Chapter-2

Elements of computer programming-I

Objectives:

i) Distinguish between flow charts, algorithms and computer programs
ii) Perform repeated operation using do loops
iii) Use if statements to effect branching in the programs
iv) Write a program for solving a quadratic equation

Key Words: flow charts, algorithms, computer programs, do loops, if statements, compilation, execution, input, output, programming errors.

2.1 Introduction.

In this chapter we will outline the basic principles of computer programming for numerical computations and give a few examples of elementary programs.

A program (or software) is a sequenceof instructions or operationswhich are so organized that they are executed or carried out on a computer. Consider the example of finding the largest of 3 numbers. Here, we need to read the three numbers, compare them with one another and write the largest number of the computer screen or a computer file. Take another example of program for calculating product of two 6x6 matrices. In this program, we have to input the values of elements of the two matrices to the program. The program has the code or the steps or the statements (program lines) to find the elements of the product matrix and write out the results either on the computer screen or in a file on the computer. On the other hand, there could be very complex programora commercial 'package' such as the one that handles the Indian railway reservations. The program runs on some "large" computer referred to as a server. The program has the complete information on the availability of the seats on all the trains. A person wanting

to reserve a seat or a berth logs into the system through the internet (using which he/she has already created an account on the reservation system's program), gives the information required for reservation. If areservation is available as required, he gets an output on the screen, which is the ticket which can be printed out. Of course, he/she has to pay the required amount through internet banking. Such a program has tremendous advantages over the laborious book keeping methods that were used in the past.

We can conclude from above that some of the ingredients in a program are input, output, repeated operations (such as computing each element of a matrix one by one), comparisons (is the seat available or not available), information storage, doing mathematical operations such as additions, multiplications and other functions such as those found on a calculator. Using these features, it is possible to design or write programs for assisting highly complex functions such as automatic landing on the moon or running unmanned trains or for mobile communications.

There are several programming languages such as FORTRAN (formula translation), C++, and java. Some languages, such as basic or Pascal are becoming obsolete. New languages may emerge. Our emphasis here will be on programming logic. While we will use mostly FORTRAN, we will also try to illustrate with otherlanguages too.We will prefer to use the Linux operating system as it is available freely in public domain. The compilers (for FORTRAN, C++, and java) which convert the program code into a form/file that can be executed on the computer (such a file is called an executable file) are freely available and so are the software such as Scilab, Xmgrace and Avogadro, whose use we will illustrate in Chapters 8 and 9.

2.2 Algorithms

In the earlier days of programming, it was a common practice to prepare the flow chart for a task you want to carry out. The important preparation before writing program is to know the algorithm for solving

the problem at hand. An algorithm is a set of well-defined steps that need to be carried out for solving a problem.The program expresses the algorithm in a suitable programming language. Of course, the steps need to be unambiguous. As an example, consider the following statements

a) If the value of a real number x is negative i.e. if $x<0$, then its absolute value is $-x$

b) Take the square root of object 'w'.

In the above statements, statement 'a' is unambiguous and statement b is vague, because if the object 'w' is a word, we cannot take its square root. If the object 'w' is a negative number, again the square root is a problem, unless we are dealing with complex numbers. Consider two examples of algorithms.

Example 1: Conversion of temperature in Celsius to Fahrenheit

Step 1: Get the value of temperature in Celsius. Let it be designated as C

Step 2: Obtain the value in Fahrenheit using $F = 32 + (9/5)\ C$

Example 2: Finding the Fibonacci series with $F(1) = 1$, $F(2) = 2$ and $F(n) = F(n-1) + F(n-2)$ for $n = 3,4,5$ ….

Step 1: Initialise the first two members of the Fibonacci series as $F(1) = 1$, $F(2)$.

Step 2: Ascertain the total number of members of the series that you want to calculate, let us say 100.

Step 3: $n = 3$

Step 4: Calculate $F(n) = F(n-1) + F(n-2)$

Step 5: Increment n by 1. If $n = 7$, then, this step converts n to 8.

Step 6: If n is less than 101, go to step 4.

Step 7: If n = 100, write all the 100 values of F(n)

2.3 Elementary functions

Let us first list a few acceptable statements in FORTRAN

```
y= x + z / c + a * b
p= sin (theta) + alog (conc)
q= exp (-energy / (boltz *temp))
 r = d ** f
t = sqrt (23.0)
a = a + b
```

In line 1, adding, dividing (/), multiplying (*) and equating (= implies replace the lhs [left hand side] variable by the value calculated on the rhs [right hand side]) are the operations that are being carried out. In mathematics a = a + b is wrong but it is perfectly fine in a computer program because the meaning of a = a + b is replace the value of 'a' by the value of a + b. The second line indicates the sin and log functions and the names of variables are theta and conc. In FORTRAN, variables beginning with i, j, k, l, m and n as the first letters are integer variables. Another vital feature is that you begin typing from the seventhcolumn and not the first. Each programming language comes with its own peculiarities! With the passage of time, the peculiarities have decreased and there are searches for more "universal" languages. Newer versions of FORTRAN contain more features of C++, and so on.

The third line uses the exponentiation function. The forth line is the FORTRAN **expression** for raising d to the power fof d^f. The fifth line has the square root function.

To run a program that does the above operations, we need to do a few more things.

1) Firstly the program has to know the values of variables x, z, a, b, etc. This can be done either by assigning values to these variables at the beginning of the program or read their values from the computer screen through a read statement.
2) We may want to write the results on the screen
3) The program should have an end statement. And most important, we need to convert the FORTRAN code into a version that can be executed on the computer.

2.4 A sample program

A program written in fortran also needs to have an extension .f , e.g. the file name should be prog.f. On a Linux system you can use the Pico or the vi editor to create a file as follows.

```
program test
    read(*,*)x,z,c,a,b,theta,conc,energy,
   1boltz,temp,d,f
    y = x + z / c + a * b
    p = sin (theta) + alog(conc)
    q = exp (-energy/(boltz * temp))
    r = d * * f
    t = sqrt (23.0)
```

```
    a = a + b
write(*,*)y,p,q,r,t,a
    stop
end
```

[In newer versions of Fortran, the stop statement (the statement before the end statement) may be left out]

After the file is saved, it needs to be compiled by typing f77 prog.f

On some linux systems, f77 may not be available, but the compiler gfortran may be available. In some cases, you may need to install these files. In the case of gfortrran compiler, the compilation statement will be

gfortran prog.f

The compilation command creates an executable file with name a.out

You may verify this by using the linux command ls

The ls command lists all the files in the present working directory.

The execution is done by typing ./a.out

After this, you need to input the values of x, z and the other variables on the screen.

If these values are 2.0, 3.0, 5.0, 6.0, 9.0, 30.0, 0.2, 10.0, 8.314, 298.0, 2.0and 3.0 (a total of 12 quantities), see what will be the values of the output on the screen.

Note:

1) The read statement in the program extends beyond one line. To continue the statement on the second line, type character such as 1 at the sixth column of the next line and continue typing.
2) Whenever quantities with units are used, ensure that proper care is taken. In the data, energy was in J/ mol and hence the value of boltz is 8.314 J/ (mol. K) and temperature is in K.
3) a.out is an executable file and the execution of the program is done by typing ./a.out

When using the computer screen for input or output, it is convenient to print out sentences or questions which will prompt you (i.e, the programmer)to give the correct input and indicate the nature of the output as well. This is done by as follows.

```
Write (*, *) 'input the value of integer n'

Read (*, *) n

Write (*, *) 'the value of n =', n
```

In the above lines the string of characters between the single quotation marks '….' are printed verbatim on the screen.

The power of computing comes from the ability to perform repeated operations quickly and the designs in the programs for conditional flow of control in the program. These two aspects will be illustrated with examples below. The meaning of (*,*) will be elaborated later. It means that the reading or the writing of the variables is done from the computer screen.The data entry is of course done by using the key board.

2.5 Do loops

Suppose we want to calculate the value of sin (x) for values of x ranging from 0 to 2π.The values of 'x' are to be given in radians. The default unit (i.e, the natural unit when nothing else is specified) of the angle in the trigonometric functions is radians.If there are only a few values of x, we can calculate sin (x) for each of those values and print them. However, if there are 100 or more values of x, we cannot write a 200 line program just for these repeated identical calculations. These are done by what are called do loops (or for loopsin C++). The program is as follows.

```
c     calculate the value of sin (x) at 101equispaced points between 0 and 2pi (including the end points).
twopi = 2.0 * 3.1415926
c     this value of pi to eight decimal places is said to be in single precision
c     value of pi in double precision (sixteen decimal places) is 3.141592653589793
c     there are higher precisions and are used occasionally when more accurate results are needed
do 10 i =1, 101
x  = real (i-1) * 0.01 * twopi
y = sin (x)
write (*, *) 'step number and the values of x and sin (x) are  =', i, x, y
 10      continue
         end
```

The meaning of the do 10 i =1, 101 is: repeat the operations till theline which is labeled 10 (the line numbers are placed in columns 1 to 5) starting with the value of i=1 and ending with the value 101. When the flow of the control in the program reaches 10 continue, the flow goes back to the do statement and increments the value of the do loop variable i by 1. When i is 102, the flow does not go back to the do statement, but to the statement following (or next to) the do statement. In place of the 10 continue, we could have written as

```
         do 10 i=1, 101,1
         x= real ( i ) * 0.01 * twopi
         y= sin (x)
10       write (*, *) 'x and sin (x) =', x,y
```

real(i) is not required, we could just use 'i'. It is however a god practice to convert real to integer and integer to real whenever the programming steps necessitate it and obtained the same result. In the above

program line numbered 10 is an executable statement and the continue statement is not needed. We have used real (i) which is a function that gives the real value of i i.e. 1.0 in place of 1 which is an integer. We need to carefully distinguish between real, integer and other types of variables. Another feature was in theline do 10i =1, 100, 1 here, the last part '1' indicates that i is incremented by 1 every time. If we want to increment the variable i in steps of 2, we use

```
do 10 i=1, 100, 2
```

There are other alternatives to the do statement such as the one given below.

```
sum=0.0
do i=1, 100
sum = sum +(real (i))**3
end do
```

What does this program do? It adds the cubes of the first hundred integers (using the arithmetic of real numbers).

Here, there are no line numbers. There are 'do' and 'enddo' statements which do the same work as our "do 10 i = 1, 100" and "10 continue" statements.

2.6 The if statement

The'if' statement allows the branching of the flow of control in the program. Ordinarily the flow is top to bottom, i.e. from one line to the next line below it. Consider the following function of x, a function initially looking like a series of steps, and later, as a straight line.

$$
\begin{aligned}
&f(x) = 0 \text{ if } x < 0.0 \\
&f(x) = 5.0 \text{ if } 0.0 \le x < 5.0 \\
&f(x) = 10.0 \text{ if } 5.0 \le x < 10.0 \\
&f(x) = x \text{ if } x \ge 10
\end{aligned}
\qquad (2.1)
$$

The sketch of the function looks as shown below

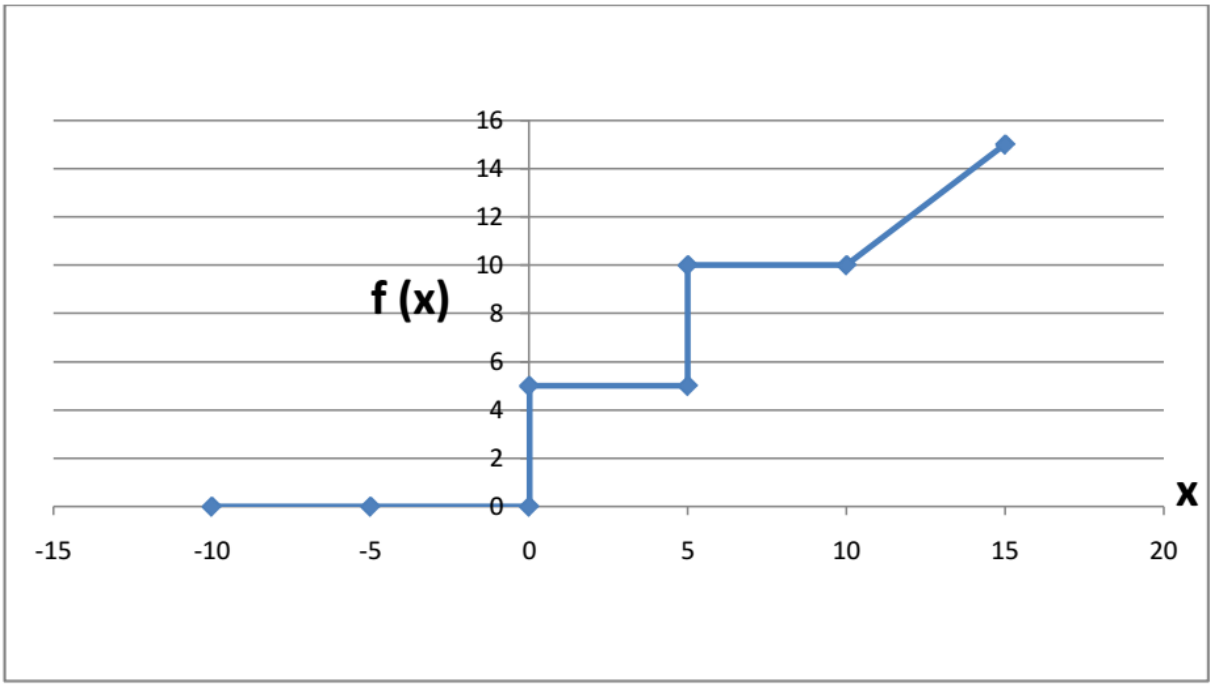

Fig. 2.1 A sketch of the function given in Eq. (2.1)

We need to write a program which gives the value of f(x) as per the above formula. This is given below.

```
read (*, *) x
    if (x.lt.0.0) then
funct = 0.0
else if (0.0 .le.x.and.x.lt.5.0) then
```

```
funct = 5.0
else if (5.0.le.x.and.x.lt.10.0) then
funct = 10.0
else
funct = x
 endif
write (*,*) 'x,funct=', x, funct
end
```

Please note the indentation conventions used above. These conventions enable one to easily visualise the original if from the several else ifs.

The symbols .lt., .eq. , .gt. , .ge. and .and. are relational statements(actually, operators). The group of statements between 'if ' … and 'endif 'is called an ' if block 'and it and executes the branching or the distribution (or redirection) of the flow of control as per the conditions satisfied.

2.7 Solution of a quadratic equation

The general form of a quadratic equation is

$$ax^2 + bx + c = 0 \quad (2.2)$$

The roots of this equation are

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The program has to read the values of a, b and c and print out the roots. If $b^2 - 4ac$ is less than zero, then the roots are complex and the program has to take care of this situation. In addition, if$a = 0$, we need to ensure that we (always) avoid dividing by zero. The program is as follows

```
program quadratic
    write (*, *) 'input the values of a, b and c:'
    read (*, *) a, b, c
    if ((a .eq. 0.0) .and. (b .ne. 0.0)) then
    x = -c/b
     twoa=2.0*a
    write (*, *) 'the solution of linear equation x=', x
    go to 100
    else if ((a .eq. 0.0) .and. (b .eq. 0.0)) then
    write (*, *) 'both coefficients a and b are zero'
    go to 100
    endif
    ww = b * b - 4.0 * a * c
    if (ww .lt. 0.0) then
    go to 50
    else
    rtofww = sqrt (ww)
    root1 =( -b + rtofww) / twoa
    root2 =( -b - rtofww) / twoa
    write (*, *) 'real roots 1 and 2 are', root1, root2
    go to 100
    endif
 50  continue
```

```
c     the roots are complex b**2 - 4 * a * c is -ve
      ww = 4.0 * a * c - b * b
      rtofww = sqrt (ww)
      realpt = -b / twoa
      impt = rtofww / twoa
      write (*, *) 'complex roots'
      write (*, *) 'root1', 'real part=', realpt, 'imaginary part=', impt
      impt2 = -impt
      write (*, *) 'root2', 'real part=', realpt, 'imaginary part=', impt2
 100  continue
end
```

In the above program, the cases of $a = 0$ and b= 0 as well as b $\neq$ 0 are considered separately at the beginning. Next, real roots are calculated when $b^2 - 4ac > 0$.

Once the calculation is completed, go to statement is used to send the flow of control to line labeled 100 or 50 as the case may be. In modern programming, line numbers are **avoided** as it is harder to debug a program where there is repeated branching. It is better to divide the program into several modules called functions or subroutines (subprograms or procedures) and access these modules as and when required in the program.

2.8 Summary

Perhaps a day will soon come when you can ask a computer to do the desired tasks by speaking into the computer. But until such time, the instructions have to be given to a computer in a language in which it understands the tasks to be performed. The main strengths of a computer are storage of a large amount of data and doing repeated operations until the user's criteria are satisfied. For example, you may want to do an SCF (self consistent field) calculation until the calculated energies are accurate to better than 1 part in a billion. Or, you may want to calculate the molecular dynamics (MD) trajectory of a liquid system for a time span of a nanosecond. The main ingredients of a program are 1) an **instruction** to carry out a mathematical operation (such as evaluating a formula for a given value of a variable), 2) **repeating** a calculation until a **condition** is satisfied, 3) **allocation of storage** space for calculated quantities such as the integrals evaluated in an SCF calculation or the coordinates and velocities in an MD trajectory, 4) reading **inputs** from files and writing the **output** to files as well as the computer screen, 5) **terminating** the program either on completion or giving messages if something has gone wrong with the execution of the program. For getting a program in an executable mode, the program written in a programming language has to be converted into an executable file. Suitable compilers are available for performing this task. A compiler converts a program in a high level language (such as Fortrtan, C, Java, etc) into a machine level code. This machine level code can not be edited as we edit a fortran program. If there is a programming error (you will see these in abundance), the program has to be corrected and recompiled. The machine level code is referred to as an executable file and you will see it as an a.out filelisted in your directory. It will be nice tofamiliarise yourself with the common unix/linux commands as mkdir, ls –l, rm, mv, cp. Also,

distinguish between the insert mode and the edit mode of the vi editor.It will also be very good if you can install a linux operating system on your computer.

2.9 Problems

1. For standard mathematical operations, a programming language will have symbols to carry out the operation. The following list will illustrate the operations with examples

$$y = a\,x^2 + e^{-x} + \sqrt{z} + \sqrt[6]{w} - z/\log(x)$$

The log and ln functions are different.Notice that the expression has several algebraic operations. In FORTRAN language, this will be expressed as

```
y = a * x ** 2 + exp (- x) + sqrt (z) + w ** ( 1/6 ) – z / log (x)
```

Rising to a power is done using the symbols **. Exponentiation is done using exp, square root through the symbol sqrt and logarithm through alog. One has to ensure that one does not take the square root of a negative number and not take the logarithm of zero.

2. Calculate the value of the hydrogen 1s and 2s orbitals for values of r ranging from zero to 10 angstroms at the interval of 0.1 angstrom.Take theformulaeof the orbitals along with the normalization constants from any physical chemistry book. The exponential part has the term r/a_0 , where r is in Angstroms and $a_0$is theBohr radius equal to 0.529 Angstroms.

3. Write a program to convert temperature in ^0F *to* ^0C .

4. Write a program to convert Joules into eV, kcal/mol, hartree(a.u), and cm^{-1}.

5. Write a program to read the radius of a circle and compute its area and circumference.

6. What does the statement IMPLICIT real*8 mean in a FORTRAN code?

7. A cylindrical tub of diameter d has water filled up to a height of 0.2m. If a tap is opened to fill it and water flows at the rate of 2 liters/min, find the height to which it will be filled after 5 minutes. [$f * t = \pi r^2 h$]

f = volume of water flowing per minute, t = time in minutes, h = height of water level.

8. Define a constant and a variable

9. List some (at least 6) intrinsic mathematical functions in FORTRAN

10. Give the equivalent FORTRAN statements:

    a. $\frac{\alpha}{\sqrt{\alpha^2+\omega^2}}\cos(\omega t+\phi)$

    b. $\log c\sqrt{\frac{x}{yz}}$

    c. $\frac{1-e^{-\alpha\sqrt{x}}}{1+xe^{-|x|}}$

11. Write a program to find the sum of the individual digits of a 5 digit number.

12. Write FORTRAN equivalent statements for the following:

    a. $\log_{10} x+\cos 32^o+\left|x^2+y^2\right|+2\sqrt{xy}$

    b. $\frac{1}{a\sqrt{2\pi}}e^{\sqrt{2\sigma(x-m)^2}}$

13. Write a program to find the polar coordinates θ and r of a given point whose Cartesian coordinates are x and y.

    $r=\sqrt{x^2+y^2}, \quad \theta=\tan^{-1}(y/x)$

14. Given a point (x, y), write a program to find out if it lies on the x- axis, y- axis or at the origin.

15. Extend the above program to check if a given point (x, y) lies in the 1^{st}, 2^{nd}, 3^{rd} or 4^{th} quadrant.