

CHAPTER 3

ELEMENTS OF COMPUTR PROGAMMING - II

3.1 INTRODUCTION

In the earlier chapter we have been introduced to elementary principles needed for writing programs. The input- output statements, do loops and if statements were illustrated with examples. As mentioned before, we need to pay attention to the ‘peculiarities’ of FORTRAN such as beginning the lines at column 7, writing line numbers in columns 1 to 5, and treating all variables whose names begin with i, j, k, m and n as integer variables. There are alternatives to this but we want to keep things simple to begin with! Other languages have their own unique features. In the present chapter, we will study subscripted variables, functions and subroutines and learn how to transfer information to and from files. We will not spend too much effort on the format statements and only give examples of different formats for reading and writing data from files.

3.2 SUBSCRIPTED VARIABLES

There are large groups of variables which are extremely similar in character and it is very laborious to give distinctive names to each value of the variable. Consider the average temperature for every hour during the whole year. If each temperature has to be given a unique and distinctive name, we will need 365×24 names and the program to even read this data will be in thousands of lines. An elegant way to circumvent this difficulty is to use subscripted variables.

Consider the statement:

dimension tempval (365, 24)

This means that the variable tempval is subscripted and has 365 rows and 24 columns. The value contained in, say tempval (101, 13) is the value in the 101st row and 13th column. It may refer to the average temperature on the 101st day of the year in the 13th hour of the day. Subscripted variables are a ‘must’ in solving matrix problems and are a compact way to store data. Storing data and retrieving it is a major hassle in modern times even though data storage devices are becoming cheaper and growing in capacity. Can you ever imagine today that early computers had no hard disks? How will you read the temperature data for the whole year? A solution will be

```
do 100 i = 1, 365

do 90 i = 1, 24

read (*, *) temp (i, j)
90 continue
100 continue
```

This is a correct solution with a severe problem. You will have to input 8760 values of temperatures in as many lines. Not only it is tiring, but imagine making a few mistakes while typing. You will have to repeat the whole process again!

3.3 READING FROM AND WRITING TO FILES

A convenient way to solve the problem is to read from files and also write the voluminous data to files rather than to the screen. This is done as follows

```
program averagetemp

DIMENSION tempval (365, 24)
open (unit = 11, file = 'input.dat')
open (unit = 12 ,file = 'output')
read (11, *) ( ( tempval (i, j), j = 1,24), i= 365)
```

```

c   calculate the average temp each day & write to file output
    do 100 i = 1, 365
      xx= 0.0
      do 90 j = 1,24
        xx = xx + tempval (i,j)
90   continue
      avtemp = xx /24.0
100  write (12, *) 'day no =', i, 'average temp = ', avtemp
      close (12)
      close (11)
    end

```

The above program reads 8760 values from file 11 (which is named input.dat and writes the average temperature of each day into the file output. The line read (11, *) implies reading from file 11, and write (12, *) means write into file labeled 12. The read statement in the program has implicit do loops incorporated into it.

```
read (11, *) (( tempval (i, j), j= 1, 24), i= 1,365)
```

The values of tempval (i, j) are read as tempval (1, 1), tempval (1, 2), tempval (1,3) tempval (1,24), tempval (2,1) tempval (2,24), tempval (3,1) tempval (365, 24). You can input the values of temperature at leisure into input.dat and check that the readings are correct and then run the program.

3.4 THE MATRIX MULTIPLICATION PROGRAM

A program to multiply two matrices is given below. We shall consider only square matrices. The (i, j) th element of the product C of two $n \times n$ A and B matrices is

$$C_{ij} = \sum_k A_{ik} * B_{kj}$$

```

program matrix multiplication
DIMENSION a(100,100), b(100,100), c(100,100)
open (unit=11, file='mata.dat')
open (unit=12, file= 'matb.dat')
open (unit=13, file= 'matc.dat')
write (*,*) 'value of n of the n x n matrix is = '
read (*,*) n
read (11, *) ( ( a(i, j), j = 1,n ), i = 1,n)
read (12, *) ( ( b(i,j), j =1,n ), i = 1 ,n)
1  continue
   do 10 i=1, n
     do 10 j=1, n
       sum = 0.0
         do 5 k= 1, n
5           sum = sum + a(i,k) * b(k,j)
         c(i,j) = sum
10        write (13, *) ' c ( i, j) = ', c(i, j)
       close(13)
     close(12)
   close(11)
   end

```

You can open the file output and check your calculated values of the elements c_{ij} in file matc.dat Make sure that there are correct number of matrix elements of a and b in files mata.dat and matb.dat respectively.

3.5 ARRANGING NUMBERS IN A GIVEN ORDER

Storing a given set of data in different ways is an important problem in computing. In a dictionary, we need to arrange all the words in well-defined ways. If a new word has to be added to the dictionary, we need to check the letters of the word from left to right with the corresponding letters of the words in the dictionary and place the new word in right place. To arrange a given set of real numbers in an ascending order we first read the numbers from a file. Next find the smallest number and place it as the first element of a new array, the next smallest element as the second array and continue this process till all the numbers are arranged in an ascending order. The program to do this operation is as follows.

```
program to arrange the number in an ascending order
c  read the data from file input and write to file output
dimension a(500),result(500)
open (unit=15, file = 'input')
open (unit=16, file = 'output')
write(*,*) 'input n(the no.of points) on screen'
read (*,*) n
do i=1,n
read(15,*) a(i)
result(i)= a (i)
end do
do 100 i= 1, n-1
do 50 j=i+1,n
small = result(i)
if (result(j) .lt. small) then
result(i)=result(j)
result(j)=small
end if
50 continue
```

```

100 continue
    do 200 j= 1,n
    write (*,*) result(j)
200 continue
    close(16)
    close(15)
    end

```

3.6 FUNCTIONS AND SUBROUTINES

In the last program, we arranged the numbers in a file in a given order. Suppose we want to arrange three more sets of data in an ascending order. One way to do this is to repeat those sets of lines which do the same job of arranging the numbers in the desired order. But this way of repeating the sets of lines is not only inefficient and laborious, but also impractical if we need to perform the task an arbitrary number of times. Programming languages solve this problem by setting aside a set or group of instructions as separate functions or procedures in a program. The way a procedure or a subprogram or a subroutine is written is given below

```

program main
c this program reads numbers from files and arranges them in an
c ascending order
dimension a(100), b(100), c(100)
dimension result(100)
open(unit=18, file= 'input1')
open(unit=19, file= 'file2')
open(unit=30, file='result1')
open(unit=40, file='result2')
write(*,*) ' no. of data points in file input1='
read (*,*) n

```

```

read (18,*) (a(i),i=1,n)
call order (a, result, n)
write(30,*) (result(i), i=1,n)
write (*,*) 'no. of data points in file file2'
read(*,*) m
read(19,*) ( b(i),i=1,m)
call order (b, result, m)
write(40,*)(result(i),i=1,m)
end
subroutine order ( a, w, kk)
dimension a(100), w(100)
do 5 i =1, kk
5 w(i)= a(i)
do 10 i=1, kk-1
small=a(i)
do 20 j=i, kk
if(w(j) .lt. w(i)) then
w(i)= w(j)
w(j) = small
endif
20 continue
10 continue
return
end

```

Execute the program and see the output files result1 and result2. Note that if there is character c in the first column, the line is a comment line and is not executed

Listed below are a few salient features of subroutines

- 1) A subroutine is accessed or invoked in the main program by a call statement. e.g,
call order (a, result, n)
- 2) The subroutine will begin with the line subroutine order (a, w, kk)
- 3) The names of variables in the call statement and subroutine statement need not be identical, but the variable types in the parenthesis should match in the call and subroutine statements e.g. 'a' has a dimension of 100 in both main and subroutine and is an array of real numbers. Result and w are both arrays of dimension 100. 'n' and kk are both integer variables.
- 4) When the calculations are done in the subroutine, it closes with a return and end statement and not just an end statement or a stop and end statements, i.e., return statement is a must in a subroutine.
- 5) The line numbers and variables in the subroutine are independent of the line numbers and variables in the main program, except for the common variables passed through the parenthesis of the subroutine statement.
- 6) The variables in the parenthesis are common between the main program and subroutine. If the subroutine requires a large number of variables of the main program, this can be done through common statements as given below.

Program main

Common/com1/a (100), b (100), c (100)

Common/com2/a, b, c, r1, r2, p1, p2

Common/com3/x, y, z

.

.

Call sub 1

Call sub2

Call sub3

End

```

Subroutine sub1
Common/com1/a1 (100), b1 (100), c1 (100)
Common/com3/x, y, z
.
.
Return
End
Subroutine sub2
Common/com2/a, b, c, a1, a2, a3, a4
.
.
Return
End
Subroutine sub3
Common/com2/ a, b, c, a1, a2, a3, a4
Common/com3/x, y, z
.
.
Return
End

```

Each common statement such as common/com3/x, y, z is called a common block. Only those variables whose values are need for a given subroutine need to be transferred to that subroutine. In subroutine sub1, com1 and com2 are ‘transferred’. In subroutine sub2, only com2 is transferred, which in subroutine sub3, com2 and com3 are transferred.

3.7 FUNCTIONS

In programs that need to transfer a lot of information to as well as from a subprogram, we need to use subroutines. If only one value is desired from a subprogram such as the value of a function such as $\sin(x)$ or $\cos(x)$, then a subprogram is called 'function' is used. The use of a function $\sinew(x)$ is illustrated below. Here $\sin(x)$ is evaluated using the Taylor series expansion of $\sin(x)$

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

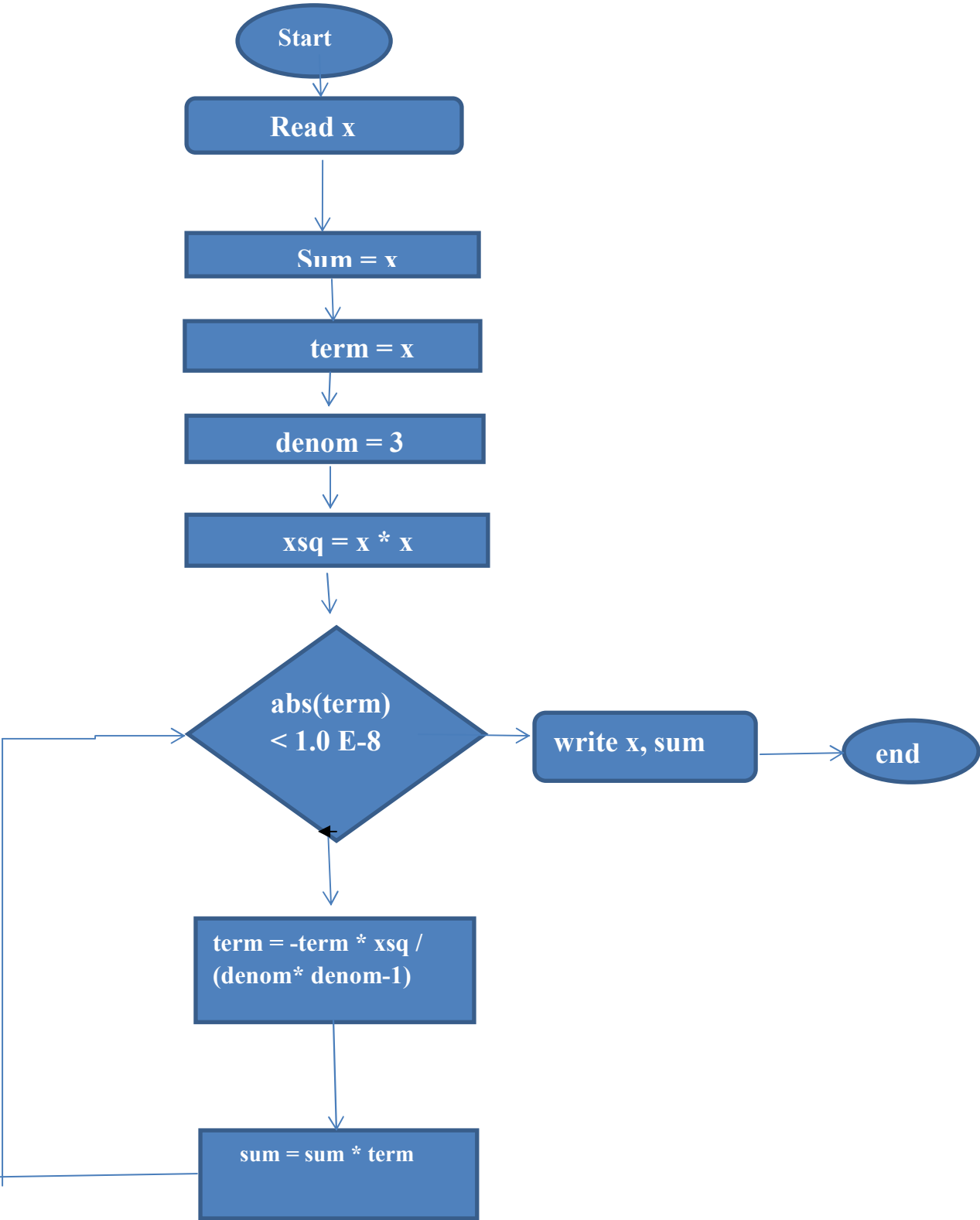
```
program for sinnew

write (*,*) 'input the value of variable x'
read (*, *) x
x=deg*3.14159/180.0
y =sinnew(x)
z= sin(x)
write (*,*) 'x, sin(x), sinew(x) =' , x, z, y
end

FUNCTION sinnew(x)

sum=x
term=x
denom=3.0
xsq=x*x
10 term=-term*xsq/(denom*(denom-1))
sum=sum+term
denom=denom+2
aterm=abs(term)
if (aterm .gt. 1.0E-8) then
go to 10
```

A flowchart for the above program function is shown below



```
endif  
sinnew= sum  
return  
end
```

In the flowcharts, start and stop are written in circles, input/output in oval shapes, statements in rectangles and conditional operations in diamond shapes. The flow of control is shown through arrows.

3.8 summary

We have seen now different aspects of programming such as input/output, do-loops, conditional (if) statements and subprograms/functions. You will improve your skills through practice. We have not spent any time in format statements which allow you to read/write data in integer, real (0.3333E+04 or 3333.0000) or character formats. You may study these using the books/websites that are freely accessible. You will also need to be comfortable with the commands of the vi editor which is quite powerful. If not, use the Pico editor which is easier to master. Programming logic and mathematical logic are different. In mathematics $1/3 + 1/3 + 1/3 = 1$ whereas on a computer, the sum is 0.99999999 to a desired precision. Real, integer, logical and character variables have to use separately and systematically. Debugging the errors is a good skill and that too comes with practice. Reading and writing to/from files has to be done with precision. If you need to read 8760 numbers and there are only 8759 data values in the file, you will get an error like end of file in until 11. While initial efforts in programming seem to cause discomfort, it is a greater challenge to make programs more efficient, compact and broad based.