Lecture Notes

On

Programming in C – Language

(Module-II)

Course Code : DSE-IV

(6th Semester)

By

Prof . Indubaran Mandal

Department of Mathematics

Bidhan Chandra College ,Asansol -4

Module –II .

Conditional statements and Loops : Decision making within a program , conditions , Relation Operators , Logical Connectives , IF statements , IF-ELSE statements ; Loops : While Loop , Do while , For Loop , Nested Loops , Switch Statement , Structured Programming .

Arrays : One Dimensional Arrays : Array Manipulation , Searching , Insertion , Deletion of an element from an Array ; Finding the largest / smallest element in an Array ; Two Dimensional Arrays : Addition and Multiplication of two matrices , Transpose of a square matrix , representation of square matrices .

Lecture 1 : Conditional statements , IF statement , IF-ELSE statement , Nested IF-ELSE statements .

Lecture 2 : ELSE-IF ladder , goto statement , switch statement , examples .

Lecture 3 : Loop Structure , while loop , do-while loop , for loop , break statement .

Lecture 4 : Continue statement , Array , One dimensional array , array manipulation .

Lecture 5 : Two-dimensional array , String , Addition and Multiplication of two matrices .

Lecture 6 : Transpose of a matrix , Searching , Inserting and deletion of an element from an array .

Lecture-1

Control Statements or Decision making statements : .

When we run a program, the statements are executed in the order in which they appear in the program. Also each statement is executed only once. But in many cases we may need a statement or a set of statements to be executed a fixed no of times or until a condition is satisfied. Also we may want to skip some statements based on testing a condition. For all these we use control statements.

In this section we are going to study the following Control statements :

(i) if statement

(ii) if-else statement

(iii) goto statement

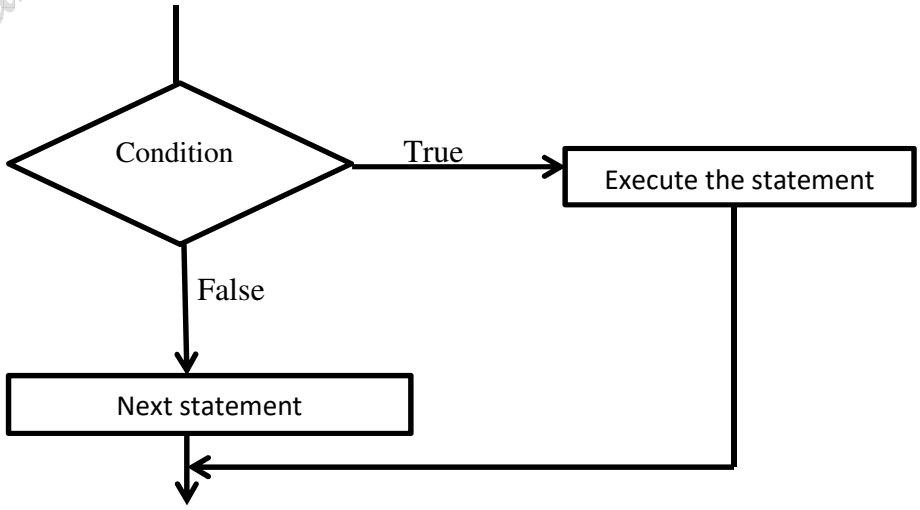
(iv) switch statement

if statement :

The if statement is a powerful decision making statement and used to control the flow of execution of statements . Its syntax is :

```
if (condition)
    statement ;
```

The statement is executed only when condition is true. If the if statement body is consists of several statement then better to use pair of curly braces. Here in case condition is false then compiler skip the line within the if block.



Example : if (n > 10)

y = n – 10 ;

In this case the condition is n > 10 . If n is greater than 10 then y is calculated as y = n -10 , otherwise the control goes to next statement .

The if statement can be used in different forms depending on the complexity of the condition to be tested . The different forms are –

1 . Simple if statement

2. if-else statement

3. Nested if-else statement

4. else if ladder .

Simple if statement :

The general form or syntax of simple if statement is :

```
if ( condition)
    {
        statement –block ;
    }
statement-x ;
```

The 'statement –block' may be a single statement or a group of statements . If the condition is true statement-block will be executed otherwise the statement-block will be skipped and the control jump to the statement-x .

Example :

```
#include<stdio.h>
 main ( )
        {
          int n ;
          printf( "Enter a number : \n");
          scanf ( "%d" , &n);
          if ( n  >10)
            printf ( " Number is greater. \n");
        }
```

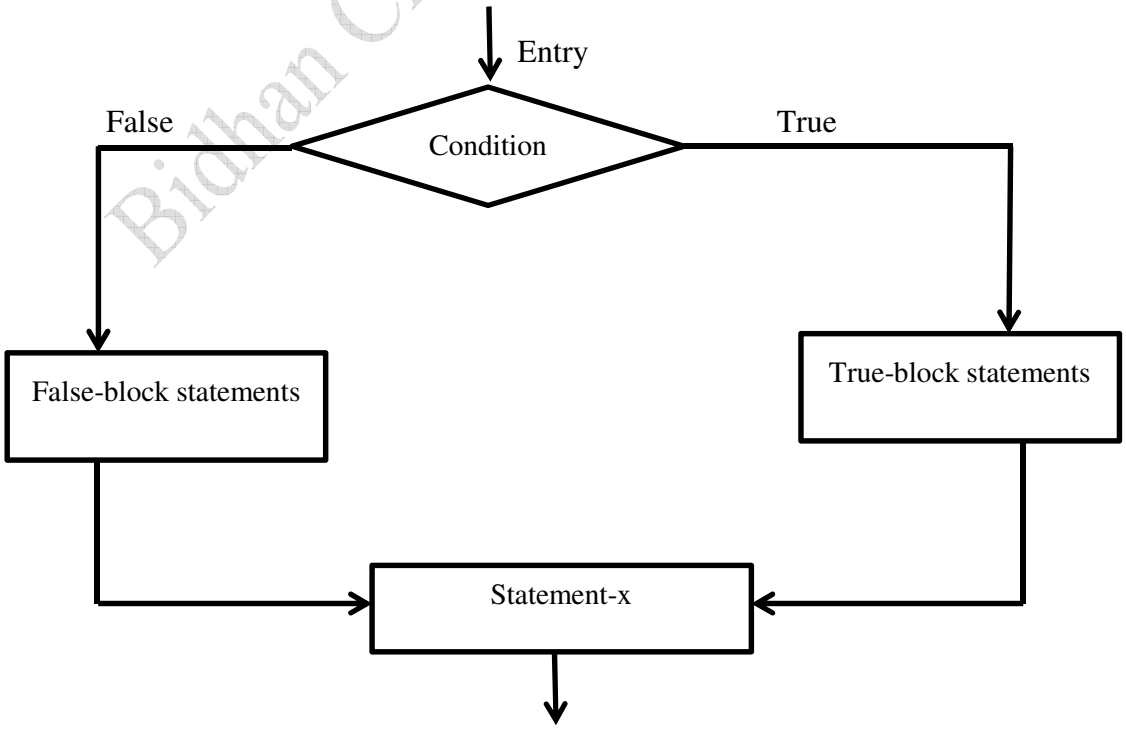
Output : Enter a number : 12

Number is greater .

if-else statement : The if-else structure is more useful and easy to handle than the if structure . The general form of the structure is

```
if (condition)
   {
       true-block statements;
    }
else
    {
        false-block statements ;
     }
```

If the condition is true then "true-block statements" are executed otherwise the "false-block statement" will be executed . In either case , either true-block or false-block will be executed , not both . Else statement cannot be used without if or no multiple else statement are allowed within one if statement. It means there must be a if statement with in an else statement.



Example : To check a given number is even or odd .

```
#include<stdio.h>
main( )
{
    int n ;
    printf(" Enter a number : \n");
    scanf("%d" , &n);

    if ( n % 2= 0)
        {
              printf (" The number is even \n");
         }
     else
          {
               printf( "The number is odd \n");
           }
```

Output : Enter a number : 123

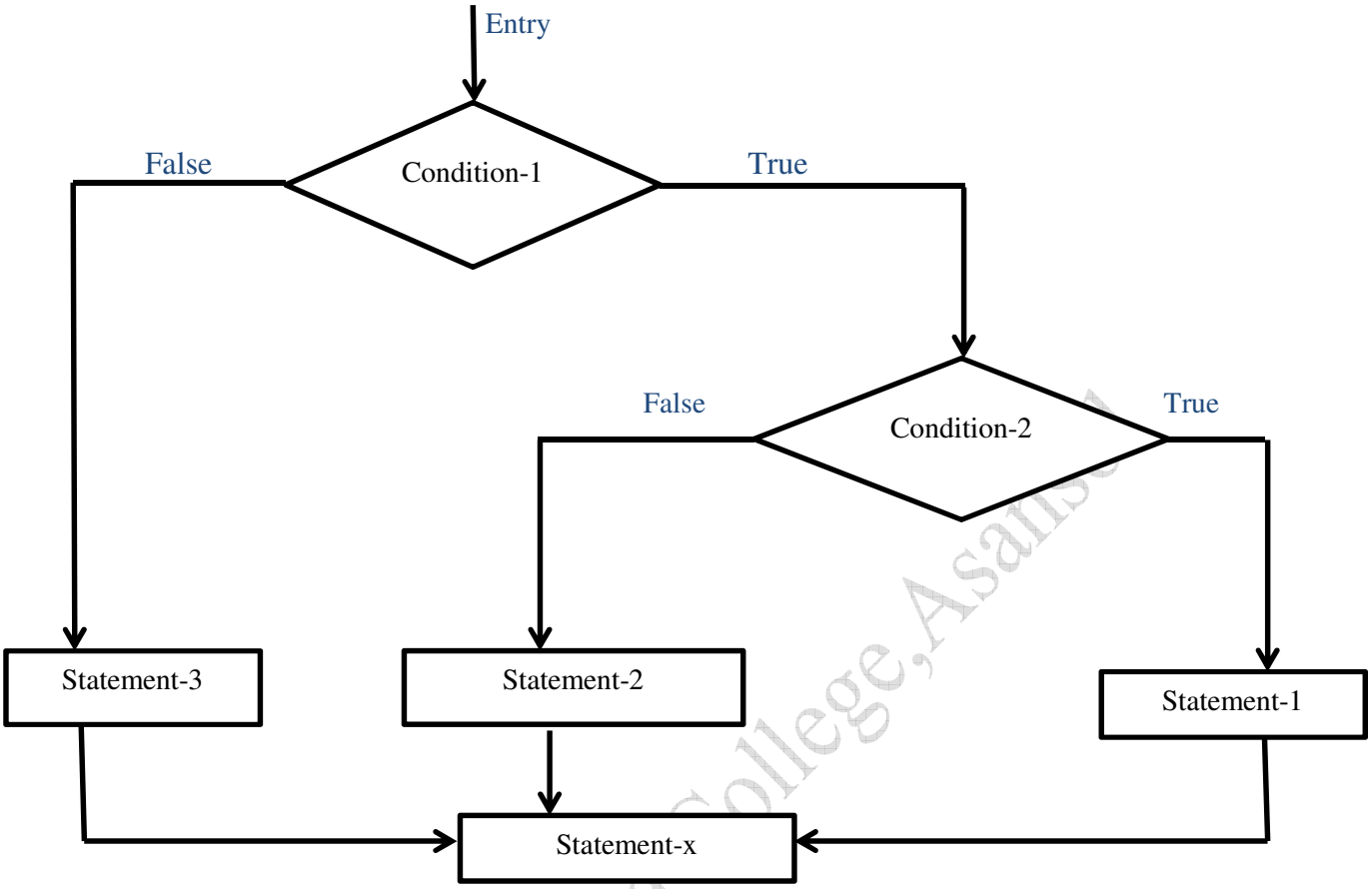
The number is odd .

Nested if-else statement :

When there are another if-else statement in if-block or else-block , then it is called nested if-else statement . The general form is –

```
if ( condition-1)
      {
          if (condition-2)
               {
                   statement-1 ;
               }
          else
               {
                   statement-2 ;
               }
       }
else
      {
          statement-3;
       }

  statement-x ;
```



Example : Grade of obtained by a student .

```
#include<stdio.h>
main( )
 {
        int m ;

        printf (“Enter the mark of a student : \n”);

        scanf( “%d” , &m);

        if ( m >40)
            {
              if ( m > 60)
                {
                   printf(“ Grade obtained : First class \n”);
                 }
               else
                 {
                   printf(“ Grade obtained : Second class \n”);
```

```
                }
            }
        else
            {
                printf( " Grade : Fail \n");
            }
}
```

If –else ladder :

In this type of nesting there is an if else statement in every else part except the last part. If condition is false control pass to block where condition is again checked with its if statement. The syntax is of the form :

```
if (condition-1)
    {
        statement-1 ;
    }

else if (condition 2)
    {
        statement-2 ;
    }

else if ( condition-3)
    {
        statement-3;
    }
else
    {
        statement-4 ;
    }
statement-x ;
```

This process continue until there is no if statement in the last block. if one of the condition is satisfy the condition other nested "else if" would not executed. But it has disadvantage over if else statement that, in if else statement whenever the condition is true, other condition are not checked. While in this case, all condition are checked.

Example : *Grade of a student*\

```
#include<stdio.h>

main( )

{
```

```
    int mark ;
    printf(" Enter the mark of the student : \n");
    scanf(" %d" , & mark);
    if (mark > = 90)
                printf ( " Grade : excellent \n");
      else if ( mark > = 80)
                printf( " Grade : Very good \n");
        else if ( mark > = 70)
                    printf( "Grade : Good \n");
            else if ( mark > = 60)
                    printf ( "Grade : Average \n");
                else
                    printf ( " Grade : Need to improve \n");
}
```

goto statement :

This statement is used to transfer the control to any other statement unconditionally . The general form is

goto label ;

where label is a valid label name .

Example : 1) goto end ;

2) goto begin ;

There should be a space between the 'goto' and the label . The following is a valid goto statement :

```
goto end ;
statements;
end : statement-x ;
```

Example : /* program for SIMPLE INTEREST */

```
#include <stdio.h>
 main( )
 {
        int n ;
        float p , r , interest ;
        begin : printf ( " Enter p , n , r \n");
        scanf (" %f %d %f " , &p , &n , &r );
        interest = (p * n * r)/100.0 ;
        printf ( "Interest =  %f  \n" , interest );
        goto begin ;
}
```

Switch statement :

The switch statement is a powerful too in C language programming . It is used to execute a particular group of statements to be chosen from several available options. The selection is based on the current value of an expression with the switch statement. The Syntax is :

```
switch ( expression)
    {
    case value 1:
            statement – block 1 ;
             break;
     case value 2:
            statement-block 2;
            break ;
    ……..
    ……...
      case value n:
            statement-block n;
            break;
       default :
            statement- x;
    }
```

All the option are embedded in the two braces { }.Within the block each group is written after the label case followed by the value of the expression and a colon. Each group ends with 'break' statement. The last may be labeled 'default'. This is to avoid error and to execute the group of statements in default if the value of the expression does not match value1, value2, …….. .

In the above format the , expression is an arithmetic expression which gives an integer value . It can also return a character value since a character also represents an integer .

value 1 , value 2 ,…. are integer value or character value depending upon the case . If it is character , it must be written within the quotes . The break statement causes the exit of the control outside the switch structure .

The value of the expression is first evaluated and it is verified whether the value is equal to any of value 1 , value 2 ,…. . If the value is equals value i ($1 \leq i \leq n$) then the statements under **case value i** are executed . If value is not equal to any value i , then the statements under default , are executed .

Example : /* Program to display Monday to Sunday */

```
#include<stdio.h>
main( )
{
   char ch;

   printf("enter m or M for Monday\n t or T for Tuesday\n w or W for Wednesday\n h
            or H  for Thursday\n f or F for Friday\n s or S for Saturday\n u or U for
             Sunday : \n");

   scanf("%c",&ch);

   switch(ch)
          {
                case 'm':
                 case 'M':
                    printf("Monday \n");
                    break;

                case 't':
                case 'T':
                   printf("Tuesday \n");
                   break;
```

```
        case 'w':
        case 'W':
            printf("Wednesday \n");
            break;

        case 'h':
        case 'H':
            printf("Thursday \n");
            break;

        case 'f ':
        case 'F':
            printf("Friday \n");
            break;

        case 's':
        case 'S':
            printf("Saturday \n");
                break;

        case 'u':
        case 'U':
            printf("Sunday \n");
            break;

        default :
            printf("wrong input \n");
            break;
    }

}
```

Output :

```
enter m or M for Monday
t or T for Tuesday

w or W for Wednesday

h or H for Thursday

f  or F for Friday

s or S  for Saturday

u or U for Sunday:

f

 Friday
```

Lecture-3

LOOP STRUCTURE :

Loop is a block of statement that performs set of instructions. It is to execute a group of instructions repeatedly, a fixed no of times or until a specified condition is satisfied.

A loop therefore consist of two segment , one known as body of the loop and the other known as control statement . The control statement tests certain conditions and then directs the repeated execution of the statements contained in the body .

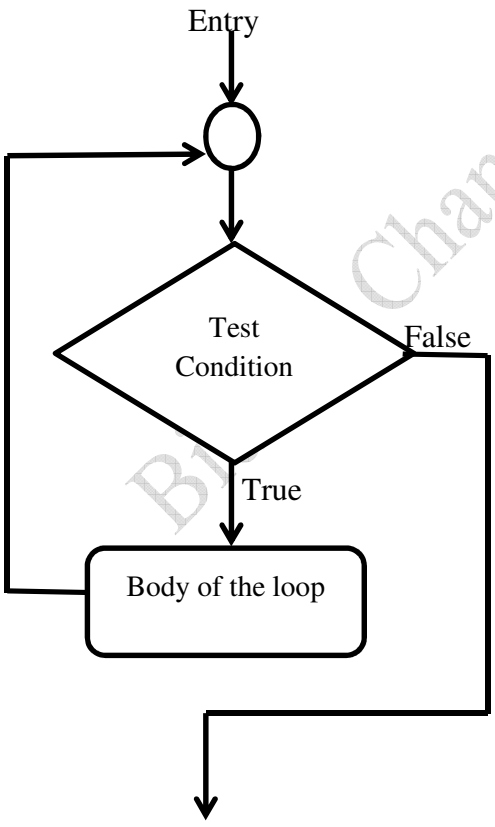
Depending on the position of the control statement in the loop , a loop may be classified as entry controlled loop or exit-controlled loop . See the figure below .

There are three type of loops –

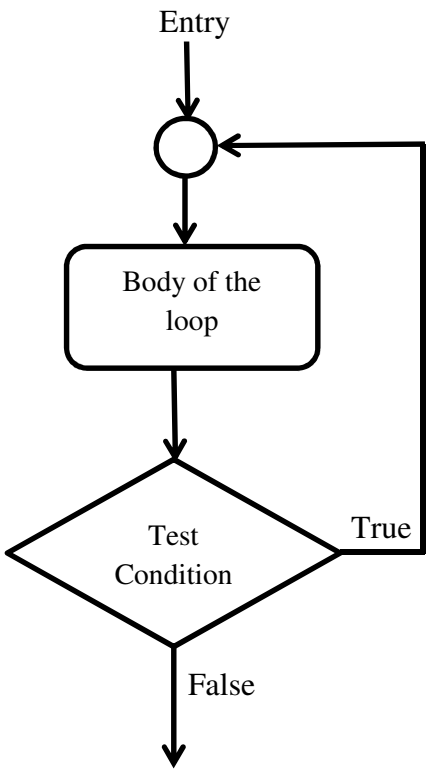
1) while loop

2) do while loop

3) for loop



Entry-Controlled Loop



Exit-Controlled Loop

While Loop :

This is to carry out a set of statements to be executed repeatedly until some condition is satisfied. It is a entry-controlled loop .The syntax is :

```
while ( test condition )
    {
        statement-block ;
     }
```

The test condition may be any expression .when we want to do something a fixed no of times but not known about the number of iteration, in a program then while loop is used.

Here first condition is checked if, it is true body of the loop is executed else, If condition is false control will be come out of loop.

Example : /* print numbers from 1 to 10 */

```
#include<stdio.h>
main ( )
{
    int i = 1 ;
    while ( i < = 10)
        {
            printf ( " %d", i);
            ++ i ;
         }
  }
```

Output **:** 1 2 3 4 5 6 7 8 9 10

do while loop :

This is also to carry out a set of statements to be executed repeatedly so long as a condition is true. It is exit-controlled loop . The syntax is :

```
do
   {
       statement-block ;
    }
 while (condition)
```

THE DIFFERENCE BETWEEN while loop and do – while loop

1) In the while loop the condition is tested in the beginning whereas in the other case it is done at the end.

2) In while loop the statements in the loop are executed only if the condition is true. Whereas in do – while loop even if the condition is not true the statements are executed atleast once.

Example :

```
#include<stdio.h>
 main ( )
  {
        int x = 4 ;
        do
              {
                 printf (" %d " , x);
                 x = x + 1 ;

               }while (x < =10)

          prinf( " .") ;
     }
```

Output : 4 5 6 7 8 9 10 .

For Loop :

It is the most commonly used loop in c programming . In a program, for loop is generally used when number of iteration are known in advance. The syntax is :

```
for ( initialization  ; test condition ; increment / decrement )
      {

          statement-block ;

       }
```

or

```
for ( expression 1; expression 2 ; expression 3)
      {

          statement-block ;
      }
```

Here expression1 is to initialize some parameter that controls the looping action , expression2 is a condition and it must be true to carry out the action , expression3 is a unary expression or an assignment expression or update expression.

Example : /* Print the number from 1 to 10 */

```
#include < stdio.h >
 main ( )
 {
     int i ;
     for ( i=1 ; i <= 10 ; i++)
          {
               printf ( "%d" , i);
           }
  }
```

Output : 1 2 3 4 5 6 7 8 9 10

Nesting of loops :

When a loop written inside the body of another loop then, it is known as nesting of loop. Any type of loop can be nested in any type such as while, do while, for. For example nesting of for loop can be represented as :

```
for ( initialization ; test condition ; increment / decrement)
     {
          statement-1 ;
          statement-2 ;
          for (initialization ; test condition ; increment/decrement)
            {
                 statement-3 ;
                 statement -4 ;
            }
     }
```

Break statement :

Sometimes it becomes necessary to come out of the loop even before loop condition becomes false then break statement is used. Break statement is used inside loop and switch statements. It cause immediate exit from that loop in which it appears and it is generally written with condition . The syntax is :

break ;

When break statement is encountered loop is terminated and control is transferred to the statement, immediately after loop or situation where we want to jump out of the loop instantly without waiting to get back to conditional state.

When break is encountered inside any loop, control automatically passes to the first statement after the loop. This break statement is usually associated with if statement.

Example : /* Program to find a given number is even or odd */

```
#include<stdio.h>
main ( )
{
   int i , n , r =1 ;
   printf (" Enter a number : \n");
   scanf ( "%d" , &n)
   for ( i = 2 ; i< n ; i++)
      {
            if ( n % i = = 0)
               {
                  r = 0;
                  break ;
               }
         }
       if ( r = = 0 )
            printf (" The number is not prime \n");
        else
            printf (" The number is prime \n");
}
```


Continue statement :

Continue statement is used for continuing next iteration of loop after skipping some statement of loop. When it encountered control automatically passes through the beginning of the loop. It is usually associated with the if statement. It is useful when we want to continue the program without executing any part of the program.

The difference between break and continue is, when the break encountered loop is terminated and it transfer to the next statement and when continue is encounter control come back to the beginning position.

In while and do while loop after continue statement control transfer to the test condition and then loop continue where as in, for loop after continue control transferred to the updating expression and condition is tested.

Example : /* print the numbers from 1 to 10 except 4 */

```
#include<stdio.h>
main ( )
{
       int n ;
       for ( n=1 ; n < =10 ; n++)
```

```
        {
          if ( n = = 4)
                continue ;
           else
              printf( " %d" , n);
        }
}
```

Output : 1 2 3 5 6 7 8 9 10

ARRAY: .

Array is the collection of similar data types or collection of similar entity stored in contiguous memory location. Array of character is a string. Each data item of an array is called an element. And each element is unique and located in separated memory location. Each of elements of an array share a variable but each element having different index number known as subscript.

An array can be a single dimensional or multi-dimensional and number of subscripts determines its dimension. And number of subscript is always starts with zero. One dimensional array is known as vector and two dimensional arrays are Known as matrix.

ADVANTAGES: array variable can store more than one value at a time where other variable can store one value at a time.

One dimensional Array :

Arrays are defined like the variables with an exception that each array name must be accompanied by the size (i.e. the max number of data it can store).For a one dimensional array the size is specified in a square bracket immediately after the name of the array.

Declaration of an Array : The syntax is :

data-type array name [size];

Example : int a[10] , int mark[20] etc .

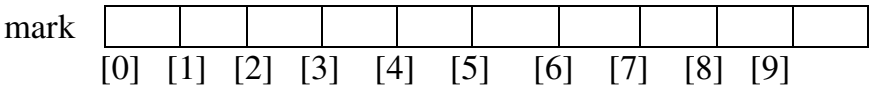
So far, we've been declaring simple variables: the declaration

int i;

declares a single variable, named i, of type int. It is also possible to declare an array of several elements. The declaration :

int mark[10];

declares an array, named mark, consisting of ten elements, each of type int. Simply speaking, an array is a variable that can hold more than one value. You specify which of the several values you're referring to at any given time by using a numeric subscript. (Arrays in programming are similar to vectors or matrices in mathematics.) We can represent the array mark above with a picture like this –



Example : int x[100] ;
float mark [50] ;
char name[30] ;

Note : With the declaration int x[100],computer creates 100 memory cells with name x[0],x[1],x[2],……,x[99].Here the same identifier x is used but various data are distinguished by the subscripts inside the square bracket.

Initialization of an Array :

After declaration element of local array has garbage value . An array be initialize as :

data-type array name [size] = { value 1 , value 2 , …… }

Example : int a[10] = { 1 , 5 , 3 , 4 , 6 , 8 , 7 , 9 , 10 , 2 }

The list of values, enclosed in braces {}, separated by commas, provides the initial values for successive elements of the array.

Note : while initializing a single dimensional array, it is optional to specify the size of array. If the size is omitted during initialization then the compiler assumes the size of array equal to the number of initializers.
For example:-

```
int marks[ ]={99,78,50,45,67,89};
```

If during the initialization of the number the initializers is less then size of array, then all the remaining elements of array are assigned value zero .
For example:-

```
int marks[5]={99,78};
```

Here the size of the array is 5 while there are only two initializers so After this initialization, the value of the rest elements are automatically occupied by zeros such as

Marks[0]=99 , Marks[1]=78 , Marks[2]=0, Marks[3]=0, Marks[4]=0

Again if we initialize an array like

```
int array[100]={0};
```

Then the all the element of the array will be initialized to zero. If the number of initializers is more than the size given in brackets then the compiler will show an error.

For example:-

```
int a[5]={1,2,3,4,5,6,7,8};  //error
```

we cannot copy all the elements of an array to another array by simply assigning it to the other array like, by initializing or declaring as

```
int a[5] ={1,2,3,4,5};
int b[5];
b=a;//not valid
```

Processing one dimensional array :

1) Reading arrays: For this normally we use for loop.
If we want to read n values to an array name called 'mark' , the statements look like

```
int mark[200],i,n;
for(i=1;i<=n;++i)
scanf("%d",&x[i]);
```

Note: Here the size of array declared should be more than the number of values that are intended to store.

2) Storing array in another:
To store an array to another array. Suppose a and b are two arrays and we want to store that values of array a to array b. The statements look like

```
float a[100],b[100];
int i ;
for(i=1;i<=100;++i)
b[i]=a[i];
```

Example : /*Write a program to input values into an array and display them*/

```
#include<stdio.h>
main( )
{
   int a[5], i;
   for(i=0;i<5;i++)
       {
```

```
            printf("enter a value for a[%d] \n",i);

            scanf("%d",&a[i]);
          }

    printf("the array elements are: ");

     for (i=0;i<5;i++)
         {
              printf("%d\t",arr[i]);
         }

  }
```

OUTPUT:

```
Enter a value for a[0] = 12
Enter a value for a[1] =45
Enter a value for a[2] =59
Enter a value for a[3] =98
Enter a value for a[4] =21
The array elements are  : 12 45 59 98 21
```

Example : / * To find the average of a set of values. */

```
#include<stdio.h>
 main( )
  {
     int n ,i ;
     float x[100], sum = 0 , avg ;

     printf(" Enter the number of values \n");
     scanf("%d",&n);

     printf("Input the numbers \n");

     for(i = 0; i<= n-1 ; ++i)
          {
            scanf("%f", &x[i]);
            sum = sum + x[i];
           }

     avg = sum/n;
     printf("\n Average = %f ",avg);


  }
```

Lecture-5

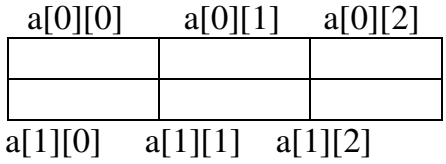
Two dimensional Array :

Two dimensional array is known as matrix . Two dimensional arrays are defined in the same manner as one dimensional arrays except that a separate pair of square brackets is required to each subscript.
The syntax is : data type array name[row][column]

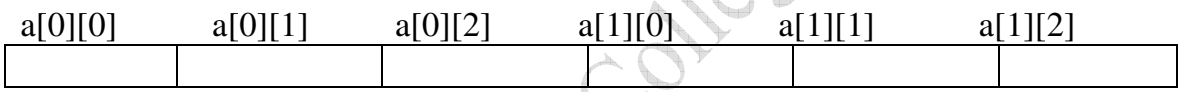
The total number of elements in 2-D array is calculated as **row * column** .

Example : int a[2][3] ;

Total number of elements is 2* 3 =6 i.e. it has 2 rows and 3 column . The elements are a[0][0] , a[0][1] , a[0][2] ; a[1][0] , a[1][1] , a[1][2] .



or



Initialization of 2-D array : 2-D array can be initialize in a similar way as 1-D . For example –

int a[2][3] = { 1, 5 , 4 , 3 , 6 , 7 };

These values are assigned to the elements row wise, so the values of elements after this initialization are –

a[0][0] = 1 , a[0][1] = 5 , a[0][2] = 4
a[1][0] = 3 , a[1][1] = 6 , a[1][2] = 7

While initializing we can group the elements row wise using inner braces. For example –

int a[2][3] = { {1,5,4 } ,{ 3, 6 ,7}};

And while initializing , it is necessary to mention the 2nd dimension where 1st dimension is optional. For example -

int mat[ ][3];
int mat[2][3];

are valid but

int mat[ ][ ];

int mat[2][ ];

are invalid .

If we initialize an array as :

int a[2][3] = {{1} , {3,6}}

Then the compiler will assume its all rest value as 0,which are not defined i.e.

a[0][0] = 1 , a[0][1] = 0 , a[0][2] = 0 , a[1][0] = 3 , a[1][1] = 6 , a[1][2] = 0 .

Strings :

Array of character is called string . It is always terminated by the NULL character. String is a one dimensional array of character.
We can initialize the string as

char name[ ]={'j','o','h','n','\o'};

Here each character occupies 1 byte of memory and last character is always NULL character. Where '\o' and 0 (zero) are not same. Array elements of character array are also stored in contiguous memory allocation .

From the above we can represent as :

j	o	h	n	\o
name[0]	name[1]	name[2]	name[3]	name[4]

The terminating NULL is important because it is only the way that the function that work with string can know, where string end.

String can also be initialized as:

char name[ ]="John";

Here the NULL character is not necessary and the compiler will assume it automatically.

String Constant :

A string constant is a set of character that enclosed within the double quotes . Whenever a string constant is written anywhere in a program it is stored somewhere in a memory as an array of characters terminated by a NULL character ('\o').

Example : "m" , "Tajmahal" , "C language" etc.

Consider : char a[ ] = "C language"

C		l	a	n	g	u	a	g	e	\o
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]

It is called the base address .

Note : Consider the example - char name[20] ;

Here name is an array that can store a string of size 20. If we want to store many strings(like many names or places) two dimensional array is used. Suppose we want to store names of 25 persons, then declare name as char name[25][ ]. Note that the second square bracket is kept empty if the length of string is not specified.

If the declaration is char name[25][30], 25 names of maximum size 30 can be stored. The various names are identified by name[0], name[1], name[2],……., name[24].

Some workout example : .

1) Program to find maximum number in an array .

```
#include<stdio.h>
#include<conio.h>
 main( )
 {
      int a[8], max, i;
      printf("Enter elements for the array: \n");
      for(i=0;i<8;i++)
          {
              scanf("%d",&a[i]);
           }
       max=a[0];
       for(i=1;i<8;i++)
          {
              if(max<a[i])
                  {
                      max=a[i];

                  }
             printf("maximum number = %d \n",max);
             getch( );
     }
```

Output : Enter elements of the array:
5
6
8
14
26

7
13
9
maximum number = 26

2) Program to display a matrix .

```
#include<stdio.h>
#include<conio.h>
 main()
   {
        int a[3][4], i, j;
        printf("Enter value for the matrix A: \n");
        for(i=0;i<3;i++)
            {
               for(j=0;j<4;j++)
                  {
                    scanf("%d",&a[i][j]);
                  }
             }
        printf("The matrix A is : \n\n");
        for(i=0;i<3;i++)
             {
                for(j=0;j<4;j++)
                   {
                     printf(" %d \t ",a[i][j]);
                   }
                 printf( "\n");
              }

         getch ( );
    }
```

Output : Enter value of the matrix :

8
9
4
5
6
7
2
1
5
12
7
13

The matrix A is :

8	9	4	5
6	7	2	1
5	12	7	13

3) Program to find sum of two matrices.

```
#include<stdio.h>
#include<conio.h>
main( )
{
    int a[3][2],b[3][2],c[3][2],i,j;
    printf("Enter value for the matrix A: ");
    for(i=0;i<3;i++)
        {
           for(j=0;j<2;j++)
               {
                  scanf("%d",&a[i][j]);
               }
         }
     printf("Enter value for the matrix B: ");
     for(i=0;i<3;i++)
         {
            for(j=0;j<2;j++)
                {
                   scanf("%d",&b[i][j]);
                }
          }
      for(i=0;i<3;i++)
          {
             for(j=0;j<2;j++)
                {
                   c[i][j]=a[i][j]+b[i][j];
                }
           }
      printf("Sum of matrix is\n");
      for(i=0;i<3;i++)
           {
              for(j=0;j<2;j++)
                  {
                     printf("%d \t",c[i][j]);
                  }
              printf("\n");
           }
       getch( );
 }
```

Output:
Enter value for 1 matrix: 1
2
3
4
5
6
Enter value for 2 matrix: 4
5
6
1
3
2
Sum of matrix is
5 7
9 5
8 8

4) Program to find multiplication of two matrices.

```
#include<stdio.h>
#include<conio.h>
 main()
  {
        int a[3][2],b[3][2],c[3][2],i,j;
        printf("enter value for the matrix A: ");
        for(i=0;i<3;i++)
             {
                for(j=0;j<2;j++)
                    {
                        scanf("%d",&a[i][j]);
                    }
             }
         printf("Enter value for the matrix B : ");
         for(i=0;i<3;i++)
              {
                  for(j=0;j<2;j++)
                       {
                           scanf("%d",&b[i][j]);
                        }
               }

         for(i=0;i<3;i++)
               {
```

```
                for(j=0;j<2;j++)
                    {
                        c[i][j]=a[i][j]*b[i][j];
                    }
                }
            printf("The required matrix is :\n");
            for(i=0;i<3;i++)
                {
                    for(j=0;j<2;j++)
                        {
                            printf(" %d \t ",c[i][j]);
                        }
                    printf("\n");
                }
            getch();
        }
```

Output:

```
enter value for the matrix A: 7
8
9
4
5
6
enter value for the matrix B: 3
2
1
2
5
6
The required matrix is
21   16
9    8
25   36
```

5) **Program to find transpose of a matrix.**

```
#include<stdio.h>
#include<conio.h>
 main()
   {
        int a[3][2],b[2][3],i,j;
        printf("Enter value for matrix: \n");
        for(i=0;i<3;i++)
          {
             for(j=0;j<2;j++)
```

```
                {
                        scanf('%d",&a[i][j]);
                 }
            }
        printf("Matrix:\n");
        for(i=0;i<3;i++)
          {
              for(j=0;j<2;j++)
                {
                    printf(" %d ",a[i][j]);
                 }
               printf("\n");
           }
         for(i=0;i<3;i++)
           {
              for(j=0;j<2;j++)
                 {
                    b[j][i]=a[i][j];
                  }
             }
          printf("Transpose matrix:\n");
          for(i=0;i<2;i++)
            {
               for(j=0;j<3;j++)
                  {
                      printf(" %d ",b[i][j]);
                   }
               printf("\n");
             }
          getch();
       }
```

Output:

Enter value for matrix: 4

5

6

1

2

3

Matrix:

4 5

6 1

2 3

Transpose matrix:

4 6 2

5 1 3

5) Program to search element in array .

```
#include <stdio.h>
#include<conio.h>
 main( )
   {
       int arr[ ];
       int size, i, x, found = 0;
       printf("Enter size of array: ");
       scanf("%d", &size);
       printf("\n Enter elements in array: ");
       for(i=0; i<size; i++)
         {
            scanf("%d", &arr[i]);
          }
       printf(" \n Enter an element to search: ");
       scanf("%d", &x);
       for(i=0; i<size; i++)
           {
              if(arr[i] == x)
                    {
                         found = 1;
                         break;
                     }
              }
        if(found == 1)
             {
                 printf(" \n %d is found at position %d \n", x, i + 1);
              }
         else
            {
               printf("\n %d is not found in the array", x);
             }

         getch( );
    }
```

Output : Enter size of array: 10
Enter elements in array: 10 12 20 25 13 10 9 40 60 5
Enter element to search: 25
25 is found at position 4

6) Program to insert element in array .

```
#include <stdio.h>
#include<conio.h>
 main( )
  {
     int arr[ ];
     int i, size, num, pos;
     printf("Enter size of the array : ");
     scanf("%d", &size);
     printf("Enter elements in array : ");
     for(i=0; i<size; i++)
          {
              scanf("%d", &arr[i]);
          }
      printf("Enter the element to insert : ");
      scanf("%d", &num);
      printf("Enter the element position : ");
      scanf("%d", &pos);
      if(pos > size+1 || pos <= 0)
          {
                printf("Invalid position! Please enter position between 1 to
                        %d", size);
           }
      else
         {
                for(i=size; i>=pos; i--)
                     {
                        arr[i] = arr[i-1];
                     }
                arr[pos-1] = num;
                size=size+1;
                printf("Array elements after insertion : ");
                for(i=0; i<size; i++)
                   {
                       printf("%d\t", arr[i]);
                    }
           }
    getch( );
}
```

Output : Enter size of the array : 5
Enter elements in array : 10 20 30 40 50
Enter element to insert : 25
Enter the element position : 3
Array elements after insertion : 10 20 25 30 40 50

7) Program to delete element from array .

```
#include <stdio.h>
#include<conio.h>
 main( )
  {
      int arr[ ];
      int i, size, pos;
      printf("Enter size of the array : ");
      scanf("%d", &size);
      printf("\nEnter elements in array : ");
      for(i=0; i<size; i++)
          {
              scanf("%d", &arr[i]);
           }
       printf("\nEnter the element position to delete : ");
       scanf("%d", &pos);
       if(pos < 0 || pos > size)
          {
              printf("Invalid position! Please enter position between 1 to
                      %d \n", size);
           }
       else
          {
              for(i=pos-1; i<size-1; i++)
                 {
                     arr[i] = arr[i + 1];
                  }
              size=size-1;
           }
      printf("\nElements of array after delete are : ");
      for(i=0; i<size; i++)
          {
             printf("%d\t", arr[i]);
           }

      getch();
   }
```

Output :

Enter size of the array : 5
Enter elements in array : 10 20 30 40 50
Enter the element position to delete : 2
Elements of array after delete are : 10 30 40 50