Arithmetic Expression Trees (Part 2)

Precedence Handling

Cpt S 321 Homework Assignment, WSU

Submission Instructions:

Submit via Git. This project will be used in future assignments.
Directory name: Spreadsheet
Git tag for submission: Spreadsheet-v3.0

Important note: This is the framework for a spreadsheet application that you will build over the course of the semester. Almost ALL remaining homework assignments will build on top of this. So think about these instructions DEEPLY!

Assignment Instructions:

Read each step's instructions carefully before you write *any* code.

In this assignment you will finish what you started in the previous expression tree homework assignment by implementing an arithmetic expression parser that builds a tree for an expression. This tree can then be used for evaluation of the expression. You may want to refer back to the instructions for the first half of this assignment to make sure that everything was properly implemented there and that you are following the required naming conventions.

Recall that your expression tree class is implemented in your engine DLL and demoed in this assignment through a standalone console application that references the DLL. Integration into the spreadsheet will happen in a future homework assignment.

Requirement Details:

Support parentheses (6 points):

- Support the addition and subtraction operators: + and -
    - You may assume that all instances of the minus character are for subtraction. In other words, you don't have to support unary negation.
- Support the multiplication and division operators: * and /
- Supporting precedence of operators is a requirement.
    - There are several choices to achieve operator precedence, but Dijkstra's Shunting Yard algorithm is a reasonable and common approach.
        - It first turns the infix expression (3 * 4) into a RPN (reverse polish notation) or postfix expression with precedence enforced: 3 4 *

        - The RPN expression is then turned into our expression tree and evaluated from there using stack-based operations:
            - https://en.wikipedia.org/wiki/Shunting-yard_algorithm
            - http://www.oxfordmathcenter.com/drupal7/node/628
            - https://brilliant.org/wiki/shunting-yard-algorithm/
- Parentheses must be supported and obeyed. Implementations without support for parentheses will result in a 50% deduction on all working operators. For example, if you support all 4 operators correctly but not parentheses, you get 3 out of 6 points for this part.

Tree Construction (5 points):

- Build the expression tree correctly internally. Non-expression-tree-based implementations will be penalized 4 points.
- If you use the Shunting-Yard algorithm, this process should be relatively straight forward if you've finished the prior homework.

Support for Variables (3 points):

- Support correct functionality of variables including multi-character values (like "A2").
- Variables will start with an alphabet character, upper or lower-case, and be followed by any number of alphabet characters and numerical digits (0-9).
- As you build the tree, every time you encounter a new variable add it to a dictionary of variable values with a default value of 0. This will be needed for integration with the spreadsheet application in a future assignment.
- Variables are stored per-expression, so creating a new expression should clear out the previous set of variable values. However, all variable values must persist as long as the expression is NOT being changed.
    - Ex: You shouldn't reset any variables after an evaluation or anything like that
    - This means you can enter an expression, set variable values, compute, then change variable values, compute, repeat -- but if you enter a new expression all variables are cleared out and set to zero

Clean, REUSABLE code (1 point):

- Have clean, well-documented code and an easy to use interface for grading.
- Obey all the naming requirements outlined in the first part of this homework (as well as any added here obviously)

10 points total

** Reminder -- code that doesn't compile for the TA is a zero. This is a 300 level class so you definitely need to be delivering code that at least compiles and runs something.

** Reminder -- Keep checking that your Git repo has your whole Visual Studio project including the solution file (sln), project file (vcproj), and source files. This is most easily done by cloning your repo into a temp directory and trying to open the sln file to build & test. If that goes well, then your grader should see the same thing as you.