INFS-519 - Assignment 2 – Calculated Expressions

Assignment Given: 9/28/2017
Assignment Due: 10/12/2017, 7:30 pm

In this assignment you are asked to create a Java program that reads an arithmetic expression, performs the arithmetic in the expression, and outputs the answer. The expression will be given in infix notation, which is the way we normally write them, the program converts the expression to postfix form, performs the calculation on the converted postfix form and outputs the answer. The program will require two steps, both involving the use of a stack data structure: (1) convert the infix expression to postfix, and (2) process the postfix expression by computing it and output the result. The outline of the two algorithms needed for each step will be given in class the day the assignment is given (9/28/2017).

In creating and coding the program:

1. Write the code in Java (using command line or your chosen IDE)
2. The input file is a text file, with the following description:
   - contains a series of arithmetic expressions, one expression per input line
   - you may assume a maximum of 100 lines will be presented
   - there is no limit to the number of operators and numbers on one line, but assume each line is 80 characters or less
   - the expressions contain a mixture of numbers and operators
   - numbers are positive integers (not negatives) and may be one or more digits
   - results might be negative numbers or non-integers (i.e. 4.75, 12.055, etc.)
   - operators in the set are {*, /, +, -, (,)} (multiplication, division, addition, subtraction, left and right parentheses) – operator precedence must be followed when calculating the expressions
   - numbers and operators may have any number of embedded spaces, which must be skipped
   - the first 'n' expressions in the evaluation test file will be error free, the remaining ones may have errors – detecting and flagging the errors is part of the assignment (but see the evaluation criteria below)
   - The submitted code must read the input file as an argument to the program since that's the way the GTA will evaluate your submission
3. Examples of expressions:
   - 3*4-5
   - 3 + 4 * 6
   - 7 * (6 + 15 – 11 *2) + 20
   - 8/5 + (10 – 5)
   - 3*(6 – 11)
   - ((5 + 11) * 3) - 19
4. You must create and use stack implementations for both steps. Both implementations presented in class – dynamic array and linked list – are to be implemented
5. Create your basic algorithm and code to read and process the expressions that uses your stack implementations. If you implement the stacks intelligently, your basic algorithm should not care which stack implementation is used and should not need to be changed. **You must create your own code for the basic algorithm**, but the stack implementations may reflect what was presented in class (with attribution).
6. I suggest testing and debugging the code using small expressions to start, then build up to larger, more complex ones
7. Open and process the input file twice, once for each stack implementation. For each stack implementation, provide the following displayed messages to the execution output stream (2 sets of messages, one per implementation):
   - Display a message stating the start of expression evaluations using the dynamic array implementation, and display on one line:
     - the expression as read from input (infix notation)

    - the expression in postfix notation
    - the arithmetic result, or a message that the expression has errors
- Display a message stating the start of expression evaluations using the linked list implementation and the same one line per expression as for the dynamic array

Measuring the Timing and Analysis Report

Since both stack implementations are very fast and constant time, this assignment has no timing requirement. No written report is required, but see the submission requirements below.

What and where to submit:

1. All items are to be submitted through Blackboard in a single directory, preferably zipped
2. The Java code used to perform the sorts, with a comment block that includes your name, date, "INFS-519 Assignment 2", appropriate code attributions (if any), and a short description of the code's input, processing, and outputs. As noted above, the code should be a single stream in a file named 'calcEx', with whatever classes are needed to implement the stack. The code will be assessed by visually inspecting the source and executed using the Java command line for a test set of expressions of at most 100 80 character lines. All code needed to execute from the command line must be included.
3. A pdf file that includes a log or screen shot of the output, including the 2 message sets described above.

How the assignment will be assessed

The Java code will be visually inspected and executed from command line using a set of test expressions. An example file for you to use in debugging is provided on Blackboard. That file does not necessarily contain all possible combinations of expressions or errors. Your program must read the input file as an argument and open it for processing. The GTA will not edit your code to hard code a file name. Programs correctly processing the first 'n' (error-free) expressions and outputting the correct arithmetic results will be given partial credit, while the ones correctly processing the entire set, including errors, will be given full credit.

Item	Assessment Description	Max Value
Java Code	All code submitted	45
	Code is readable, commented, attributed when necessary	5
Execution Output	Code correctly executes against the first 'n' error-free expressions and outputs the correct arithmetic results (GTA assigns partial credit for less than 100% correct output, using his judgment)	30
	All the above plus the code flags expression having errors after finding the first in the expression (errors in the same expression after that can be ignored, only the first needs to be detected)	20
Total		**100**

Extra Credit: As with Assignment #1, for the class being held the night the assignment is due (10/12), I would like a few students to volunteer to present their code and results to the class. This would be at most a 10 minute presentation that includes (1) a code walk-through, (2) reviewing the results, (3) discussing results, and (4) Q&A from the class. Contact me separately if interested. Priority will be given to those who haven't presented before. Since this assignment is less complicated in terms of analytical results, I will limit the number of presentations to five. Extra credit would be +5% added to your cumulative assignment score (i.e. if the collective grade for all your assignments is 85% for the semester, this increases it to 90%). Presenting your work in front of peers is good experience (professionally and academically).