Data Structures: Lab 1

Following the American Psychological Association's Guidelines

Markham M. Shofner

Johns Hopkins University

Data Structures: Lab 1

Analysis Document

I chose a stack to solve the problems presented in this lab, not just because it seemed require, but also because a stack seemed like the best solution. A postfix expression naturally lends itself to a stack, as the way we have learned (and one would natural decide) to evaluate a postfix expressions is through a LIFO structure. A simple way to evaluate a postfix expression is by walking through the string, one character at a time. If the character is an operand (A, B, C, etc…), push it onto the stack. If the character is an operator (+, -, *, etc…), pop the stack twice and perform an the operator operation with the popped values and then push the stored result back onto the stack. This is fairly easy to describe and understand, but was a slightly different ballgame to code – especially since we were not able to use standard libraries.

I feel like this was certainly a learning process, and a valuable and a times challenging one at that. It wasn't the most efficient way to build a stack to be honest, given that I wrote another implementation before the one I landed on. That said, I think I have a much stronger understanding than I did before around what a stack (or array or list or whatever) actually requires from a coding perspective, and also a much greater appreciation for those who came before me that have largely abstracted these problems away from my day to day coding life.

In terms of what I would do differently – I would set up my classes earlier, and also modularize my code on the front end of development, instead of midway through. Before any of that though, I would spend a bit more time with pen and paper actually planning out deeper parts of the application and thinking long and hard about overall structure and requirements before touching the keyboard. I think this would lead to more efficient work, or at least less lost work that I end up having to rewrite or delete. I also think taking cyclical chances to pause and reorient is something I can do moving forward to develop stronger more robust code.

My data structure setup was fairly simple. I relied on a handrolled stack to accept the postfix expression. The stack was based on a PostfixStack class that I built that kept track of the size of the stack, as well as a pointer to the top of the stack. The stack also supported basic stack functionality such as push (which accepted a data value and then created a PostfixStackNode based on that data value, while also tying the needed next pointer there as well). A pop subroutine (that remove and returns the top PostfixStackNode of the stack) was also included. And finally, the PostfixStack class included an isEmpty method that returns a Boolean the checks on whether or not the stack has contents (based on the size of the stack tracking int).

The main method handles opening up a filereader, creating a string based on input values, passing that to a translate function that uses the above PostfixStack, and then outputs the appropriate string values using a filewriter.

In terms of things I would like to do better next time, ideally I would have a few more hours to clean up error/edge case handling. Right now it's fairly basic, and I don't think this program is production ready if I were to want to share it with clients or use it in a professional since. It's more at the "let's hammer this code and find its weaknesses to fix before we ask people to pay for it" stage. But so it goes as a working professional. If I had a bit more time, I would also like to set up accepting arguments from the command line as inputs to read from. This one is fairly trivial in terms of implementation difficulty, but fairly major in terms of user experience – so would be good to get in at some point.

Regarding recursion, I guess I can see how a recursive solution could be applied to parts of this application – but honestly I don't see it as a better or more efficient solution. It could probably be cleverly implemented, but I don't think it would serve to work faster or more smoothly, or even be more human readable necessarily. That lack of vision may be on me, and my lack of truly deep familiarity and comfort with recursion. But an iterative version of a stack (which seems to be the crux of this whole assignment – not the error handling, or the file i/o, or the string building, etc…) seems to be the easiest/best way to think about that data structure, and the requirements it fulfills.

Efficiency wise, I think this program can quickly consume inputs and produce outputs based on the majority of .txt files that would reasonably be inputted, or postfix expressions that would commonly need to be expressed in machine language. In terms of space consumption, there is not a lot of information being stored in memory (as garbage collection will clear out locally stored variables), so I believe this application would be able to handle a fairly heavy load in terms of input file size.

References

zyBooks. 605.202: Data Structures. Module 2 – Stacks.