C++ Lab 03 - C++ Functions

2.680 Unmanned Marine Vehicle Autonomy, Sensing and Communications

Ten Short CPP Labs

IAP 2024

Michael Benjamin, mikerb@mit.edu
Department of Mechanical Engineering
MIT, Cambridge MA 02139

1 Lab Three Overview and Objectives

This lab addresses

- Simple C/C++ functions
- Parameter Passing and Return Values
- Default Parameters
- Pass by Reference vs. Pass by Value
- Forward Declared Functions and Functions in Separate Files

2 Simple C/C++ Functions

C and C++ handle functions in pretty much the same way. In C++ however, it is more common to associate functions with classes/objects, which we haven't touched yet. The core ideas of function declarations, return types and parameter passing are the same in either case. Here are few good pages to get started:

- http://www.cplusplus.com/doc/tutorial/functions:
  Up to and including the section *The return value of main.*
- http://www.tutorialspoint.com/cplusplus/cpp_functions.htm
- http://www.learncpp.com/cpp-tutorial/14-a-first-look-at-functions

2.1 Exercise 1: The Simplest of Simple Functions

Write a "Hello World!" program that simply invokes a function to do the dirty work of outputting the message. Call your file `function_hello.cpp` and build it to the executable `function_hello`. When your program runs, it should be invocable from the command line with:

```
$ ./function_hello
Hello World!
```

The solution to this exercise is in Section 6.1.

2.2 Exercise 2: A Function with Parameters and Return Value

This exercise is similar to the first one in that we'll write a "Hello World!" program that invokes a function to do the dirty work of outputting the message. But this exercise, get your message from the command line and pass it as a string to the function outputting the message. Your function should also return a Boolean (type `bool`) indicating if the message was greater than 10 characters long or not. The `main()` routine should use this result to comment on the length of the message.

Call your file `function_hellocmd.cpp` and build it to the executable `function_hellocmd`. When your program runs, it should be invocable from the command line with:

```
$ ./function_hellocmd 'Hello Again World!!!'
Hello Again World!!!
That was a long message.

$ ./function_hellocmd 'Hey!'
Hey!
That was a short message.
```

The solution to this exercise is in Section 6.2.

Comments on Exercise 2:

- Note that the two quote characters are needed around the message provided on the command line. Arguments passed on the command line are delimited by white space, so in order for `"Hello"` and `"World!"` to be considered one single argument, it must be surrounded by quotes, `'Hello World!!!'`.
- The `'!'` character is also a special character to the command line shell. If you are running the `bash` shell, the single quote as above is enough to ensure the `'!'` character is treated literally. If you're using the `tcsh` or `csh` shells, you will need to take the further step of putting a back-slash in front as in:

  ```
  $ ./function_hellocmd 'Hello Again World\!\!\!'
  ```

3 C++ Functions with Default Parameters

In C++ there is the option to define a function with one or more default parameters. For example you could invoke a function to post a message with an optional prefix and suffix provided, and invoke in in a few different ways:

```
postMessage("Hello there!", "Good Evening", "Good bye");
postMessage("Hello there!", "Good Evening");
postMessage("Hello there!");
```

Default function parameters in C++ are described here:

- The section titled *Default values in parameters* in:
  http://www.cplusplus.com/doc/tutorial/functions:
- The section titled *Default Values for Parameters* in:
  http://www.tutorialspoint.com/cplusplus/cpp_functions.htm

In practice I've found that the main utility of default parameters is that one can modify a previously written function implementation that now takes a new parameter without worrying about breaking existing code that invokes the original function without the new parameter.

3.1 Exercise 3: A Function with Default Parameters

In this exercise, extend your previous program to have a message handling function with a default message of `"Hello there.  How are things?"`.

Call your file `function_hellodef.cpp` and build it to the executable `function_hellodef`. When your program runs, it should be invocable from the command line with:

```
$ ./function_hellodef
Hello there. How are things?
That was a long message.

$ ./function_hellodef 'Hey!'
Hey!
That was a short message.
```

The solution to this exercise is in Section 6.3.

4 Pass by Reference vs. Pass By Value

Parameters in C/C++ functions as described so far have passed a *copy* of the parameter to the function. You can modify the value of that parameter within the function and it will have no bearing on the local value of that variable in the code that invoked it. This is called *pass by value*. An alternative is *pass by reference* where modifications made on the variable within the function persist for the local value of that variable in the code that invoked it. This is an important concept that transcends C/C++, so if it is new to you, please read one or more of the tutorial pages links below and we'll do a couple short exercises.

- The section titled *Arguments passed by value and by reference* in:
  http://www.cplusplus.com/doc/tutorial/functions
- Or the following few pages from learncpp.com:
  http://www.learncpp.com/cpp-tutorial/71-function-parameters-and-arguments
  http://www.learncpp.com/cpp-tutorial/72-passing-arguments-by-value
  http://www.learncpp.com/cpp-tutorial/72-passing-arguments-by-value

4.1 Exercise 4: Using Pass By Reference to Have a Function Return Two Values

Functions in both C and C++ return at most one value. While true that the return value could be an object containing multiple values, often a simple second return value is all that is desired, without the need to define a new structure or class to hold two values. So a common use of pass-by-reference parameters is to create a function that "returns" a value by altering a provided parameter.

In this exercise, modify the previous version of your Hello World function to not only return a Boolean indicating whether the message was longer than ten characters, but also a Boolean indicating whether the message began with a capital letter.

Call your file `function_hellocap.cpp` and build it to the executable `function_hellocap`. When your program runs, it should be invocable from the command line with:

```
$ ./function_hellocap
Hello there. How are things?
That was a long message.
The message was properly capitalized.

$ ./function_hellocap 'hey!'
hey!
That was a short message.
The message was not properly capitalized.
```

The solution to this exercise is in Section 6.4.

Comments on Exercise 4:

- Don't forget that if you have a function parameter with a default value, it must be the last parameter. If you have N parameters with default values, they have to be the last N parameters.
- In this exercise (and solution provided), the check is made if the first character in the message is a capital letter. Can you improve on this to make sure that a message `" Hello"`, with leading white space, is reported as being capitalized?

5 Forward Declared Functions and Functions in Separate Files

In all our examples so far in this lab, the function invoked by the `main()` routine was defined prior to defining `main()`. If the order were switched, the C++ compiler would give you an error message. For example if you switched the order of `main()` and `message()` in the solution to exercise 1 (try it), you would get an error message along the lines of:

```
function_hello.cpp:14:3: error: use of undeclared identifier 'message'
  message();
  ^
1 error generated.
```

When the compiler gets to the invocation of the `message()` function, it simply knows nothing about this method and cannot discern this from a simple typo, so the compilation stops. There is a way around this by declaring the function prior to using it. Read about this topic discussed here:

- The section titled *Declaring functions* in:
  http://www.cplusplus.com/doc/tutorial/functions:

5.1 Exercise 5: Using a Forward-declared Function in a Separate File

In this exercise, modify the solution to the previous pass-by-reference exercise to move the function implementation into a separate file, leaving only the `main()` function in its own file with the function simply declared before the `main()` function.

Call your files `function_sepdef.h`, `function_sepdef.cpp` and `function_main.cpp` and build it to the executable `function_sepdef`. Note that both source code (`.cpp`) files must now be provided on the command line:

```
$ g++ -o function_sepdef function_sepdef.cpp function_main.cpp
```

As before, when your program runs, it should be invocable from the command line with:

```
$ ./function_sepdef
Hello there. How are things?
That was a long message.
The message was properly capitalized.

$ ./function_sepdef 'hey!'
Hey!
That was a short message.
The message was not properly capitalized.
```

The solution to this exercise is in Section 6.5.

6 Solutions to Exercises

6.1 Solution to Exercise 1

```
/*-------------------------------------------------------------------------*/
/*  FILE:  function_hello.cpp  (Third C++ Lab Exercise 1)                  */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/function_hello.cpp          */
/*  BUILD: g++ -o function_hello function_hello.cpp                        */
/*  RUN:   ./function_hello                                                */
/*-------------------------------------------------------------------------*/

#include <iostream>

using namespace std;

void message()
{
  cout << "Hello World!" << endl;
}

int main()
{
  message();
  return(0);
}
```

6.2 Solution to Exercise 2

```
/*-------------------------------------------------------------------------*/
/*  FILE:  function_hellocmd.cpp  (Third C++ Lab Exercise 2)               */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/function_hellocmd.cpp       */
/*  BUILD: g++ -o function_hellocmd function_hellocmd.cpp                  */
/*  RUN:   ./function_hellocmd 'Hello World!'                              */
/*-------------------------------------------------------------------------*/

#include <iostream>

using namespace std;

bool message(string msg)
{
  cout << msg << endl;

  if(msg.length() > 10)
    return(true);
  return(false);
}

int main(int argc, char **argv)
{
  if(argc != 2) {
    cout << "Usage: hellocmd MESSAGE" << endl;
    return(1);
  }

  bool msg_is_long = message(argv[1]);
  if(msg_is_long)
    cout << "That was a long message." << endl;
  else
    cout << "That was a short message." << endl;

  return(0);
}
```

6.3 Solution to Exercise 3

```
/*-------------------------------------------------------------------------*/
/*  FILE:  function_hellodef.cpp  (Third C++ Lab Exercise 3)               */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/function_hellodef.cpp       */
/*  BUILD: g++ -o function_hellodef function_hellodef.cpp                  */
/*  RUN:   ./function_hellodef 'Hello World!'                              */
/*-------------------------------------------------------------------------*/

#include <iostream>

using namespace std;

bool message(string msg="Hello there. How are things?")
{
  cout << msg << endl;

  if(msg.length() > 10)
    return(true);
  return(false);
}

int main(int argc, char **argv)
{
  if(argc > 2) {
    cout << "Usage: hellodef MESSAGE" << endl;
    return(1);
  }

  bool msg_is_long;
  if(argc == 2)
    msg_is_long = message(argv[1]);
  else
    msg_is_long = message();

  if(msg_is_long)
    cout << "That was a long message." << endl;
  else
    cout << "That was a short message." << endl;

  return(0);
}
```

6.4 Solution to Exercise 4

```
/*-------------------------------------------------------------------------*/
/*  FILE:  function_hellocap.cpp  (Third C++ Lab Exercise 4)               */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/function_hellocap.cpp       */
/*  BUILD: g++ -o function_hellocap function_hellocap.cpp                  */
/*  RUN:   ./function_hellocap 'Hello World!'                              */
/*-------------------------------------------------------------------------*/

#include <iostream>
using namespace std;

bool message(bool& capitalized, string msg="Hello there. How are things?")
{
  cout << msg << endl;

  bool long_msg        = false;
  if(msg.length() > 10)
    long_msg = true;

  capitalized = false;
  // Recall that a character in a string is represented by an ASCII
  // integer. 'A' is 65 and 'Z' is 90.
  if((msg.length() > 0) && (msg[0] >= 65) && (msg[0] <= 90))
    capitalized = true;

  return(long_msg);
}

int main(int argc, char **argv)
{
  if(argc > 2) {
    cout << "Usage: function_hellocap MESSAGE" << endl;
    return(1);
  }

  bool capitalized;
  bool msg_is_long;
  if(argc == 2)
    msg_is_long = message(capitalized, argv[1]);
  else
    msg_is_long = message(capitalized);

  if(msg_is_long)
    cout << "That was a long message." << endl;
  else
    cout << "That was a short message." << endl;

  if(capitalized)
    cout << "The message was properly capitalized." << endl;
  else
    cout << "The message was not properly capitalized." << endl;

  return(0);
}
```

Comments on the Solution to Exercise 4:

- When a Boolean expression is evaluated, C++ will only evaluate as much as it needs to. For example, in the below expression:

```
if( (1 > 2) && (my_string == "pears"))
```

The string comparison will never be made since the first condition failed and that is enough to know the whole conjunction will fail. Likewise in

```
if( (2 > 1) || (my_string == "pears"))
```

the success of the first term ensures the entire disjunction is successful, so the string comparison is again not made.

In our example, we can safely assume we will not be accessing the string at an index out of bounds with expression `msg[0]` since the first term in that conjunction ensures we have a non-empty string before evaluating `msg[0]`.

6.5 Solution to Exercise 5

```
/*-------------------------------------------------------------------------*/
/*  FILE:  function_main.cpp  (Third C++ Lab Exercise 5)                   */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/function_main.cpp           */
/*  BUILD: g++ -o function_sepdef function_sepdef.cpp function_main.cpp    */
/*  RUN:   ./function_sepdef 'Hello World!'                                */
/*-------------------------------------------------------------------------*/

#include <iostream>
#include "function_sepdef.h"

using namespace std;

int main(int argc, char **argv)
{
  if(argc > 2) {
    cout << "Usage: function_sepdef MESSAGE" << endl;
    return(1);
  }

  bool capitalized;
  bool msg_is_long;
  if(argc == 2)
    msg_is_long = message(capitalized, argv[1]);
  else
    msg_is_long = message(capitalized);

  if(msg_is_long)
    cout << "That was a long message." << endl;
  else
    cout << "That was a short message." << endl;

  if(capitalized)
    cout << "The message was properly capitalized." << endl;
  else
    cout << "The message was not properly capitalized." << endl;

  return(0);
}
```

```
/*-------------------------------------------------------------------------*/
/*  FILE:  function_sepdef.h    (Third C++ Lab Exercise 5)                 */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/function_sepdef.cpp         */
/*  BUILD: g++ -o function_sepdef function_sepdef.cpp function_main.cpp */
/*  RUN:   ./function_sepdef 'Hello World!'                                */
/*-------------------------------------------------------------------------*/

#ifndef SEPDEF_MESSAGE
#define SEPDEF_MESSAGE

#include <string>

bool message(bool& capitalized, std::string msg="Hello there. How are things?");

#endif
```

```
/*-------------------------------------------------------------------------*/
/*  FILE:  function_sepdef.cpp  (Third C++ Lab Exercise 5)                 */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/function_sepdef.cpp         */
/*  BUILD: g++ -o function_sepdef function_sepdef.cpp function_main.cpp */
/*  RUN:   ./function_sepdef 'Hello World!'                                */
/*-------------------------------------------------------------------------*/

#include <iostream>
#include "function_sepdef.h"

using namespace std;

bool message(bool& capitalized, string msg)
{
  cout << msg << endl;

  bool long_msg = false;
  if(msg.length() > 10)
    long_msg = true;

  capitalized = false;

  // Recall that a character in a string is represented by an ASCII
  // integer. 'A' is 65 and 'Z' is 90.
  if((msg.length() > 0) && (msg[0] >= 65) && (msg[0] <= 90))
    capitalized = true;

  return(long_msg);
}
```

Comments on the Solution to Exercise 5:

- This may be our first example of creating a file such as `function_sepdef.h` that we include from one of our other files. In this case the file is included from both `function_sepdef.cpp` and `function_main.cpp`.
- The three pre-processor lines in `function_sepdef.h` are there to prevent an infinite cycle of includes, e.g., file A includes file B which include file A and so on.

- It may be a good time to dig into the topic of the C/C++ pre-processor, at the links below. The pre-processor runs prior to the compiler. So it gets first dibs at defining things or blocking out code altogether. My favorite trick is the use of the `#if 0` to temporarily comment out or disable a whole bunch of code. It's much quicker than using comments. here's an example:

```
#if 0
 // Some code we want the compiler to ignore
 if(argc > 2) {
   cout << "Usage: function_sepdef MESSAGE" << endl;
   return(1);
 }
#endif
```

Further readingon the C preprocessor:

- http://www.learncpp.com/cpp-tutorial/110-a-first-look-at-the-preprocessor
- http://www.cplusplus.com/doc/tutorial/preprocessor
- http://www.cprogramming.com/tutorial/cpreprocessor.html