

## CHAPTER-2: Elements of computer programing-1

### 2.1 Introduction.

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

A program is a collection of a set of instructions or operations which are so organized that they are executed or carried out in a systematic manner. Take the example of program for calculating product of two 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 on a file on the computer. Take another example of a complex program such as the program that handles the Indian railway reservations. The program runs on some “large” computer referred to 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 a reservation 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.

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, C++, and java. Some languages are becoming obsolete such as basic or Pascal. New languages may emerge. Our emphasis is on programming logic. While we will use mostly FORTRAN, we will illustrate with others 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. Nowadays it is less common. 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. If there is any ambiguity in a step then we cannot solve the problem. 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.

## 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 seventh column 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 fourth line is the FORTRAN (formula translation) of  $d^f$ . 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

The program also needs to have an extension .f i.e. 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

lboltz, 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

end
```

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

The execution is done by typing `./a.out`

After this, you need to input the values of  $x, z, \dots$  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, 2.0, 3.0 (a total of 12 quantities), the output on the screen will be:

Note:

- 1) The read statement in the program extends beyond one line. To continue the statement on the second line, type character such as `&` 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/(mole K) and hence the value of boltz is 8.314 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 key words which will prompt you 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 phrases between the single quotation marks '....' are printed 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 eth variables in done from the computer screen.

## 2.5 Do loops

Suppose we want to calculate the value of  $\sin(x)$  for values of  $x$  ranging from 0 to  $2\pi$ . 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 (or for in C++) loops. The program is as follows

c calculate the value of  $\sin(x)$  at 100 points

```
twopi = 2.0 * 3.1415
```

```
Do 10 i =1, 100
```

```
  x = real (i) * 0.01 * twopi
```

```
  y = sin (x)
```

```
  write (*, *) 'x and sin (x) =', x, y
```

```
10  continue
```

```
end
```

The meaning of the do 10 i =1, 100 is: repeat the operations till line 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 100.

When the flow of the control in the program reaches 10 continue, the flow goes back to the do statement and increments  $i$  by 1. When  $i$  is 100, 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, 100,1
```

```
  x = real ( i ) * 0.01 * twopi
```

$$y = \sin(x)$$

```
10    write (*, *) 'x and sin (x) =', x, y
```

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 line do 10 i=1, 100, 1 here, the last part '1' indicates that i is incremented by 1 every time. If we want to increment I by 2 each time, we use

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

There are other alternatives to the do statement such as

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

Here, there are no line numbers. There are 'do' and 'end do' 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 linear, i.e. from one line to the next line below it. Consider the following function of x.

$$f(x) = 0 \text{ if } x < 0.0$$

$$f(x) = 5.0 \text{ if } 0.0 \leq x < 5.0$$

$$f(x) = 10.0 \text{ if } 5.0 \leq x < 10.0 \quad (2.1)$$

$$f(x) = x \text{ if } x \geq 1$$

The sketch of the function looks as shown below

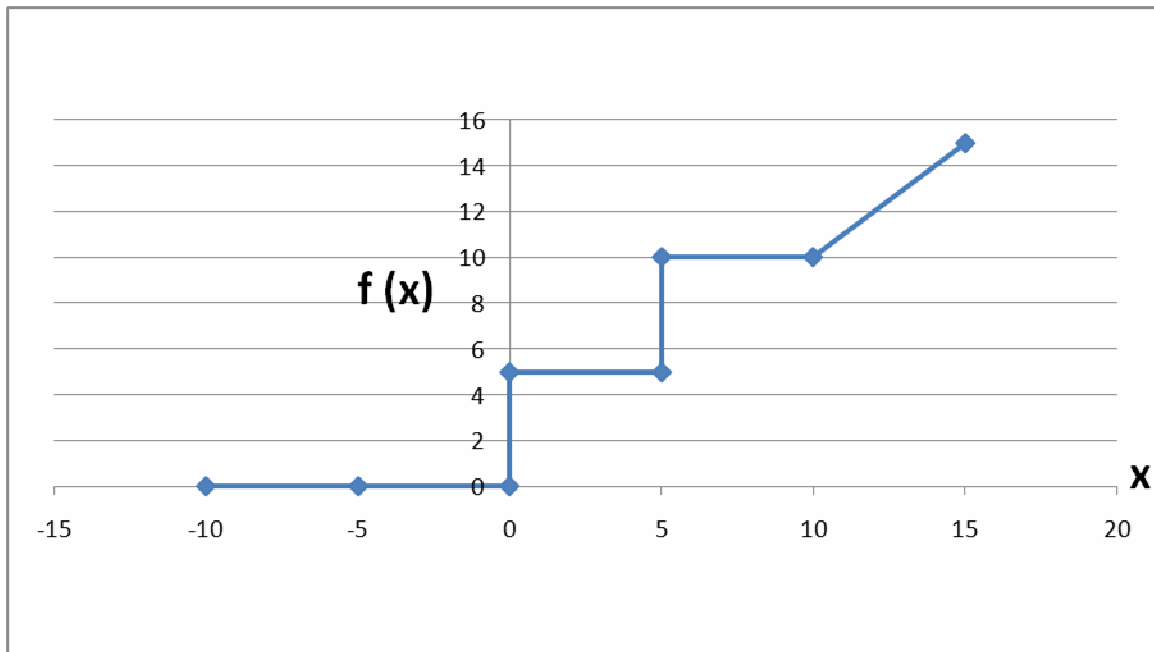


Fig. 2.1 A sketch of the function given in Eq. (2.1)

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

```

func = x
endif
write (*,*) 'x,func=', x, func
end

```

The terms `.lt.`, `.eq.`, `.gt.`, `.ge.` and `.and.` are relational statements. The group of statements between 'if' ... and 'endif' is called an 'if block' and it 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 the quadratic equation is

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

The roots of the 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
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 + rtofw) / (2.0 * a)
root2 = (-b - rtofw) / (2.0 * a)
write (*, *) 'real roots 1 and 2 are', root1, root2
go to 100
endif

50 continue

c the roots are complex  $b^2 - 4ac$  is -ve
ww = 4.0 * a * c - b * b
rtofww = sqrt (ww)
realpt = -b / (2.0 * a)
impt = rtofw / (2.0 * a)
write (*, *) 'complex roots'

```

```

write (*, *) 'root1', 'real part=', realpt, 'imaginar part=', impt
impt2 = -impt
write (*, *) 'root2', 'real part=', realpt, 'imaginay 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. We will take examples of these tasks in this lecture.

## 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

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

Notice that the expression has several algebraic operations. In Fortran language, this will be expressed as

$$y = a * x ** 2 + \exp(-x) + \text{sqrt}(z) + w ** (1/6) - z / \log(x)$$

Raising 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.

