C++ Lab 01 - Basic Program Structure

Ten Short CPP Labs

IAP 2024

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

1 Introduction to the Ten Short C++ Labs

The MIT 2.680 *Ten Short C++ Labs* is a short tutorial sequence on C++ that leverages the existing wealth of on-line material for self-teaching C++. It may be fair and reasonable at MIT to ask students to teach themselves C++. After all, there really is a ton of material available online and in hard-print or e-books. The goal of the *Ten Short Labs* is to offer a reasonable path through all that material in a way that (a) sets up a student well for taking MIT 2.680 where C++ is the primary language, and (b) assumes that the reader is working with about a 3-4 week window with about 10-20 hours per week.

1.1 Why this Tutorial and Assumptions about Our Readers

In any C++ tutorial there are virtually unlimited ways to provide code examples and pose challenge exercises. In the *Ten Short Labs* the goal is to provide a good C++ guided tour regardless of what the user wishes to do with this skill. But we have a further goal of dual-using the exercises to be also relevant to later MIT 2.680 labs to make those labs easier.

This tutorial is also tailored a bit to fit some of the assumptions we make about our readers, who are often engineering majors with widely varying computer science backgrounds. Some of these assumptions are:

- We assume the reader has some kind of programming experience, and that this isn't the first time you have seen a for-loop. But it may be the first time you have seen a for-loop in C++. If it *is* the first time you have seen a for-loop or have programmed at all, you may just have to lean on the on-line material a bit more heavily, and this lab sequence may take at least twice as long.
- We assume some users may be new to the Linux command line environment. Tips on the command line will be included throughout the tutorial.
- We assume some users are new to using a text editor like Emacs or vi. For these labs we endorse Emacs, but you are free to use any text editor you like.
- We don't encourage (at least initially) the use of a graphical IDE like XCode, Eclipse, or other powerful integrated development environments. This is because we assume our readers will soon need to remotely log into a robot via a terminal shell and will need to know how to make edits to code with a text editor like Emacs or Vi.

1.2 C++ Resources Used in the Ten Short Labs Sequence

These *Ten Short Labs* will rely heavily on external sources found at the below two web sites. They provide pretty good stand-alone tutorials and we will be picking and choosing reading topics from them. We also list a few books worth considering as stand-alone tutorials. Lately I'm fond of any material that has an e-book or kindle version to have an electronic reference independent of a web connection.
Web sites:

- The www.cplusplus.com web site has a lot of resources. In particular there are a number of good tutorials:
  `http://www.cplusplus.com/doc/tutorial`

- The www.learncpp.com is also a great website with a tutorial for learning C++:
  http://www.learncpp.com
- The tutorialspoint.com web site is pretty thorough and aimed at a beginner with some programming experience:
  http://www.tutorialspoint.com/cplusplus/index.htm

Text books:

- The closest to a standard, comprehensive treatment is *The C++ Programming Language (4th Edition)*, Bjarne Stroustrup, Addison-Wesley, 2013. This edition also has kindle version available.
- Another fairly recent and gentle introduction, with a kindle version is *Jump into C++*, Alex Allain, published by cprograming.com in 2013.
- A gentler text book I also recommend is *Practical C++ Programming*, Steve Oualline, O'Reilly Publishers, 2003. This book is available on Amazon with used copies typically under ten dollars.

2 Lab One Overview and Objectives

This lab will be incredibly short if you already are familiar with the command line and a text editor such as Emacs or Vi. If you are not familiar with either the command line or Emacs, you will have more self-learning to do (and we have a few help pages of our own to point you to so you won't have to dive down a deep Emacs/Command-line rat-hole). The good news is that you will only need a few skills in each tool to get the job done in this lab. Also, if you are not familiar with the command line or text editors, the great news is that you will be augmenting your skills in two really important and powerful areas in this lab sequence!!

If you are fairly familiar with the command line and comfortable with a text editor on your machine, you can safely skip to Section 5.

- Introduction to Text Editors (Emacs in our case)
- Introduction to the Command Line
- Basic Structure of a C++ Program
- Building and Running a C++ Program from the Command Line

3 Lab Preparation - Software Required for the Ten Short Labs

You will need some (easy to find and free) software to do this lab sequence, which we will discuss below. But first, a word about machines or operating systems. This lab sequence assumes you are either running GNU/Linux or OSX. We have links to help-pages for getting started in either case. We are assuming our readers will before long be wanting to operate on a robotic platform, and this is just about always GNU/Linux in labs at MIT, especially marine robotics. The OSX environment is essentially Linux under the hood and is also a suitable platform for these labs and development in general.

Notably missing in this list is Windows, and Windows is still to-date the most ubiquitous operating system in the world. But not in robotics (I'm pretty sure). So the assumption of Linux or OSX reflects the assumptions of this lab sequence - that you will be soon doing all this stuff on robotic platforms in or around MIT, and this means Linux. So if you were planning on doing these labs on a Windows machine, you can either run a Linux virtual machine in Windows, or get your hands on a Linux machine or a Mac laptop. If you're an incoming MIT 2.680 student, we can help you with the latter option - come see us.

Back to the software - this lab will ask you to compose C++ programs in a text editor, built and run from the command line. So you will need to make sure the following are installed and findable on your computer:

- A terminal application
- C++ (or more specifically the C++ compiler)
- A text editor such as Emacs, vi or some ability to edit text files.

3.1 Access to the Command Line with the Terminal Application

In both Ubuntu and OSX, the terminal application is simply called `Terminal`. In Ubuntu it can be found by searching the Applications folder using the Dash utility, which is usually the launchable via icon on the top-left in a vanilla Ubuntu install. In OSX the Terminal app can be found by opening the Finder utility and clicking on the Applications folder on the left. The Terminal icons are pretty similar and look like the below for Linux and OSX respectively.

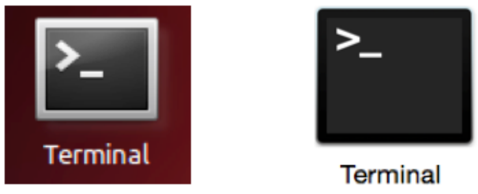

Figure 1: The icons for the Terminal Application in GNU/Linux and OSX.

In both Linux and OSX you will want to ensure this application is at your fingertips always. Typically one drags the icon from the Applications folder to the Dock of applications permanently residing on the edge of your screen. Make sure you do this step before moving on. If you're new to both Linux and OSX, you may have to investigate the setup of your dock in your environment. In Ubuntu the term to search is "Launcher". In OSX the same is called the "Dock".

For OSX, see the following help page regarding the Terminal and the Dock:
`http://oceanai.mit.edu/ivpman/help/osx_terminal`

Once you have the Terminal application going, it's not too early to start exploring the command line if it's new to you. In particular, have a look at the first few topics in the "Help with the Command Line" section of the help pages here:
`http://oceanai.mit.edu/ivpman/help`

3.2 Access to the C++ Compiler

The C++ compiler is the program that will convert (compile) your programs into an executable program, and is called `g++` in both Linux and OSX. If you are running Ubuntu you very likely have this installed automatically right from the get-go. In OSX you may need a few more steps. In either case you can verify that it is installed by typing the following from the command line:

```
$ which g++
/usr/bin/g++
```

In the above, don't type the dollar-sign. This just represents the shell prompt you likely see in your terminal window. The shell is the interactive program running in your terminal. By typing `which g++` you are asking your shell where it thinks the `g++` program is located. If it replies with a non-empty answer like `/usr/bin/g++`, that means it's installed an runnable from your shell. If it replies with no answer, you need to install it. As mentioned above, if you're running Linux, it is very unlikely it's not already installed, so we're not going to address how to fix this. You'll need to get some external help.

In the case of OSX, if `g++` is not installed you likely need install XCode and any optional command-line tools. Some tips on that can be found here:
http://oceanai.mit.edu/ivpman/help/osx_get_xcode

3.3 Access to the Emacs Text Editor

We assume and encourage the use of the Emacs text editor unless you already have an allegiance to another suitably competent text editor. To determine if this is already installed, we again use the `which` command:

```
$ which emacs
/usr/bin/emacs
```

If you get a reply like the above, you're all set. Emacs is not automatically installed in typical GNU/Linux distros such as Ubuntu, and not in OSX either. For help on getting Emacs in Linux, see:
http://oceanai.mit.edu/ivpman/help/emacs_get_for_gnulinux

For help in getting Emacs in OSX, see:
http://oceanai.mit.edu/ivpman/help/emacs_get_for_osx

4 A Few Things Before Diving into C++

If you're new to Emacs and/or the command-line, knowing a few basics will come in handy in making your first C++ program. Below covers the absolute minimum to get started with making your first C++ program.

Emacs text editor help: http://oceanai.mit.edu/ivpman/help/emacs_survival

- Open and close a file

- Saving changes
- Undoing mistakes
- Cut and Paste

Command line help: http://oceanai.mit.edu/ivpman/help/cmdline_ls_pwd

- Knowing what directory you are in (pwd command)
- Changing directories (cd command)
- Seeing the contents of a directory (ls command)

More command line help: http://oceanai.mit.edu/ivpman/help/cmdline_mkdir_rmdir

- Making a directory or folder (the `mkdir` command)
- Removing a directory or folder (the `rmdir` command)
- Removing files or folders recursively (the `rm` command)

5 Exercise 1: A Simple First Program

The goal of this first exercise is to build a ”Hello World” program. To do this you will need to:

- Read the page: http://www.cplusplus.com/doc/tutorial/program_structure

- Use your new-found skills in Emacs to compose this program in a file called `hello.cpp`. Start by creating a new file with Emacs with the following command:

  ```
  $ emacs hello.cpp
  ```

- Build the program from the command line per the instructions below.

Please read through the full contents of the tutorial page above. It basically gives you the program of this exercise with step by step explanation of what is going on in C++ syntax. Take the time to get your initial comfort in Emacs in building this program.
To build the program, we will do this on the command line. After you are done composing the file, save the file and quit Emacs. Confirm file is present:

```
$ ls
hello.cpp
```

You can even confirm the contents of the file by using the `cat` command, which just dumps the contents of a given file to the terminal:

```
$ cat hello.cpp
#include <iostream>

using namespace std;

int main ()
{
  cout << "Hello World!" << endl;
  return 0;
}
```

Now build the program into an executable. On the command line:

```
$ g++ hello.cpp
$ ls
a.out hello.cpp
```

In this case we have only one source (`.cpp`) file. In later cases where the source code comprising a program is contained in several files, all files must be given as arguments on the command line. This should generate a file called `a.out` as seen above when you invoke the `ls` command after building. This is the default name given to a C++ program if we don't otherwise specify a name we want instead. Now you can verify that it runs by running it:

```
$ ./a.out
Hello World
```

NOTE: For those new to the command-line and shell paths, the `"./"` in the above invocation is important. It tells your shell where to look for the executable `a.out`. Normally the shell looks only in the set of directories in its *path*. By default, the present working directory is not in the shell's path. The notation `"./"` is shorthand for the present working directory. The notation `"../"` refers to the directory above the present directory. Try running `"ls -a"` and you will see that both `"./"` and `"../"` are listed.

One more step:
Rather than building the default (but somewhat cryptic) `a.out` executable, try building it to have a more meaningful name, using the `g++` command line argument `-o`. First remove the file `a.out`:

```
$ rm a.out
$ g++ -o hello hello.cpp
$ ls
hello hello.cpp
```

Now you should be able to run the program by:

```
$ ./hello
Hello World
```

6 Lab Summary and Further Tips

Here are a few links to the `learncpp.com` website that may be worth exploring:

- If you are new to programming in general, or would like to know a bit about the difference between a compiled language and interpreted language, see:
  `http://www.learncpp.com/cpp-tutorial/02-introduction-to-programming-languages/`

- If you'd like a bit of historical context of C++ vs C and prior languages, see:
  `http://www.learncpp.com/cpp-tutorial/03-introduction-to-cc`