CS206 Introduction to Data Structures

Lab 5

Stacks, Debugging in Visual Studio Code

Thursday, Feb 20

Part 1: Postfix and stacks

Within VSC create a new workspace folder (I will assume you called the folder Lab05). Import into this folder PostfixEvaluator.java and TestPostfixEvaluator.java from `/home/gtowell/Public206/lab05/`. Recall that the procedure is:

a. Open a MATE terminal from the menus
b. `cd cs206/Lab05`
c. `cp /home/gtowell/Public206/lab05/*.java .`
(do not forget the dot at the end of the line)

This code implements a simple postfix evaluator (using the java Stack class) that only handles addition and only works on integers. Adapt / extend this code to do the math operations of subtraction, multiplication and division (as well as addition). To add other math operators you will need to change the value of the `OPERATORS` variable and edit the `evalop` function (both in PostfixEvaluator). `evalop` uses the Java `switch` statement; a sort of shorthand for repeated if-then-else. If you are unfamiliar with switch, you can read about, and experiment with, it at https://www.w3schools.com/java/java_switch.asp

To run the postfix evaluator, use the main function in TestPostfixEvaluator.

Postfix notation in math, briefly:
In postfix you put the operator last so: `3 3 +` results in 6. You can to a lot more: for instance `3 3 4 + *` results in 21 (why?). `3 4 + 3 *` also results in 21. The cool thing about postfix is that you do not need order of operations or parenthesis.

Use your expanded postfix evaluator to come up with a postfix expression that yields the value 131 and with uses at least 3 different operands (operands can be used more than once) and does not multiply or divide by 1. So for instance `131 1 /` is not acceptable; the formula does result in 131 but fails on the other two conditions.

Part 2: Debugging in VSC
VSC includes a debugging facility far superior to System.out.println.
This is well described in https://code.visualstudio.com/docs/java/java-debugging. Realistically, this article covers much more on debugging than you will need for this course, the sections on Breakpoints, Pause and Continue, Step In/Out/Over, Variables, and Call Stacks are more than enough (for today).
Using the code from Part 1 (or any previous assignment) experiment with breakpoints and the other things described in this article. Be sure you understand how to set breakpoints (and why), how to view the value of variables, how to execute a program line by line, and how to resume execution.

I expect that you will never again use print statements for debugging.

After reading about debugging, go back to the postfix evaluator program and figure out how to set a breakpoint(s) so you can observe the actual contents of the stack as a postfix operation is is evaluated.

When you are complete with both parts, hand in a this paper with your name and the formula you wrote to get 131. Also show the contents of the stack immediately prior to the execution of each operator. Put the stacks in the whitespace below.