Computer Science 62
Lab 10

Wednesday, April 8, 2015

Our goal today is to get you started with C++, particularly its input and output.

We will be writing a few short C++ programs. There are compiled examples of the programs in the course directory so that you can see how they work. Just type the following command, with a digit between 1 and 4 substituted for the question mark, in a terminal window.

```
/common/cs/cs062/labs/lab10/lab10_?
```

Before lab, take a look at the C++ references posted on the course "Documentation and Handouts" page. Spend some time on these websites familiarizing yourself with the different C++ libraries and classes. You may not understand everything but that's okay.

Getting started

Begin by creating a working directory for this laboratory, inside your `cs062` directory, and open Aquamacs to edit your source files.

Program 1: Your first C++ program

The first part of our exercise is to familiarize you with terminal input and output using the C++ "streams" model. (Think of output as a "stream" of characters flowing out of the program.) The streams associated with the terminal are `cin` and `cout`. They are analogous to Java's `System.in` and `System.out`.

You can send information to the terminal by "shifting it" with the shift operator `<<` onto the output stream.

```
cout << "Hello, world!" << endl;
```

Shifting is a new notation for a familiar operation – moving data onto an output stream. As you see, one can shift several items in succession. You may include any number of shifts in fact. The symbol `endl` represents the end of line character.

Start by writing a short program `lab10_1.cpp` that prints two lines (with whatever content you like) to the terminal. Here is a skeleton for the file you will create in Aquamacs:

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

int main () {

   // output commands go here

   return 0;
}
```

Here are the commands to compile and run your code in a terminal window. Note you must be in the same directory as your cpp file when executing these commands.

```
g++ -o lab10_1 lab10_1.cpp
./lab10_1
```

`g++` is the Gnu C++ compiler. The `-o` flag tells the compiler to write the compiled version of your code to the executable file `lab10_1`. The last argument `lab10_1.cpp` is the file to be compiled. See the appendix for more variants of `g++`.

After using `g++` to compile your code, you can run `lab10_1` by simply referring to it using the command `./lab10_1`.

Program 2: Reading input

We have seen that output is a stream of characters flowing out of a program. Analogously, input is a stream of characters flowing *into* a program. The stream `cin` (analogous to `System.in`) consists of the characters typed on the keyboard. There is a shift operator `>>` that moves data from the input stream to objects inside your program, but we will not use it today. (The streams library likes to ignore white space, and we want to include it.) Instead, we will use the `getline` function to read an entire line. The following command copies a line of input characters from the first parameter, which should be a stream, into the second parameter, which should be a string:

```
getline(cin, str);
```

Write a program `lab10_2.cpp` that echoes lines typed on the keyboard. The program will consist of a file containing a single function `main` as in the skeleton above.

Your program will contain a `while` loop that reads in what is typed on the keyboard and then prints it back out for the user to see. You can signal end-of-input at the keyboard by typing ctrl-D. You can test for end-of-input in your program by testing the value of `cin.eof()`.

You will need to add a line at the top of your program to include the `string` package. Notice that the name of the C++ class `string` begins with a lowercase letter.

```
#include <string>
```

Problem 3: File I/O

The next step is to learn about input and output with files. Include the `fstream` package at the top of your program.

The data structure for an output file is of type `ofstream`. It is created with a constructor whose argument is the name of the file.

```
ofstream outfile("somefile.txt");
```

You can then shift data into an output file variable in the same way as `cout`. When you are finished with a file, close it.

```
outfile.close();
```

The data structure for an input file is of type `ifstream`. It is created and closed in the same way as an `ofstream` object. The functions `getline` and `eof` work just as they do with `cin`.

Write a file copying program `lab10_3.cpp`. It should copy one line at a time from an input file to an output file. For the purposes of this exercise, the file names may be hard-coded into the program.

Be sure that the output file is *identical* to the input file. Use the `diff` command to detect differences and use the `ls -l` command to compare the file sizes. You can also use the `less` command to open up and look at a file. To exit from `less`, simply type `q`.

Problem 4: STL

The last component of today's laboratory is to use one of the C++ classes from the STL. We will create a simple stack of strings and use it to echo lines from the terminal—in reverse order.
Read the documentation for `stack` in the C++ Library Reference. Note particularly the use of `top` and `pop`. They are *different* from those we discussed in class.

For any of the classes in the STL library you will need an appropriate "include" command at the beginning of the file, that includes the class. For example,

```
#include <stack>
```

for the stack class.

Write a program `lab10_4.cpp` to read lines from the terminal and push them on a stack until the user enters the end-of-input command, then pop them off the stack one at a time and print them.

Appendix

- **g++ commands**

  Often, you have a `Makefile` which encapsulates these commands, and all you need to do is type `make`. Today, we will use the commands directly.

  The process normally takes two steps. The `.cpp` file is the one you create in Aquamacs. The `.o` file is the object file containing the compiled code; it is created with the following command.

  ```
  g++ -O -c filename.cpp
  ```

  To compile `.o` files into an executable program:

  ```
  g++ -O -o programname fileA.o fileB.o ...
  ```

  Here is a shortcut when there is only one file.

  ```
  g++ -O -o programname filename.cpp
  ```

  The `-O` switch is optional; it turns on optimization. The letter is an uppercase "Oh," not a zero.

- **strings**

  The `string` package in C++ is very much like Java's `String` class. You can concatenate strings with the `+` operator, and you can use string literals (inside double quotation marks).

  String indices start at 0, and there are `substr` and `length` methods. If `st` is of type `string` of length at least 3, here is how we would pick out the third character and the last three characters.

  ```
  char third = st[2];
  string lastThree = st.substr(st.length() - 3, 3);
  ```

  C++ strings are *not* the same as the primitive C strings. The latter are simply arrays of characters. There are some functions, like the `fstream` constructors, that require C strings. In those cases, you will have to use the `c_str` method to convert from a C++ string.

  ```
  ofstream outfile(filename.c_str());
  ```