Introduction to Scientific & Engineering Computing BIL 102FE (Fortran) Course for Week 6

Dr. Ali Can Takinacı

Assistant Professor

in

The Faculty of Naval Architecture and Ocean Engineering

80626

Maslak – Istanbul – Turkey

REPEATING PARTS OF A PROGRAM (LOOPS)

A very large proportion of mathematical techniques rely on some form of iterative process,
while the processing of most types of data requires the same,
or similar actions to be carried out repeatedly for each set of data.

One of the most important of all programming concepts
is the ability to repeat sequences of statements
either a predetermined number of times
or until some condition is satisfied.

Program repetition and the do construct

The repetition of a block of statements a number of times is called **a loop**.
It is called **a do construct** and takes one of the following forms:

```
do count = initial,final,inc
    block of statements
end do
```

or

```
do count = initial,final
    block of statements
end do
```

or simply

```
do
    block of statements
end do
```

A loop created by use of a do construct is called a do **loop**.

Count-controlled do loops

The first statement of a do loop is called a do statement.

```
do count = initial,final,inc
do count = initial,final
do
```

The first two alternatives define **a count-controlled do loop**, in which an integer variable known as the **do variable**, is used to determine how many times the block of statements which appears between the **do statement** and the **end do** is to be executed.

The meaning of the second alternative, in which *inc* is absent, is that the loop is executed for count taking the value *initial* the first time and the loop is executed with *initial+1* for next time, and so on, until it takes the value *final* on the last pass through the loop.

The formal definition of this process is that
when the do statement is executed
an **iteration count** is first calculated using the formula

`max``((`*`final-initial+inc`*`)/`*`inc`*`,0)`

do statement	**Iteration count**	**do variable values**
do i=1,10	10	1,2,3,4,5,6,7,8,9,10
do j=20,50,5	7	20,25,30,35,40,45,50
do p=7,19,4	4	7,11,15,19
do q=4,5,6	1	4
do r=6,5,4	0	(6)
do x=-20,20,6	6	-20,-14,-8,-2,4,10,16
do n=25,0,-5	6	25,20,15,10,5,0
do m=20,-20,-6	7	20,14,8,2,-4,-10,-16

At the beginning of the execution the value of *count* is *initial*, and on each subsequent pass its value is increased by *inc*.

If *inc* is absent then its value is taken as 1.

The effect of the **max** function is that if *final<initial* then the loop will not be executed at all.

The do variables *count, initial* and *final* must be a scalar integer (for Fortran Power Station User's can use the variables as real also).

Because of its special role the do variables between the initial do statement and the corresponding end do statement can not be altered while the do loop is under processing.

It must always be remembered that once the loop has been completed the value *count* will be *final+inc*.

Example: Write a program which first reads the number of people taking an exam.
It should then read their marks (or scores) and
print the highest and lowest marks,
followed by the average mark for the class.

Analysis: It will be used a do loop to repeatedly read a mark and use it to update the sum of all the marks, the maximum mark so far, and the minimum mark so far. The initial value of the cumulative sum obviously starts zero. The maximum and minimum marks are both set to the first value (first mark or score) of the do loop by using an if construct, which obviously works for only the first loop (i=1 case).

```
      program examination_marks
      integer :: i,number,mark,maximum,minimum,total
      real :: average
!     initialize variable
      total = 0
!     read number of marks , and then the marks
      print *,"how many marks are there"
      read *,number
      print *," please type ",number," marks, one per line"
!     Loop to read and process marks
      do i = 1 , number
      read *, mark
!     initialize max. and min. marks for only the first loop.
      if (i==1) then
!     this if construct is executed for the case only i=1.
      maximum = mark
      minimum = mark
      end if
!     on each pass ,update sum,maximum and minimum
      total = total + mark
      if (mark > maximum)then
      maximum = mark
      else if (mark < minimum)then
      minimum = mark
      end if
      end do
!     calculate average mark and print out results
      average = real(total) / number
      print *,"highest mark is",maximum
      print *,"lowest mark  is",minimum
      print *,"average mark is",average
      end program examination_marks
```

More flexible loops

The third form of the **do** statement together with a statement, **exit**, which causes a transfer control to the statement immediately following the **end do** statement. When executed all the remaining statements in the loop are omitted and it is always used in association with one of the control statements (if constructs).

For example, the following loop will continue to be executed until the value of term becomes less than the value small:

```
do
 .
if (term < small) then
exit
end if
 .
 end do
! After obeying the exit statement execution continues
! from the next statement (that is, after this comment)
```

This form of do statement does incur the risk that the condition for obeying the **exit** statement may never occur. In that situation the loop will continue executing until the program is terminated by some external means such as exceeding a time limit or switching off the computer. In order to avoid this situation a fail-safe mechanism in which a do variable is used to limit the number of repetitions a predefined maximum, should be used. Therefore the simple example given above should be designed with such a mechanism:

```
do count=1,max_iterations
  .
if (term < small) then
exit
end if
  .
end do
! After obeying the exit statement, or after obeying
! the loop max_iterations times, execution continues
! from the next statement (that is, after this comment)
 .
 .
```

Giving names to do constructs

It is possible to give a name to do construct by preceding the do statement by a name,
which follows the normal F rules for names
and is separated from the do by a colon,
and by following the corresponding end do by the same name.

```
block_name:    do
                .
                .
          end do block_name
```