Dr. Scot Morse
Department of Computer Science
Western Oregon University

October 5, 2011

PROGRAMMING LAB 2 — Using a Stack to Perform Arithmetic

In this lab you'll use a Stack ADT implementation to write a calculator program. As discussed in lecture and the notes a stack is integral to evaluating postfix expressions. So our calculator will be a postfix calculator. You may ultimately write your program as a GUI program or a command line program. However, applets will not be allowed, so if you choose to write a GUI program it must be a stand-alone application.

Let's get a few requirements out of the way first:

1. We'll turn this in the same way as Lab 1 so follow the same procedures (*username_Lab2.zip*).
2. No matter what, the top level Java file for your program must be called `Calculator.java`. I should be able to compile and run your GUI or command line application by typing the following on the command line from within your folder: `javac Calculator.java` and then `java Calculator`. The files you must submit are:
   - `StackADT.java`
   - `LinkedStack.java`
   - `Node.java`
   - `Calculator.java`
   - Plus anything else needed for your GUI or command line program
3. All filenames and method names and signatures must be exactly as directed below. The automatic grading code assumes it can find exact methods inside exact files or it will not work. Just pay attention to the requirements in the Problems sections below.
4. Print out all your code and turn in at the beginning of class on the due date. Please, please, please make sure your code is nicely formatted. Do not copy and paste into Word – it looks horrible. I'll pay a quarter if you print it out in color with proper syntax highlighting (as all IDE's do) and no sloppy line wrapping.

Problem 1. StackADT implementation

To use a stack we must first have one. Your first task is to implement a linked stack and get it working perfectly. Type or copy in the `StackADT.java` file from the lecture notes and then complete `LinkedStack.java`. It is all there in the lecture notes but make sure you understand what is going on. Feel free to implement it yourself without relying on the notes, but make sure you implement the `StackADT` interface as it is found in the lecture notes. (A reminder: you cannot use code from anyplace other than from yourself or from me. So if you acquired a textbook, use it only as a guide and do not copy anything.)

To test your implementation with JUnit, feel free to use the `Lab2Test.java` file located on the class website (right next to where you got this file). It is written for both `LinkedStack` and the calculator part of this lab.

Problem 2. Postfix calculator

For this part of the lab, you are to implement the functionality of a postfix calculator that uses your linked stack to perform the calculation. To simplify things, this calculator simply takes an input expression and returns the answer. String in; string out. The user submits an input expression, in postfix form, and is handed back the answer. Note, ours *must* handle decimal numbers in addition to integers. That is, your calculator should be able to return an answer of 14.7186875 when given an input of: 2.71 3.95 $*$ 4 / 2.2 3.3 + $*$. Let's not bother with accepting literals in other bases such as hexadecimal or octal.

Begin by writing a class called `Calculator.java`. At the very least, this class must:

(a) use a default constructor to set itself up,

(b) contain a `main` method (the usual `public static` method used to run standalone programs)

(c) and contain the following method:

```
/**
 *  Evaluate an arithmetic expression written in postfix form.
 *
 *@param  input  Mathematical expression as a String
 *@return        Answer as a String
 *@exception  IllegalArgumentException  Invalid input string
 */
public String evaluatePostFixInput(String input)
{
//... Your code here
}
```

The unit testing code will test your stack by itself, and then test this method to see if you implemented the calculator correctly. Within this method, evaluate the input expression using your linked stack class and format the output as a string which you return.

Please see the unit testing file for examples of postfix input and the answer I expect. A basic problem is the accuracy of the results expected (since we're doing floating point

math). You will be fine if your code uses `double` for the arithmetic. Do not try to handle integers differently.

Problem 3. User Interface

Once you have the calculator implemented please create a GUI or command line interface so that you can actually use your calculator. There are no requirements for this part other than that it must somehow be usable by a normal person (who knows postfix!) and run from the main method inside the `Calculator` class (i.e. I should be able to run your code by typing this from the command line: `java Calculator`.

If your program requires special instructions to use, make sure I get them automatically when the program is run.

Hint

To extract the numbers and operators from the input string I suggest using the `java.util.Scanner` class.

Submitted by Dr. Scot Morse on October 5, 2011.