

Fortran95 程序设计

彭国伦 编著

第5章 流程控制与逻辑运算

5-0-1 自顶向下设计

- 当给定一个新问题，如上一章的作业，一个很自然的倾向就是坐在电脑前面开始编程，而不是“浪费”时间进行思考。
- 对于很小的问题，这种飞跃式的方法编程通常都是可行的。
- 现实世界中，问题一般都是大型的，如果尝试采用这种方法，程序员将陷入绝望的困境。
- 在编写程序解决大型问题之前，彻底考虑一下问题以及将要采取的方法是非常值得的。



5-0-1 自顶向下设计

➤ 自顶向下设计就是一个过程：

从大型任务开始，将其分解为更小的、更容易理解的块（子任务），执行所需任务的一部分。如果需要的话，每个子任务还可以依次再细分为更小的子任务。

对每一个块单独编码和测试，直到每一个子任务都能够独立的正确工作。

将子任务集成为一个完整的任务。

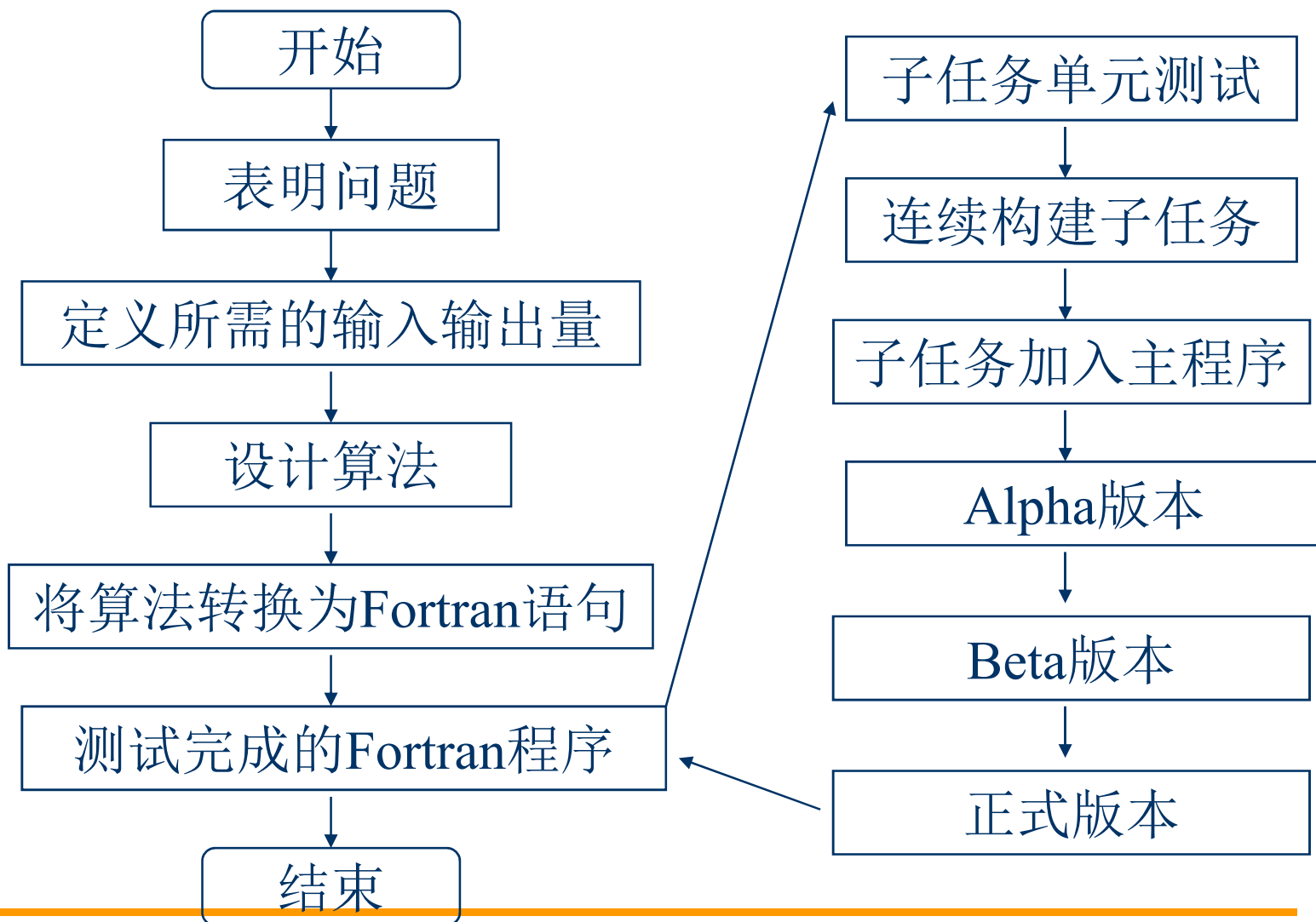


5-0-1 自顶向下设计

1. 清楚的陈述要解决的问题
2. 定义程序所需的输入和程序产生的输出。
3. 设计要在程序中实现的算法。
4. 将算法转换为Fortran语句。
5. 测设完成的Fortran程序。



5-0-1 自顶向下设计



5-0-2 伪代码与流程图

- ▶ 作为设计过程的一部分，有必要对欲实现的算法进行描述。为便于你和他人理解，应该以一种标准的形式对算法进行描述，并且这个描述还应该便于设计思路转换为Fortran代码。
- ▶ 描述算法的形式称为结构(construction)，利用这些结构描述的算法称为结构化算法(constructed algorithm)。当算法在Fortran程序中实现时，结果程序称为结构化程序(structured program)。



5-0-2 伪代码与流程图

- ▶ 用来建立算法的结构可以用两种不同的方法描述：伪代码和流程图。
- ▶ 伪代码是Fortran语句和自然语言掺杂在一起的混合体，构成类似Fortran程序，对每个不同的想法或代码段都有单独的一行来表示。
- ▶ 伪代码具有灵活性和易修改的特点。
- ▶ 伪代码中经常使用←代替=号，表示一个值存在于一个变量中，或者叫赋值。
- ▶ 流程图是描述算法的图形化方法，修改比较麻烦。



5-0-2 伪代码与流程图

椭圆框表示算法的开始和结束

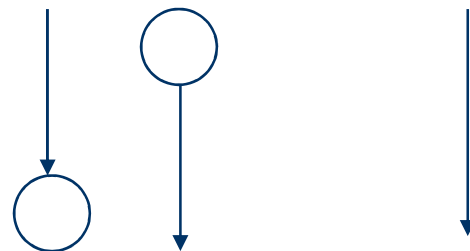
表示输入或输出操作

子程序的引用

反复的或计数的循环回路

矩形框表示计算，并将计算结果赋给一个变量

两种选择中的选择指向



5-1 IF语句

能够在程序执行当中自动选择转向、跳过某些程序模块来执行程序代码，这是IF关键字的功能。

5-1-1 IF基本用法

➤ 最基本的使用方法是 由一个程序模块所构成，当IF所赋值的逻辑判断式成立时，这模块中的程序代码才会执行。

IF (逻辑判断式) THEN

.....

.....

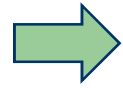
.....

End IF

逻辑成立时才会执行
这里面的程序代码



[ex0501.f90]



```
program ex0501
implicit none
  real(kind=4) :: speed
  write(*,*) "speed:" !信息提示
  read(*,*) speed     !读入车速
  if ( speed > 100.0 ) then
    ! speed > 100 时才会执行下面这一行程序
    write(*,*) "Slow down."
  end if
stop
end
```



程序的核心部分：

```
if ( speed > 100.0 ) then
```

! speed > 100 时才会执行下面这一行程序

```
write(*,*) "Slow down." ←
```

```
end if
```

```
stop ←
```

从IF到END IF之间的程序算是一个区块，
IF中判断式成立时会执行这个区块中的程序

IF中判断式不成立时，会跳跃到END IF后的
地方继续执行

➤ IF括号中的判断式成立时，如果所需要执行的程序模块只有一行程序代码，可以把IF跟这行程序代码些在同一行。

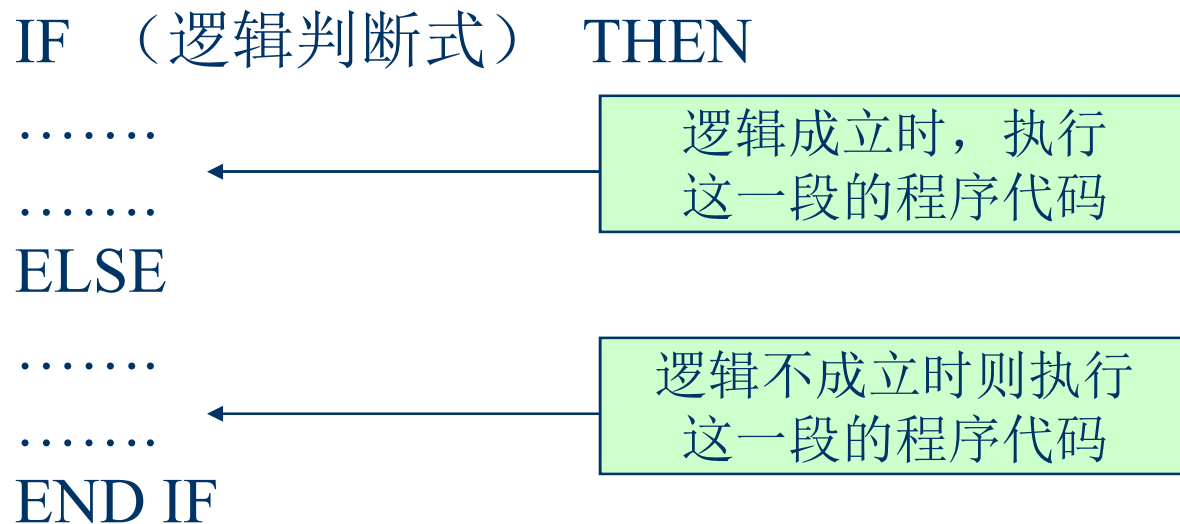
➤ 上面程序中if到end if之间的内容可以改写成下面这一行程序代码：

```
If (speed>100.0) write (*,*) “Slow down”
```



5-1 IF语句

➤ IF命令还可以搭配ELSE，用来赋值当判断式不成立时，会去执行某一段程序代码。



➤ 如果.... 那么就....， 否则就.....。



ex0502

```

program ex0502
implicit none
  real(kind=4) :: height      ! 记录身高
  real(kind=4) :: weight     ! 记录体重

  write(*,*) "height:"
  read(*,*) height           ! 读入身高
  write(*,*) "weight:"
  read(*,*) weight           ! 读入体重

```

```

if ( weight > height-100 ) then  ! 如果体重大于身高减去100, 会执行下面的程序
  write(*,*) "Too fat!"

```

```

else                               ! 如果体重不大于身高减去100, 会执行下面的程序
  write(*,*) "Under control."
end if

```

```

stop
end

```



```

C:\> "G:\chap05\Debug\ex0502.exe"
height:
65
weight:
10
Too fat!
Press any key to continue

```



```

C:\> "G:\chap05\Debug\ex0502.exe"
height:
170
weight:
60
Under control.
Press any key to continue

```



5-1-2 逻辑运算

IF命令需搭配逻辑表达式才能使用。
逻辑运算符号：

F90以上	F77	说明
==	.EQ.	判断是否“相等”
/=	.NE.	判断是否“不相等”
>	.GT.	判断是否“大于”
>=	.GE.	判断是否“大于或等于”
<	.LT.	判断是否“小于”
<=	.LE.	判断是否“小于或等于”



[[ex0501.f90](#)]



[ex0501.for]

```
PROGRAM ex0501
IMPLICIT NONE
REAL speed
WRITE(*,*) "speed:"
READ (*,*) speed
IF ( speed .GT. 100 ) then
! FORTRAN 77要用缩写.GT.代表大于">"
  write(*,*) "Slow down."
END IF
STOP
END
```



5-1-2 逻辑运算

逻辑表达式除了可以单纯对两个数字比较大小之外，还可以对两个逻辑表达式间的关系来运算。如：

If (a>=80 .and. A<90) then

“`.and.`”是并且的意思。

表示相互关系的集合运算符：

<code>.AND.</code>	交集，如两边表达式都成立，整个表达式就成立。
<code>.OR.</code>	并集，两边表达式只要有一个成立，整个表达式就成立。
<code>.NOT.</code>	逻辑反向，如后面的表达式不成立，整个表达式就成立。
<code>.EQV.</code>	两边表达式的逻辑运算结果相同时，整个表达式就成立。
<code>.NEQV.</code>	两边表达式的逻辑运算结果不同时，整个表达式就成立。

大于小于等式的运算符优先级高于集合运算符



[ex0503.f90]

```
program ex0503
implicit none
integer rain, windspeed
write(*,*) "Rain:"
read(*,*) rain
write(*,*) "Wind:"
read(*,*) windspeed
if ( rain>=500 .or. windspeed >=10 ) then
write(*,*) "停止上班上课"
else
write(*,*) "照常上班上课"
end if
stop
end
```



集合运算符的使用方法

A	B	.NOT. A	A .AND. B	A .OR. B
.T.	.T.	.F.	.T.	.T.
.T.	.F.	.F.	.F.	.T.
.F.	.T.	.T.	.F.	.T.
.F.	.F.	.T.	.F.	.F.

A	B	A .EQV. B	A .NEQV. B
.T.	.T.	.T.	.F.
.T.	.F.	.F.	.T.
.F.	.T.	.F.	.T.
.F.	.F.	.T.	.F.



集合运算符的使用举例

【.AND.】

10>5 .AND. 6<10	True
2>1 .AND. 3>1	True
10>5 .AND. 6<10	False
1>2 .AND. 1>3	False

【.OR.】

1>5 .OR. 2<5	True
2>1 .OR. 3>1	True
1>5 .OR. 2>5	False

【.NOT.】

.NOT. 3>5	True
.NOT. 1<2	False



【.EQV.】

$1 > 3$.EQV. $2 > 3$	True
$1 < 2$.EQV. $2 < 3$	True
$1 < 2$.EQV. $2 > 3$	False

【.NEQV.】

$1 > 2$.NEQV. $2 < 3$	True
$1 < 2$.NEQV. $2 < 3$	False
$1 > 2$.NEQV. $2 > 3$	False



逻辑运算可以通过利用AND/OR/NOT/EQV/NEQV/这几个运算符号连接出很长的表达式，也可以用括号()括起来以确定他们的运算先后顺序。

If((A>=10) .AND. (A<=20)) then.....

! 变量A>=10并且A<=20时，也就是变量A在10~20之间时条件成立。

If((key=='Y') .OR. (key=='y')) then.....

! 如果变量key等于字符Y或y时，条件成立

If(.NOT.(A==10)) then.....

! 变量A等于10时，条件不成立。



逻辑表达式中包含算术运算符、关系运算符和逻辑运算符，它们的运算优先次序如下表：

运算类别	运算符	优先级
括号	()	1
算术运算	**	2
	* /	3
	+ -	4
关系运算	>(.GT.) >=(.GE.) <(.LT.) <=(.LE.) ==(.EQ.) /= (.NE.)	5
逻辑运算	.NOT.	6
	.AND.	7
	.OR.	8
	.EQV. .NEQV.	9



5-1-2 逻辑运算

程序代码中可以使用逻辑表达式来设置逻辑变量的内容

`Logical_var = A > B`



当A的数值大于B时，logical_var这个逻辑变量会被设定成“真”(.TRUE.)，否则会被设定为“假”(.FALSE.)

使用IF时，可以先把逻辑运算的结果存放到逻辑变量中，再利用逻辑变量来做条件判断。



[ex0504.f90]

```
program ex0504
implicit none
integer rain, windspeed
logical r,w
write(*,*) "Rain:"
read(*,*) rain
write(*,*) "Wind:"
read(*,*) windspeed
r = (rain>=500) ! 如果rain>=150, r=.true, 不然r=.false.
w = (windspeed>=10) ! 如果windspeed>=10, w=.true, 不然w=.false.
if ( r .or. w ) then ! 只要r或w有一个值是true就成立
write(*,*) "停止上班上课"
else
write(*,*) "照常上班上课"
end if
stop
end
```



5-1-3 多重判断 IF-ELSE IF

多重判断可一次列出多个条件及多个程序模块

IF (条件1) then

.....



条件1成立时，执行这个模块程序

.....

else if (条件2) then

.....



条件2成立时，执行这个模块程序

.....

else if (条件3) then

.....



条件3成立时，执行这个模块程序

.....

else if (条件4) then

.....



条件4成立时，执行这个模块程序

.....

else



Else这个模块可以省略

.....



每个条件都不成立时，
才执行这个模块程序

End if



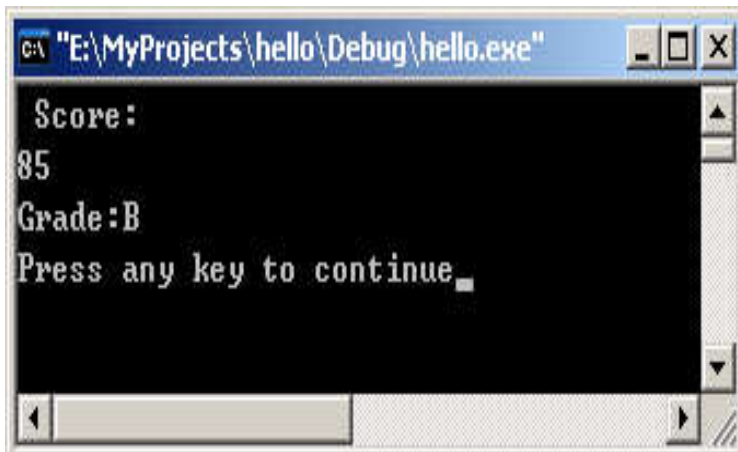
[ex0505.f90]

```
program ex0505
implicit none
integer score
character grade

write(*,*) "Score:"
read(*,*) score
```

```
if ( score>=90 .and. score<=100 ) then
  grade='A'
else if ( score>=80 .and. score<90 ) then
  grade='B'
else if ( score>=70 .and. score<80 ) then
  grade='C'
else if ( score>=60 .and. score<70 ) then
  grade='D'
else if ( score>=0 .and. score<60 ) then
  grade='E'
else
  ! score<0 或 score>100的不合理情形
  grade='?'
end if

write(*, "('Grade:',A1)") grade
stop
end
```



```
C:\E:\MyProjects\hello\Debug\hello.exe
Score:
85
Grade: B
Press any key to continue
```



因为IF-ELSE IF所组合出来的多重判断式只会执行第一个符合条件的程序模块，执行完后就跳到END IF后继续执行程序，每个条件都不成立时才会执行ELSE中的模块。所以EX0505可以简化为：

[ex0506.f90]

```
program ex0506
implicit none
integer score
character grade
write(*,*) "Score:"
read(*,*) score
```

```
if ( score>100 ) then
  grade='?'
else if ( score>=90 ) then ! 会执行到此, 代表score<=100
  grade='A'
else if ( score>=80 ) then ! 会执行到此, 代表score<90
  grade='B'
else if ( score>=70 ) then ! 会执行到此, 代表score<80
  grade='C'
else if ( score>=60 ) then ! 会执行到此, 代表score<70
  grade='D'
else if ( score>=0 ) then ! 会执行到此, 代表score<60
  grade='E'
else ! 会执行到此, 代表score<0
  grade='?'
end if
write(*, "('Grade:',A1)") grade
stop
end
```

```
c:\E:\MyProjects\hello\Debug\hello.exe
Score:
85
Grade: B
Press any key to continue.
```



利用多个独立的IF语句也能实现多重判断的效果
如：判别成绩等级

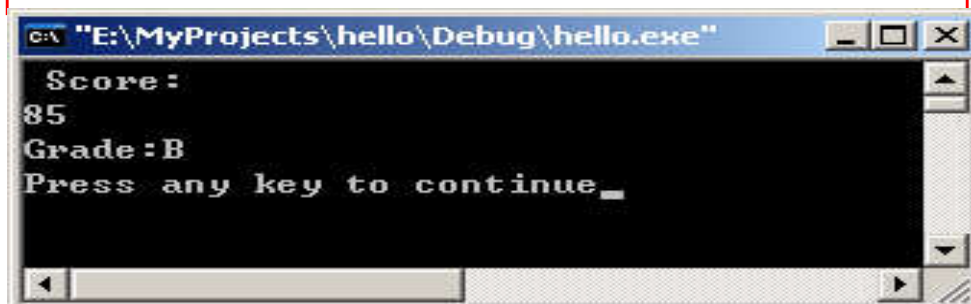
[ex0507.f90]

```
program ex0507
implicit none
integer score
character grade
```

```
write(*,*) "Score:"
read(*,*) score
```

```
if ( score>=90 .and. score<=100 ) grade='A'
if ( score>=80 .and. score<90 ) grade='B'
if ( score>=70 .and. score<80 ) grade='C'
if ( score>=60 .and. score<70 ) grade='D'
if ( score>=0 .and. score<60 ) grade='E'
if ( score>100 .or. score<0 ) grade='?'

write(*,"('Grade:',A1)") grade
stop
end
```



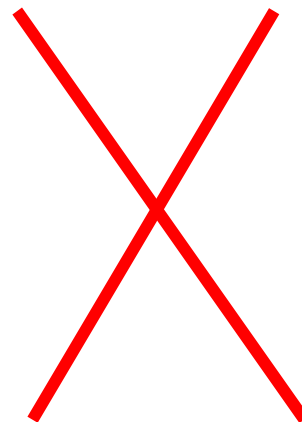
```
C:\ "E:\MyProjects\hello\Debug\hello.exe"
Score:
85
Grade: B
Press any key to continue_
```



5-1-3 多重判断 IF-ELSE IF

如果把EX0507.F90中的6个IF语句改为如下的样子

```
If (score>=90) grade='A'  
If (score>=80) grade='B'  
If (score>=70) grade='C'  
If (score>=60) grade='D'  
If (score>=0) grade='E'  
If (score>=100 .or. score<0) grade='?'
```



执行后会发现，永远都只能得到“E”或“？”两种结果。



例子1：二次方程式求解

- 设计并编写程序，无论是何种类型，都可以求出二次方程式的根。
- 定义输入输出

$$ax^2+bx+c=0$$

提示用户输入系数a, b, c

- 设计算法：
读取输入数据，计算根，输出根



例子1: 二次方程式求解

! Data dictionary: declare variable types, definitions, & units

REAL :: a **! Coefficient of x^2 term of equation**

REAL :: b **! Coefficient of x term of equation**

REAL :: c **! Constant term of equation**

REAL :: discriminant **! Discriminant of the equation**

REAL :: imag_part

! Imaginary part of equation (for complex roots)

REAL :: real_part **! Real part of equation (for complex roots)**

REAL :: x1 **! First solution of equation (for real roots)**

REAL :: x2 **! Second solution of equation (for real roots)**



例子1：二次方程式求解

! Prompt the user for the coefficients of the equation

```
WRITE (*,*) 'This program solves for the roots of a quadratic '
```

```
WRITE (*,*) 'equation of the form A * X**2 + B * X + C = 0. '
```

```
WRITE (*,*) 'Enter the coefficients A, B, and C: '
```

```
READ (*,*) a, b, c
```

! Echo back coefficients

```
WRITE (*,*) 'The coefficients A, B, and C are: ', a, b, c
```

! Calculate discriminant

```
discriminant = b**2 - 4. * a * c
```



例子1: 二次方程式求解

! Solve for the roots, depending upon the value of the discriminant

IF (discriminant > 0.) THEN **! there are two real roots, so...**

x1 = (-b + sqrt(discriminant)) / (2. * a)

x2 = (-b - sqrt(discriminant)) / (2. * a)

WRITE (*,*) 'This equation has two real roots:'

WRITE (*,*) 'X1 = ', x1

WRITE (*,*) 'X2 = ', x2

ELSE IF (discriminant < 0.) THEN **! there are complex roots, so ...**

real_part = (-b) / (2. * a)

imag_part = sqrt (abs (discriminant)) / (2. * a)

WRITE (*,*) 'This equation has complex roots:'

WRITE (*,*) 'X1 = ', real_part, ' +i ', imag_part

WRITE (*,*) 'X2 = ', real_part, ' -i ', imag_part

ELSE IF (discriminant == 0.) THEN **! there is one repeated root, so...**

x1 = (-b) / (2. * a)

WRITE (*,*) 'This equation has two identical real roots:'

WRITE (*,*) 'X1 = X2 = ', x1

END IF

END PROGRAM roots



5-1-4 嵌套IF语句

```
IF (.....) THEN           ←第1层IF开始
  IF (.....) THEN         ←第2层IF开始
    IF (.....) THEN       ←第3层IF开始
      ELSE IF (.....) THEN
        ELSE
          END IF           ←第3层IF结束
        END IF           ←第2层IF结束
      END IF           ←第1层IF结束
    END IF
  END IF
END IF
```



嵌套IF举例：读入坐标点(x, y),
判断其落在哪个象限。

[ex0508.f90]

```

program ex0508
implicit none
real x,y
integer ans
write(*,*) "Input (x,y)"
read(*,*) x,y
if ( x>0 ) then
  if ( y>0 ) then ! x>0,y>0
    ans=1
  else if ( y<0 ) then ! x>0, y<0
    ans=4
  else ! x>0, y=0
    ans=0
  end if
end if

```

```

else if ( x<0 ) then
  if ( y>0 ) then ! x<0, y>0
    ans=2
  else if ( y<0 ) then ! x<0, y<0
    ans=3
  else ! x<0, y=0
    ans=0
  end if
else ! x=0, y=任意数
  ans=0
end if
if ( ans/=0 ) then ! ans不为0时, 代表有解
  write(*, "('第',I1,'象限')") ans
else
  write(*,*) "落在轴上"
end if
stop
end

```



5-1-5 命名的IF结构

可以给IF结构指定一个名称。对IF结构中的所有组成部分进行命名，可以快速地指明某一特定的ELSE或ELSE IF语句属于那个IF结构。可使程序员的注意力更加清晰。

带有名称的IF结构的形式：

```
[名称:] IF(逻辑表达式_1) THEN
```

```
语句1
```

```
语句2
```

```
.....
```

```
ELSE IF(逻辑表达式_2) THEN [名称]
```

```
语句1
```

```
语句2
```

```
.....
```

```
ELSE [名称]
```

```
语句1
```

```
语句2
```

```
.....
```

```
END IF [名称]
```



5-2 浮点数及字符的逻辑运算

5-2-1 浮点数的逻辑判断

- 使用浮点数做逻辑运算时，要避免使用“等于”的判断
 - 使用浮点数计算，有效位数是有限的，难免会出现计算上的误差，理想中的等号不一定会成立
例：EX0509.f90
 - 浮点数的计算误差经常发生，在判断式中要给误差预留一点空间
例：EX0510.f90



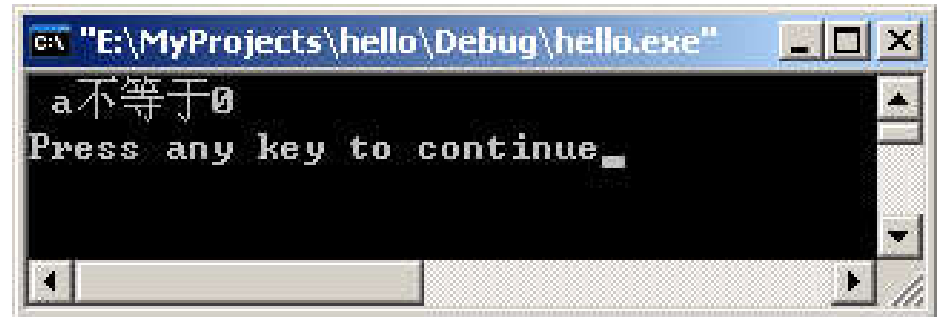
[ex0509.f90]

```
program ex0509
implicit none
real :: a
real :: b = 3.0
```

$a = \sqrt{b}^2 - b$! 理论上a应该要等于0

```
if ( a==0.0 ) then
    write(*,*) "a等于0"
else
    write(*,*) "a不等于0"
end if
```

```
stop
end
```



```
c:\ "E:\MyProjects\hello\Debug\hello.exe"
a不等于0
Press any key to continue
```

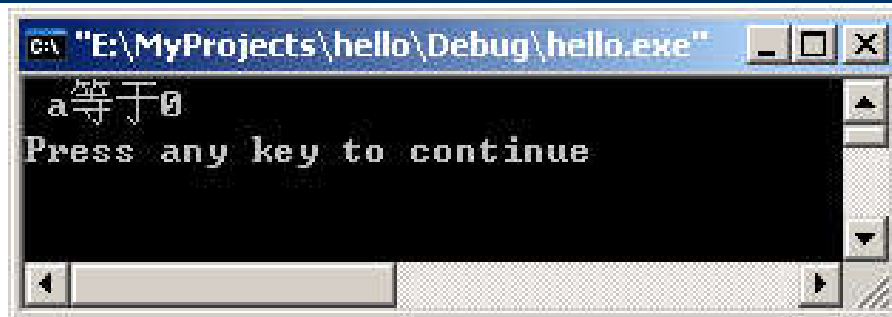


```
program ex0509_2
implicit none
real :: a
real :: b = 4.0
```

$a = \sqrt{b}^2 - b$! 理论上a应该要等于0

```
if ( a==0.0 ) then
    write(*,*) "a等于0"
else
    write(*,*) "a不等于0"
end if
```

```
stop
end
```



```
cmd "E:\MyProjects\hello\Debug\hello.exe"
a等于0
Press any key to continue
```

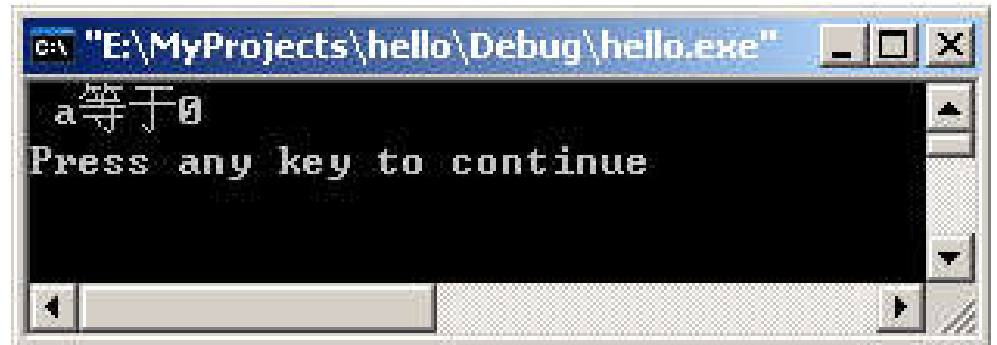


[ex0510.f90]

```
program ex0510
implicit none
real :: a
real :: b = 3.0
real, parameter :: e = 0.0001 !设置误差范围

a=SQRT(b)**2-b ! 理论上a应该要等于0

if ( abs(a-0.0)<=e ) then
    write(*,*) "a等于0"
else
    write(*,*) "a不等于0"
end if
stop
end
```



```
"E:\MyProjects\hello\Debug\hello.exe"
a等于0
Press any key to continue
```



5-2-2 字符的逻辑判断

字符也可比较大小，根据是比较它们的字符码，个人计算机中比较字符用字符的ASCII码。

‘a’<‘b’

!因为a的ASCII码为97，b的ASCII码为98

‘A’<‘a’

!因为A的ASCII码为65，a的ASCII码为97

“abc”<“bcd”

!根据字母顺序来比较，字符串“abc”的第1个字符小于字符

!串“bcd”的第1个字符

“abc”<“abcd”

!根据字母顺序来比较，两个字符串的前3个字符都一样

!但字符串“abcd”比字符串“abc”多了一个字符。



[ex0511.f90]

```
program ex0511
```

```
implicit none
```

```
character(len=20) :: str1,str2
```

```
character relation
```

```
write(*,*) "String 1:"
```

```
read(*,"(A20)") str1
```

```
write(*,*) "String 2:"
```

```
read(*,"(A20)") str2
```

```
if ( str1>str2 ) then
```

```
    relation = '>'
```

```
else if ( str1==str2 ) then
```

```
    relation = '='
```

```
else
```

```
    relation = '<'
```

```
end if
```

```
write(*,"('String1',A1,'String2')") relation
```

```
stop
```

```
end
```



5-3 SELECT CASE语句

```
Select case (变量) ← 放入所要判断的变量
case (数值1)
..... ← 变量等于数值1时, 执行此程序段
.....
case (数值2)
..... ← 变量等于数值2时, 执行此程序段
.....
case (数值n)
..... ← 变量等于数值n时, 执行此程序段
.....
case default
..... ← 变量不等于任何数值时, 执行此程序段
End if
```



case里的冒号前后放入两个数字代表在这两个数字之间的所有数值。**case**的括号里面可以用逗号来放多个变量。

case(1)	!变量=1时，会执行这个case中的程序模块
case(1: 5)	!1<=变量<=5时，会执行这个case中的程序模块
case(1:)	!1<=变量时，会执行这个case中的程序模块
case(: 5)	!变量<=5时，会执行这个case中的程序模块
case(1, 3, 5)	!变量等于1或3或5时，会执行这个case中的程序模块



使用SELECT-CASE语句写的判断分数等级的程序:

[ex0512.f90]

```
program ex0512
```

```
implicit none
```

```
integer score
```

```
character grade
```

```
write(*,*) "Score:"
```

```
read(*,*) score
```

```
select case(score)
```

```
case(90:100) ! 90到100分之间
```

```
grade='A'
```

```
case(80:89) ! 80到89分之间
```

```
grade='B'
```

```
case(70:79) ! 70到79分之间
```

```
grade='C'
```

```
case(60:69) ! 60到69分之间
```

```
grade='D'
```

```
case(0:59) ! 0到59分之间
```

```
grade='E'
```

```
case default ! 其它情形
```

```
grade='?'
```

```
end select
```

```
write(*, "('Grade:',A1)") grade
```

```
stop
```

```
end
```



5-3 SELECT CASE语句

- ▶ 使用SELECT CASE来取代某些使用IF-ELSE IF的多重语句，会让代码看起来比较简洁。
- ▶ SELECT CASE有一些限制：
 - ▶ 只能使用**整数**，**字符和逻辑变量**，不能使用**浮点数和复数**。
 - ▶ 每个**case**中所使用的数值必须是固定的**常量**，不能使用**变量**。



```
integer :: a=65
```

```
integer :: b=97
```

```
integer, parameter :: c=87
```

```
read(*, *) key
```

```
select case(key)
```

```
case(a) !这一行程序错误, case中不能使用变量
```

```
.....
```

```
case(b) !这一行程序错误, case中不能使用变量
```

```
.....
```

```
case(c) !c是声明成parameter的常量, 可以使用
```

```
.....
```

```
end select
```



```
program ex0513
```

```
implicit none
```

```
real a,b,ans
```

```
character operator
```

```
read(*,*) a
```

```
read(*,“(A1)”) operator ! 不使用格式有些机器会  
!读不到除号"/"
```

```
read(*,*) b
```

```
select case(operator)
```

```
case('+')
```

```
ans = a+b
```




```
case('-')
```

```
  ans = a-b
```

```
case('*')
```

```
  ans = a*b
```

```
case('/')
```

```
  ans = a/b
```

```
case default ! 输入其它符号不处理
```

```
  write(*,("Unknown operator ',A1)") operator
```

```
    stop ! 结束程序
```

```
end select
```

```
write(*,("F6.2,A1,F6.2,'=',F6.2")) a,operator,b,ans
```

```
stop
```

```
end
```



5-4 其他流程控制

5-4-1 GOTO

- 从Fortran 77之前就流传下来的古老的“跳转”语句
- 不建议使用
- 使用GOTO会使编写的程序在结构上变乱，导致程序代码难以阅读，这里介绍是希望读职能看懂一些用古典风格编写的程序。
- GOTO命令就是提供程序员一个任意跳跃到所赋值“行代码”的那一行程序位置来执行程序的能力



判断一个人是否过重的程序ex0502.f90使用GOTO改写后的形式。

[ex0514.for]

```
PROGRAM ex0514
  IMPLICIT NONE
  REAL height ! 记录身高
  REAL weight ! 记录体重

  WRITE(*,*) "height:"
  READ(*,*) height ! 读入身高
  WRITE(*,*) "weight:"
  READ(*,*) weight ! 读入体重

  IF ( weight > height-100 ) GOTO 200
  ! 上面不成立, 没有跳到200才会执行这里
100  WRITE(*,*) "Under control."
     GOTO 300 ! 下一行不能执行所以要跳到300.
200  WRITE(*,*) "Too fat!"

300  STOP
     END
```



使用GOTO编写的“循环”

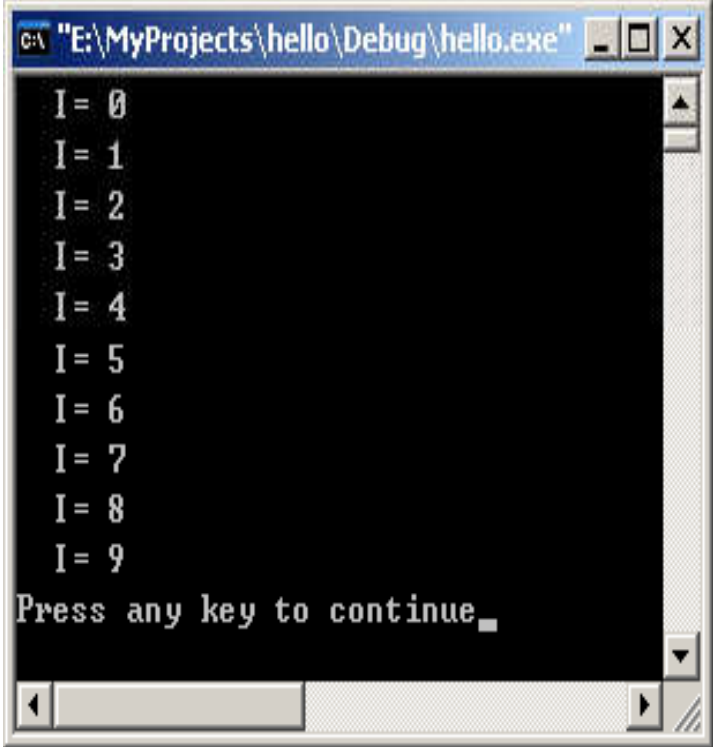
[ex0515.for]

```
PROGRAM ex0515
  IMPLICIT NONE

  INTEGER I
  INTEGER N
  PARAMETER(N=10)
  DATA I /0/

10  WRITE(*, '(1X,A3,I2)') 'I=',I
     I=I+1
     IF ( I.LT. N ) GOTO 10

  STOP
  END
```



```
C:\E:\MyProjects\hello\Debug\hello.exe
I= 0
I= 1
I= 2
I= 3
I= 4
I= 5
I= 6
I= 7
I= 8
I= 9
Press any key to continue
```



GOTO还有一种用法，程序代码中可以一次提供好几个跳跃点，根据GOTO后面的算式来选择要使用哪一个跳跃点。称为计算GOTO语句。

形式: goto(label1, label2, label3, ..., labelk), int_expr

label1到labelk是可执行语句的行代码。int_expr表达式计算出一个1和k之间的整形数。如果int_expr的值为1，转去行代码为label1的语句执行，int_expr的值为k，转去行代码为labelk的语句执行。如果label1的值小于1或者大于k，将引发错误。



[ex0516.for]

```
PROGRAM ex0516
IMPLICIT NONE
```

```
INTEGER I
INTEGER N
```

```
DATA I,N /2,1/
```

```
C I/N=1时GOTO 10, I/N=2时GOTO 20, I/N=3时GOTO 30
```

```
C I/N<1或I/N>3时不做GOTO, 直接执行下一行
```

```
GOTO(10,20,30) I/N
```

```
10 WRITE(*,*) 'I/N=1'
```

```
GOTO 100
```

```
20 WRITE(*,*) 'I/N=2'
```

```
GOTO 100
```

```
30 WRITE(*,*) 'I/N=3'
```

```
100 STOP
```

```
END
```



5-4-2 IF与GOTO的联用

▶ IF判断还有一种叫做算术判断的方法，做法和GOTO类似。

<0 =0 >0

▶ 形式：**if(arithmetic_expression) label1, label2, label3**

其中arithmetic_expression可以是任何整形、实型算术表达式。label1, label2和label3是可执行语句的行代码。

当arithmetic_expression的值为负时，执行行代码为label1的语句，当arithmetic_expression的值为0时，执行行代码为label2的语句，当arithmetic_expression的值为正时，执行行代码为label3的语句。



[ex0517.for]

```
PROGRAM ex0517
IMPLICIT NONE
REAL A, B
REAL C
DATA A, B /2.0, 1.0/
```

```
C=A-B
```

```
C      C<0就GOTO 10, C=0就GOTO 20, C>0就GOTO 30
```

```
IF ( C ) 10, 20, 30
```

```
10     WRITE(*,*) 'A<B'
```

```
GOTO 40
```

```
20     WRITE(*,*) A=B '
```

```
GOTO 40
```

```
30     WRITE(*,*) 'A>B'
```

```
40     STOP
```

```
END
```



5-4-3 PAUSE, CONTINUE, STOP

- ◆ PAUSE：暂停执行，直到用户按下Enter键才会继续执行。
- ◆ CONTINUE：继续向下执行程序，f90后很少使用
- ◆ STOP：结束程序执行



5-5 二进制的逻辑运算

◆ 二进制的逻辑运算和IF中的逻辑判断式不太相同，比较接近单纯的数学运算。

$$0 \text{ .and. } 0 = 0$$

$$0 \text{ .and. } 1 = 0$$

$$1 \text{ .and. } 0 = 0$$

$$1 \text{ .and. } 1 = 1$$

◆ F90的库函数：IAND() IOR() 用来做二进制的AND和OR运算，是把输入的两个整数中，同样位置的位进行逻辑运算。

a=2 !a等于二进制的010

b=4 !b等于二进制的100

c=iand(a, b) !c=0，也就是二进制的000

c=ior(a, b) !c=6，也就是二进制的110



Fortran 90在设置整数时，可以不用十进制的方法，而使用其他进制的方法来做设置。把数字用双引号括起来，最前面加上**B**代表这段数字是二进制数字，同理最前面用**O**代表要使用八进制，用**Z**代表要使用**16**进制。

```
integer :: a
```

```
    a=B"10"      ! a=2
```

```
    a=O"10"      ! a=8
```

```
    a=Z"10"      ! a=16
```



Fortran程序的调试

- 编写包含分支和循环结构的程序比编写简单的顺序结构程序更容易犯错误。
- 如果只是发现输出值有错误，如何定位排错？
- 最好方法：符号调试器，但是需要查看系统手册以便如何使用。
- 一般方法：在程序中插入**write**语句，在一些关键位置点输出变量的值，以观察程序中间结果与预期结果的差异。
- 如果是**IF**块的问题：重点检查逻辑表达式。
- 另外一个常见错误：实数型变量的等值测试。



作业

P98

1、2、3、4

- 5、一束光线从折射率为 n_1 的区域穿透到折射率为 n_2 的另一个区域，光线被折射，折射的角度由斯涅尔定律给出。编写Fortran程序，给定区域1中的入射角 θ_1 和折射率 n_1 、 n_2 ，计算光束在区域2中的出射角度 θ_2 。计算时以空气和水为例，并大致估计一下发生全反射的入射角度。可以根据 $n_1=1.0$ ， $n_2=1.7$ 测试不同的角度，并适当变化 n_2 值的大小，看看全反射角的变化。

