

Computer Science II — Spring 2008

Lab 1 — Getting Started

Overview

Three points are associated with each lab. These are “checkpoints” for completion of work. When you have completed each checkpoint, raise your hand and one of the TAs will quickly check your work.

Organization and Rules

Here are a few significant organizational issues and rules about labs. The rules will be enforced in all labs.

- No IM, no email, no network! With the exception of downloading lab files provided by the instructor at the start of lab, you are not allowed to use the network at all, in any lab. Unplug your network connection, and disable your wireless ethernet connection. Anyone caught using the network during lab will be given an immediate 0 for that lab. There will be no exceptions.
- Get to know the other students in the lab. Introduce yourself to your neighbors. You may ask your fellow students questions about the lab. This will help reduce the burden on the TAs and will reduce your waiting time in lab. Even though students may give and receive help, **each student must produce his/her own solution.**
- Part of earning a checkpoint for a lab may involve answering a short question that the TA asks you. If you have done the checkpoint and understood it, you should have no trouble earning this credit. If you have relied on help from other students too much, you may find the question hard to answer.

Checkpoint 1

There are three goals in this checkpoint:

- Getting started with the C++ development environment you intend to use for all the labs and homework in this class. As discussed during Lecture 1, you may use whatever operating system, compiler, editor, and general environment that you choose for code development.
- For those of you who do not use the cygwin environment and g++, learning to compile and run your programs using cygwin and g++ before you submit your homework.
- Practicing submission to the homework website. Remember, we will grade your code using the g++ 4.0 compiler.

Let's get started:

1. **Create a directory (a.k.a. “folder”)** on your laptop to hold CS II files. Create a sub-directory to hold CS II labs. And finally, create a sub-directory named `lab1`. Please make sure to save your work frequently and periodically back-up all of your data.

2. Using a web browser, **copy the following file** to your lab1 directory:

```
http://www.cs.rpi.edu/academics/courses/spring08/cs2/labs/lab1_intro/julian.cpp
```

Compiling Using g++ on Linux, FreeBSD, or Windows/Cygwin

Even if you intend to use Visual Studio to write your programs, you should learn to compile them using the g++ compiler. This will prevent problems when submitting your programs.

1. **Open a shell/terminal/command prompt window.** If you have never used cygwin before, you should be able to find it in your list of programs. Within the cygwin directory, you want the **bash shell**. **If you are on a Windows machine and if you can not find cygwin, please contact a lab TA immediately.**
2. Change directories into your new lab1 directory. Use the command

```
cd c:
```

to get to the top level of your C drive. Use

```
ls
```

to list the contents of the directories. Then you can use **cd** to move down in these directories. In doing so, remember that directory names are separated by '/' and when you have a space in the name of the directory, then **bash** requires that you precede the blank with a '\'. Thus, on my windows machine, I type

```
cd Documents\ and\ Settings/Stewart/cs2/lab1
```

in order to get to the directory containing my code.

3. Now you are ready to attempt to **compile/build the program** for this lab (or other simple one file projects) by typing:

```
g++ julian.cpp -o julian.exe
```

4. If you will be using g++ on a regular basis for later labs and assignments, you may eventually want to learn to use a *Makefile*.
5. Note: To check which version of gcc is installed on your machine, type:

```
g++ -v
```

If you are not using g++ 4.0 you *may* notice slight differences between your compiler and the version on the homework submission server when we get to advanced topics.

When Using the Microsoft Visual C++ Development Environment:

The following instructions show you how to get started using the Visual Studio development environment.

1. **Start up Developer Studio.** Spend some time exploring the user interface, especially the online help. Help can be easily accessed through the “Help/Search” menu. Also notice that if you leave the mouse over an icon for a second or two, a “tool tip” will pop up telling you what the icon does.

Work within Developer Studio is organized around projects and solutions. Projects are collections of files used to create an executable program; a solution is a collection of one or more projects. Browse the online help on solutions and projects.

2. **Create a workspace.** Use the **File->New->Project** menu and click the **Visual C++ Projects** folder. There are multiple kinds of projects, but you only need to worry about the one named: **Win32 Project**. Make this choice, select the folder you created for your CS II labs as the Location, and then give your project a name such as Lab 1 (as you type, Visual Studio will confirm what the actual path of your project will be on your computer, which will make finding the executable program much easier). Click OK.

You will then see a very short wizard dialog that makes sure the project that is being created fits all of your needs — click on Applications Settings and change the program from a Windows Application to a Console Application using the row of check boxes. You should also check the Empty Project option to make sure that you are starting from a completely clean slate. Click OK.

3. **Add your file to the project.** Use the **Project->Add Existing Item** menu and browse to the location where you made your local copies of the lab file (**frame.cpp**). Select the file that you want to add (note you can select multiple files by holding down the Ctrl key while clicking) and click OK. Then go to the Solution Explorer window (if it does not appear on the screen, you can open it from View menu) of the workspace, click the “+” next to your project’s Source Files folder, and you should see the file listed.
4. Attempt to **build the project.** Go to the Build menu and select **Build Lab 1**. This is the Developer Studio term for **compiling** and **linking**.

The Rest of Checkpoint 1

The process of *compiling* a program translates the high-level C++ code into machine-level, “object” code. The compiler itself is a complicated program and is the most important part of any programming environment. The *linking* step gathers the object code from your program together with other code (“libraries”) that your program needs which has already been compiled (“pre-compiled”) and stored elsewhere. For your program this is just the standard library, which includes code for i/o streams and strings. Once the linking step is complete, you have an executable program.

We have introduced a number of errors into this program so that it will *not* compile correctly to produce an executable program. If you are using g++, the compile and link errors will be displayed in the terminal/shell/command-prompt. Within Visual Studio,

these errors are directed to the status pane along the bottom of your screen. If you double click on an error, the status pane will scroll to the error, and the position of the cursor in the editing pane will move to the file and line number where the error occurred.

“Submit” the buggy version of the lab code to the homework server: Go to the course webpage, click on the “Homework” link and follow the instructions under the “Electronic Submission” section (substitute `lab1` for `hw1`). After submitting the buggy code you should receive confirmation of your submission and be notified of the compile-time errors in the program.

To complete Checkpoint 1: Show one of the TAs

1. the compiler errors that you obtained in your chosen development environment on your machine,
2. if your chosen environment is not `g++`, show the errors you obtain using `g++`, *and*
3. and show the response from the homework submission server.

Checkpoint 2

The compiler errors we have introduced are pretty simple to fix. Please do so, and then compile the program. Once you have removed all of the errors, you are ready to execute the program. If you are using `g++`, you can run the program by typing:

```
julian.exe
```

Within Visual Studio you can do this by clicking on the **Debug** menu and selecting the “!” option. This will produce a new pane on which you will have to type the input to the program.

“Re-submit” the fixed version of the lab code to the homework server: Assuming your fixes are cross-platform compatible, the re-submission should successfully compile and run without error. Save this submission result, you will need to show it to a TA to complete the checkpoint. *You will not need the network for the remainder of this lab. Please disable your internet connection now.*

For the rest of this lab we are going to review more about arrays and the logic of manipulating them. We will also practice a bit with functions and with input and output.

Modify the main program so that it defines two arrays that hold 10 integer values each. One of these arrays should store months and the other should store days. (Assume the year is 2008.) Write a `for` loop that reads 10 month / day combinations into these two arrays. Create a third array that holds Julian days. Now write another `for` loop that computes the Julian day from the month / day combinations and stores it in the array. Finally, write a `for` loop that outputs the Julian days.

To complete Checkpoint 2: show a TA

1. your debugged, extended, and tested program,
2. the results of compiling and running your program using `g++` on your machine, and
3. the results of submitting your debugged code for the single-date problem to the homework server.

Checkpoint 3

Note that solving the Checkpoint 2 problem did not actually require arrays. You were asked to solve the problem in this manner just to give you some practice. Checkpoint 3, however, will require the use of arrays, and will give you practice with functions.

Checkpoint 3 will also require using the function `fabs`, which returns the absolute value of the expression it is passed. The prototype for this function is in header file `cmath`. Thus, for the problem in Checkpoint 3, the complete set of headers is

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

If you did not have the `using namespace std;` statement, you would have to refer to `fabs` as `std::fabs`. Here is an example snippet of code using `fabs`:

```
float x = 3.5, y = 7.4;
float diff = fabs(x-y);
cout << diff << '\n';
```

For the third checkpoint, write a program that reads in a sequence of floats into an array, computes the average, and decides which value is closest to the average. The program should output both the average and the closest value. The main computation of the program — the computing of the average and the value closest to the average — *must* be accomplished in a separate function. Since the function must produce two results, these results must be communicated back to the main function using *reference parameters*.

The first value input will be an integer giving the number of values in the sequence. The remaining input will be the values. You may assume there are at most 100 values.

As an example, given the sequence:

```
3    1.5  3.5  4.0
```

the program should output the two values 3.0 and 3.5, while given the sequence

```
15
```

```
11.5 17.8 9.9 12.2 -1.7 89.2 65.1 27.1 40.2 9.4 -2.5 26.1 37.1 44.3 56.8
```

the program should output the two values 29.5 and 27.1.

If you are using Visual Studio, you will need to

- Create a new project within the current solution.
- Create a new C++ source file. To do this, click

```
Project -> Add New Item
```

and select a C++ source file (.cpp file). Be sure it is being added to the new project (it should be) and then give it a name (something like `checkpt3.cpp`).

Regardless of what environment you are using, type in the source code, save it, compile it (fixing any compiler errors you make), and run it. Fix any logic errors that you see in the results.

To complete this checkpoint and the entire lab, show the resulting program and its output to a TA.