**+CSC 221 Data Structures**

**Spring 2014**

**Professor Burg**

**Programming Assignment 1 – Stacks, Postfix Expressions, and Expression Evaluation**

**Due Wednesday, February 12 at 8:00 AM. Please put your program in the Assignment folder on Sakai (not the Dropbox).**

Write a C++ program that (1) converts infix expressions to postfix expressions and (2) evaluates the postfix expressions to produce scalar values. (Postfix expressions are described on p. 105.) The *convert* and *evaluate* functions should be in the same file as *main*. Both these functions will use a Stack for their implementation. For an explanation of how to do the *convert* function, see pp. 108-111. For an explanation of how to do the *evaluate* function, see pp. 105-107.

* Use operators +, -, \*, and /. Observe proper precedence of operators and left-to-right execution.
* Implement your own Stack and StackNode classes as a class templates. Stacks are described in Section 3.6 of your book.
* Your *main* function should prompt the user for the name of a file containing input expressions. Print your output to the screen, including the converted expression (in postfix form) and the scalar value resulting from the evaluation.
* Create your own file of infix expressions. Each expression in the file should be on a separate line, and your program can operate under that assumption. All the numbers in the infix expressions should be one-digit integers. This makes the implementation easier.
* Your *main* function should have a loop that continues to process infix expressions until none remain.
* Divide your program files into interface and implementation files. If your class has only small functions that can be written in-line in the header file, you don’t need a corresponding .*cpp* file. (For example, you might have a StackNode.h file with no StackNode.cpp file.)
* Create and run your program with an IDE, preferably Eclipse.

**Warning.** Sometimes the thing that gives students the most trouble is the input part of a program like this. If you combine the use of *cin >> x* or *getline()* with *getchar()* and *getc()* you can get into trouble. The thing to watch out for is whether or not the input function is consuming the end-of-line character, *‘\n’*.

**Implementing the Stack and StackNode classes**

* The StackNode class should have private data members for
  + data (of the type for the class template)
  + next (a pointer to the next StackNode in the Stack
* You can make the Stack class a friend of the StackNode class.
* You’ll need a *forward declaration* of the Stack class within the StackNode class.
* The Stack class should have a private data member for *top* and public member functions for *push*, *pop*, and *printStack*.
* You don’t need to use an iterator for the Stack class. You can traverse the Stack in the “old fashioned” way by keeping a pointer to the current StackNode and advancing it as you print the Stack.
* You may include other things in these classes if they are appropriate for the implementation.