City University
School of Engineering and Mathematical Sciences
Common part 1

COMPUTING

Fortran Lecture 3 Program flow control

3.1Normal program flow

Normally the computer will execute each line of the program in strict sequence, starting with the first and ending with the last line in the program. There can be many exceptions to this, for example in solving a quadratic equation. We know from mathematics that the roots of a*x*x + b*x +c =0 can be either two real numbers, coincident or a pair of complex conjugates with real and imaginary parts. In English we could write if b*b is less than 4*a*c then we have complex roots, if b*b equals 4*a*c then we have coincident roots , and if b*b is greater than 4*a*c then we have two real roots.

3.2The single-line if statement

The following is permissible.

```
if(nmax.eq.0)write(*,*)'nmax = 0'
```

In this type of if statement if the defined condition is not met then control passes to the line immediately below the if statement.

Table of rational operators allowed in an if statement

Fortran	Example	Meaning
.eq.	x.eq.y	x equal to y
.ne.	x.ne.y	x not equal y
.lt.	x.lt.y	x less than y
.le.	x.le.y	x less than or equal to y
.gt.	x.gt.y	x greater than y
.ge.	x.ge.y	x greater than or equal to y

3.3.The goto statement

This statement allows direct, unconditional, branching. –

```
GO TO 100
```

Note that the single-line if statement is commonly used in combination with a goto statement.

```
10 write(*,*)'enter total number of vaules (>1)'
```

```
read(*,*)nmax
if(nmax.le.1)goto 10
```

3.4**Quadratic equation solver**

Copy program Prog9 into U: and execute it.

```
Program prog9
!=======
! program to solve a quadratic
! user types in the coefficients
!program works out if roots are complex, real or coincident
!=======
!  ax*x + b*x + c =0
! calculates b*b - 4(a*c)
!
    Complex croot1,croot2
    Print *,'program to find roots of a quadratic'
    Print *,'solves a*x*x + b*x + c =0 '
    Print *, 'type in three coefficients'
    read (*,*) a,b,c
    d=b*b - 4*a*c
    if (ABS (d) < 0.00000001) go to 101
    if (d < 0.0) go to 100
!------------------
! discriminant is positive so square root ok
!------------------
    root1= (-b + SQRT (d))/(2.0*a)
    root2= (-b -SQRT (d))/(2.0*a)
    print *,"  root1 = ",root1
    print *,"  root2 = ",root2
    stop
!------------------
! complex roots - print out message
!-----------------
 100 print *, '  roots are complex'
       croot1= CMPLX((-b/(2.0*a)), SQRT(ABS(d)))
       croot2=CONJG (croot1)
        Print *, croot1,croot2
    stop
 !------------------
 !  roots are coincident - floating point arithmetic never
 ! gives exactly zero
 !--------------------
  101 root1= (-b)/(2*a)
     print *, "  both roots are ",root1
     stop
     end
```

3.5Problems with Quadratic equation solver

What happens if we type 0.0 for the first coefficient ? change the program to correct this.

3.6Program Loops – DO Loops

Do loops

The repetition of a series of commands is accomplished by the use of a do loop. The basic structure of the do loop can be seen in the following simple example, which sums a number of values.

```
      Program prog10
           write(*,*)'enter total number of values to be summed'
           read(*,*)nmax
           sum = 0.0
           do 20 i = 1,nmax
           write(*,*)'enter value',i
            read(*,*)val
            sum = sum+val
     20   continue
            write(*,*)' '
            write(*,*)'sum =',sum
            stop
            end
```

The do loop starts with the statement do 20… and end at the statement 20 continue. The continue statement does nothing other than signify the end of the do loop. By default, the counter, i in the above example, is incremented by one. However any integer step can be defined explicitly (including negative values). For example:

```
      Program prog11
          do 30 i = 10,2,-2
          write(*,*)i
     30   continue
          stop
          end
```

Would print the values 10, 8, 6, 4, 2.

Do loops can be nested; i.e., a do loop can contain other do loops. For example:

```
      Program prog12
      do 20 i = 1,2
      do 14 j = 3,6,3
      write(*,*)i,j
     14   continue
      do 16 k = 4,1,-2
      write(*,*)i,k
```

```
16  continue
20  continue
    stop
    end
```

Would print the values 1 3, 1 6, 1 4, 1 2, 2 3, 2 6, 2 4, 2 2. Note that the line "do 16 k = 4,1,-2" only produces values for k of 4 and 2.

3.7.The if...then statement

This construct allows conditional branching in programs. The basic structure is show in the following example:

```
      Program prog13
      write(*,*)'enter total number of values to be summed'
      read(*,*)nmax
       if(nmax.gt.1)then
           sum = 0.0
          do 20 i = 1,nmax
              write(*,*)'enter value',i
              read(*,*)val
              sum = sum+val
20        continue
          write(*,*)' '
          write(*,*)'sum =',sum
      endif
      stop
      end
```

In the above, the sum is now only performed if the total number of values is indicated to be greater than 1. If this condition is not met all commands are skipped until the one immediately below the endif statement, which in this case is stop.

3.8.The if...then...else statement

This construct allows the choice of two outcomes, as the example below illustrates.

```
Progran prog14
write(*,*)'enter total number of values to be summed'
    read(*,*)nmax
    if(nmax.le.1)then
     write(*,*)'value entered not greater than 1'
    else
```

```
      sum = 0.0
      do 20 i = 1,nmax
      write(*,*)'enter value',i
      read(*,*)val
      sum = sum+val
20    continue
      write(*,*)' '
      write(*,*)'sum =',sum
     endif
     stop
     end
```