Spreadsheet Formula Evaluation
Cpt S 322 Homework Assignment #7
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 combine your arithmetic expression evaluator from homework #6 with the spreadsheet application from homework #4.

Part 1 (1 point): Edit the spreadsheet application from homework 4 so that it has the following functionality:

1. In its normal, non-editing state each cell will display the **Value** property of the cell in the logic engine. Recall that this value represents the *evaluation* of the formula in the cell, if present. The **Text** property represents what the user actually typed into that cell (which may be a formula if it starts with '='). If there is no formula in a cell then the **Value** just matches the **Text** property.
2. When the user starts editing the cell, the cell value should change to the **Text** property of the cell. When the user finishes editing it must go back to the **Value**. This functionality closely resembles that of other spreadsheet applications such as Microsoft Excel.

You'll need to use the CellBeginEdit and CellEndEdit events.

Part 2 (4 points): Incorporate your expression evaluator into the spreadsheet functionality so that when the spreadsheet is updating the value for a cell, it is properly evaluating the formula. You may assume the following:

- Each cell only contains a formula to be evaluated if it starts with the '=' character. The substring to the right of this is the actual formula that your expression evaluator should be able to parse and evaluate.
- No formulas will contain whitespace. It would be *nice* if your code handled whitespace characters (by ignoring them), but it's not required.
- Every formula contains only things that were required in homework assignments 5 and 6: add, subtract, multiply, divide, exponents, constants, variable names and parentheses.

- All variable names will be cell names that start with a single capital letter character and then are followed by a row number. Rows start at 1 in the user interface and therefore in the formulas as well, so keep this in mind when adjusting array indices in your code (actual index in array will be 1 less).
- There will be no circular references. This means that if we have some cell, say A1, it won't have A1 in its own formula. Also, it won't reference any cell that in return references back to it. You WILL have to deal with this in a future assignment, so keep that in mind. But for this assignment right now you don't need to worry about it.

There are sub-tasks within this task that require extending your arithmetic tree code. As discussed in class you need a way for the spreadsheet to enumerate all the cell names referenced in the formula. Some of this has additional requirements tied to it, as discussed in part 4.

Part 4 (3 points): Make sure that when a cell is changed all other cells that reference that cell in their formulas get updated. This means that the cell Text property change is not the only circumstance where you need to update its value.

There's some work behind doing this in an efficient way. Don't just try to update all cells in the spreadsheet every time any cell changes. Come up with a more efficient design than that. You'll get 1 point if you get this working in any way, but to get the other two you need something better than that.

Part 5 (1 point): Have a good design with respect to how the spreadsheet gets information about variables used in a particular expression. Don't expose your tree nodes publically, unless you can do so in a fashion that guarantees that the outside world can't modify it, but even this is not recommended. As we discussed in class, the ExpTree class that you built could have a "GetVarNames" function that returns a list or array of all variable names in the expression. Alternatively it could have a reference to a function to call when it needs to lookup a variable value. Whichever design you choose, the following must be true:

1. The world outside of your ExpTree class cannot edit the tree. There should be no way for the outside world to change a child reference in any node, change a variable name or value within any node, and so on. If all your nodes are declared privately within the ExpTree class then this should be taken care of.
2. This is similar to number 1, but the world outside of ExpTree must never have to deal with nodes. It's not the responsibility of objects outside this class to know how to iterate through an expression tree. Provide an easy-to-use interface for finding relevant information about the expression.

Part 6 (1 point): Make sure your code has a clean style and good comments.

1. Put your name and ID number in comments at the top of each source file.
2. Make simple, intuitive, and well commented utility methods. If you have to do the same 10 lines of code in 5 different places, don't copy and paste those 10 lines in those 5 places. Make a

function for it and call it in those 5 instances. It’s cleaner code and will also help you think about breaking things up into manageable parts.