Elementary “Principles” or ingredients of Computer Programming

- Input Output
- Declaration statements: real; integer; dimension; parameter; etc
- Execution statements: y = cos (x)
- Iterative statements or loops
- Control of transfer (“if” or “while” statements)
- Functions, subprograms or Procedures
- End of lines

Calculation of the value of sin (x) at 101 equispaced points

- c calculate the value of sin (x) at 101 equispaced points between 0 and 2pi (including the end points).
- twopi = 2.0 * 3.14159265
- c
- write (*,*) ‘101 values of sin(x)’
- do 10 i =1, 101
- x = real (i-1) * 0.01 * twopi
- y = sin (x)
- write (*, *) ‘x and sin (x) =’, x, y

c the above statement can be improved)

- 10 continue
- end

What you are expected to learn in this course

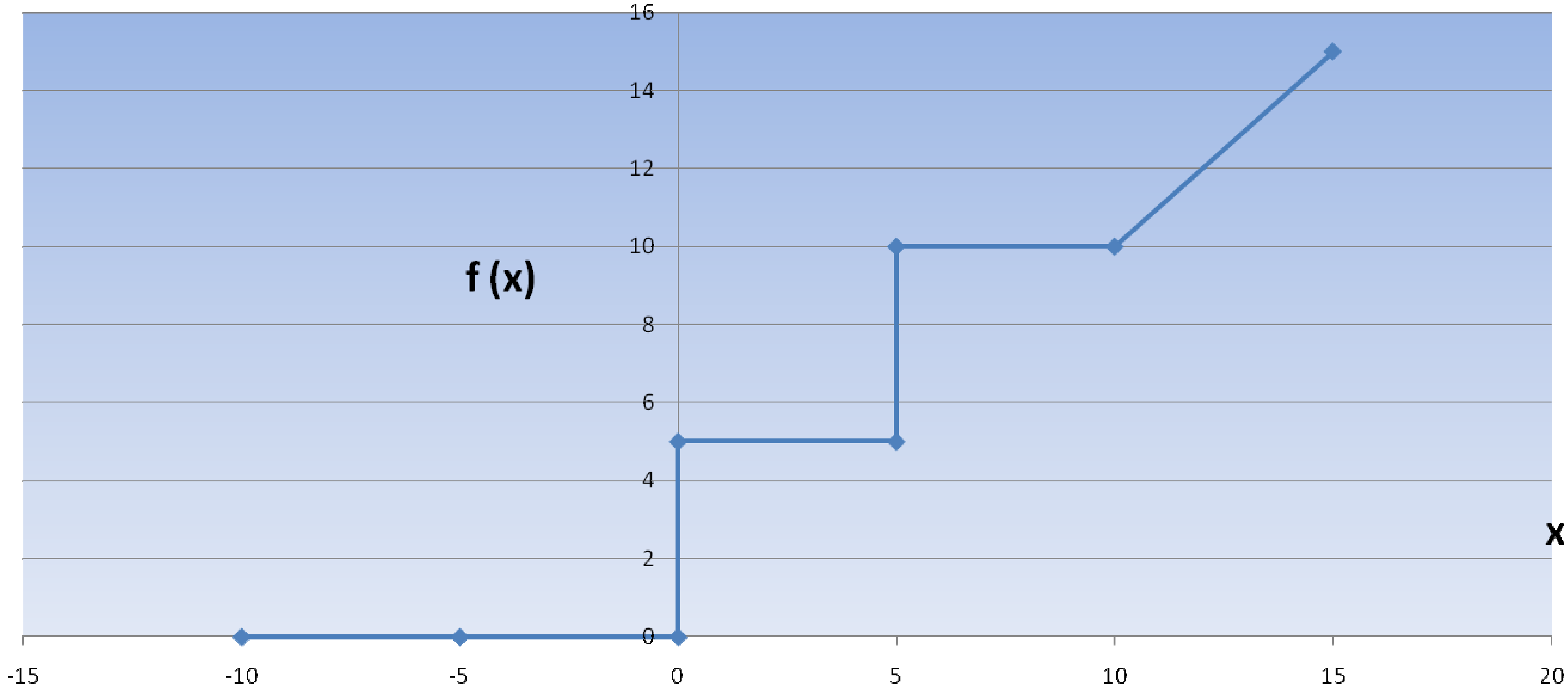
- Run elementary programs for calculating
- Fibonacci series, the sine function,
- energy conversion factors, use of dimensioned variables,
- arrranging numbers is ascending order,
- solve a quadratic eq.,
- elementary interpolation and integration,
- matrix multiplication, reading and writing to files, and

- Using scilab to diagonalize matrices,
- plot elementary wavefunctions, and
- solve differential equations.

do 10 i =1, 100, 1

- 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
-                          executable statements
-                   10   continue
- There are other alternatives to the do statement such as
- ```
  sum = 0.0
  do i=1, 100,2
  sum = sum +(real (i))**3
  end do
  ```
- Computer treats real numbers and integers very differently
  ```

Observe the graph



We need to write a program which gives the value of f (x) as per the above formula. This is given below. The use of “relational operators”: .lt. , .eq. , ..

```
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
c  the above program illustrates the use of the if statement
```

Some useful linux commands

ls – l (list all the files in a directory)

- mkdir newdir
- cd newdir (change directory to newdir)
- cd .. (go back to the earlier directory)
- cp file1 file2 (copy file1 to file2); rm file3
- help (help on a command)
- man f77 (manual for a command)

Solution of a quadratic equation

- The general form of the quadratic equation is
- a x * x + bx + c

The roots of this equation are

- [- b + (b * b – 4.0 * a * c) **(1/2)] / (2*a)

- [- b - (b * b – 4.0 * a * c) **(1/2)] / (2*a)

program quadratic

- c 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
c     do not use impt1 as it will treat it as in integer variable !!!!
      ximpt1 = rtofww / twoa
      write (*, *) 'complex roots'
      write (*, *) 'root1 ', ' real part =  ', realpt, ' imaginary part = ' , ximpt1
       ximpt2 = -ximpt1
       write (*, *) 'root2', ' real part=  ', realpt, ' imaginary part=' , ximpt2
 100  continue
       end
```

Summary: The main ingredients of a program

- 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 matrices
- 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.

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 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
- dimension tempval (365, 24)

Using Arrays

```
        dimension temp(365,24)
        do 100 i = 1, 365
        do 90 i = 1, 24
         read (*, *) temp (i, j)
  90   continue
100   continue
```

- ___________________________________
- Meaning of temp(22,33)
- Element of 22nd row and 33rd column of a two dimensional array temp
- Ex: vect(3), coach(10111, 325, 4, 45)

Reading and writing to files

- DIMENSION tempval (365, 24)
- open (unit = 11, file = 'input.dat')
- open (unit = 12 ,file = 'output')
- C the following line contains implicit ‘do’ statements
- read (11, *) ((tempval (i, j), j = 1,24), i 1,= 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 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

$$c(i,j) = \sum a(i,k) * b(k,j), \quad k = 1, n$$

```
c          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
```

PROGRAMME TO ARRANGE NUMBERS IN AN ASCENDING ORDER

```
c       Let us see how to exchange the values of two numbers.
         a= 2.0
         b = 3.0
c        the simplest way to attempt this is by doing
         b = a
         a = b
c        but this does not achieve the desired result !!!  The correct way is
         temp = a
         a = b
         b = temp
---------------------------------------------------------------------------------------------------------
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
      do 200 j= 1,n
       write (*,*) result(j)
200  continue
      close(16)
      close(15)
      end
```

Summary of lectures 4 and 5

- More ingredients of programmes
- If statements to transfer control
- Some more linux commands
- Programme for a quadratic equation
- Dimensioned variables/arrays
- Reading and writing to files
- Programs for matrix multiplication and arranging numbers in an ascending order