CSCI-1200 Computer Science II — Spring 2006
Lab 1 — Getting Started

Overview

There are three points 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 remove/disable your wireless Ethernet card. 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. **However, 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

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 environment that you choose. Just be sure the code you submit for homeworks compiles with either the latest Vision Studio compiler or gcc 3.x.

If you're using g++ on Linux, FreeBSD, or Windows/Cygwin:

1. **Create a directory** on your laptop to hold CS II files. Create a sub-directory to hold CS II labs. Create a sub-directory for Lab 1.
2. Using a web browser, **copy the following file** to your Lab 1 folder:

   `http://www.cs.rpi.edu/academics/courses/spring06/cs2/labs/01_intro/julian.cpp`

   From this point on, you may not use the network during this lab.
3. **Open a shell/terminal/command prompt window** and go to that directory.
4. Attempt to **compile/build the program** for this lab (or other simple one file projects) by typing:

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

   In later labs/assignments with more files and more complex build parameters you will probably want to use a *Makefile* (which you can learn about later).

If you're using the Microsoft Visual C++ Development environment:

1. **Create a folder** on your laptop to hold CS II files. Create a sub-folder to hold CS II labs. Create a sub-folder for Lab 1.

2. **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.

3. **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.

4. Using a web browser, **copy the following file** to your Lab 1 folder:

   `http://www.cs.rpi.edu/academics/courses/spring06/cs2/labs/01_intro/julian.cpp`

   From this point on, you may not use the network during this lab.

5. **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.

6. 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 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.

To complete Checkpoint 1: Show one of the TAs the compiler errors that you obtained to demonstrate that you have reached this point.

Note: Learn how to save all of your work (in Visual Studio, look under the File menu). It is a good idea to save your work frequently, so that if the computer crashes you do not lose too much.

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'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.

Show the TA that you have done this as well before proceeding. (Note that this is an intermediate step toward completing Checkpoint 2.)

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 2006.) 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: compile and test the resulting program. Show the result to a TA.

Checkpoint 3

Note that solving Checkpoint 2 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. The details of Checkpoint 3 will be distributed in lab. To prepare, you can look up the function `fabs`, which returns the absolute value of the expression it is passed. The prototype for this function is in header file `cmath`. Here is an example snippet of code using `fabs`:

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

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

If you did not have the `using namespace std;` statement, you would have to refer to `fabs` as `std::fabs`.