CSCI-1200 Data Structures — Spring 2014
Lab 1 — Getting Started

Welcome to CSCI 1200 Data Structures lab! Please listen carefully when your graduate lab TA and undergraduate programming mentors introduce themselves at the start of class. They are here to answer any questions about the course materials and work with you one-on-one to master strong programming and debugging skills. Also, introduce yourself to the other students in your lab section. 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. **Note: each student must produce his/her own exercise solutions.**

There will be three graded "checkpoints" associated with each lab. When you have completed each checkpoint, raise your hand and your graduate TA or one of the programming mentors will check your work. Part of earning each checkpoint for the lab will involve answering one or more short questions about the material. 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 questions hard to answer.

If the TA & mentors are busy helping other students, add your name to the "help queue" or "checkpoint queue" at the front of the room. **Do not wait until the end of lab to be checked off for multiple checkpoints.** If there is a queue the TA/mentor will only check you off for one checkpoint at a time and ask you to add your name to the end of the queue for the next checkpoint.

> *IMPORTANT NOTE: No phones, no email, no texting, no social media, no web surfing, no game-playing, no distraction! With the exception of downloading lab files provided by the instructor at the start of lab, and occasional use of online C++ reference material (e.g., to look up the the details of a particular built-in function or class), you are not allowed to use the internet during lab. Unplug or disable your wireless connection if necessary to remove the temptation. Anyone caught using the network or programs not directly relevant to this course will be given an immediate 0 for that lab and asked to leave.*

Getting Started

The first checkpoint involves getting started with the C++ development environment that you plan to use for all the labs and homeworks in this class. You may use whatever operating system, compiler, editor, and general runtime environment that you choose for code development in lab and for the homeworks. However, for the homework assignments, your code must compile and run correctly under gcc/g++ 4.6.x. This streamlined grading process allows the TAs to spend more time giving you constructive feedback on programming style, individual tutoring, and debugging help.

- **Create a directory (a.k.a. "folder")** on your laptop to hold Data Structures files. Create a sub-directory to hold the 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.
- Using a web browser, copy the following files to your `lab1` directory:
  http://www.cs.rpi.edu/academics/courses/spring14/ds/labs/01_getting_started/julian.cpp
  http://www.cs.rpi.edu/academics/courses/spring14/ds/labs/01_getting_started/README.txt

Compiling using gcc/g++ on Linux, FreeBSD, MacOSX, 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 (and/or the `clang++` compiler). This will help you develop portable code and prevent problems when submitting your assignments.

Windows users will need Cygwin to follow the instructions below. The default installation of Cygwin from RCS does not include all of the packages you will use this semester. If you are missing some packages, re-run the setup.exe installer, and search and click to enable installation of "g++" and "zip" and "clang". Upgrading is much faster than the original installation. Read more here:

http://www.cs.rpi.edu/academics/courses/spring14/ds/development_environment.php

- **Open a shell/terminal/command prompt window**. Windows users should be able to find Cygwin in the list of programs. Within the Cygwin directory, you want the `bash shell`. *Please ask for help if you have problems installing Cygwin or finding your Cygwin shell.*

- **Change directories into your new `lab1` directory**. On Windows, 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 a forward slash ('/') and when you have a space in the name of the directory, then `bash` requires that you precede the blank with a backslash ('\'). Thus, on a windows machine, you may type something like this:

  ```
  cd Users/username/My\ Documents/Data\ Structures/labs/lab1
  ```

  Note you can use the tab key to auto-complete the directory or file name (if unique) after typing the first few letters. Type `ls` to confirm that the downloaded `julian.cpp` and `README.txt` files are in the *current* directory.

- 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
  ```

- Note: To check which version of gcc is installed on your machine, type:

  ```
  g++ -v
  ```

  If you are not using g++ 4.6.x, you *may* notice slight differences between your compiler and the version on the homework submission server when we get to advanced topics. You may also try to compile using `clang++` instead of `g++`. The LLVM/clang++ compiler has earned much praise for having clear and concise compiler error messages that are especially helpful for new C++ programmers.

To compile using the Microsoft Visual C++ development environment:

- **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.

- **Create a workspace.** Under the `File` menu, create a new "Empty Project" 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 Data Structures 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.

- **Add your file to the project.** Use the `Project->Add Existing Item` menu and browse to the location where you saved your lab files. 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.
- 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**.

Checkpoint 1

The process of *compiling* a program translates the high-level C++ code into machine-level, "object" code. The compiler itself is an extremely complicated program and is the most important part of the 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're 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 on the left pane and follow the instructions under the "Electronic Submission" section to zip up the `julian.cpp` and `README.txt` files. After submitting the buggy code you should receive confirmation of your submission and be notified of the compile-time errors in the program. Note that all graded assignments will require submission of a plaintext README.txt file to receive full credit.

To complete Checkpoint 1: Show one of the TAs the compiler errors that you obtained in the g++ development environment on your machine *and* the response from the homework submission server indicating the same compiled errors. Also, give the TA your signed "Collaboration Policy and Academic Integrity" form (handed out in lecture on Tuesday).
http://www.cs.rpi.edu/academics/courses/spring14/ds/csci1200_collaboration_and_academic_integrity.pdf

Checkpoint 2

The compiler errors we have introduced are pretty simple to fix. Please do so, and then re-compile the program. Once you have removed all of the errors, you are ready to execute the program. If you're 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. Keep this submission results page open in your browser, you will need to show it to a TA to complete checkpoint 2. *You will not need the network for the remainder of this lab.*

For the rest of this lab we are going to review more about arrays and the logic of manipulating them. We'll also practice a bit 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 2014.) 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. Note: You won't submit this modified version of the program to the homework server!

To complete Checkpoint 2: Also show the TA the results of submitting your debugged code for the single-date problem to the homework server *and* your debugged, extended, and tested multi-date program (not submitted to the server).