有名常数来声明，这样可以在以后很方便的修改数组的大小。
- 所有数组数据在使用前都必须初始化，没有初始化的数组，其后果难于预测，且因为处理器不同也会出现不同的现象。
- 程序中使用数组，大多数常见问题是发生越界操作。为检测出这些问题，在测试和调试程序时，应该打开编译器的越界检测选项。调试完毕后关闭越界检测。



作业1：求中值

- 常用数据统计中的两个量为平均值和标准偏差（参看第6章例题）。另外一个常用的数据统计量为中值：

  数据集里面一半的数值比这个值更大，数据集里面一半的数值比这个值更小。如果数据集有偶数个数据，那么在中间位置没有精确的数值存在。在这种情况下，中值通常定义为中间位置两个数据元素的平均值。

  数据集的中值常接近于数据集的平均值，但也不完全如此。如1、2、3、4、100的中值就跟平均值差异很大。

- 计算数据中值的方法之一是升序排列数据，然后选择数据集的中间数值作为中值。如果数据集中有偶数个数据，那么中间两个数值的平均值就是得到的中值。
- 算法描述：
  1. 用户取得欲排列的数据；
  2. 选择法进行排序；
  3. 计算出中值、平均值和标准偏差等；
  4. 将结果输出到屏幕或其他设备。
- 请把几个分离的程序有机的“合并”起来，形成新的程序。【数组、获取数据、排序、求中值（偶数的情况下求中间两个数的平均值）、平均值和标准偏差。】



- 计算中值的算法：

  假设有**N**个已经排好序的数据，可以用大小为N的一维数组保存整个数据。

  假如N为偶数：

  中值=[a(N/2)+a(N/2+1)]/2

  假如N为奇数：

  中值=[a(N/2)+1]



- 温习平均值和标准偏差公式：
- 两个式子中的求和可以很容易的一个一个读入作为数据值，在开始求和前不需要得到全部数据读入，因此，计算数据集的平均值和标准偏差的程序不需要使用数组。
- 而计算数据集的中值，因为需要对存储的数据进行升序排列，而升序排列的数据都必须存在于内存中，因此在开始计算之前就必须使用数组来存储所有的输入数据。

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$s = \sqrt{\frac{N\sum_{i=1}^{N} x_i^2 - \left(\sum_{i=1}^{N} x_i\right)^2}{N(N-1)}}$$

作业2

P153

1

2. 假设values是101个元素数组，含有科学实验的测量值，数组用下列语句声明：
REAL, DIMENSION(-50:50):: VALUES
编写一个程序，计算数组中正数、负数和零的个数并输出计算结果。

3. 编程计算空间点(-1,4,6)到空间点(1,5,-2)间的距离，输出计算结果。

4. 编写程序计算两个矢量$\mathbf{v}_1=5\mathbf{i}-3\mathbf{j}+2\mathbf{k}$，$\mathbf{v}_2=2\mathbf{i}+3\mathbf{j}+4\mathbf{k}$的点乘和叉乘。输出计算结果。假定作用于一个物体的力为$\mathbf{F}=4\mathbf{i}+3\mathbf{j}-2\mathbf{k}$，物体的运动速度为$\mathbf{V}=4\mathbf{i}-2\mathbf{j}+\mathbf{k}$，计算出作用于物体的功率。