UNIT-4

Programming in C++

Statements:-Statements are the instructions given to the Computer to perform any kind of action.

Null Statement:-A null statement is useful in those case where syntax of the language requires the presence of a statement but logic of program does not give permission to do anything then we can use null statement. A null statement is nothing only a ;. A null (or empty statement have the following form:

```
;        // only a semicolon (;)
```

Compound Statement :-A compound statement is a group of statements enclosed in the braces { }. A Compound statement is useful in those case where syntax of the language requires the presence of only one statement but logic of program have to do more thing i.e., we want to give more than one statement in place of one statement then we can use compound statement.

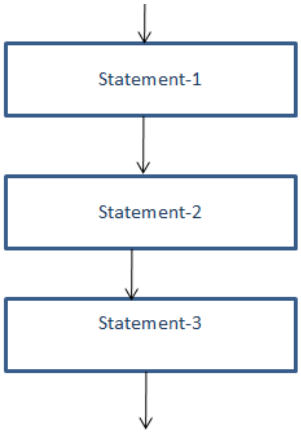
```
{
        St-1;
        St-2;
        :
        :
}
```

Statement Flow Control:-In a program , statements may be executed sequentially, selectively, or iteratively.

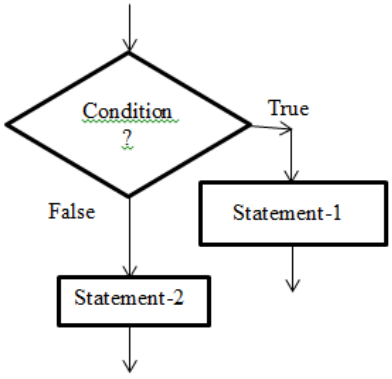
Every programming language provides three constructs:

1. Sequence Constructs
2. Selection Constructs
3. Iteration Constructs

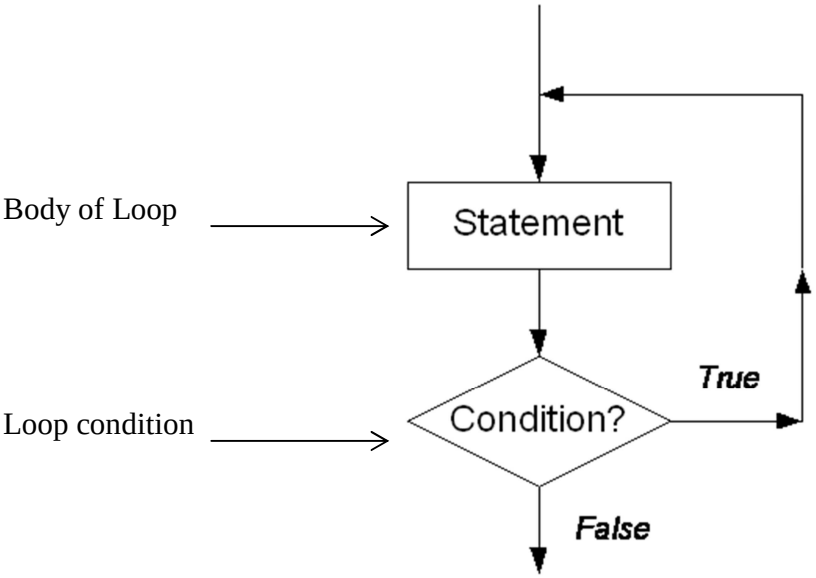
Sequence Construct:-The sequence construct means the statements are being executed sequentially. It represents the default flow of statements**.**



Selection Construct:- The selection construct means the execution of statement(s) depending on a condition. If a condition is true, a group of statements will be execute otherwise another group of statements will be execute.

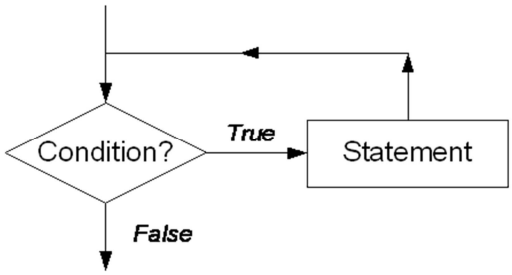


Looping or Iteration Statements:-Looping the iteration construct means repetition of set of statements depending upon a condition test. The iteration statements allow a set of instructions to be performed repeatedly until a certain condition is true.

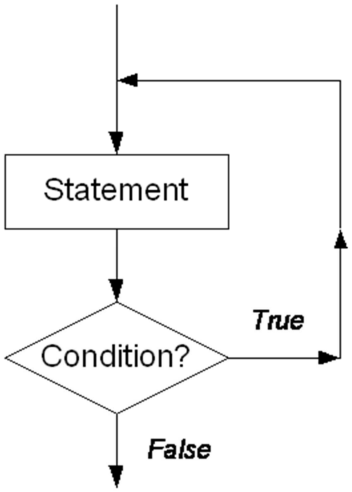


There are two types of loops:-

1. **Entry-controlled loop :-**In entry-controlled loop first of all loop condition is checked and then body of loop is executed is condition is true. If loop condition is false in the starting the body of loop is not executed even once.



2. **Exit-controlled loop :-**In exit-controlled loop first body of loop is executed once and then loop condition is checked. If condition is true then the body of loop will be executed again.
   It means in this type of loop, loop body will be executed once without checking loop condition.

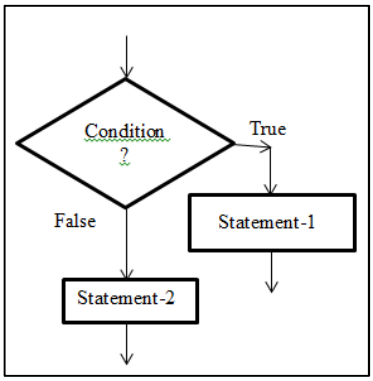


Selection Statements :-There are two types of selection statements in C++ :

1. if statement
2. switch statement

1. if Statement : If statement have three forms

(a) if … else Statement :- It is useable, when we have to performs an action if a condition is True and we have to perform a different action if the condition is false. The syntax of if…else statement is:




```
if ( < conditional expression > )
{
< statement-1 or block-1>;
    // statements to be executed when conditional expression is true.
            }
else
{
< statement-2 or block-2>;
    // statements to be executed when conditional expression is false.
}
```

If the <conditional expression> is evaluated to true then the < statement-1 or block-1> (statement under if ( ) block) will be executed otherwise the <statement-2 or block-2> (statements under else block) would be executed. if there exists only one program statement under if( ) block then we may omit curly braces { }.

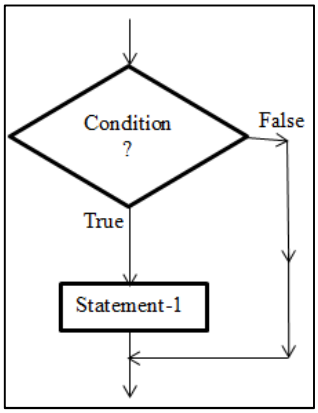
For example, a program to check whether a given number is greater than 100 or not is given below:

```
#include <iostream.h>
void  main()
{
int x;
cout<< "\nEnter a number: ";
cin>> x;
if( x > 100 )
cout<< "That number is greater than 100\n";
else
cout<< "That number is not greater than 100\n";
}
```

In this program if the conditional expression (x>100) in the if statement is true, the program prints one message; if it isn't, it prints the other.

Here's output from two different invocations of the program:

```
Enter a number: 300
That number is greater than 100
Enter a number: 3
That number is not greater than 100
```



(b) **Simple if statement:-** The else part in if … else statement is optional, if we omit the else part then it becomes simple if statement. This statement is usable, when we have to either perform an action if a condition is True or skips the action if the condition is false. The syntax of simple if statement is:

```
if ( < conditional expression > )
{
    < statement-1 or block-1>;
    // statements to be executed when conditional expression is true.
}
```

Here <statement-1 or block-1> will be executed only if <conditional expression > evaluates true. if there exists only one program statement under if( ) block then we may omit curly braces { }. eg.,

```
#include <iostream.h>
void  main()
{
int x;
cout<< "Enter a number: ";
cin>> x;
if( x > 100 )
cout<< "That number is greater than 100\n";

}
```

In the above example, program's output when the number entered by the user is greater than 100:

```
Enter a number: 2000
That number is greater than 100
```

If the number entered is not greater than 100, the program will terminate without printing the second line.

(c) **The if-else-if ladder :-**This statement allows you to test a number of mutually exclusive cases and only execute one set of statements for which condition evaluates true first.

The syntax is:

```
if ( <condition -1> )
        statement-1;        // do something if condition-1 is satisfied (True)
else if ( <condition – 2 >)
        statement-3 ;    // do something if condition -2 is satisfied (True)
else if (<condition – 3 >)
        statement-3 ;        // do something if condition- 3 is satisfied (True)
                                :
                            :    // many more n-1 else - if ladder may come
                            :
else if( < condition – n >)
        statement-n ;            // do something if condition – n is satisfied (True)
else
        statement-m ;            // at last do here something when none of the
                                 // above  conditions gets satisfied (True)
}
```

<>	in syntax is known as a place holder, it is not a part of syntax, do not type it while writing program. It only signifies that anything being kept there varies from program to program.
[ ]	is also not a part of syntax , it is used to mark optional part of syntax i.e. all part of syntax between [ ] is optional.

In the above syntax there are ladder of multiple conditions presented by each if( ) , all of these conditions are mutually exclusive. If one of them would evaluates true then the statement followed that condition will be executed and all the conditions below it would not be evaluated (checked).

Say suppose if condition-3 gets satisfy (i.e. evaluates true value for the condition), then statement-3 gets executed and all other conditions below it would be discarded.

If none of the n if ( ) conditions gets satisfied then the last else part always gets executed. It is not compulsory to add an else at the last of the ladder.

We can also write a group of statement enclosed in curly braces { } (as a compound statement) in place of any statement (statement-1. Statement-2,……., statement-n) if required in above syntax.

For example, a program which accept number of week's day (1-7) and print is equivalent name of week day (Monday for 1,……, Sunday for 7). using the if-else-if ladder is given below:

```
#include< iostream.h>
#include<conio.h>
void main()
{
        int day;
        cout<<" Enter a  number (between 1 and 7):"<< endl;
        cin>>day;
        if (day= =1)
                cout<<" Monday"<< endl;
        else if (day= =2)
                cout<<" Tuesday"<<endl;
        else if( day= =3)
                cout<<" Wednesday"<<endl;
        else if(day= =4)
                cout<<" Thursday"<<endl;
        else if(day= =5)
                cout<<" Friday"<< endl;
        else if(day= =6)
                cout<<" Saturday"<< endl;
        else if(day= =7)
                cout<<" Sunday"<< endl;

        else
                cout<<" You enter a wrong number"<< endl;
        getch();
}
```

Nested if Statement:- If an if statement is written in the if or else clause of another if statement then it is known as nested if. Some possible syntax of nested if statements given below:

Syntax 1:-

```
if ( <outer- condition > )
 {
        if ( <inner-condition> )
        {

                //some statements to be executed
                // on satisfaction of  inner  if ( ) condition.

        } // end of scope of inner if( )
        //some statements to be executed
        // on satisfaction of outer  if ( ) condition.

} // end of the scope of outer if( )
```

Syntax 2:-

```
if ( <outer- condition > )
 {
        if ( <inner-condition> )
        {

                //some statements to be executed
                // on satisfaction of  inner  if ( ) condition.

        }
        else
        {
                // statements on failure of inner if( )
        }

                //some statements to be executed
                // on satisfaction of outer  if ( ) condition.

        }
 else
 {

                // statements on failure of outer if( )
 }
```

2. switch Statement :-This is multi-branching statement. Syntax of this statement is as follows:

```
        switch (expression/variable)
        {       case value_1:  statement -1;
                               break;
                case value_2:  statement -2;
                               break;
                :
                :
                case value_n:  statement -n;
                               break;
                [  default:  statement -m    ]
        }
```

Note: expression/variable should be integer or character type only.

When the switch statement is executed, the expression/variable is evaluated and control is transferred directly to the statement whose case label value matches the value of expression/ variable. If none of the case label value matches the value of expression/variable then only the statement following the default will be executed. If no default statement is there and no match is found then no action take place. In this case control is transferred to the statement that follows the switch statement.

For example, a program which accept number of week's day (1-7) and print is equivalent name of week day (Monday for 1,……, Sunday for 7). using switch-case statement is given below:

```
#include <iostream.h>
#include<conio.h>
void main()
{
```

```
        int day;
        cout<<" Enter a number (between  1 and 7):"<< endl;
        cin>>day;
        switch(day)
        {
        case  1:
                cout<< " Monday"<< endl;
                break;
        case 2:
                cout<<" Tuesday"<< endl;
                break;
        case 3:
                cout<<" Wednesday"<< endl;
                break;
        case 4:
                cout<<" Thursday"<< endl;
                break;
        case 5:
                cout<<" Friday" << endl;
                break;
        case 6:
                cout<<" Saturday" << endl;
                break;
        case 7:
                cout<<" Sunday"<< endl;
                break;
        default:
                Cout<<" You enter a wrong number "<< endl;
        }
                getch();
}
```

Loops in C++:-There are three loops or iteration statements are available in C++

1. for loop
2. while loop
3. do…. while loop

1. **The for Loop**:For loop is an entry control loop the syntax of for loop is :

```
for(initialization_expression(s); loop_Condition; update_expression)
        {
                Body of loop
        }
```

Working of the for Loop:-

1. The *initialization_expression* is executed once, before anything else in the for loop.
2. The *loop condition* is executed before the body of the loop.
3. If loop condition is true then body of loop will be executed.
4. The *update expression* is executed after the body of the loop
5. After the *update expression* is executed, we go back and test the *loop condition* again, if loop_condition is true then body of loop will be executed again, and it will be continue until loop_condition becomes false.

Example:

```
for (int i = 0; i < 7; i++)
cout<< i * i << endl;
```

Interpretation:

An int i is declared for the duration of the loop and its value initialized to 0. i^2 is output in the body of the loop and then i is incremented. This continues until i is 7.

Example: A C++ Program to Print a table of factorials (from 0 to 9).

```
#include <iostream.h>
void main( )
{
        int factorial =1;
        for(int i=0; i<10; i++)
        {
                if (i!=0)
                factorial*=i;
                cout<<i<<"\t"<<factorial;
        }
}
```

2. **while Loop:-** while loop is also an entry controlled loop. The syntax of while loop is :

```
while (loop_condition)
        {
                Loop_body
        }
```

Where the Loop_body may contain a single statement, a compound statement or an empty statement.

The loop iterates (Repeatedly execute) while the loop_condition evaluates to true. When the loop_condition becomes false, the program control passes to the statement after the loop_body. In while loop , a loop control variable should be initialized before the loops begins. The loop variable should be updated inside the loop_body.

For example the program:

```
const int MAX_COUNT = 10;
count = 0;
while (count < MAX_COUNT)
        {
                cout<< count << " ";
                count ++;
                }
```

Will give the output:

0 1 2 3 4 5 6 7 8 9

3. **do-while loop:-** do-while loop is an exit-controlled loop i.e. it evaluates its loop_condition at the bottom of the loop after executing its loop_body statements. It means that a do-while loop always executes at least once. The syntax of do-while loop is:

```
do
        {
                Loop_body
        }while (loop_condition);
```

In do-while loop first of all loop_body will be executed and then loop_condition will be evaluates if loop_condition is true then loop_body will be executed again, When the

loop_condition becomes false, the program control passes to the statement after the loop_body.

For example the program:

```
const int MAX_COUNT = 10;
count = 0;
do
{
cout<< count << " ";
count ++;
} while (count < MAX_COUNT) ;
```

Will give the output:

0 1 2 3 4 5 6 7 8 9

Nested Loops :-Any looping construct can also be nested within any other looping construct . Let us look at the following example showing the nesting of a for( ) loop within the scope of another for( ) loop :

➢

```
for(int i = 1 ; i<=2 ; i++)  ← Outer for( ) loop
{
    for( int j = 1 ; j<=3 ; j++)  ← Inner for( ) loop
        {
            cout<< i * j <<endl ;
        }
    }
```

For each iteration of the outer for loop the inner for loop will iterate fully up to the last value of inner loop iterator. The situation can be understood more clearly as :

1st Outer Iteration
i= 1
1st Inner Iteration
j = 1 , output : 1 * 1 = 1
2nd Inner Iteration
j = 2 , output : 1 * 2 = 2
3rd Inner Iteration
j = 3 , output : 1 * 3 = 3

2nd Outer Iteration
i= 2
1st Inner Iteration
j = 1 , output : 2 * 1 = 1
2nd Inner Iteration
j = 2 , output : 2 * 2 = 4
3rd Inner Iteration
j = 3 , output : 2 * 3 = 6

You can observe that j is iterated from 1 to 2 every time i is iterated once.

Jump Statements:-These statements unconditionally transfer control within function . In C++ four statements perform an unconditional branch :

1. return
2. goto
3. break
4. continue

1. **return Statement:-** The return statement is used to return from a function. It is useful in two ways:

(i) An immediate exit from the function and the control passes back to the operating system which is main's caller.

(ii) It is used to return a value to the calling code.

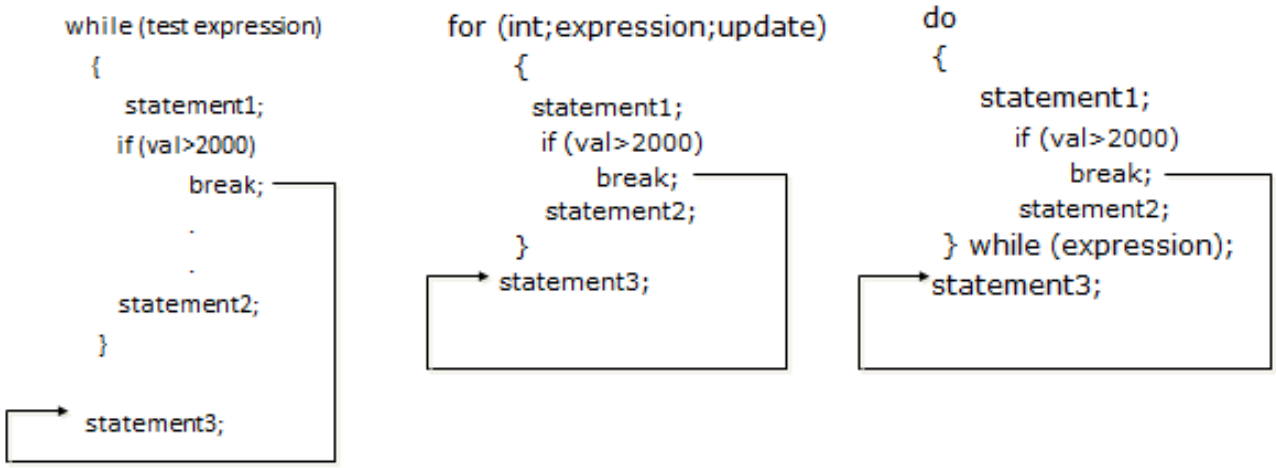
2. **goto statement :-** A goto Statement can transfer the program control anywhere in the program. The target destination of a goto statement is marked by a *label*. The target label and goto must appear in the same function. The syntax of goto statement is:

```
goto label;
        :
label :
```

Example :

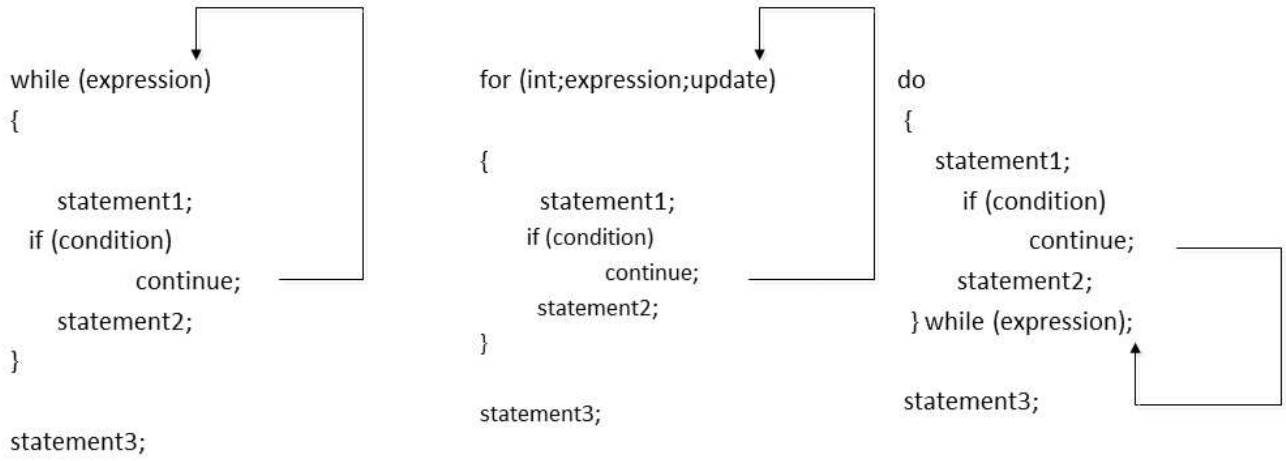
```
a= 0;
start :
        cout<<"\n" <<++a;
if(a<50) goto start;
```

3. **break Statement :-** The break statement enables a program to skip over part of the code. A break statement terminates the smallest enclosing while, do-while, for or switch statement. Execution resumes at the statement immediately following the body of the terminated statement. The following figure explains the working of break statement:



The Working of a Break Statement

4. **continue Statement**:- The continue is another jump statement like the break statement as both the statements skip over a part of the code. But the continue statement is somewhat different from break. Instead of forcing termination, it forces the next iteration of the loop to take place, skipping any code in between. The following figure explains the working of continue statement:



The Working of Continue Statement

Functions :- Function is a named group of programming statements which perform a specific task and return a value.

There are two types of functions:-

1. Built-in (Library)functions
2. User defined functions

Built-in Functions (Library Functions) :- The functions, which are already defined in C++ Library (in any header files) and a user can directly use these function without giving their definition is known as built-in or library functions. e.g., sqrt( ), toupper( ), isdigit( ) etc.

Following are some important Header files and useful functions within them :

stdio.h (standard I/O function)	gets( ) , puts( )
ctype.h (character type function)	isalnum( ) , isalpha( ), isdigit ( ), islower (), isupper ( ), tolower ( ), toupper( )
string.h (string related function)	strcpy ( ), strcat ( ), strlen( ), strcmp( ) , strcmpi( ) , strrev( ),strupr( ) , strlwr( )
math.h (mathematical function)	fabs ( ), pow ( ), sqrt ( ), sin ( ), cos ( ), abs ( )
stdlib.h	randomize ( ), random ( )

The above list is just few of the header files and functions available under them , but actually there are many more. The calling of library function is just like User defined function , with just few differences as follows:

i) We don't have to declare and define library function.

ii) We must include the appropriate header files , which the function belongs to, in global area so as these functions could be linked with the program and called.

Library functions also may or may not return values. If it is returning some values then the value should be assigned to appropriate variable with valid datatype.

gets( ) and puts( ) : these functions are used to input and output strings on the console during program run-time.

gets( ) accept a string input from user to be stored in a character array.
puts() displays a string output to user stored in a character array.

Program: Program to use gets( ) and puts( )

```
#include<iostream.h>
#include<stdio.h>  // must include this line so that gets( ) , puts( ) could be linked and
                   //      called
void main( )
{
        char myname[25] ;   //declaring a character array of size 25

        cout<<"input your name : ";
        gets(myname)   ;   // just pass the array name into the parameter of the function.
        cout<< "You have inputted your name as : " ;
        puts(myname);
}
```

i**salnum( ) , isalpha( ), isdigit( ) :** checks whether the character which is passed as parameter to them are alphanumeric or alphabetic or a digit ('0' to '9') . If checking is true functions returns 1.

Program : Program to use isalnum( ) , isalpha( ), isdigit( )

```
#include<iostream.h>
#include<ctype.h>
void main( )
{
        char ch;
        cout<<"Input a character";
        cin>>ch;
        if( isdigit(ch) = = 1)
          cout<<"The inputed character is a digit";
        else if(isalnum(ch) = = 1)
          cout<<"The inputed character is an alphanumeric";
        else if(isalpha(ch) = = 1)
          cout<<"The inputed character is an alphabet.
}
```

islower ( ),isupper ( ), tolower ( ), toupper( ) : islower( ) checks whether a character is lower case , isupper( ) check whether a character is upper case . tolower( ) converts any character passed to it in its lower case and the toupper( ) convert into upper case.

Program: Program to use islower ( ),isupper ( ), tolower ( ), toupper( )

```
#include<iostream.h>
#include<ctype.h>
void main( )
{
        char ch;
        cout<<"Input a character";
        cin>>ch;
        if( isupper(ch) = = 1)    // checks if character is in upper case converts the character
                                  //  to lower case

        {
        tolower(ch);
          cout<<ch;
        }
        else if(islower(ch) = = 1)  // checks if character is in lower case converts the
                                    // character to uppercase

        {
          toupper(ch);
          cout<<ch;
        }
}
```

fabs ( ), pow ( ), sqrt ( ), sin ( ), cos ( ), abs ( ) :

Program 3.4

```
#include <iostream.h>
#include <math.h>
#define PI 3.14159265 // macro definition PI will always hold 3.14159265

void  main ()
{

cout<<"The absolute value of 3.1416 is : "<<fabs (3.1416) ;
                                                    // abs( ) also acts similarly but only on int data
cout<<"The absolute value of -10.6 is "<< fabs (-10.6) ;
cout<<"7.0 ^ 3 = " <<pow (7.0,3);
cout<<"4.73 ^ 12 = " <<pow (4.73,12);
cout<<"32.01 ^ 1.54 = "<<pow (32.01,1.54);

double param, result;
param = 1024.0;
result = sqrt (param);
cout<<"sqrt() = "<<result ;
result = sin (param*PI/180);   // in similar way cos( ) , tan() will be called.
cout<<"The sine of " <<param<<" degrees is : "<< result
}
```

randomize ( ), random ( ) :
The above functions belongs to header file stdlib.h . Let us observe the use of these functions :

randomize( ) : This function provides the seed value and an algorithm to help random( ) function in generating random numbers. The seed value may be taken from current system's time.

random(<int>) : This function accepts an integer parameter say x and then generates a random value between 0 to x-1
for example : random(7) will generate numbers between 0 to 6.
To generate random numbers between a lower and upper limit we can use following formula
random(U – L +1) + L
where U and L are the Upper limit and Lower limit values between which we want to find out random values.
For example : If we want to find random numbers between 10 to 100 then we have to write code as :
random(100 -10 +1) + 10 ; // generates random number between 10 to 100
User-defined function :- The functions which are defined by user for a specific purpose is known as user-defined function. For using a user-defined function it is required, first define it and then using.
Function Prototype :- Each user define function needs to be declared before its usage in the program. This declaration is called as function prototype or function declaration. Function prototype is a declaration statement in the program and is of the following form :
Return_type function_name(List of formal parameters) ;
Declaration of user-defined Function: In C++ , a function must be defined, the general form of a function definition is :

```
Return_type  function_name(List of formal parameters)
    {
       Body of the function
     }
```

Where Return_type is the data type of value return by the function. If the function does not return any value then void keyword is used as return_type.
List of formal parameters is a list of arguments to be passed to the function. Arguments have data type followed by identifier. Commas are used to separate different arguments in this list. A function may be without any parameters, in which case , the parameter list is empty.
statements is the function's body. It is a block of statements surrounded by braces { }.
Function_name is the identifier by which it will be possible to call the function.

e.g.,

```
int addition (int a, int b)
        {
                int r ;
                r=a+b ;
                return (r) ;
        }
```

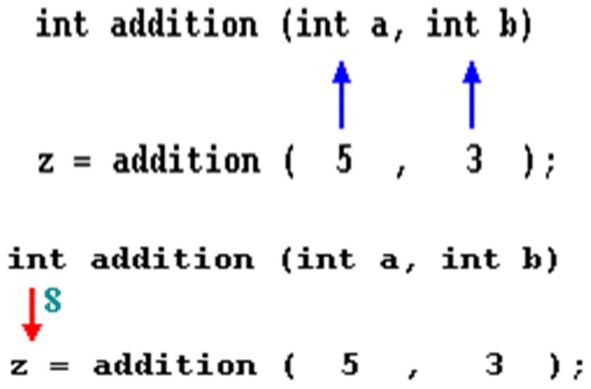
Calling a Function:- When a function is called then a list of actual parameters is supplied that should match with formal parameter list in number, type and order of arguments.
Syntax for calling a function is:

function_name (list of actual parameters);

e.g.,

```
#include <iostream>
 int addition (int a, int b)
{  int r;
   r=a+b;
return (r); }
void main ( )
{  int z ;
   z = addition (5,3);
cout<< "The result is " << z;
  }
```

The result is 8



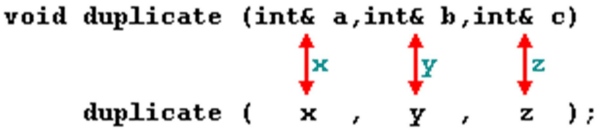
Call by Value (Passing by value) :- The call by value method of passing arguments to a function copies the value of actual parameters into the formal parameters , that is, the function creates its own copy of argument values and then use them, hence any chance made in the parameters in function will not reflect on actual parameters . The above given program is an example of call by value.

Call by Reference (Passing by Reference) :- The call by reference method uses a different mechanism. In place of passing value to the function being called , a reference to the original variable is passed . This means that in call by reference method, the called function does not create its own copy of original values , rather, its refers to the original values only by different names i.e., reference . thus the called function works the original data and any changes are reflected to the original values.

// passing parameters by reference

```
#include <iostream.h>
void duplicate (int& a, int& b, int& c)
{
        a*=2;
        b*=2;
        c*=2;
}

void main ()
        {
                int x=1, y=3, z=7;
                duplicate (x, y, z);
                cout <<"x="<< x <<", y="<< y <<", z="<< z;
        }
```



output :x=2, y=6, z=14

The ampersand (&) (address of) is specifies that their corresponding arguments are to be passed *by reference* instead of *by value*.

Constant Arguments:-In C++ the value of constant argument cannot be changed by the function. To make an argument constant to a function , we can use the keyword const as shown below:

int myFunction(const int x , const int b);

The qualifier const tell the compiler that the function should not modify the argument. The compiler will generate an error when this condition is violated.

Default Arguments :- C++ allows us to assign default value(s) to a function's parameter(s) which is useful in case a matching argument is not passed in the function call statement. The default values are specified at the time of function definition. e.g.,

float interest (float principal, int time, float rate = 0.70f)

Here if we call this function as:

si_int= interest(5600,4);

then rate =0.7 will be used in function.

Formal Parameters:- The parameters that appear in function definition are formal parameters.

Actual Parameters :- The parameters that appears in a function call statement are actual parameters.

Functions with no return type (The use of void):- If you remember the syntax of a function declaration:

Return_type function_name(List of formal parameters)

you will see that the declaration begins with a type, that is the type of the function itself (i.e., the data type of value that will be returned by the function with the return statement). But what if we want to return no value?

Imagine that we want to make a function just to show a message on the screen. We do not need it to return any value. In this case we should use the void type specifier for the function. This is a special specifier that indicates absence of type.

The return Statement :- The execution of return statement, it immediately exit from the function and control passes back to the calling function (or, in case of the main( ), transfer control back to the operating system). The return statement also returns a value to the calling function. The syntax of return statement is:

return (value);

Scope of Identifier :- The part of program in which an identifier can be accessed is known as scope of that identifier. There are four kinds of scopes in C++

(i) Local Scope :- An identifier declare in a block ({ }) is local to that block and can be used only in it.

(ii) Function Scope :- The identifier declare in the outermost block of a function have function scope.

(iii) File Scope (Global Scope) :- An identifier has file scope or global scope if it is declared outside all blocks i.e., it can be used in all blocks and functions.

(iv) Class Scope :- A name of the class member has class scope and is local to its class.

Lifetime :- The time interval for which a particular identifier or data value lives in the memory is called Lifetime of the identifier or data value.

ARRAYS :-Array is a group of same data types variables which are referred by a common name.

An array is a series of elements of the same type placed in contiguous memory locations that can be individually referenced by adding an index to a unique identifier.

Need for Arrays:- Suppose we have to store roll number & marks of 50 students in a program , then in all we need 100 variable each having a different name. Now remembering and managing these 100 variables is not easy task and it will make the program a complex and not-understandable program.

But if we declare two array each having fifty elements one for roll numbers and another for marks. Now, we have only two names to remember. Elements of these array will be referred to as Arrayname[n], where n is the element number in in the array . Therefore arrays are very much useful in a case where quit many elements of the same data types need to be stored and processed. The element number s in parenthesis are called subscripts or index. The subscript, or index of an element designates its position in the array's ordering.

Types of Array :-Arrays are of two types:

1. One-dimensional Arrays
2. Multi-dimensional Arrays (But we have to discussing only Two-dimensional arrays in our syllabus)

One –dimensional Arrays :- The simplest form of an array is one - dimensional. The array itself is given a name and its elements are referred to by their subscripts. In C++ , an array must be defined before it can be used to store information. The general form of an array declaration is:

Data_type Array_name[size];

For example the following statement declares an array marks of int data type , which have 50 elements marks[0] to marks[49].

int marks[50] ;

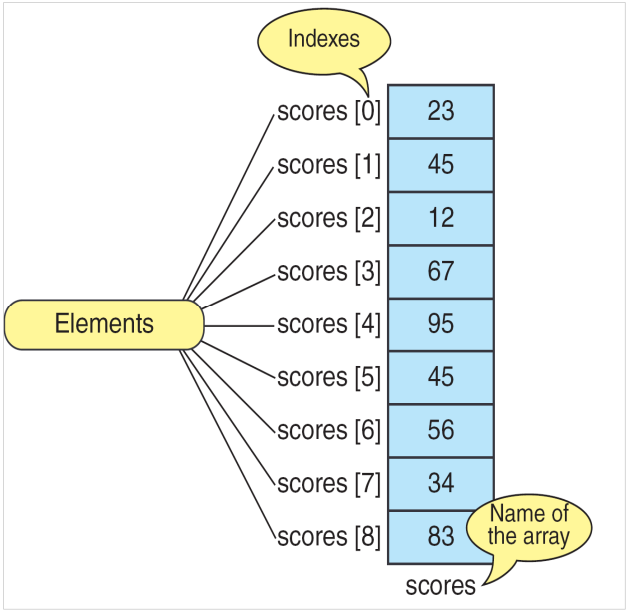
Referencing Array Elements:- We can reference array elements by using the array's name & subscript (or index). The first element has a subscript of 0. The last element in an array of length *n* has a subscript of *n-1*.

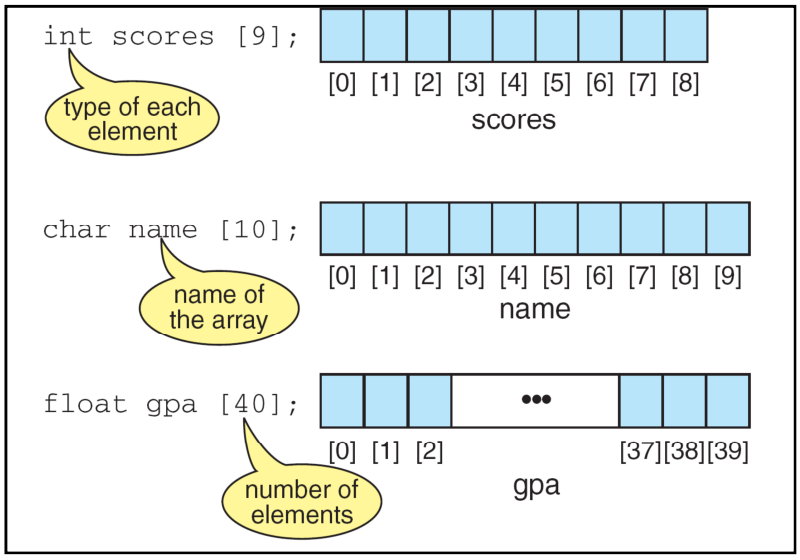
If we declare an array:

int scores[9];

then its 10 element will be referred as:

scores[0], scores[1], ……., scores[8]

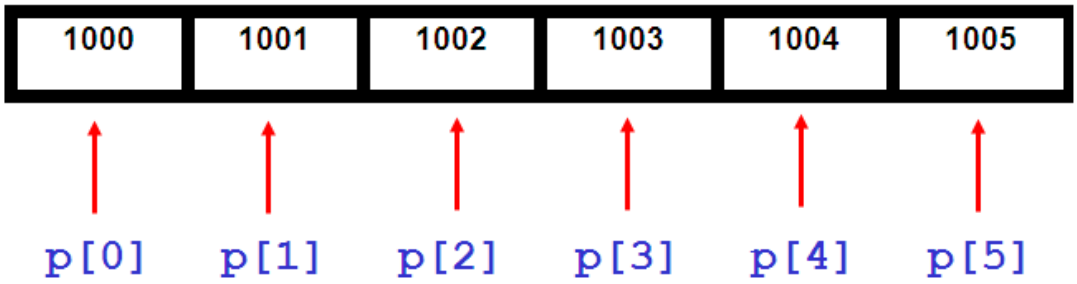




Memory Representation of Single Dimensional Arrays:-Elements of single dimensional array are stored in contiguous memory location in their index order. For example , an array p of type char with 6 elements declared as:

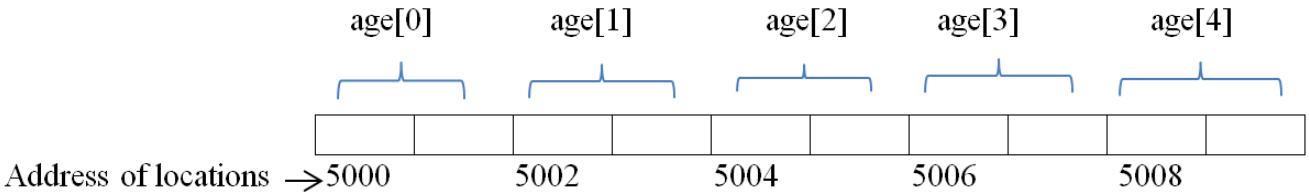
char p[6];

will be represented in memory as shown below:

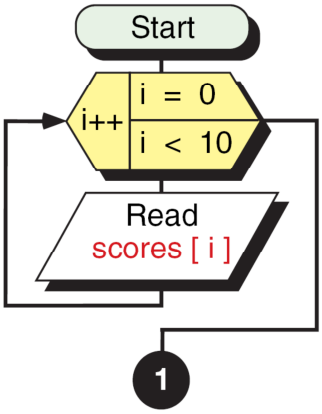


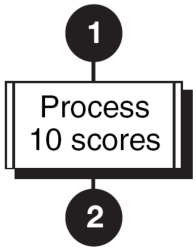
If we declare an int array age with 5 elements as:

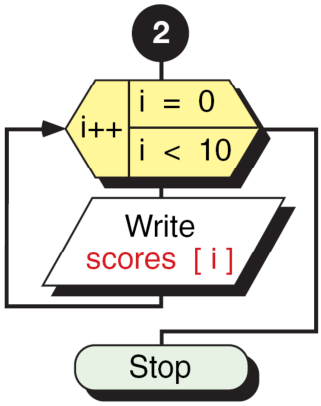
int age[5];



Accepting Data in Array from User (Inputting Array elements) and Printing Array elements:- Since we can refer to individual array elements using numbered indexes, it is very common for programmers to use **for** (or any) loops when processing arrays.







General form of for loop for Reading elements of array (1-D)	Generally processing part may be include with in the loop of reading or printing, otherwise a same type separate loop may be used for processing	General form of for loop for printing elements of array (1-D)

for (int i=0; i< size; i++)
{
cout<<"Enter Array Element "<<i+1;
cin>>Array_Name[i];
}
		for (int i=0; i< size; i++)
		{
		cout<<Array_Name[i]<< ", ";
		}

Program : A C++ program to read an array of 10 elements and print the sum of all elements of array:

```
#include <iostream.h>
void  main()
{
int a[10], sum=0;
for(int j=0; j<10; j++)                    //loop for reading
{
cout<< "Enter Element "<< j+1<<":";
cin>> a[j]
sum=sum+a[j]`;
}
cout<<"Sum : "<<sum;
}
```

Program: Here's another example of an array this program , invites the user to enter a series of six values representing sales for each day of the week (excluding Sunday), and then calculates the average sales.

```
#include <iostream.h>
void  main()
{
const int SIZE = 6;                        //size of array
double sales[SIZE];                        //array of 6 elements
double total = 0;
cout<< "Enter sales for 6 days\n";
for(int j=0; j<SIZE; j++)                  //put figures in array
cin>> sales[j];
for(j=0; j<SIZE; j++)                      //read figures from array
total += sales[j];                         //to find total
double average = total / SIZE;             // find average
cout<< "Average = " << average << endl;
}
```

Output:-

```
Enter sales for 6 days
352.64
867.70
781.32
867.35
746.21
189.45
Average = 634.11
```

Linear Search:- In linear search, each element of the array is compared with the given item to be searched for , one by one either the desired element is found or the end of the array is reached.

```
#include<iostream.h>
void main( )
{
int size, i, n, c=0;
cout<<"Enter the size of array:"
cin>>size;
int a[size];
cout<<"Enter the elements of Array: ";
```

```
for(i-0; i<size; i++)
{
cin>>a[i];
}
cout<<" Enter the number to be search : ";
cin>>n ;
for (i=0; i<size; i++)
{   if (a[i]= =n)
    {
       c=1;
break;
     }
}
if (c= =0)
     cout<<" The number is not in the array.";
else
cout<<" The Number is found and location of element is: "<<i+1:
}
```

Initializing arrays.

When we declare an array, we have the possibility to assign initial values to each one of its elements by enclosing the values in braces { }. Syntax for initialization of one-dimensional array is:

Data_type Array_name[sise] = { value list } ;

For example:

int num [5] = { 16, 2, 77, 40, 12071 };

This declaration would have created an array like this:

num

0	1	2	3	4
16	2	77	40	12071

The amount of values between braces { } must not be larger than the number of elements that we declare for the array between square brackets [ ].

C++ allows us to skip the size of the array in an array initialization statement (by leaving the square brackets empty [ ]). In this case, the compiler will assume a size for the array equal to the number of values given between braces { }:

int num [ ] = { 16, 2, 77, 40, 12071 };

After this declaration, the size of array num will be 5, since we have provided 5 initialization values.

Program : Program to find the maximum and minimum of an array :

```
#include<iostream.h>
void  main( )
 {
int arr[ ] = {10, 6 , -9 , 17 , 34 , 20 , 34 ,-2 ,92 ,22 };
int max = min = arr[0];
for(int i = 0 ; i<10 ; i++)
  {
                if(arr[i] < min)
                    min= arr[i];
```

```
                    if(arr[i] > max)
                      max = arr[i];
    }
  cout<<"The minimum of all is : " << min<<"and the maximum is : " <<max;
}
```

String as an array:- C++ does not have a string data type rather it implements string as single dimensional character array. A string is defined as a character array terminated by a null character '\0'. Hence to declare an array strg that holds a 10 character string, you would write :

```
char  strg[11];
```

Program : Program to count total number of vowels present in a string :

```
#include<iostream.h>
    #include<stdio.h>
    void  main( )
    {
      char string[35]; int count = 0;
      cout<<"Input a string";
      gets(string);                    // library function in stdio.h to input a string
      for(int i = 0 ; string[i] != '\0' ; i++)
      {
 if( string[i] = = 'a' || string[i] = = 'e' || string[i] = = 'o' || string[i] = = 'u' || string[i] = = 'i' ||
string[i] = = 'A' ||  string[i] = = 'E' || string[i] =  = 'O' || string[i] = = 'U' ||  string[i] == 'I' )
        {
            count++;
        }
      }
    }
```

Two dimensional array : A two dimensional array is a continuous memory location holding similar type of data arranged in row and column format (like a matrix structure).

Declaration of 2-D array:- The general syntax used for 2-Darray declaration is:

Data_type Array_name [R][C] ;

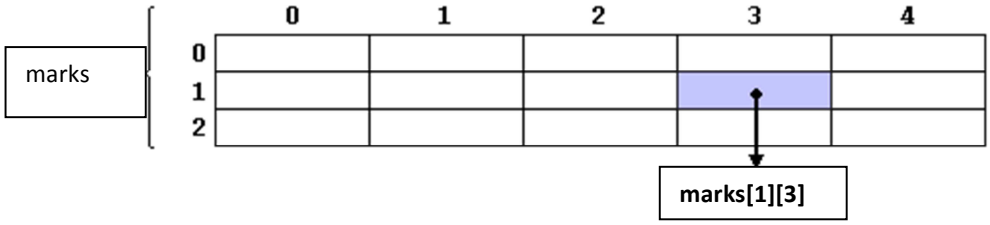
Where R represent number of rows and C represent number of columns in array.

For example,

int marks [3][5];

and, for example, the way to reference the second element vertically and fourth horizontally in an expression would be:

marks[1][3]



Initialization of 2-D Array:- Two dimensional arrays are also initialize in the same way as single dimensional array . e. g.,

int cube[5][2] = {1,1, 2,8,3,27,4,64,4,125 };

will initialize cube[0][0]=1, cube[0][1]=1, cube[1][0]=2, cube[1][1]=8 , cube[2][0]=3 and so on.

2-D array can also be initialized in unsized manner. However the first index can be skiped , the second index must be given. e.g.,

```
int cube[ ][2] = {1,1, 2,8,3,27,4,64,4,125 };
```

is also valid.

Accepting Data in 2-D Array from User (Inputting 2-D Array elements) and Printing 2-D Array elements:- Since We must reference both, a row number and a column number to input or output values in a two-dimensional array. So, we use a nested loop (generally nested for loop) , when processing 2-D arrays.

General form of for loop for Reading elements of 2-D array	Generally processing part may be include within the loop of reading or printing, otherwise a same type separate nested loop may be used for processing	General form of for loop for printing elements of 2-D array

for (int i=0; i< R; i++)
{ cout<<"Enter Row "<<i+1;
for (int j=0; j<C ; j++)
cin>>Array_Name[i][j];
}
		for (int i=0; i< R; i++)
		{ for (int j=0; j<C ; j++)
		cout<<Array_Name[i][j]
		<<'\t';
		cout<<'\n';
		}

Where R represent number of rows and C represent number of columns in array.

Program : Program to subtract two 3x3 matrices and then print the resultant matrix.

```
#include <iostream.h>
#include<conio.h>
main()
{
int m, n, c, d, first[10][10], second[10][10], sub[10][10];
cout<<"Enter the elements of first matrix\n";

for ( i = 0 ; j < 3 ; i++ )
for ( j = 0 ; j <3 ; j++ )
cin>>first[i][j];

cout<<"Enter the elements of second matrix\n";

for ( i = 0 ; j < 3 ; i++ )
for ( j = 0 ; j <3 ; j++ )
cin>>second[i][j];
cout<<"Subtraction of entered matrices:-\n";
for ( i = 0 ; j < 3 ; i++ )
  {
for ( j = 0 ; j <3 ; j++ )
        {
                sub[i][j] = first[i][j] - second[i][j];
                cout<<sub[i][j]<<'\t';
        }
  cout<<'\n' ;
}
getch();
}
```

Array of Strings:- An array of string is a two-dimensional character array. The size of first index determines the number of strings and size of second index determines maximum length of each string. The following statement declares an array of 10 strings , each of which can hold maximum 50 valid characters.

```
char string[10][51] ;
```

Notice that the second index has been given 51, because 1 extra size is given for store null character '\0'.

User Defined Data Types:-

The C++ language allows you to create and use data types other than the fundamental data types. These types are called user-defined data types.

Structures:- In C++, a structure is a collection of variables that are referenced by a single name. The variable can be of different data types. They provide a convenient means of keeping related information together.

Defining Structure :-

A Structure is defined by using the Keyword struct. The General Syntax for defining a Structure is :

Syntax :

```
struct< Name of Structure >
{
<datatype>< data-member 1>;
<datatype>< data-member 2>;
<datatype>< data-member 3>;
        …
        …
<datatype>< data-member n>;
} ;
```

Example :

A Proper example following the previous syntax could be :

```
struct  WORKER                  Structure name
{
char name[45];
char gender ;                   data-member
int age ;
float rate ;

} ;
```

Declaring Structure Variable :-

This is similar to variable declaration. We can declare the variable of structure type using two different ways either with structure definition or after structure definition.

The following syntax shows the declaration of structure type variables with structure definition:

```
struct< Name of Structure >
{
<datatype>< data-member 1>;
```

```
<datatype>< data-member 2>;
<datatype>< data-member 3>;
                …
                …
<datatype>< data-member n>;
} var1, var2,….., varn ;
```

We can declare the structure type variables separately (after defining of structure) using following syntax:

```
Structure_name  var1, var2, …… ….., var_n;
```

Accessing Structure Elements :- To access structure element , dot operator is used. It is denoted by (.). The general form of accessing structure element is :

```
Structure_Variable_Name.element_name
```

Initializing of Structure elements:- You can initialize members of a structure in two ways. You can initialize members when you declare a structure or you can initialize a structure with in the body of program. For Example:

First Method:-

```
#include <iostream.h>
#include <conio.h>
void main()
{
        // Declaring structure at here
        struct Employee
        {
                int empcode;
                float empsalary;
        } ;
        Employee emp1 = {100, 8980.00} ;          // emp1 is the structure variable ,
                                                  // which is also initialize with declaration
        clrscr();
        int i;          // declares a temporary variable for print a line
                        // Printing the structure variable emp1 information to the screen
        cout<< "Here is the employee information : \n";
        for (i = 1; i <= 32; i++)
                cout<< "=";
        cout<< "\nEmployee code : " << emp1.empcode;
        cout<< "\nEmployee salary : " << emp1.empsalary;
}
```

Second Method:-

```
#include <iostream.h>
#include <conio.h>
void main()
{
        // Declaring structure  here
        struct Employee
        {
                int empcode;
                float empsalary;
```

```
        } emp1;          // emp1 is the structure variable
        clrscr();
        int i;           // declares a temporary variable for print a line
        // Initialize members here
        emp1.empcode = 100;
        emp1.empsalary = 8980.00;
        // Printing the structure variable emp1 information to the screen
        cout<< "Here is the employee information : \n";
        for (i = 1; i <= 32; i++)
                cout<< "=";
        cout<< "\nEmployee code : " << emp1.empcode;
        cout<< "\nEmployee salary : " << emp1.empsalary;
}
```

Structure variable assignments

We know that every variable in C++ can be assigned any other variable of same data type i,e :

if int a = 7 ; b = 3;

we can write :

a = b ; // assigning the value of b to variable a

Similar is the case with Structure variables also , i,e :

if WORKER w1 = {“Ramu” , ‘M’ , 17 , 100.00};

WORKER w2;

we can write :

w2 = w1 ; // assigning the corresponding individual // data member values of w1 to Worker w2;

or

WORKER w2 = w1;

Note : Both structure variables must be of same type (i,e WORKER in this case)

There is a member wise copying of member-wise copying from one structure variable into other variable when we are using assignment operator between them.

So, it is concluded that :

```
Writing : strcpy (w1. name,w2.name) ;
w1.gender = w2.gender ;
w1.age = w2.age ;
w1.rate = w2.rate ;
```

is same as :

```
w1 = w2 ;
```

Array of Structure :- We can define an array of structure by using following syntax :

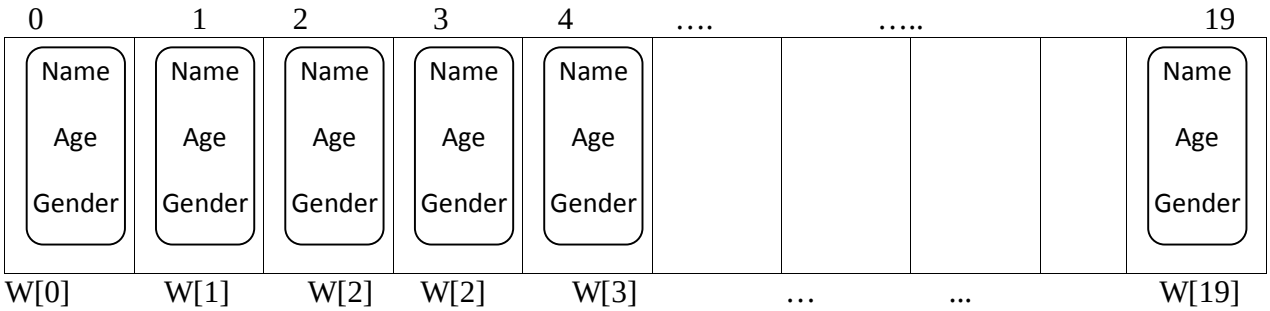
<structure_name><array_name>[size];

where : structure_name is the name of the structure which you have created.
array_name is any valid identifier
size is a positive integer constant.

Example : to create array of 20 Workers we can have :

Worker W[20];

The above structure could be visualized as :



Each of the elements of the array is itself a structure hence each of them have all the four components.

Function with Structure variable:-

We can also pass a structure variable as function parameter / arguments. The benefit is that the structure carries a bundled information to the structure. The prototype of such a function would be

<return_type> function_name(<structure_name><var> , … , …);

Let us understand the concept with following function ,which increases the wage of a female worker by passed percent

```
void increaseWage(   Worker  & w ,  float incr )
{
        if( w.gender == 'F' || w.gender == 'f' )
        {
                w.wage + = w.wage* (incr /100) ;
        }
}
```

Look at the highlighted parameter, we have passed formal structure variable as reference, so that the increment is reflected back in actual parameter.

Similarly, if we don't want to pass our structure variable as reference we can do so , but then we have to return the modified structure variable back. This can be achieved in above function, if we take return type as structure name. Let us modify the function so that it returns a structure :

```
Worker increaseWage( Worker w , float incr)
{
        if( w.gender == 'F' || w.gender == 'f' )
        {
                w.wage + = w.wage* (incr /100) ;
        }
 return w ;
}
```

Nested structure :- A Structure within another structure is known as nested structure. A structure containing an another structure as valid element is known as nested structure.

e.g.

```
struct address
{
int houseno;
char city[20];
char area[20];
long int pincode;
} ;

struct employee
{
int empno;
char name[30];
char design[20];
char department[20];
address ad; // nested structure
}
```

Declaration:

employee e;

Input /Output :

```
cin>>e.ad.houseno;          // members are accessed using dot(.) operator.
cout<<e.ad.houseno;
```

typedef :- The typedef keyword allows to create a new names for data types. the syntax is:

typedef existing_data_type new_name ;

e.g.,

```
typedef int num;
```

#**define Preprocessor Directive :-**The # define directive create symbolic constant, constants that are represent as symbols and macros (operation define as symbols). The syntax is:

```
#define identifier replacement_text ;
```

When this line appears in a file, all subsequent occurances of identifier in that file will be replaced by replacement_text automatically before the program is compiled. e.g.,

Program: Program to calculate area of a circle by using #define preprocessor directive.

```
#include <iostream.h>
#define PI 3.14159
#define CIRCLE_AREA(x) (PI * (x) * (x))
void main()
{
    float area;
    int r;
    cout<< "Enter the radius : ";
    cin>> r;
    area = CIRCLE_AREA(r);
    cout<< "Area is : " << area;
}
```