COMP 2150 – Spring 2014
Project 3: Evaluating Infix Expressions
(Posted Apr. 18 – Due May 5 by 11:59 pm)

Note: You have the option of working in teams of two for this assignment. If you choose to do so, please include both your name and your teammate's name on all your source code files. Only one person needs to submit the final product to eCourseware, but please include a note with the submission that lists both of your names. If an honors student and a regular student choose to work together, both agree to be graded on the honors scale.

Submission: Please zip your source code into a single file (you can zip the entire project folder if you're using BlueJ) and upload it to the proper folder in the eCourseware dropbox at https://elearn.memphis.edu. The dropbox will cut off all submissions after the indicated deadline, so please don't wait until the very last minute to submit your work!

Grading: This assignment is worth a maximum of 80/75 points for regular students (75/75 for honors students). The assignment will be graded by the TA Rahul Vemuri (rvemuri@memphis.edu). If you have questions or concerns about your grade, please contact him first. I'll be happy to look over your assignment myself if he's not able to resolve the situation to your satisfaction. Also remember that as stated on the syllabus, all submissions MUST compile and run to receive credit. The TA does not have time to find and correct your syntax errors!

Coding Style: Be sure to follow the good coding practices that were discussed in COMP 1900:

- Use descriptive variable and method names. Avoid using single-character names unless it's very obvious what the variable's being used for (like a loop counter).
- Follow standard Java programming conventions for **variableAndMethodNames**, **ClassNames**, **CONSTANT_NAMES**.
- Consistently indent your code.
- Use comments judiciously. Every source code file should include a comment block at the top that lists your name and the assignment number. Every non-trivial method should have a comment preceding it that specifies what the method does, what its parameters are, and what it returns. For simple methods like accessors or mutators this isn't necessary, but it doesn't hurt to put in a simple comment like `// accessor methods`. I also expect to see comments throughout your code explaining what actions are being taken!

Background Information

As discussed in class, you can evaluate an infix expression by using two separate algorithms: one to convert the infix expression into postfix, and one to evaluate the postfix expression. This requires two passes to evaluate the infix expression.

It's possible to execute both algorithms at the same time to allow you to evaluate an infix expression in a single pass. Here's how the combined algorithm works:

1. Maintain two stacks, one for **operands** and one for **operators**/parentheses.
2. Scan the infix expression one token at a time, from left to right. Here a "token" is defined as an operand, operator, or parentheses symbol.
    a. If the token is an operand, push it onto the **operand stack**.
    b. If the token is an operator or a parentheses symbol, handle it as described in the infix to postfix conversion algorithm. Every time you pop a non-parentheses operator off the **operator stack**, also pop the top two elements off the **operand stack**, perform the indicated operation, and push the result back onto the **operand stack**.
3. Once all tokens in the infix expression have been scanned, pop the remaining operators off the **operator stack** while also modifying the **operand stack** as described in step (2b).
4. The final result will be the top (and only) element left on the **operand stack** at the end.

The Assignment

Using this combined algorithm, write a *SUPER DUPER JAVA CALCULATOR*™. Your program should allow the user to enter an infix expression involving any combination of non-negative real operands (including decimals), the six math operators + - * / % ^ (where % and ^ mean mod and exponentiation, respectively – remember that mod's precedence is the same as multiplication/division, while exponentiation is higher), and parentheses/brackets/braces () [] {}. Expressions entered by the user may or may not include whitespace. The program should check the expression to ensure it's a valid infix expression, then evaluate the expression, display the final result, and keep allowing the user to enter expressions until s/he chooses to exit. Your infix expressions should be evaluated in a single pass using the combined algorithm above! Do not convert them to postfix first. (However, you can make more than one pass when checking the expression to see if it's valid.)

Your program should be as resilient as possible. In particular:

- Display an appropriate error message if the user's infix expression is not well-formed with its parentheses characters.
- Display an appropriate error message if the user's infix expression isn't an infix expression at all. For example, there's nothing wrong (parenthetically) with the expression `+ 5 * (6 1 -)`, but it's obviously not an infix expression.
- Display an appropriate error message if the user's infix expression contains things besides parentheses characters, operators, and numerical operands. Your program should never crash due to user input!
- The program should work correctly regardless of how many or few spaces the user puts in the infix expression.

Note that your textbook provides code for much of this project in Section 3.4 (pp. 170-188)! The only things I'm asking you to change/add are:

- Support for the % and ^ operators
- Improved input validation, as listed above
- Using the combined algorithm instead of converting infix to postfix first

To implement the stack data structure, you can use either one of the stack implementations we wrote in class, or Java's built-in **Stack<E>** class.

Below are some examples of user inputs and the corresponding output that your program should give.

If the user enters…	The program's output should be…
`8*(9-2)`	56.0
`[{100+  90 + 75 - 30}  / 4 ]`	58.75
`(57 -  50)  *4 + 8  %  4`	28.0
`((57-50)^1.2+8)*4`	73.32164852464746
`2**3`	An error message of some sort
`([57 -50)*4+8]*4`	An error message of some sort
`Jka;se8fajf`	An error message of some sort

Need Help?

If you choose to work with a teammate for this assignment, the two of you can jointly write your code. I don't mind if you consult with students outside your team, but the code you submit should be written exclusively by you and your teammate (or just you, if you opt to fly solo). If you find yourself stuck, please feel free to contact me (Top) anytime. The Computer Science Learning Center in Dunn Hall 208 is also open, where you can get help from graduate students. Hours are posted at www.cs.memphis.edu/cslc.