Lesson 8: Functions in C++

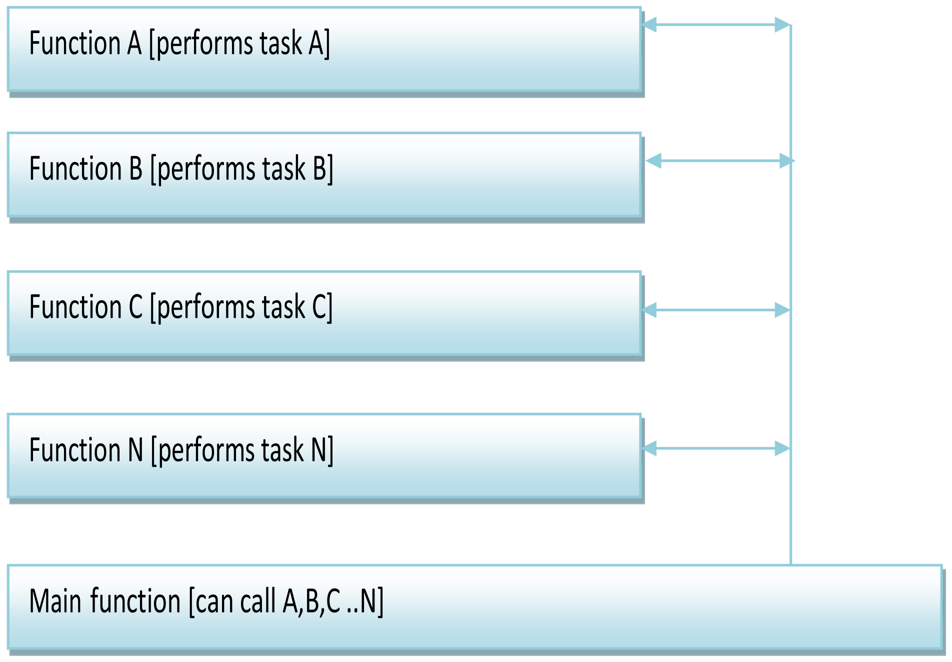
8.1. Introduction
In our previous lesson we discussed about basic structure of a c++ program, operators, variables, data types and modifiers in c++. In our lesson today, we will be learning about functions. A program can be broken down into smaller programs called modules/functions. Each of these functions performs a specific task. Breaking a program to functions reduces its complexity.

8.2. Lesson objectives
By the end of this lesson you will be able to:

- Declare and define a function
- Call a function
- Describe function overloading

8.3. Lesson outline:
This lesson is structured as follows:

8.4. Overview of functions
A function is a block of statements for performing a certain task. A given program may be broken down to several tasks where each task is accomplished by a function. Breaking a program into functions makes it easier to code and debug. Programmers can also collaborate in which case each team member can be assigned some module/function to develop. However, there are some limitations when working with functions. For instance when integrating the separate modules, errors may arise. Also challenges of maintaining consistent naming conventions.


Function A [performs task A]
Function B [performs task B]
Function C [performs task C]
Function N [performs task N]
Main function [can call A,B,C ..N]

8.5. Function declaration

Function declaration tells the compiler how the function works. It is also referred to as function prototype. The declaration does not have function body. The declaration must end with a semi colon.
The declaration consists of:

- Return type e.g. int, float, void etc
- Name or identifier e.g. sum
- Type and name of the parameters enclosed in brackets (optional).

```
int sum(int a, int b); void sum();
float sum(float a, float b, float c);
```

8.6. Function definition

Function definition involves giving the function prototype as well as the function body.

Syntax of a function definition

```
type name ( parameter1, parameter2, ...)
 {
Statement(s);
}
```

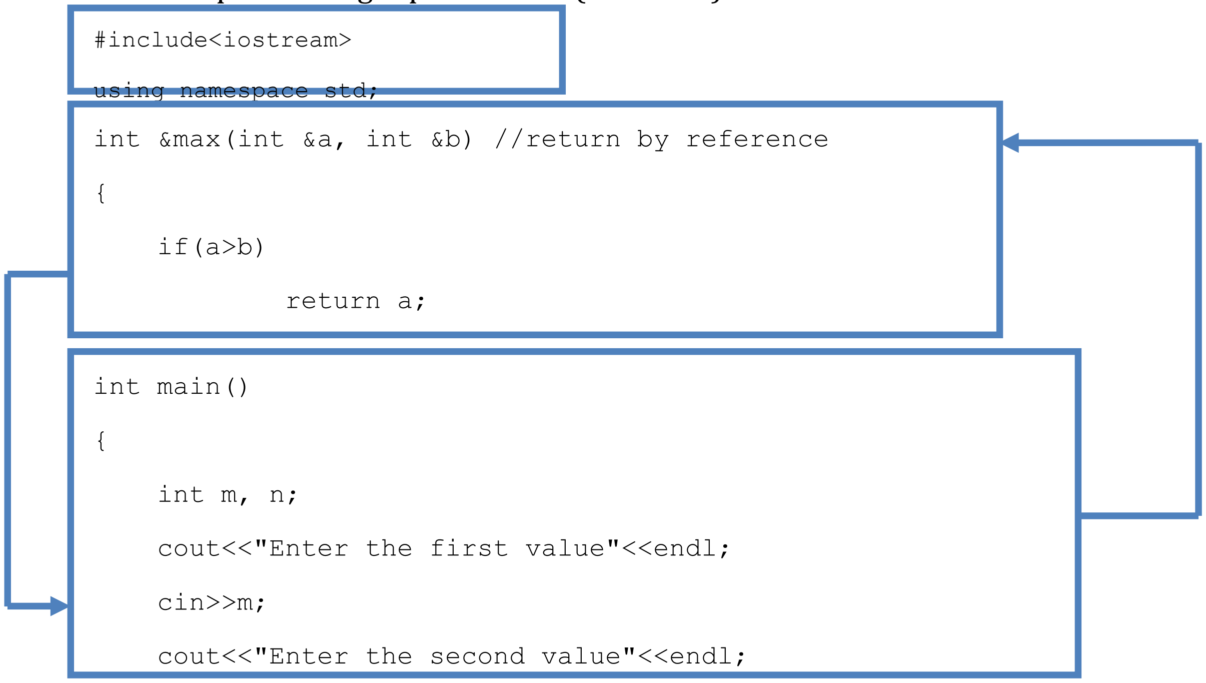
Type: This is the type of the value returned by the function.
Name/identifier: This is the identifier by which the function can be called.

Type and name of parameters: Each parameter consists of a type followed by an identifier, with each parameter being separated from the next by a comma. Each parameter looks very much like a regular variable declaration (for example: int x), and in fact acts within the function as a regular variable which is local to the function. The purpose of parameters is to allow passing arguments to the function from the location where it is called from.

Function body/Statements: This is the function's body. It is a block of statements surrounded by braces { } that specify what the function actually does.

8.7. Calling a function

Whenever you want the task to be performed you only need to call the right function and pass the right parameters (variables).



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

```
int &max(int &a, int &b) //return by reference
{
    if(a>b)
            return a;
```

```
int main()
{
    int m, n;
    cout<<"Enter the first value"<<endl;
    cin>>m;
    cout<<"Enter the second value"<<endl;
```

[a]. **Call by value**: In this technique, arguments are passed by value while calling a function in the program. This method copies the actual value of an argument into the formal parameter of the function. This mechanism is used when you don't want to change the value of passed parameters. When parameters are passed by value then functions in C create copies of the passed in variables and do required processing on these copied variables. In this case, changes made to the parameter inside the function have no effect on the argument.

[b]. **Call by reference:** The arguments are passed by address while calling a function. This method copies the address of an argument

into the formal parameter. This mechanism is used when you want a function to do the changes in passed parameters and reflect those changes back to the calling function. In this case only addresses of the variables are passed to a function so that function can work directly over the addresses. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affect the argument. One advantage of the call by reference method is that it uses pointers, so there is no doubling of the memory used by the variables (as with the copy of the call by value method).

Example: A program in c++ to swap values using functions and pass by value.

```
#include<iostream> using namespace std; void swap(int a,
int b) //parameters will be passed by value
{   int tmp;  tmp = a;  a = b;  b = tmp;  cout<<"Values after swap, first
value="<<a<<" and second value="<<b<<endl;
}
int main()
{ int m, n; cout<<"Enter first value"<<endl; cin>>m; cout<<"Enter second
value"<<endl; cin>>n; cout<<"values before swap: first value= "<<m<<" and
second value="<<n<<endl; swap(m,n);// calling swap function by value return
0;
}
```

Example: A program to swap values in c++ using a function that is called by reference.

```
#include<iostream> using namespace std; void swap(int *a, int *b)
//the formal parameters are pointers a and b to store addresses
{     int tmp;  //declare temp variable     tmp
= *a;  //assign the address of a to temp
    *a = *b;   //assign address of b to a
    *b = tmp;   //assign the value of temp to b(swap value)
cout<<"Values after swap are"<<*a<<" and"<<*b<<endl;//prints the swapped
values
}
int main()
{     int m, n;     cout<<"Enter the first value"<<endl;
cin>>m;     cout<<"Enter the second value"<<endl;
cin>>n;     cout<<"Values before swap "<<m<<" and
"<<n<<endl;     swap(&m, &n);  //  calling swap function
by reference         return 0;
```

Example: A program that uses references to swap two values.

```
#include<iostream> using namespace std; void swap(int &a, int
&b) //the formal parameters are references
{     int t;  //declare t variable     t= a;  //assign the address of a to
t     a = b;   //assign address of b to a     b = t;   //assign the value
of t to b(swap value)     cout<<"Values after swap are "<<a<<" and
"<<b<<endl;//prints the swapped values }
int main()
{     int m, n;     cout<<"Enter the first
value"<<endl;     cin>>m;     cout<<"Enter the second
value"<<endl;     cin>>n;     cout<<"Values before
swap "<<m<<" and "<<n<<endl;     swap(m, n);  //
calling swap function by reference         return 0;
}
```

8.8. Inline functions

Every time the function is called it takes a lot of time in executing a series of instructions e.g. jumping to the function, saving to registers, pushing the arguments into the stack, returning to the calling function e.t.c. For a small function, you do not need to go through all these instructions. The inline functions replace the function call with respective function code.

The syntax for inline function is:

```
inline function header
{
statement(s);
}
```

```
#include<iostream>

using namespace std;
```

```
inline int sum(int a, int b)
{
    return (a+b);
}
```

```
void main()
{
    int m, n;
    cout<<"Enter the first value"<<endl;
    cin>>m;
    cout<<"Enter the second value"<<endl;
    cin>>n;
    cout<<"The largest value is  "<<sum(m,n)<<endl;
 }
```

However, the inline function cannot be used if:

- Function returns value, if a loop, a switch or go to exists.
- Function doesn’t value, if a return statement exists Function contains static variables Inline functions are recursive.

8.10. Recursive functions

A function that calls itself. Not many programming languages support recursion. However, C++ allows a function to call itself (recursion).Caution must however, must be exercised to avoid infinite loop.
Example: Consider the program below.

```
#include<iostream>
using namespace std;
int Sum(int num)
{ int currentSum; if(num == 1) currentSum = num; else
currentSum = num + Sum(num - 1);//Sum is calling itself
return(currentSum);
}
int main()
{ int x; for(x = 1; x <= 10; ++x) cout << "When x is " << x <<"
then Sum(x) is " <<Sum(x) << endl; return 0;
}
```

8.11. Revision

[a]. Explain the main advantage of passing arguments by reference

[b]. Explain why it may be necessary to make a function inline

[c]. Explain two advantages of modularization

8.12. Summary

In this lesson you have learnt about functions in C++. In particular, you have learnt how to declare and define a function. You have also learnt how to call a function by value as well as by reference. Finally, you have learnt about inline function and recursive functions.

8.13. Suggestions for further reading

[5]. Sams teach yourself c++ in 24 hours by Jesse Liberty and Rogers Cadenhead.

[6]. Object oriented programming in c++ by Joyce Farrel