Ekaba Bisong
Programming in C++
University of Calabar

Enrichment Note #1
June 01, 2015

Sources: http://www.cplusplus.com/doc/tutorial/functions/

Enrichment: Functions

Functions

Functions allow to structure programs in segments of code to perform individual tasks. It is a black-box that receives inputs and provides an output.

The most common syntax to define a function is:

```
return-type   name   ( parameter1, parameter2, ...)
{
        // statements
}
```

Where:

- ***return-type:*** is the type of the value returned by the function, it is the data-type of the output.
- ***name***: is the identifier by which the function can be called.
- ***parameters:*** (as many as needed): 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.
- ***statements:*** is the function's body. It is a block of statements surrounded by braces ***{ }*** that specify what the function actually does.

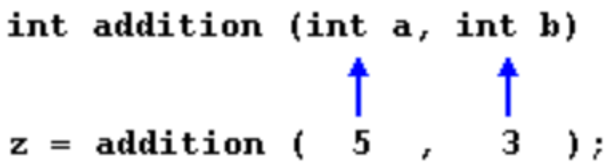
An example.

```
1  // function example
2  #include <iostream>
3  using namespace std;
4
5  int addition (int a, int b)
6  {
7    int r;
8    r=a+b;
9    return r;
10 }
11
12 int main ()
13 {
14   int z;
15   z = addition (5,3);
16   cout << "The result is " << z;
17 }
```

```
The result is 8
```

This program is divided in two functions: addition and main. Remember that no matter the order in which they are defined, a C++ program always starts by calling main. In fact, main is the only function called automatically, and the code in any other function is only executed if its function is called from main (directly or indirectly).

In the example above, main begins by declaring the variable z of type int, and right after that, it performs the first function call: it calls addition. The call to a function follows a structure very similar to its declaration. In the example above, the call to addition can be compared to its definition just a few lines earlier:



The parameters in the function declaration have a clear correspondence to the arguments passed in the function call. The call passes two values, 5 and 3, to the function; these correspond to the parameters a and b, declared for function addition.

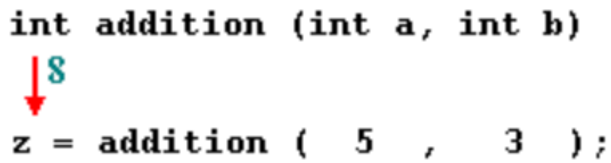
At the point at which the function is called from within main, the control is passed to function addition: here, execution of main is stopped, and will only resume once the addition function ends. At the moment of the function call, the value of both arguments (5 and 3) are copied to the local variables int a and int b within the function.

Then, inside addition, another local variable is declared (int r), and by means of the expression r=a+b, the result of a plus b is assigned to r; which, for this case, where a is 5 and b is 3, means that 8 is assigned to r.

The final statement within the function:

```
return r;
```

Ends function addition, and returns the control back to the point where the function was called; in this case: to function main. At this precise moment, the program resumes its course on main returning exactly at the same point at which it was interrupted by the call to addition. But additionally, because addition has a return type, the call is evaluated as having a value, and this value is the value specified in the return statement that ended addition: in this particular case, the value of the local variable r, which at the moment of the return statement had a value of 8.



Therefore, the call to addition is an expression with the value returned by the function, and in this case, that value, 8, is assigned to z. It is as if the entire function call (addition(5,3)) was replaced by the value it returns (i.e., 8).

Then main simply prints this value by calling:

```
cout << "The result is " << z;
```

But what if the function does not need to return a value? In this case, the type to be used is void, which is a special type to represent the absence of value. For example, a function that simply prints a message may not need to return any value:

```
// void function example
#include <iostream>
using namespace std;

void printmessage ()
{
  cout << "I'm a function!";
}

int main ()
{
  printmessage ();
}
```

```
I'm a function!
```

void can also be used in the function's parameter list to explicitly specify that the function takes no actual parameters when called. For example, printmessage could have been declared as:

```
void printmessage (void)
{
  cout << "I'm a function!";
}
```