Functions in C++

Introduction

Developers often need the ability to break larger programs into smaller programs, which are easier to develop. The real world programs can be many thousands of lines long, without dividing the programs into sub programs, and developing such large program could be impossible and even debugging and adding new features to that create so many problems. C++ allows developers to divide their program into smaller programs called as functions. A function is a simple and well-defined sub program, which performs some specific task. Functions are used to provide modularity to the program. Developing application using functions makes it easier to understand and modify and easy to debug the program.

Advantages of using functions:

- The program can be easily understandable and easy to modify and debug the code.
- Functions can be stored in library and can be reused many number of times in the programs.
- Sometimes part of code in the program is to be used more than once at different places in the program in that context the functions can be written once and called at different places in the program which avoids repetition of code in the program.

Function definition

The function definition defines the description of a function. It gives information about what function going to do and its inputs and outputs. The function definition consists of two parts:

- A Function header
- Function body

Syntax:

```
return_type func_name (parameter declarations)
{
    variable declarations;
    statements;
```

return; or return (expression);

}

The first line in the function definition is known as the function header, after that the function of body starts in curly braces. The return_type defines the type of the value the function returns. A function can return value or it does not return a value. A function which does not return any value called as void function and 'void' should be return in the place return_type.

If no return type is mentioned by default, it will assume 'int' return type. func_name specifies the name of the function. After function name there will parameter declarations inside the parentheses, which mentions the type and name of the parameters. These parameters are also called formal parameters and are used to accept the values from a function call.

A function can have no parameters or any number of parameters. If there are no parameters then 'void' is written inside the parenthesis, which denotes that this function does not have any parameters. The body of a function is block of code, which consists of declaration of variables, statements and return statement. The variables declared inside the function are called as local variables and scope is local to that function only. They are created when the function body enters and destroyed when the function is exited.

The return statement is used for exit from the called function and returns to calling function. If there is no return statement in the function then that function is executed till the close brace is encountered. The return can be used in two ways:

- Only 'return;' – This statement is used to terminate the function without returning any value to function.
- return with expression 'return(expression) – This statement is used to terminate the function and return a value to the calling function. The expression can be constant value or any valid expression.

  Example:

  return 10;

  return (a+b);

  return a++;

Let us consider an example program in which define a function, which add two numbers, and return the sum to calling function and which will display result on the console.

```
/* Function example program addition of two numbers */

#include <iostream>
using namespace std;

int add (int a, int b )// function defination
{
        int sum=0;
        sum = a+b;
        return sum;
}

int main()
{
        int x,y,z;
        cout << "x value:";// Asking input for x
        cin >> x;
        cout << "y value:";// Asking input for y
        cin >> y;
        z = add (x, y); // function call
        cout << "The sum is:" << z << endl;
        return 0;
}
"fun1.cpp" 23L, 403C                                  1,1           All
```

Figure1: Function Definition example program

```
bala@ubuntu:~/Documents/C++$ vim fun1.cpp
bala@ubuntu:~/Documents/C++$ g++ fun1.cpp
bala@ubuntu:~/Documents/C++$ ./a.out
x value:7
y value:7
The sum is:14
bala@ubuntu:~/Documents/C++$
```

Figure2: Output of the program

There are two types of functions:

1. Library functions
2. User defined functions

Library functions

The library functions are built in functions in C++ programming. The developer can use library functions by invoking the function call directly. To use the library function we need to include corresponding header file using preprocessor directive “#include” in the starting of the

program. For example, the ‘cin’, ‘cout’ and ‘endl’ functions are available in the library ‘iostream’. Let us take one example, which illustrates the use of library functions.

```
/* library function example */
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
        float n, root;
        cout << "Enter the number:";
        cin >> n;
        root = sqrt(n);// sqrt is in built library function to calculate
                       // square root of a given number
        cout << "square root of the number is:" << root << endl;
        return 0;
}
~
~
~
```

Figure3: Library function example program

```
bala@ubuntu:~/Documents/C++$ vim lib.cpp
bala@ubuntu:~/Documents/C++$ g++ lib.cpp
bala@ubuntu:~/Documents/C++$ ./a.out
Enter the number:2
square root of the number is:1.41421
bala@ubuntu:~/Documents/C++$ ./a.out
Enter the number:4
square root of the number is:2
bala@ubuntu:~/Documents/C++$ ./a.out
Enter the number:3
square root of the number is:1.73205
bala@ubuntu:~/Documents/C++$
```

Figure4: Output of the program

In the above program we used “sqrt()” library function to calculate the square root of a given number. We included this function using “#cmath” header file in the program. Using “#include <cmath>” we can use all functions defined in cmath library.

User defined functions

C++ allows programmers to define their own functions. The programmers can define their own functions for performing any specific task. When that functions are invoked from any part of the program, it will executes the code defined in the body of function. The following figure shows the user defined function. Here the program starts begin with main () function and starts executing the statements in the program, when the control reaches to "func_name()" statement in the main () function the control of the program moves to the "int func_name()" and starts executing that function. After executing all the statements in that function the control jumps to the main program exactly after the "func_name ()" call and starts executing remaining part of the main program.

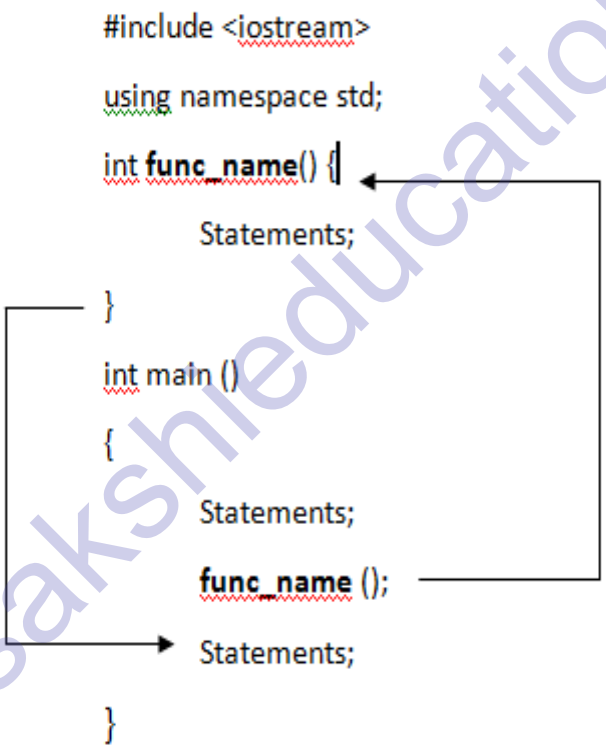

Figure5: User defined function

To use User defined functions we need to know the following points:

- Function declaration
- Functions call
- Function definition

Function declaration

Functions declaration also called function prototype. Functions declaration is used to give specific information about the function to the compiler so that it can check the function calls. The calling function needs information about called function. It can be used in two ways; if the definition of the called function is defined before the calling function then function declaration is not needed. In this case the function is defined before main () function so that the main () function knows the information about called function. Generally the main () function is defined at the starting of the program and the remaining functions are defined after main function (). In this situation function declaration is required, and which gives the information about following points to compiler:

- Name of the function
- Type of arguments and number of arguments
- Type of value returned by the function

Syntax:

return_type func_name(type_arg1, type_arg2, ….);

Example:

int add(int a, int b);

The syntax of function declaration looks like function definition except there is comma at the end of the line. The arguments in the function declaration can be used without name also, these are optional and the compiler ignores the name as shown below example.

int add (int, int);

Function definition

The function definition already explained in the above sections, which gives the information about the function. When the function is called the control point is transferred to first statement of function, starts executing the statements sequentially till the last statement reached and then the control point moves to the calling function.

Function call

To execute the function defined in the program, it needs to be invoked or called from somewhere in the program. A function is called by simply writing function name followed by the arguments list in the parentheses. The functions calls are called in the main () function. The arguments used here are actual arguments, which are passed to called function.

Syntax:

return_type func_name(arg1, arg2, arg3, …)

```
#include <iostream>
using namespace std;
int add (int, int);        // function declaration
int main ()
{
        int n1, n2, sum;

        sum = add (n1, n2) // Actual arguments //function call
        Statements;
}
int add (int x, int y) {   // formal arguments // function definition
        add = a+b;
        return add;
}
```

Figure6: Example for user defined function

Let us consider example program for user-defined function which defines two functions addition and subtraction of two numbers and returns the result of sum and difference to the calling function.

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

int addition (int, int);    //function declaration
int subtraction (int, int); //function declaration
                  //or
//int addition (int x, int y);    //function declaration
//int subtraction (int x, int y); //function declaration

int main(){
        int a = 4, b = 3, sum, diff;
        sum = addition (a, b);      // function calls for add and sub
        diff = subtraction (a, b); // a and b are actual arguments
        cout << "The addition is: " << sum << endl;
        cout << "The subtraction is: " << diff << endl;
        return 0;
}

int addition (int x, int y){   // function definition
        int sum;                // x and y are the formal arguments
        sum = x+y;
        return sum;
}

int subtraction (int x, int y){ // function definition
        int diff;
        diff = x-y;
        return diff;
}
"fun_user.cpp" 29L, 765C                              1,1           All
```

Figure7: Example program for User defined function

```
bala@ubuntu:~/Documents/C++$ vim fun_user.cpp
bala@ubuntu:~/Documents/C++$ g++ fun_user.cpp
bala@ubuntu:~/Documents/C++$ ./a.out
The addition is: 7
The subtraction is: 1
bala@ubuntu:~/Documents/C++$
```

Figure8: Output of the program

Passing arguments to function

The variables used during function call are called actual arguments and these values are initialized to other variables in the function definition are called formal arguments. In the above

example program the variables ‘a’ and ‘b’ used in function call are called actual arguments and the values of these variables are initialized to variables ‘x’ and ‘y’ in the function definition are called formal arguments.

- The number of actual arguments and formal arguments should be same.
- The type of each argument in function call should match the type of each argument in the function definition.
- The type of arguments should be any type but should match in function call and function definition.
- We can use any number of arguments or no arguments.

Inline functions

The inline functions are C++ added feature to increase the execution time of a program. The one problem with function is that every time a function is called there is a certain amount performance overhead because the CPU must store the address of the current instruction it is executing along other registers, all the function parameters must be created and assigned values and the program has to branch a new location.

The inline function is used in the program is significantly faster and this is a powerful concept used with classes. If a function is inline then the compiler places a copy of the code of that function at every place where the function is called at compiler time. The ‘inline’ keyword is used to request the compiler treat your function as an inline function.

Any change in the inline function could requires to recompile the program because the compiler would need to replace all the code once again in the program otherwise it runs with previous functionality. The problem with inline functions is expanded the code for every function call in the program, this can make your compile code bit a larger. The below example program is using inline function, in which we are finding out which is a small number in a given numbers.

```
/* Example program for inline function*/

#include <iostream>
using namespace std;

inline int min(int x, int y){

                return(x>y)?y:x;
        }

int main()
{
        cout << "min value is:" << min (5,10) << endl;
        cout << "min value is:" << min (32,3) << endl;
        return 0;
}
~
```

Figure9: Example program for Inline function

```
bala@ubuntu:~/Documents/C++$ vim min.cpp
bala@ubuntu:~/Documents/C++$ g++ min.cpp
bala@ubuntu:~/Documents/C++$ ./a.out
min value is:5
min value is:3
bala@ubuntu:~/Documents/C++$
```

Figure10: Output of the program

C++ function types

Based on the return value and arguments the functions can be classified into four types:

- Functions with no argument and no return value
- Functions with no argument but return value
- Functions with argument but no return value
- Functions with argument and return value

Functions with no argument and no return value

In the below example program the function 'func ()' is called from main () function and the 'func ()' definition is written after the main function. The 'func ()' does not have arguments

and no return type, so it cannot send any data to 'func ()' and the 'func ()' cannot return any data to main () function.

```
#include <iostream>
using namespace std;
void func (void);
int main ()
{
        Statements;
        func(void);
        Statements;
}
void func(void) {
        Statements;
}
```

Figure11: Functions with no argument and no return value

Functions with no argument but return value

These functions does not have any arguments but they can return a value to main () function.

```
#include <iostream>
using namespace std;
int func (void);
int main ()
{
        int add;
        add = func(void);
        Statements;
        return 0;
}
int func(void) {
        int sum;
        Statements;
        return sum;
}
```

Figure12: Functions with no argument but return value

Functions with argument but no return value

These functions have arguments so the calling function send arguments to called function but it does not have return value so they cannot return any value to calling function.

```
#include <iostream>
using namespace std;
void func (int, int);
int main ()
{
        int a, b;
        func(a,b);
        Statements;
        return 0;
}
void func(int x, int y) {

        Statements;
}
```

Figure13: Functions with argument but no return value

Functions with argument and return value

These types of functions have arguments and return type, so the calling function sends the arguments to the called function and the called function return some data to the calling function using return statement.

```
#include <iostream>
using namespace std;
int func (int, int );
int main ()
{
        int add;
        add = func(void);
        Statements;
        return 0;
}
int func(int x, int y) {
        int sum;
        Statements;
        return sum;
}
```

Figure14: Functions with argument and return value