Name: SGT Schoenwald-Oberbeck, Jesse
Date: 4FEB2017
Current Module: Data Structures and Algorithms
Project Name: Ticker

Project Goals:

Create a file that takes in a file, parses stock symbols, values and company names from each line, and allows the user to add or change stocks/values, and print the stocks in order of value, low to high at the end.

Considerations:

o File data must be able to be affected by user input.
o Operation must be quick, so efficiency must be a factor in handling the data, and searches thereof.

Initial Design:

Main exists within ticker.c, all other functions are in "tickerlib.c," which is included via the header file "tickerlib.h." Made file will be "ticker."

Data Flow:

A file will be taken in, parsed line by line into "symbol," "value," and optional "name." These parsed lines will be added to "stock" structs, which are assembled into a binary search tree via the insert function to which a function is passed allowing nodes to be sorted by symbol.

User input is taken, parsed by the same function, and added to the tree if unique, or modifying the existing value if symbol is matched.

The tree is traversed, and each node is passed back to the insert function, along with a sort on value function to make a new tree sorted by value instead of symbol.

The second tree is printed in order.

Communication Protocol:

No networking was required, and none were implemented or used.

Potential Pitfalls:

o Merging user input with file data would be easiest to accomplish with a function to take and digest file data, and a separate one to handle user input. However, a single function which takes a modifying compare function would be far fewer lines, and potentially be more dynamically usable in the future.

Test Plan:

User Test:

Small lists of stocks made by the tester.
Long lists of stocks.
Long lists of stocks, and large amounts of user input.

Test Cases:

All test cases completed with correct output.
Blank lines are skipped, stocks with matches have their values appropriately adjusted, unique input stocks are added. Sorts occur correctly, and operation/completion time is quick.

Conclusion:

My approach to this project was planned out before execution in a fair amount of detail. And although the trees are never balanced, operation is quick and efficient. Splay trees would only provide an advantage if a small selection of the larger body of stocks were being adjusted in quick succession. Otherwise, the constant moving of the tree is going to slow down operation if stock adjustments are widely varied. Re-sorting a tree by making a new tree with a new insert method is efficient, as it requires only one traversal of the initial tree, and insertion into the second one has the efficiency inherent in binary trees. Overall, I am satisfied with my methods and their efficiency.