Overview for functions

Overview for functions

A function is a set of statements designed to accomplish a particular task. The advantage of using functions is that it is possible to reduce the size of a program by calling and using them at different places in the program. C++ has added many new features to functions to make them more reliable and flexible.

General Format of a Function Definition:

Functions can be define before the definition of the main() function, or they can be declared before it and define after it.

Declaring a function means listing its return type, name, and arguments. This line is called the function prototype. A function prototype tells the compiler the type of data returned by the function. It is usually defined after the preprocessing statements at the beginning of the program.

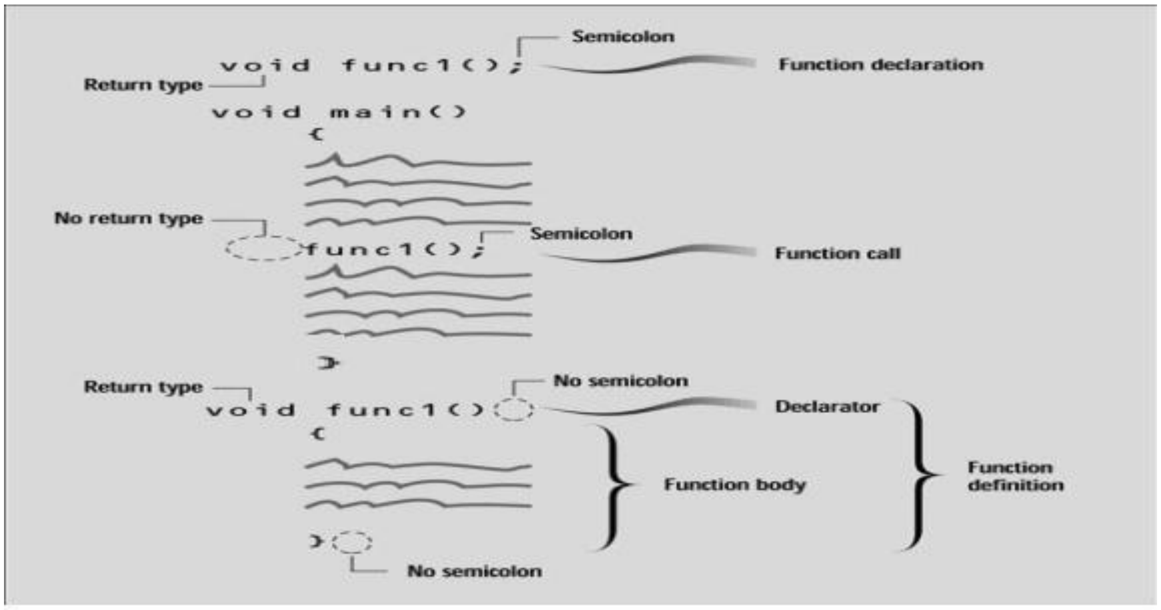


Figure: ***Function syntax***

Example 1: Write a C++ program to find the maximum value between two values using function

```
#include <iostream.h>
float max ()
{
   float a, b;        a=3.5;        b=10.25;
    float c;
    if (a > b)
           c = a;
      else
           c = b;
    return (c);
}
void main ( )
{    float k;
        k=max();
     cout <<k;
}
```

Output:-

```
10.25
```

Local and Global Variables:

The variables in general bay be classified as local or global variables.

(a) **Local variables:** Identifiers, variables and functions in a block are said to belong to a particular block or function and these identifiers are known as the local parameters or variables. Local variables are defined inside a function block or a compound statement. For example,

```
#include <iostream.h>
void sum ()
{        int a,b,s;        // a,b,s  are local variables in a function sum()
        a=3;
        b=5;
         s=a+b;
        cout<<s;
}
void main( )
{
         sum();
}
```

(b) Global variables: these are variables defined outside the main function block. These variables are referred by the same data type and by the same name through out the program in both the calling portion of the program and in the function block.

```
#include <iostream.h>
int  a,b,s;         // a,b,s are Global variables
void sum ()
{       s=a+b;
cout<<s;
}
void main( )
{       a=3;
        b=5;
        sum();
}
```

Inline Function:

C++ suppliers" programmers with the inline keyword, which can speed up programs by making very short functions execute more efficiently. Normally a function resides in a separate part of memory, and is referred to by a running program in which it is called. Inline functions save the step of retrieving the function during execution time, at the cost of a larger compiled program.

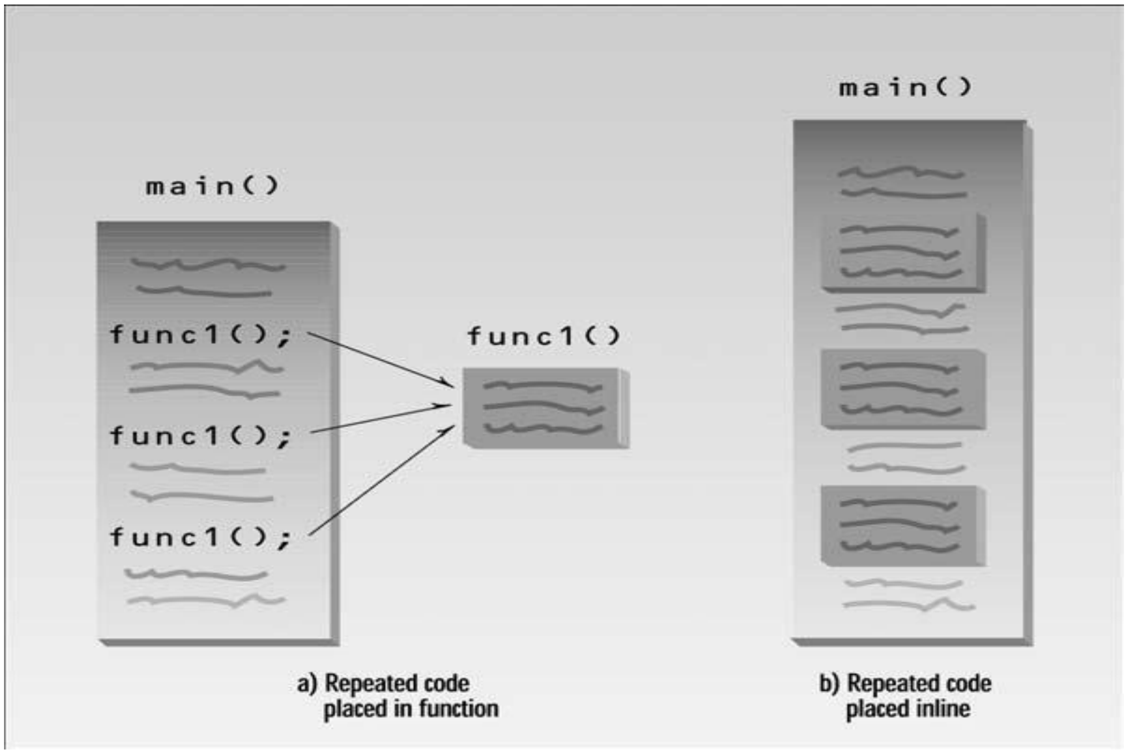

main()
func1();
func1();
func1();
func1()
main()
a) Repeated code placed in function
b) Repeated code placed inline

Figure: ***Functions versus inline code***

Example 2: Write a C++ program to find the square of a number using inline function

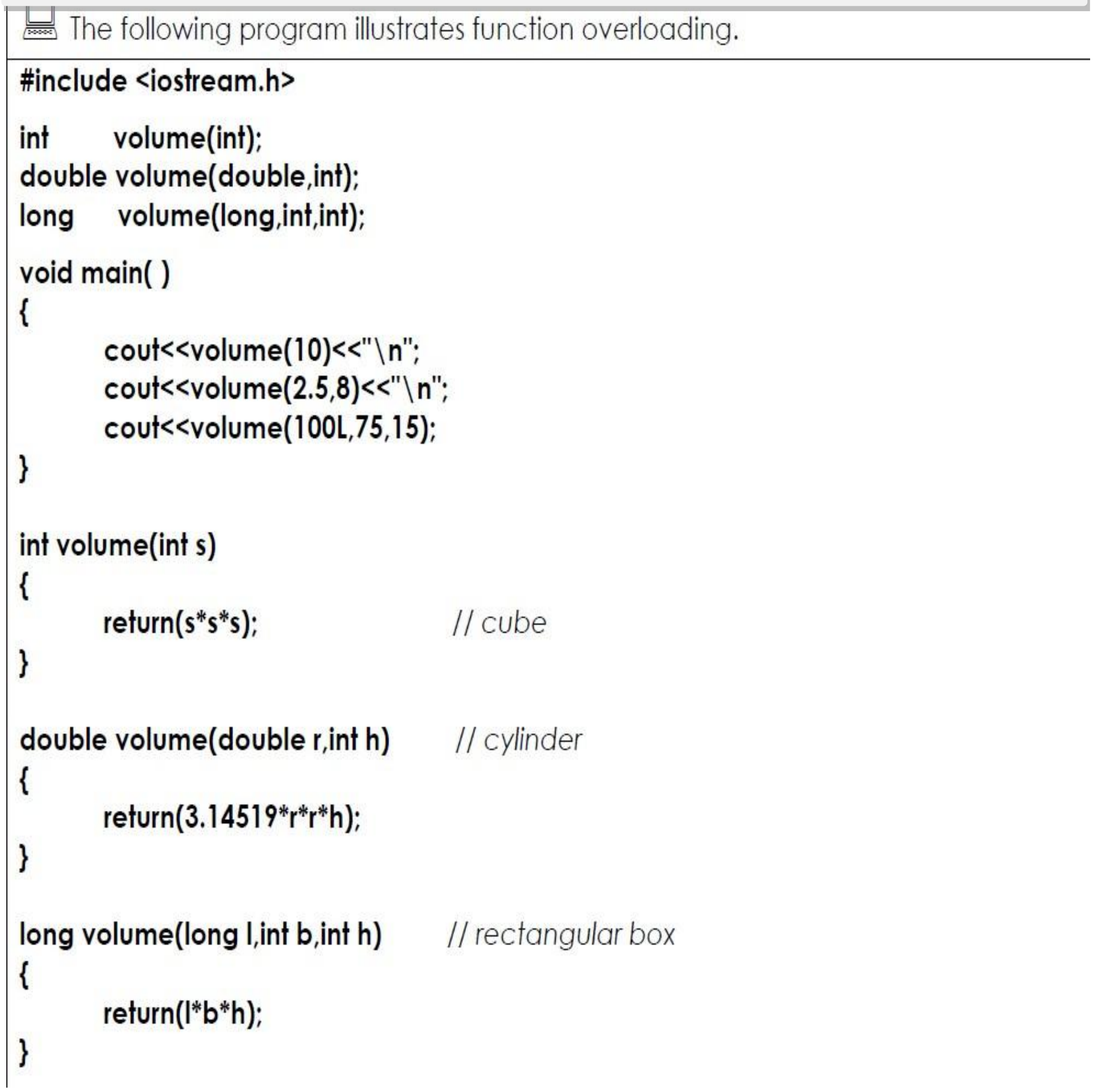
```
#include<iostream.h>
inline int square ( int y )
   {
       return ( y*y );
     }
void main( )
 {
          int m;
                m=5;
                cout<< square ( m ) ;
}
```

Function Overloading:

Overloading refers to the use the same thing for different purposes. C++ also permits overloading of functions. This means that we can use the same function name to create functions that perform a variety of different tasks.

We can design a family of functions with one function name but with different argument lists. The function would perform different operations depending on the argument list in the function call.

Example3:



The following program illustrates function overloading.

```
#include <iostream.h>

int       volume(int);
double volume(double,int);
long     volume(long,int,int);

void main( )
{
        cout<<volume(10)<<"\n";
        cout<<volume(2.5,8)<<"\n";
        cout<<volume(100L,75,15);
}

int volume(int s)
{
        return(s*s*s);                    // cube
}

double volume(double r,int h)       // cylinder
{
        return(3.14519*r*r*h);
}

long volume(long l,int b,int h)       // rectangular box
{
        return(l*b*h);
}
```

Passing Parameters:

There are two main methods for passing parameters to a program:

1- Passing by Value:

When parameters are passed by value, a copy of the parameters value is taken from the calling function and passed to the called function. The original variables inside the calling function, regardless of changes made by the function to it are parameters will not change. All the pervious examples used this method.

Example 4

Write C++ program using function to calculate the average of two numbers entered by the user in the main program:

```
#include<iostream.h>

void aver (int x1, int x2)
{
        float z;
        z = ( x1 + x2) / 2.0;
      cout<<z;
}
void main( )
{
        int   num1,num2;
        cout << "Enter 2 numbers \n";
        cin >> num1 >> num2;
        aver (num1, num2);

}
```

Output:-

```
4.5
```

Input:-

```
Enter 2 numbers
  6    3
```

2- Passing by Reference:

When parameters are passed by reference their addresses are copied to the corresponding arguments in the called function, instead of copying their values. Thus pointers are usually used in function arguments list to receive passed references.

Example 5:

The following program illustrates passing parameter by reference.

```
#include <iostream.h>
void swap(int *a,int *b);
void main( )
{
        int x=10;
        int y=15;
        cout<<"x before swapping is:"<<x<<"\n";
        cout<<"y before swapping is:"<<y<<"\n";

        swap(&x,&y);
        cout<<"x after swapping is:"<<x<<"\n";
        cout<<"y after swapping is:"<<y<<"\n";
}
void swap(int *a,int *b)
{
        int c;
        c=*a;
        *a=*b;
       *b=c;
}
```

Example 6:

Write a C++ program using functions to print the contents of an integer array of 10 elements.

```
#include <iostream.h>
int const size=10;
void pary(int y[]);
void main( )
{
        int s[size]={2,4,6,8,10,12,14,16,18,20};
        pary(s);
}
void pary(int x[])
{
        int i;
        for (i=0;i<size;i++)
            cout<<x[i]<<endl;
}
```

Example 7:

Design a function that takes the third element of an integer array defined in the main program. The function must multiply the element by 2 and return the result to the main program.

```
#include<iostream.h>
int elementedit(int);

void main( )
{
        int a[4]={2,5,3,7};
        int k;
        k=elementedit(a[2]);
        cout<<"k after change is: "<<k;
}

int elementedit(int m)
{
        m=m*2;
        return(m);
}
```

Default Arguments

As with global functions, a member function of a class may have default arguments. The same rules apply: all default arguments should be trailing arguments, and the argument should be an expression consisting of objects defined within the scope in which the class appears.

Surprisingly, a function can be called without specifying all its arguments. This won‟t work on just any function: The function declaration must provide default values for those arguments that are not specified.

Example 8: Write a simple program to represent a default argument

```
// missarg.cpp
// demonstrates missing and default arguments
#include <iostream>
using namespace std;

void repchar(char='*', int=45);     //declaration with
                                    //default arguments
int main()
   {
   repchar();                       //prints 45 asterisks
   repchar('=');                    //prints 45 equal signs
   repchar('+', 30);                //prints 30 plus signs
   return 0;
   }
//--------------------------------------------------------------
// repchar()
// displays line of characters
void repchar(char ch, int n)        //defaults supplied
   {                                //   if necessary
   for(int j=0; j<n; j++)           //loops n times
      cout << ch;                   //prints ch
   cout << endl;
   }
```

In this program the function repchar() takes two arguments. The first time it‟s called with no arguments, the second time with one, and the third time with two. Why do the first two calls work? Because the called function provides default arguments, which will be used if the calling program doesn‟t supply them. The default arguments are specified in the declaration for repchar(): **void repchar(char='*', int=45);** The default argument follows an equal sign, which is placed directly after the type name. You can also use variable names, as in **void repchar(char reptChar='*', int numberReps=45);** If one argument is missing when the function is called, it is assumed to be the last argument. The repchar() function assigns the value of the single argument to the **ch** parameter and uses the default value 45 for the n parameter. If both arguments are missing, the function assigns the default value „*‟ to **ch** and the default value 45 to n.