Lab 0: Hello, Boxes

CSCI 41

Objectives:

- Get familiar with the autograder and the terminal
- Implement a class with a custom constructor, overloaded operators, and friend functions
- Write custom tests to verify that your implementation is correct
- Draw pretty boxes on the terminal

Overview

This lab is half about getting settled into the virtual machine, and half about practicing our class-implementation skills. You'll first make a simple program to get used to the whole process, and then you'll implement and test a custom class.

Part 1: Get the starter code

First, `cd` into the folder where you want to store your class files (e.g., `~/labs`). Copy the starter code there with the following line:

```
gimme lab00@csci41
```

This will make a lab00 folder in whatever directory you were in. Then, if you `cd` into that folder and run `ls` you should see the following files:

```
$ ls
box.h          runBox.cpp      testBox.cpp
```

Now you're ready to start working.

Part 2: Write a hello world program

You will write the classic "hello world" program as a warmup. You must write your program in a file called hello.cpp, and your program must print **exactly** "Hello, world! I am ready for CSCI 41!" and then a newline.

Compile your code with `g++ hello.cpp -o hello` and verify that it's working correctly.

Part 3: Implement a box-drawing class

Take a look at box.h—it has some comments that explain how the class works. You will make a file called box.cpp that contains the implementations of these functions.

Look at runBox.cpp. It includes box.h and provides an example of using the `Box` class. Play around with compiling and running this file (using `g++ runBox.cpp box.cpp -o runBox`) to test your program. runBox.cpp is just for you—you won't be turning it in. Here is what the unedited runBox.cpp should look like when you run it (after you've implemented the Box class):

```
lawtonnichols:solution/ $ ./runBox
Enter a width: 8
Enter a character to draw the box with: $
$$$$$$$$
$$$$$$$$
$$$$$$$$
$$$$$$$$
$$$$$$$$
$$$$$$$$
$$$$$$$$
$$$$$$$$
```

Now that you know what's supposed to happen, go forth and implement box.cpp!

Part 4: Test your box-drawing class

Look at testBox.cpp. It includes box.h and tests that the `Box` class was implemented correctly—these are the benchmarks that you will be graded against. Compile it with `g++ testBox.cpp box.cpp -o testBox`.

You might want to look into how the `std::stringstream` class works: here's a link to the documentation page. It lets you create a "fake" output stream that you can `<<` into and get the resulting built-up string with `.str()`.

In addition to **getting the current tests to pass**, your job is to **write 6 more tests** in the same style as the ones given.

Part 5: Submit your code

When you're satisfied with your solutions, you can submit them to the autograder for grading. **Do not submit and assume you got 100%—you may be unpleasantly surprised. Always check your grade.**

From your solution directory, submit your code to the autograder using the following command on the terminal:

```
turnin lab00@csci41 box.cpp testBox.cpp hello.cpp
```

The autograder will grade your code within a minute or so. If it's not working, please yell at Lawton to fix it. Run `view-grades` to look at your grades on the terminal, or go to the class website to view them there. You may resubmit as much as you want before the due date—just follow this same process after you've updated your programs.

Rubric

Rubric Item	**Points (35 pts total)**
hello.cpp runs and produces the correct output	3 pts
testBox.cpp runs and passes the tests that I've provided	20 pts (4 for each test)
testBox.cpp runs and passes the 6 additional tests that you wrote	12 pts (2 for each test)