Chapter 3

Elements of computer programming– II

Objectives:

i) Using dimension statements and subscripted variables
ii) Writing function subprograms
iii) Writing and calling subroutines
iv) Writing a program for matrix multiplication
v) Writing a program for arranging numbers in an ascending order
vi) Reading from files and writing date into files

Key Words: subscripted variables, dimension statements, functions, subroutines, matrix multiplication, reading from files, writing to files, file unit numbers.

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, l,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)
90continue
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! We have considered a two dimensional array. One dimensional arrays arise naturally when we have repeated observations on a single quantity such as the marks of a student in a course. If there are 100 students, the marks can be stored in an array

scores(100)

Let us see how to calculate the average marks contained in this array through a do loop without using a continue statement.

```
sum = 0.0

do i = 1,100

sum = sum + scores(i)

end do

ave = sum/real(100)
```

The end of a do loop is effected by end do without using line numbers. Note that we have divided by real(100) rather than the integer value of 100. One has to be very careful with integer arithmetic as, m = 2/3 will give a value of zero (integer division), while ratio = 2.0/3.0 gives a value 0.66666666

While defining arrays, we need to have a good idea of the computer memory. This is particularly important for large programmes with a large number of subscripted variables. If the computer memory is 500 MB, can you have an array of size 600000000? It will not just fit into the memory!

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)
ccalculate the average temp each day & write to file output
c      this is for a non-leap year. For a leap year, we have 366 days
c      How will you write a more general program valid for both types of years?
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
```

Through the statement, open (unit = 11, file = 'input.dat') the file with name input.dat is assigned to a unit (or number) 11. This statement opens the file input.dat for reading data. When the job of reading is completed, it is good to close the file by the statement close (11). In the read statements, the number 11 is used while reading data from this file.

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. What we get from the program is the average temperature of the year. The problem of keying in all these values remains!! For this there is no choice and the same is true for making the measurements. The problem of keying in the data on to the computer when a large amount of data is involved, such as in the case of the NMR free induction decay data has been solved by having good analog to digital conversion cards. These cards enable a direct transfer of digital data to the computer (from an instrument) for direct processing.

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 matrix C of two n×n matrices A and B is

$$C_{ij} = \sum A_{ik} * B_{kj}$$, and the summation is from 1 to n.

```
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 nxn 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)
1continue
do 10 i=1, n
do 10 j=1, n
sum = 0.0
do 5 k= 1, n
5sum = sum + a(i,k) * b(k,j)
c(i,j) = sum
```

```
10 write (13, *) 'c(i, j)=', c(i, j)
c     note that we have written the product matrix in the loop itself.
c   as an exercise, write the matrix outside the main loop and write the product
c     in such a way that the rows and columns of the product are nicely arranged
c       in rows and columns
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 anew 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.There are many types of 'sorting' algorithms and the present one is the most direct, even if not very efficient.

```
 c    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
c     you may work through the logic of the program with let us say, 4 numbers.
do 200 j= 1,n
write (*,*) result(j)
200  continue
      close(16)
      close(15)
      end
```

3.6FUNCTIONS 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. This feature gives the programmer the flexibility of modularity, i.e, she can separately program different modules that are required to solve a major problem. Take the case of the railway reservation program. Data collection (on line tickets as well as tickets at various counters) is done by a separate module of the program. Taking the print outs for a given train from a specific starting point will be done by another different module of the program. Conversion of RAC and waiting list tickets into confirmed reservations will be done by another module. As an exercise, see how many modules will be required to print the grade cards of all the students of a university after the semester-end examinations.

```
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=w(i)
do 20 j=i+1, 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 parentheses 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**.**In some compiles, return may be sufficient and end may not be needed.
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 parentheses 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, while in subroutine sub3, com2 and com3 are transferred.

3.7 FUNCTIONS

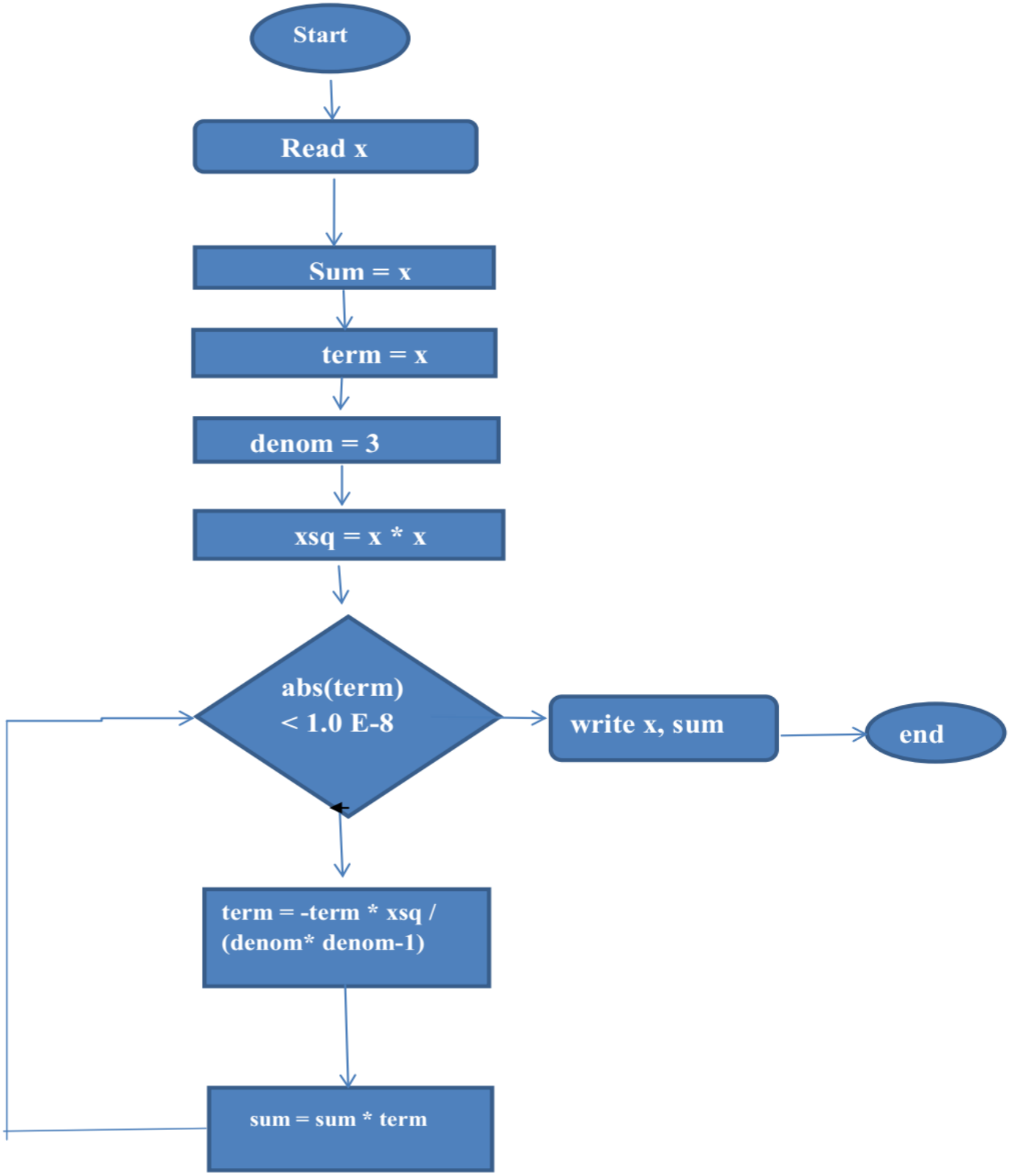
In programs that need to transfer a lot of information to as well as from a subprogram, we need to use subroutines. If only one value is desired from a subprogram such as the value of a function such as sin(x) or cos (x), then a subprogram is called 'function' is used. The use of a function sinnew (x)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

$$sinx = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} \ldots$$

program for sinnew

```
      write (*,*) 'input the value of variable xin degrees'
      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
endif
      sinnew= sum
      return
      end
```

A flowchart for the above program function is shown below



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/functionscan be written and connected. 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 be used 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.

PROBLEMS

1. Given a 4 digit number representing a year, write a program to find out if it is a leap year
2. Give the flow chart to find the sum of digits of an integer,(say, up to 8digits).
3. Given a set of integers, write a program to find the numbers which are palindromes [e.g. 123321 is a palindrome. A **palindrome** is a word, phrase, number, or other sequence of units that may be read the same way in either direction, with general allowances for adjustments to punctuation and word dividers.

4. Construct an algorithm and a flowchart to calculate the value of a, where

   $a = \frac{x-y}{x^y} - \frac{y-x}{y^x}$ For values of x =1, 2, 3… 10 and for values of y = 0.1, 0.2, 0.3… 1.0.

5. Construct an algorithm and a flow chart for a module to approximate the cosineof an angle(given in radians):

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots$$

Then use this module to test the identity $\cos(3a) = 4\cos^3(a) - 3\cos(a)$ for a = 1, 2 …5

6. Write a function that computes the factorial (i!) of a number i. (You cannot go too far with this in integer mode). Then use that function in a second subprogram to compute the series shown below.

$$\sum_{i=1}^{i=n} (-1)^{(i+1)} \frac{1}{i!}$$

   The main program should read in the value of n, then compute the series for the appropriate number of terms, and print out the series terms. All computations should be done in the function subprograms.