COMP 530 Assignment 0: Simple C++ Warm-Up

1. The Task

In Assignment 0, your task is to implement a simple LIFO stack template using C++. In case you don't know what a *template* is: a temple is C++'s variant on the idea of generic programming, somewhat analogous to a Java generic. However, there are some key differences. For example, C++ templates do not use type erasure and are generally much higher performant than Java generics, but the cost of this is that each time you instantiate a template you are actually creating a new class that must be compiled into your code base. If you are new to C++, I'd start be reading this page:

http://www.cplusplus.com/doc/tutorial/classes/

And this page:

http://www.cplusplus.com/doc/tutorial/templates/

But the real point of this assignment is not to learn about templates. The assignment is designed so that you can make sure that you are able to write code that correctly manipulates pointers and memory, creating and freeing memory as needed. Specifically, when you push a new value on the stack, you'll need to allocate a new stack node to contain the new value. When you pop, you'll need to deallocate the stack node.

If you struggle with this assignment because you are new to C++, that is OK. There is going to be a bit or a learning curve. If you struggle with this assignment because you simply can't figure out how to make pointers and memory management work, then you need to seriously consider your future in this class. A1 will be at least an order of magnitude more complex than A0!

One thing to be careful of: the test cases as published are not exhaustive. You will need to create a few of your own test cases. Make sure to test your template with a few other, complex types, such the built in C++ `String` type.

2. Testing And Grading

In this class, we're using a very simple little test harness for C++ called `Qunit`. I have written some unit tests for the stack class I've asked you to implement, and included it in the skeleton that I'll distribute. It is located in `StackTest/source/StackQUnit.cc`. When you build your project using the `SCons` build tool (see below), this file will be complied and an executable will be produced.

When you turn in your code, and it's time for us to grade, we'll run the test cases I've suppled, as well as several others that won't be made public until after the turnin. You'll

be graded on your code's success in passing all of the test cases. You won't be graded on style and comments. However, I strongly encourage you to take this opportunity to put your best software engineering practices to use.
Important:

3. Getting Started

Lastly, I'm going to describe how to get started on the assignment. I'm going to give instructions on how to get started on Clear, *because it is required that everyone get their code to run on Clear.* Let me say this again: **to get *any* credit on A0, your code must compile and run on Clear**. That way, we have a common environment for grading and we don't have to spend time getting your code to compile. That said, I suspect that almost everyone is going to do the assignment on their own laptop or home computer, and then copy over to clear to verify that everything works right before submission. Which is fine.

You will likely need to modify this process at least somewhat to get things to work in your own environment.

First you need to install the `SCons` build tool. `SCons` is a modern build tool (sort of like make, but far more useful because it is Python-based, and it allows you to include scripting code within your build tool). We'll be using `SCons` throughout the class.

Clear does not have `SCons` installed. So in your home directory, download `SCons`:

```
[cmj4@glass bin]$ wget --no-check-certificate https://pypi.python.org/
packages/source/S/SCons/scons-3.1.2.tar.gz
```

Unpack it:

```
[cmj4@glass ~]$ gunzip scons-3.1.2.tar.gz
[cmj4@glass ~]$ tar xvf scons-3.1.2.tar
```

Build it:

```
[cmj4@glass ~]$ mkdir scons
[cmj4@glass ~]$ cd scons-3.1.2/
[cmj4@glass ~/scons-3.1.2]$ python setup.py install --prefix=../scons
[cmj4@glass ~]$ cd ..
[cmj4@glass ~]$ rm -r scons-3.1.2/
```

And now you can run it (though nothing is going to happen because there is no `SConstruct` file in this directory to tell `SCons` what to do):

```
[cmj4@glass ~]$ ~/scons/bin/scons-3.1.2
```

Next, download the starter `A0.zip` and unzip it. Then go into the build directory and use `SCons` to build the project:

```
[cmj4@glass ~]$ cd A0/Build/
[cmj4@ring Build]$ ~/scons/bin/scons-3.1.2
scons: Reading SConscript files ...

What do you want to build/clean?

1. Stack unit tests (using default compiler)
2. Stack unit tests (use clang++ compiler for Clear)

Select the module(s) you want to build or clean. 2

OK, building stack unit tests using clang++.
scons: done reading SConscript files.

scons: Building targets ...
```

Once the build completes, you can run the code:

```
[cmj4@ring Build]$ bin/stackUnitTest
```

At this point you are ready to begin work on the project. Note that as long as you keep your header and source files in the directories that are currently there, `SCons` will automatically find and build them.

Hint: don't just test your code by putting integers in the stack. Also test by putting more complex objects in there.

4. Turnin

Simply zip up all of your source code and then turn it in on Canvas (make sure to archive into the zip format, and not some other archiving format. If you choose to use something else, we'll take off a few points!). Please name your archive A0.zip. Please do not change the original directory structure, except for perhaps adding some new files. The root should be a directory called A0, with two subdirectories Build and Main. And so on.

And remember, this needs to compile and run on Clear.

Good luck and have fun!