# Arithmetic Expression Trees (Part 2 of 2)
# Cpt S 322 Homework Assignment #6
# by Evan Olds

## Submission Instructions:

Submit source code (zipped) to Angel BEFORE the due date/time. If the Angel submission is not working, then submit to TA via email BEFORE the due date/time. "Angel wasn't working" is never an excuse. Optional: Include a readme.txt file in the zip with any relevant information that you want the grader to be aware of.

## Assignment Instructions:

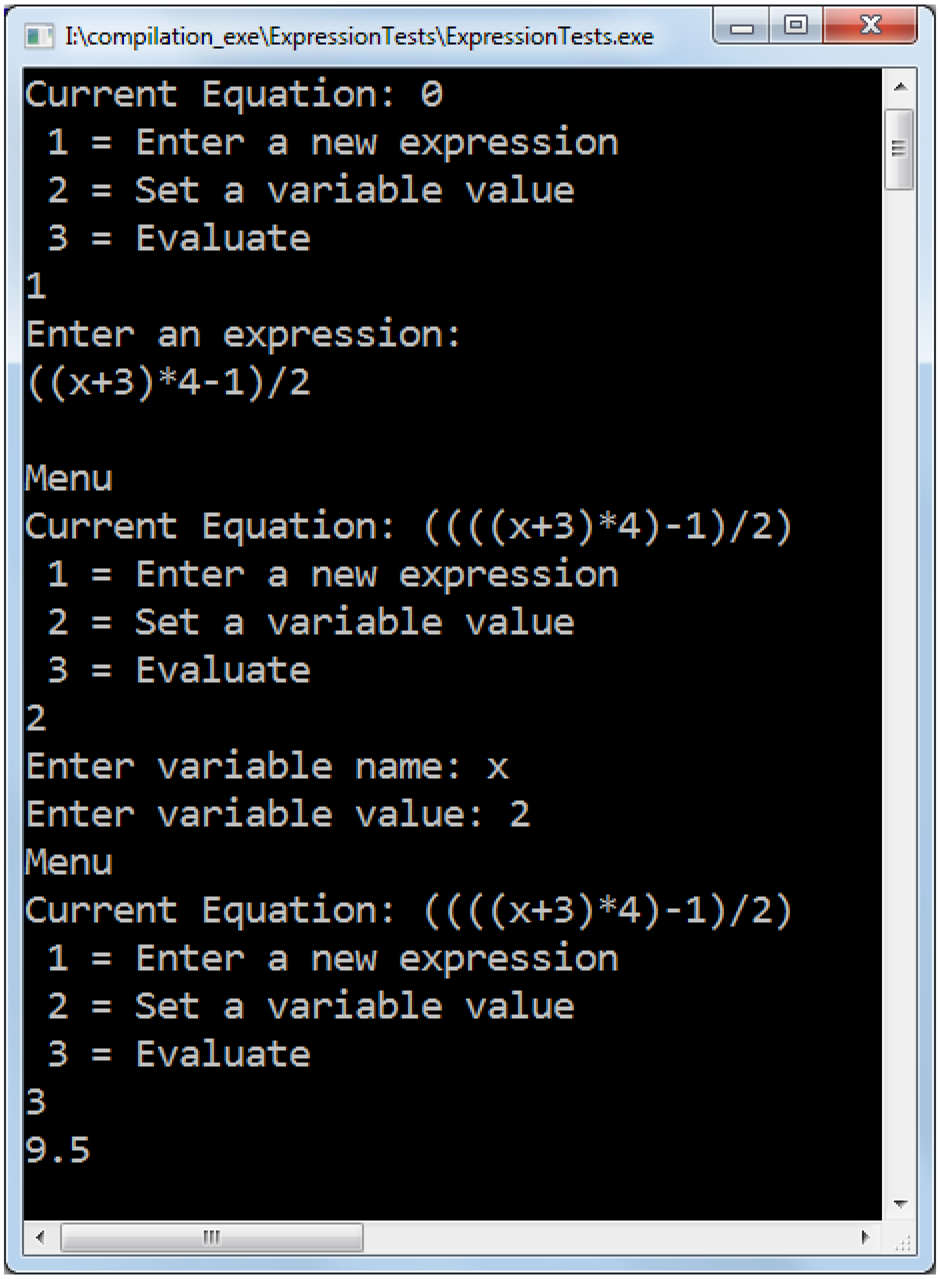
**Read each step's instructions *carefully* before you write any code.**

In this assignment you will finish what you started in the previous homework assignment by implementing an arithmetic expression parser that builds a tree for the expression. This tree can then be used for evaluation of the expression. Recall that this is a standalone console application and integration into the spreadsheet will happen in a future homework assignment. Make sure you implement the code in a class or collection of classes that can be easily moved to another application later on.

Create a console application that presents a menu when run. This menu must contain the following options:

1. The option to enter an expression string. You may assume that only valid expressions will be entered with no whitespace during grading.
2. The option to set a variable value in the expression. This must prompt for both the variable name and then the variable value.
3. The option to evaluate to the expression to a numerical (double) value.

The screenshot below shows what this might look like:



```
Current Equation: 0
 1 = Enter a new expression
 2 = Set a variable value
 3 = Evaluate
1
Enter an expression:
((x+3)*4-1)/2

Menu
Current Equation: ((((x+3)*4)-1)/2)
 1 = Enter a new expression
 2 = Set a variable value
 3 = Evaluate
2
Enter variable name: x
Enter variable value: 2
Menu
Current Equation: ((((x+3)*4)-1)/2)
 1 = Enter a new expression
 2 = Set a variable value
 3 = Evaluate
3
9.5
```

## Requirement Details:

Support parentheses and operators with proper precedence (3 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 negation.
- Support the multiplication and division operators: * and /
- Each operator must be implemented with proper precedence.
- 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 1.5 / 3 points for this part.

Tree Construction (4 points):

- Build the expression tree correctly internally. Non-expression-tree-based implementations will be penalized 4 points.

Support for Variables (2 points):

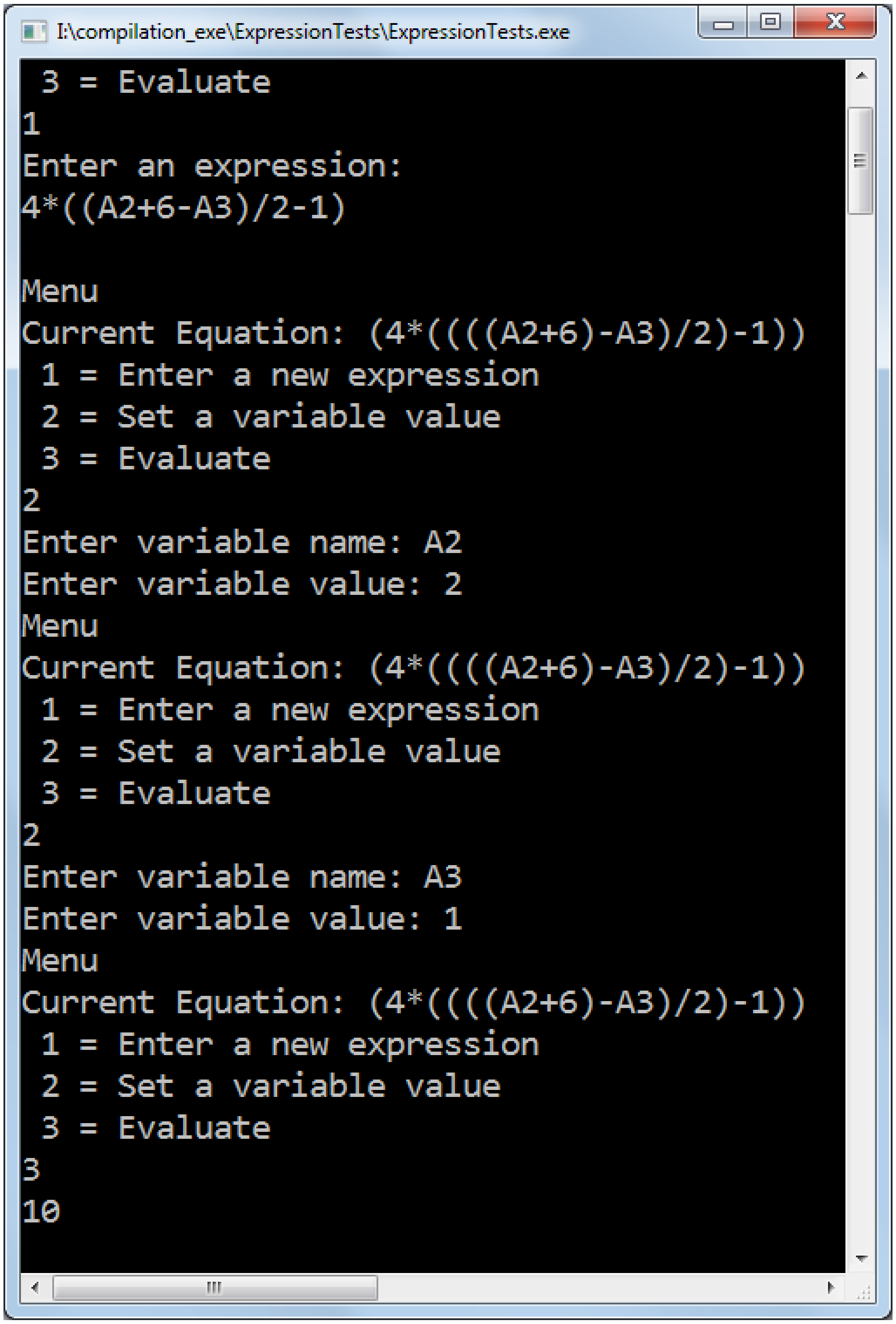
- Support correct functionality of variables including multi-character values (like "A2").
- Variables will start with an alphabet character, upper or lower-case, and be followed by any number of alphabet characters and numerical digits (0-9).
- As you build the tree, every time you encounter a new variable add it to the variable values dictionary with a default value of 0. As we discussed in class, this will be needed for the extensions that you'll need to do to integrate it with the spreadsheet stuff later on.
- 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.
    - You shouldn't reset any variables after an evaluation or anything like that

Clean, REUSABLE code (1 point):

- As we discussed in class you can have this functionality in a class called "Expression", "ExpTree", "Exp", or something else similar. You can name it whatever you want, but make sure all the relevant functionality is in this class. Do not have things like the variable dictionary declared as a local variable in Main(). Such implementations will not allow you to easily bring your code into the WinForms app for the next homework.
- Have clean, well-documented code and an easy to use interface for grading.

10 points total

Screenshot of another example (with multi-character variable names being used):


I:\compilation_exe\ExpressionTests\ExpressionTests.exe

```
 3 = Evaluate
1
Enter an expression:
4*((A2+6-A3)/2-1)

Menu
Current Equation: (4*((((A2+6)-A3)/2)-1))
 1 = Enter a new expression
 2 = Set a variable value
 3 = Evaluate
2
Enter variable name: A2
Enter variable value: 2
Menu
Current Equation: (4*((((A2+6)-A3)/2)-1))
 1 = Enter a new expression
 2 = Set a variable value
 3 = Evaluate
2
Enter variable name: A3
Enter variable value: 1
Menu
Current Equation: (4*((((A2+6)-A3)/2)-1))
 1 = Enter a new expression
 2 = Set a variable value
 3 = Evaluate
3
10
```