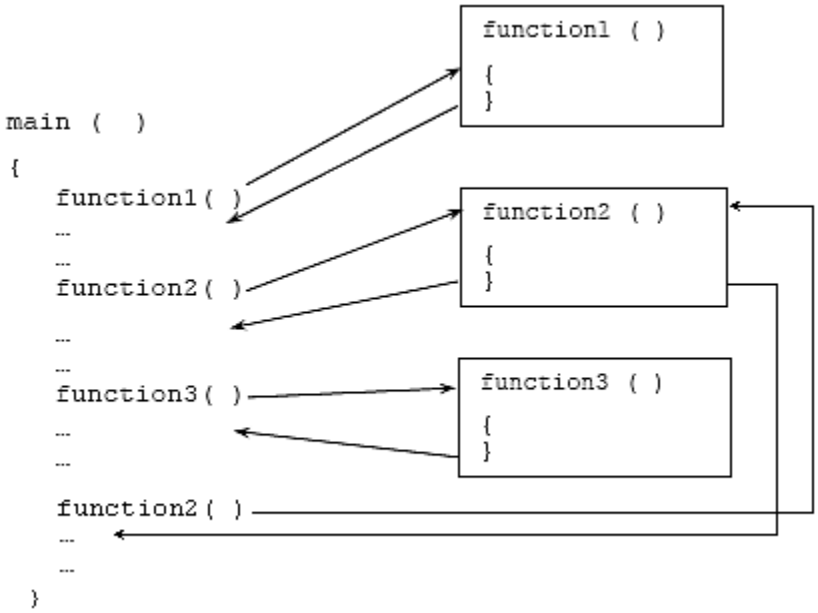
Computer Programming

Lab Manual No 08

Functions in C++

Part II



Semester:

Spring 2016

Objectives:-

The objectives of this session is:

- Learn about passing structures to Functions.
- Returning data from Functions.
- Returning Structures from Functions.
- Local and Global Variables.
- Local and Global Functions.
- Reference parameters of Functions.
- Function Overloading.
- Built-in Functions.

Passing Structures to Functions:-

The structures can also be passed as arguments to a function. If a structure is to be passed to a function, then it must be defined before declaration of the function that uses the structure type data as argument.

The program example given below explains the use of structure type data as argument of a function.

```
#include<iostream>

#include<cstdlib>

using namespace std;


struct xyz

{

        char name[15];

        int age;

};

int main()
```

```
{
    xyz s1;
    void rec(xyz);
    cout<<"Enter your name\n ";
    gets(s1.name);
    cout<<"Enter your Age\n";
    cin>>s1.age;
    rec(s1);
    cout<<"Okay\n";
    system("PAUSE");
    return 0;

}

void rec(xyz fs)
{
     cout<<"Name = "<<fs.name<<endl;
     cout<<"Age = "<<fs.age<<endl;
}
```

The variable **s1** is declared as structure type (i.e. xyz) inside the main function and the function **"rec"** is declared after this. The function **"rec"** uses an argument of type **"xyz"** i.e. of structure type. The function **"rec"** is called by passing the structure variable **s1** to function definition **"rec".**

In function definition the variable **"fs"** of type **"xyz"** receives the data of **s1** and the same is printed on the screen. An array of structure type can also be passed as argument to a function in the similar way as any other array is passed as argument.

Returning Data from Functions:-

A function can return only one value. It can be of any type except string and array. The type of data that a user defined function returns is declared in function declaration.

The value of data is returned from the function to the calling function by the **return** statement.

Return Statement:-

The return statement is used to return the calculated value from function definition to the calling function. In some compilers of C and C++ it is used as a last statement of body of the function to return the control as well as the calculated value to the calling function.

Its syntax is:

```
return expression;
```

where

expression — represents and arithmetic expression or constant value. The value of the expression is returned to the calling function.

If the function has a **"void"** return type then the use of this statement is optional.

Declaration of Function that returns a Value:-

The type of function specifies the type of the data returned by it. It is specified in the function declaration. It is also called the data type of the function.

The data type of a function is declared in the same way as the type of a variable is declared, e.g.

```
float temp (void);
```

In the above function declaration;

temp — name of the function

float — returned data type

void in the brackets specifies that it has no arguments.

Calling a function that returns a value:-

The function that returns a value can be called in an arithmetic expression. In this case the function is treated as a variable. Similarly the value returned by the function may be assigned to a variable or it can be directly sent to an output device, i.e. printer, screen or file.

For example to assign the returned value of the function "temp" to a variable "r" the statement is written as:

```
r= temp();
```

where "r" is a variable that receives the returned value of function "temp". Since the function will return float type value the variable "r" should also be of float type.

To print the returned value directly on the screen the function "temp" is called as:

```
cout<<"The returned value is : "<<temp();
```

To use this function in an arithmetic expression it is called as:

```
r = x + temp( ) * 1000;
```

where

`r` represents the receiving data variable of numeric type.

`x` represents a variable of numeric type.

`temp()` represents a function call. It is also treated as variable.

Function Definition that returns a Value:-

The function that returns a value is defined in the similar way as other functions are defined but in the declarator the return type is mentioned.

The returned data type must be the same as mentioned in the function declaration. The "return" statement must be used in the function body to return the value to the calling function.

For example a program is given below that converts the kilogram into grams by using user defined function.

Example:-

```
#include<iostream>
#include<cstdlib>
using namespace std;
int main()
{
    int kg,gm;
    int grams(int);
    cout<<"Enter values in kilograms\n";
    cin>>kg;
    gm=grams(kg);
    cout<<"Answer in grams = "<<gm<<endl;
    system("PAUSE");
    return 0;
}
int grams(int val)
{
    int k;
    k=val*1000;
    return k;
}
```

Returning Structures from Functions:-

Since a structure type data can be passed to a function, a function can also return a structure type data. Usually the structure type data returned by the function is assigned to a structure type variable. For example:

```
#include<iostream>
#include<cstdlib>
using namespace std;


struct temp
{
        char mn[15];
        float marks;
};
int main()
{

    temp abc(void);
    temp xyz;
    xyz=abc();
    cout<<"Name = "<<xyz.mn<<endl;
    cout<<"Marks = "<<xyz.marks<<endl;
    cout<<"Okay\n";
    system("PAUSE");
```

```
    return 0;


}


temp abc(void)
{
      temp rec;
      cout<<"Enter Name "<<endl;
      gets(rec.mn);
      cout<<"Enter Marks "<<endl;
      cin>>rec.marks;
      return rec;
}
```

In the above program the **"temp"** is the tag name of the structure. The function **"abc"** is declared that has no arguments and its returned data type is temp i.e. structure type. When the function is called then it returning data is assigned to variable **"xyz"** after executing body of the function.

In the function definition the variable **"rec"** is also declared as data type **"temp".** The data is entered into the **"rec"** and the same is returned to the calling program by return statement.

Local and Global Variables:-

The variables can be declared inside the main function inside user defined functions or outside the main functions. The effect of the variables declared in these places is different. Based upon these effects variables are divided into two classes.

1. Local Variables
2. Global Variables

Local Variables:-

The variables that are declared inside the main function or inside any user defined function are called local variables. These variables are also called the automatic variables. The keyword "auto" can be used to declare these variables. For example,

```
auto int a,b,c;
```

The use of keyword "auto" is optional. The variables used inside the functions are automatic by default.

Life time of a Local Variables:-

The life time of a local variable is the time period between the creation and destruction of the variable. A variable remains available and can be used only during its lifetime.

When the control is transferred to a function, the local variable or variable declared inside that function are automatically created and they occupy memory spaces to store data. When the control returns to the calling function or calling program, the variables of that function are destroyed and their data is lost. The local variables can only be accessed from within outside that function.

The following program example explains the concept of local variables.

```
#include <iostream>

#include<cstdlib>

using namespace std;


int main ()

{

    int a,b,s;

    int sum(int,int);

    a=10;

    b=100;
```

```
    s=sum(a,b);
    cout<<"Sum is "<<s<<endl;
    system("PAUSE");
    return 0;
}


int sum(int x,int y)
{
    int rs;
    rs=x+y;
    return rs;
}
```

In the above program the variables a, b and s are declared in the main function. These are local variables for main function. The variables declared in the "sum" function are x, y and rs. The function declared in function declaratory are also treated as local variables.

When the function "sum" is called control transfer to the function definition "sum". The variables x , y and rs are created into the memory. The value 10 is stored in variable x and 100 in variable y. The sum of **"x"** and **"y"** is calculated inside the function and the result is returned to the calling function. When the control returns, the variables x, y and rs are destroyed from memory.

Global Variables:-

The variables that are declared outside the main function or any other function are called global variables or external variables. These variables can be accessed in any function during execution of the program.

The global variables are used when variables are to be accessed by more than one function in a program. The global variables are not normally used in program because:

Since they occupy a large amount of memory permanently during program execution and the data accessing speed of the program becomes low.

Since they are accessible to all functions, it is difficult to track changes in their values.

Life Time of a Global Variables:-

The global variables exist in the memory throughout the program execution. They are created into the memory when the program execution starts. They are destroyed only when the program ends. Therefore the lifetime of a global variables is between the starting and termination of the program.

A program is given that uses global as well as local variables is given below:

```
#include <iostream>
#include<cstdlib>
using namespace std;
int a,b,s;
int main ()
{
    int sum(int,int);
    a=10;
    b=100;
    s=sum(a,b);
    cout<<"Sum is "<<s<<endl;
    system("PAUSE");
    return 0;
}
```

```
int sum(int x,int y)
{
    s=x+y;
    return s;
}
```

In the above program a , b and s are global variables and are outside the main function. These can be accessed by sum function as well as main function. The variables x and y are local variables of function **"sum".**

Static Variables:-

Static variables are special variables that are declared inside a function by using the keyword **"static".** Like local variables these can only be accessed in the function in which they are declared but they remain in existence for the lifetime of the program.

Another difference between the local variables and static local variables is that the initialization in the static variables take place only once when the function is called for the first time.

A program example is given below that uses the static variables.

```
#include <iostream>
#include<cstdlib>
using namespace std;
int main ()
{
    int i;
    int temp(int);
    for(i=1;i<=3;i++)
    {
```

```
                    cout<<temp(i)<<endl;
    }
    cout<<"Okay\n";
    system("PAUSE");
    return 0;
}
int temp (int x)
{
    static int co=10;
    co=co*x;
    return(co);
}
```

The output of the above program will be as

```
10
20
60
```

Note:- The static global variables can also be declared outside the main function but these are normally used in multiple file programs.

Local and Global Functions:-

The functions that are declared inside the main function or inside any other function are called local function. They can be called only in that function in which they are declared.

The functions that are declared outside the main function are called global functions. They can be called by any function.

The function prototypes are defined in the header file are treated as global functions because they can be called in any function.

Reference Parameters:-

There are two ways to pass arguments to a function. These are:

1. Arguments passed by value
2. Arguments passed by reference.

When an argument is passed by value to a function a new variable of the data type of the argument is created and the data is copied into it. The function accesses the value in the newly created variable and the data in the original variable in the calling function is not changed.

The data can be passed to a function by reference of a variable name that contain data. The reference provides the second name for a variable name. When a variable is passed by reference to a function, no new copy of the variable is created. Only the address of the variable is passed to the function. The original variable is accessed in the function with reference to its second name or alais. Both variables use the same memory location. Thus any change in the reference variable also changes the value in the original variable.

The reference parameter are indicated by an ampersand (&) sign after the data type both in the function prototype and in the function definition.

Example:-

Write a program to exchange the values of two variables by using reference arguments in the function.

```
#include <iostream>

#include<cstdlib>

using namespace std;

int main ()

{

    void exch(int & , int &);

    int a,b;
```

```
    cout<<"Assign value to variable A ? "<<endl;
    cin>>a;
    cout<<"Assign value to variable b ? "<<endl;
    cin>>b;
    exch(a,b);
    cout<<"Value after exchange"<<endl;
    cout<<"Value of A is "<<a<<endl;
    cout<<"Value of B is "<<b<<endl;
    system("PAUSE");
    return 0;
}


void exch(int& x, int& y)
{
     int t;
     t=x;
     x=y;
     y=t;
}
```

In the above program the function **"exch"** is declared with two int type parameters. The ampersand sign (&) is used with each data type. It indicates that both the parameters of the function **"exch"** are reference parameters.

When the function **"exch"** is called no ampersand sign (&) is used.

The ampersand sign is also used with the data type of x and y variables used in the declaratory of the function definition. The **"x"** and **"y"** variables are the alais of **"a"** and **"b"** variables, respectively. The memory location of **"x"** and **"a"** is the same and similarly memory location for **"y"** and **"b"** are same.

The structure and array variables can also be passed to the function by reference in the similar way as simple data types are passed. By default the array variables are passed by reference.

Example:-

Write a program to find the factorial of a given number by using function and also by using reference parameters.

```
#include <iostream>

#include<cstdlib>

using namespace std;

int main ()

{

    void fact(int & , int &);

    int f=1,n;

    cout<<"Enter any number "<<endl;

    cin>>n;

    fact(f,n);

    cout<<"Calculated Factorial is  "<<f<<endl;;

    system("PAUSE");

    return 0;

}
```

```
void fact(int& fact, int& a)
{
      for (;a>=1;a--)
      fact*=a;
}
```

Default Arguments:-

The arguments in which data is initialized during the declaration are called default arguments. If the values of the arguments are specified in the function declaration then the arguments of the functions in the function call can be omitted. If the arguments are omitted in function call then the default values are automatically passed to the function definition.

For example in the following program two default argument parameters are defined in the function declaration.

```
#include <iostream>
#include<cstdlib>
using namespace std;
int main ()
{
     void temp(char[]= "Pakistan" ,int=2);
     temp();
     temp("Islamabad",10);
     system("PAUSE");
     return 0;
}
```

```
void temp(char x[], int y)

{

      for (int c=1;c<=y;c++)

      cout<<x<<endl;

}
```

In the above program, the function "temp" has two default arguments, first of string type and second of integer type. These values are initialized at the time of function declaration. When the function "temp" is called without using the parameter values, the default parameter values are passed to the function and the string "Pakistan" will be printed twice.

In the second call the parameter values "Islamabad" and 10 are also passed with the function call. In this case the string "Islamabad" will be printed ten times.

Function Overloading:-

Declaring more than one function with the same name but different set of arguments and return data types is called function overloading.

For example three function with the same name "sum" and have different parameters are declared as:

```
int sum (int, int);

int sum (int, int, int );

double sum (double, double, double);
```

The first function **"sum"** has two parameters both of int type and returned data type is also of int type.

The second function **"sum"** has three parameters, all of int data type. The return type is also of int type.

The third function **"sum"** has three parameters all of double type and its return type is also double.

When the program that has overloaded function is compiled the C++ compiler checks the number of arguments, their order and data type and marks a proper function name for each function. At function call it executes that function whose return type or parameters matched with the return type or parameter given in the function call. The following program explains the concept of function overloading.

```
#include <iostream>
#include<cstdlib>
using namespace std;
int main ()
{
    int sum(int,int);
    int sum(int,int,int);
    double sum(double, double, double);
    cout<<"Sum of 3 integer values = " <<sum(2,4,5)<<endl;
    cout<<"Sum of 3 float values = " <<sum(2.6,4.7,5.6)<<endl;
    cout<<"Sum of 2 integer values = " <<sum(2,4)<<endl;
    system("PAUSE");
    return 0;
}

int sum(int x,int y)
{
    return x+y;
}
```

```
int sum(int x,int y,int z)
{
    return x+y+z;
}

double sum(double x,double y, double z)
{
    return x+y+z;
}
```

In the above program three functions are defined with the same name **"sum".** When the function **"sum"** is called by passing three parameters of integer type the control shift to second function that has three integer type arguments. Similarly when the function is called by passing three float data type values control shifts to the 3rd function that matches the parameters.

Built-in Functions:-

The functions which already has been defined and are part of language and can be used in any program are called built in functions.

Accessing Built in Functions:-

The built in functions can be accessed through header files.

Each built in function is defined in its related header file. If a built in function is to be used in a program its header files must be included in the program.

For example "**clrscr**" function is defined in "**conio.h**" header file. If this function is to be used in the program, the **conio.h** header file must be included in the program. Similarly commonly used mathematical functions are defined in the "**math.h**" header file and when a mathematical function is to be used, "**math.h**" file is included in the program.

Arguments of the functions:-

To perform an operation, a function may require data. It perform operations on the data and produces result. The data that is passed to a function is called arguments of a function.

The arguments of the function may be of different types. These may be int, float, string, double etc. Some functions need more than one parameters, separated by commas. Some functions need no parameters.

Similarly some built in functions also return a single value to the calling program after executing all its statements. The returned value may be stored in a variable or directly sent to the output device.

The commonly used built in functions are discussed below:

Built in Functions of "stdio.h":-

The "stdio" stands for standard input/output.

This header file contain a large number of standard input and output functions that are used to get input from the input devices and also to print result on the output device.

The commonly used functions of this header file and their description are given below.

The "getchar" Function:-

This function is used to get a single character from a standard input device i.e. from the keyboard during program execution.

The main features of this function are:

1. When a character is entered it is displayed on the screen.
2. Enter key must be pressed to complete the input.

Its syntax is:

```
var = getchar();
```

The use of var is optional. It is used to store the value of the character entered from the keyboard.

The "putchar" Functions:-

The "putchar" stands for put character.

It is used to put or print a single character on the output device. i.e. computer screen.

The "gets" Functions:-

This function is used to get input data into a string variable from the keyboard. Any type of characters, including space and special characters can be entered using this function. Enter key is pressed after entering the string. When the Enter key is pressed, a null character is appended at the end of the string.

Its syntax is:

```
gets(strvar);
```

where

strvar represents the string type variable into which data is to be entered.

The data in a string type variable can also be entered by using "cin" object but this object is not the most suitable method of inputting data into string variables. When a space key is pressed during input process the "cin" object immediately appends the null character ('\0') at that place in the string. The characters entered after the space are not stored in the string variable.

The "puts" Functions:-

It is used to print a string on the computer screen. The new line is inserted after printing the string.

Its syntax is:

```
puts(string);
```

The "string" may be a string constant or a string type variable.

In case of a string constant, the string is enclosed in double quotation. In case of a string variable, it is written without quotes.

Example:-

Write a program to input a string by using "gets" function print it on the screen by using the "puts" function.

```
#include <iostream>
#include<cstdlib>
#include<conio.h>
using namespace std;
int main ()
{
    char str[20];
    char var;
    puts("Enter a String");
    gets(str);
    puts("The string is ");
    puts(str);
     var= getchar();
    system("PAUSE");
    return 0;
}
```

Built in Functions of "string.h":-

The "string.h" header file contains the functions that are used to process strings. The commonly used functions of this header file are given below.

The "strlen" Function:-

The "strlen" stands for string length. This function is used to find the length of the string. Is counts the total number of characters including spaces. Null characters are excluded.

Its syntax is:

```
int strlen(string);
```

where

string represents the string whose numbers of characters are to be counted. It may be a constant string type variable.

Example:

Write a program to get a string, find out its length, print the string and its length on the screen.

```
#include <iostream>
#include<cstdlib>
#include<conio.h>
#include<string.h>
using namespace std;
int main ()
{
    char str[100];
    puts("Enter a String");
    gets(str);
    cout<<"Length of string is "<<strlen(str)<<endl;
    system("PAUSE");
    return 0;
}
```

The "strcmp" Function:-

The "strcmp" stands for string compare.

This function is used to compare two strings. It compares first string with the second string character by character.

Its syntax is:

```
int strcmp(string1 , string2 );
```

where

string1 represents the first string. It may be a string constant or a string type variable.

string 2 represents the second string. It may be a string constant or a string type variable.

The function returns integer type output as follow:

- If string1 is equal to string2 then it returns 0.
- If string1 is less than string 2 it returns less than 0.
- If string 1 is greater than string 2 then it returns greater than 0.

For example, if:

S1="ISLAMABAD"

S2="ISLAMABAD"

S3="QUETTA"

```
strcmp(S1,S2)

strcmp(S3,S2) returns less than 0

strcmp(S2,S3) returns greater than 0.
```

The "strncmp" Function:-

This function is similar to the "strcmp" function but it compares specified number of characters of the two strings.

Its syntax is:

```
int strncmp (string1, string2, n);
```

where

string1 represents the first string. It may be a string constant or a string type variable.

string 2 represents the second string. It may be a string constant or a string type variable.

n represents a specified numbers of characters that are to be compared. It may be an integer constant or integer type variable.

This function returns the int type output as follows:

If the first "n" characters of the string 1 are equal to the first "n" characters of string 2 it returns 0.

If the first "n" characters of the string 1 are less to the first "n" characters of string 2 it returns less than 0.

If the first "n" characters of the string 1 are greater than the first "n" characters of string 2 it returns greater than 0.

The "strcpy" Function:-

The "strcpy" stands for string copy.

It is used to copy the contents of one string variable including the null character ('\0').

Its syntax is:

```
strcpy(string1, string2);
```

where

string 1 represents the first string. It must be a string type variable because the contents of string2 are to be copied into it.

string2 represents the second string. It must may be a string constant or string type variable.

For example to copy the string "I love Pakistan" into the string variable. "Pak" the statement is

written as:

```
strcpy(pak,"I love Pakistan");
```

The "strncpy" Function:-

It is similar to strcpy function but it is used to copy a specified number of characters from one string to another string variable.

Its syntax is:

```
strncpy(string1, srting2, n);
```

where,

string 1 represents the first string. It must be a string type variable because the contents of first "n" characters of string2 are to be copied into it.

string2 represents the second string. It must may be a string constant or string type variable. The first "n" characters of this string are copied into the string1.

n represents the number of characters of string 2 that are to be copied into string1. It may be an integer constant or an integer type variable.

For example to copy the first three characters of string "Pakistan" into the string variable acon,

the statement is written as:

```
strncpy(acon,"Pakistan",3);
```

The "strcat" Function:-

The "**strcat**" stands for string concatenate.

It is used to append or combine the contents of one string to another string variable including the null character. Its syntax is:

```
strcat(string1,string2);
```

where,

string 1 represents the first string. It must may be a string constant or string type variable. The contents of string2 are added at the end of this string.

string2 represents the second string. It must may be a string constant or string type variable. The contents of this string are added at the end of the string1.

For example to add the string "istan" to the string variable "acon" that already contain the string "Pak" the statement is written as:

```
strcat(acon , "istan");
```

After executing the above statement the contents of the variable "acon" will be "Pakistan".

The "strncat" Function:-

It is similar to strcat function but it is used to append a specified number of characters of one string to another string variable. Its syntax is:

```
strncat(string1, string2, n);
```

where,

string 1 represents the first string. It must be a string type variable because the contents of first "n" characters of string2 are to be appended into it.

string2 represents the second string. It must may be a string constant or string type variable. The first "n" characters of this string are appended into the string1.

n represents the number of characters of string 2 that are to be appended into string1. It may be an integer constant or an integer type variable.

For example, to append the first 8 characters of string "Pakistan is my country" into the string variable "str" that already contains the contents "I live in" the statement is written as:

```
strncat(str, "Pakistan is my country",8);
```

After executing this statement the contents of the variable "str" will be "I live in Pakistan".

Built in Functions of "math.h":-

The functions that are need to perform the mathematical calculations are defined in "math.h" header file.

The commonly used math library functions are discussed below.

The "pow" Function:-

It is used to calculate the exponential power of a given integer number.

Its syntax is

```
pow(x , y);
```

where

x is a integer whose power is to be calculated.

y is an integer number that represents the exponent of the number.

Example:

If the value of 2^3 is to be calculated the function is written as pow(2 ,3);

The "sqrt" Function:-

It is used to calculate the square root of a given positive number.

Its syntax is:

```
sqrt(x);
```

where

x is a positive number of type double. It also returns a value of type double.

Example: If x=9.0 then

sqrt(x) returns 3.0 (i.e. 9=3)

The "floor" Functions:-

It is used to round a given float value to the integer value. The given float value is converted into largest integer value that is not greater than the given float value.

Its syntax is:

```
floor(x);
```

where,

x is a float or a double value/variable.

Example:

If x=9.6 then floor(x) returns 9.0

If x=-9.6 then floor(x) returns -10.0

The "ceil" Function:-

It is similar to the floor function but it returns the rounded integer value greater than the given float or double number.

Its syntax is:

```
ceil (x);
```

where

x is a given float or double type given number or value.

Example:-

If x=19.2 then ceil(x) returns 20.0

If x=-19.2 then ceil(x) returns -19.0

The "fmod" Functions:-

It is used to calculate the remainder by dividing one floating point number by another floating point number.

Its syntax is:

```
fmod(x,y);
```

where

x represents the nominator. It is a floating point number or variable.

y represents the denominator. It is a floating point number or variable.

Example:

If x=9.50 and y=4.00 then

fmod(x , y) returns 1.5

When y=0 fmod returns 0.

The "cos" Function:-

It is used to calculate the trigonometric cosine of a given angle. The angle is given in radius as a floating number. Its syntax is:

```
cos(x);
```

where

x represents an angle in radians.

Example:

If x=0.0 then cos(x) returns 1.0.

The "sin" Function:-

It is used to calculate the trigonometric sine of a given floating number. The angle is given in radians as a floating number. Its syntax is:

```
sin(x);
```

where

x represents an angle or value in radians.

Example:

If x=0.0 then sin(x) returns 0.

The "tan" Function:-

It is used to calculate the trigonometric tangent of a given floating number. The angle is given in radians as a floating number. Its syntax is:

```
tan(x);
```

where

x represents an angle or value in radians.

Example If x=0.0 then tan(x) returns 0.

The "log" Functions:-

It is used to calculate the natural logarithm (base e) of a given floating number. Its syntax is:

```
log(x);
```

where

x represents a given floating number.

The "log10" Function:-

It is used to calculate the logarithm (base 10) of a given floating number. Its syntax is:

```
log10(x);
```

where,

x represents a given floating number.

Note:- You can look for different header files and their associated functions on the internet. Some of the links are provided below:-

http://www.cppforschool.com/tutorial/libraryfunc.html

http://www.datarecovery.com/cfunctions.asp

http://udel.edu/~caviness/Class/CISC181-03F/C++HeaderFiles.html

http://www.cs.uah.edu/~rcoleman/Common/C_Reference/HeaderFiles.html

These are just a few references not a complete list.