Function

A function is a subprogram that acts on data and often returns a value. A program written with numerous functions is easier to maintain, update and debug than one very long program. By programming in a modular (functional) fashion, several programmers can work independently on separate functions which can be assembled at a later date to create the entire project. Each function has its own name. When that name is encountered in a program, the execution of the program branches to the body of that function. When the function is finished, execution returns to the area of the program code from which it was called, and the program continues on to the next line of code.

Creating User-Defined Functions

Declare the function.

The declaration, called the **FUNCTION PROTOTYPE**, informs the compiler about the functions to be used in a program, the argument they take and the type of value they return.

Define the function.

The function definition tells the compiler what task the function will be performing. The function prototype and the function definition must be same on the return type, the name, and the parameters. The only difference between the function prototype and the function header is a semicolon.
The function definition consists of the function header and its body. The header is EXACTLY like the function prototype, EXCEPT that it contains NO terminating semicolon.

```
//Prototyping, defining and calling a function
#include <iostream>
using namespace std;
```

```
void starline();          // prototype the function

int main()
{
     starline( );       // function call
     cout<< "\t\tBjarne Stroustrup\n";
     starline( );     // function call
     return 0;
}

// function definition
void starline()
{
     int count;          // declaring a LOCAL variable
     for(count = 1; count <=65; count++)
          cout<< "*";
     cout<<endl;
}
```

Argument To A Function

Sometimes the calling function supplies some values to the called function. These are known as parameters. The variables which supply the values to a calling function called **actual parameters**. The variable which receive the value from called statement are termed **formal parameters**.

Consider the following example that evaluates the area of a circle.

```
#include<iostream>
using namespace std;
void area(float);

int main()
{
    float radius;
    cin>>radius;
    area(radius);
    return 0;
}
```

```
void area(float r)
{
    cout<< “the area of the circle is”<<3.14*r*r<<”\n”;
}
```

Here radius is called **actual parameter** and r is called **formal parameter**.

Return Type Of A Function

// Example program

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

int timesTwo(int num);    // function prototype
int main()
{
     int number, response;
     cout<<"Please enter a number:";
     cin>>number;
     response = timesTwo(number);  //function call
     cout<< "The answer is "<<response;
     return 0;
}
//timesTwo function
int timesTwo (int num)
{
     int answer;    //local variable
     answer = 2 * num;
     return (answer);
}
```

Calling Of A Function

the function can be called using either of the following methods:
i) call by value
ii) call by reference

Call By Value

In call by value method, the called function creates its own copies of original values sent to it. Any changes, that are made, occur on the function’s copy of values and are not reflected back to the calling function.

Call By Reference

In call be reference method, the called function accesses and works with the original values using their references. Any changes, that occur, take place on the original values are reflected back to the calling code.

Consider the following program which will swap the value of two variables.

using call by reference

```
#include<iostream>
using namespace std;
void swap(int &, int &);
int main()
{
   int a=10,b=20;
   swap(a,b);
   cout<<a<<" "<<b;
   return 0;
}
void swap(int &c, int &d)
{
   int t;
   t=c;
   c=d;
   d=t;
}
```

output:
20 10

using call by value

```
#include<iostream>
using namespace std;
void swap(int , int );
int main()
{
   int a=10,b=20;
   swap(a,b);
   cout<<a<<" "<< b;
   return 0;
}
void swap(int c, int d)
{
   int t;
   t=c;
   c=d;
   d=t;
}
```

output:
10 20

Function With Default Arguments

C++ allows to call a function without specifying all its arguments. In such cases, the function assigns a default value to a parameter which does not have a mathching arguments in the function call. Default values are specified when the function is declared. The complier knows from the prototype how many arguments a function uses for calling.

Example :

```
float result(int marks1, int marks2, int marks3=75);
```

a subsequent function call

```
average = result(60,70);
```

passes the value 60 to marks1, 70 to marks2 and lets the function use default value of 75 for marks3.

The function call

```
average = result(60,70,80);
```

passes the value 80 to marks3.

Inline Function

Functions save memory space because all the calls to the function cause the same code to be executed. The functions body need not be duplicated in memory. When the complier sees a function call, it normally jumps to the function. At the end of the function. it normally jumps back to the statement following the call.

While the sequence of events may save memory space, it takes some extra time. To save execution time in short functions, inline function is used. Each time there is a function call, the actual code from the function is inserted instead of a jump to the function. The inline function is used only for shorter code.

```
inline int cube(int r)
{
    return r*r*r;
}
```

Some important points to be noted

- Function is made inline by putting a word inline in the beginning.
- Inline function should be declared before main() function.
- It does not have function prototype.
- Only shorter code is used in inline function If longer code is made inline then compiler ignores the request and it will be executed as normal function.

Global Variable And Local Variable

Local Variable : a variable declared within the body of a function will be evaluated only within the function. The portion of the program in which a variable is retained in memory is known as the **scope of the variable**. The scope of the local variable is a function where it is defined. A variable may be local to function or compound statement.
Global Variable : a variable that is declared outside any function is known as a global variable. The scope of such a variable extends till the end of the program. these variables are available to all functions which follow their declaration. So it should be defined at the beginning, before any function is defined. *See Assignment Solutions*

Unary Scope Resolution Operator (::)

It is possible to declare local and global variables of the same name. C++ provides the **unary scope resolution operator (::)** to access a global variable when a local variable of the same name is in scope. A global variable can be accessed directly without the unary scope resolution operator if the name of the global variable is not the same as that of a local variable in scope.

Variables and storage Class

The storage class of a variable determines which parts of a program can access it and how long it stays in existence. The storage class can be classified as automatic register static external
Automatic variable
All variables by default are auto i.e. the declarations int a and auto int a are equivalent. Auto variables retain their scope till the end of the function in which they are defined. An automatic variable is not created until the function in which it defined is called. When the function exits and control is returned to the calling program, the variables are destroyed and their values are lost. The name automatic is used because the variables are automatically created when a function is called and automatically destroyed when it returns.

Register variable

A register declaration is an auto declaration. A register variable has all the characteristics of an auto variable. The difference is that register variable provides fast access as they are stored inside CPU registers rather than in memory.

Static variable

A static variable has the visibility of a local variable but the lifetime of an external variable. Thus it is visible only inside the function in which it is defined, but it remains in existence for the life of the program.

External variable

A large program may be written by a number of persons in different files. A variable declared global in one file will not be available to a function in another file. Such a variable, if required by functions in both the files, should be declared global in one file and at the same time declared external in the second file.