605.202: Introduction to Data Structures

Omar Ismail

Lab1 Analysis

Due Date: March 2nd, 2021

Dated Turned In: March 2nd, 2021

Lab1 Analysis

The stack data structure was a great way to convert prefix to postfix expressions. My program had three main components: user input/output through .txt file system, conversion of prefix to postfix expression, and the use of a stack, made up of string arrays, to hold contents of the conversion.

The conversion of the expression can seem a little complicated but it's not too hard to understand. We could have just used an array and a for loop to do the conversion, but it would have been pretty complicated. We would also need to know the length of the expression in order to use arrays, which is not optimal for our purposes. Using a stack allows us to use this program in more instances. A stack also allowed us to move around and group variables very easily. I think a stack is the preferred way to implement a solution for this problem.

When considering recursion vs iteration for this project, I think there are a lot of benefits to utilizing a recursive solution. If memory does not play a role, a recursive solution would make my code much more concise and easier to read. The base case would be when the algorithm reads every character in the expression from right to left. So when expression.length() = 0 would be the base case.

Due to the requirements of this project, we could not use the built in library Stack class (java.util.Stack) which essentially moves through any type of element not matter the type. It is really nice to use and extremely useful. In my implementation of my array stack, I had to continuously convert between string and char based on what step I was in the conversion. On the other hand, the Java stack class does not care about the type as it parses through anything so that only the available memory is a restriction. Therefore, creating my own array stack class as quite challenging.

What I Learned

This project gave me a lot of insight on how to tackle a program from multiple angles. To be more specific, the project seemed daunting at the start because I did not know where to start. I had to learn how to focus on smaller parts of the project in order to be productive. For example, I started by building a separate program using the Java stacks library in order to get the base of my algorithm. I build my own stacks class. Eventually I had multiple pieces of the puzzle and all I had left was to just put them together. It was a really rewarding experience.

What I might do differently Next Time

I would test my program as I went in order to save time. I found myself not testing enough and when I finally would try to run my program, I would run into several different issues, which would overwhelm me. I think that if I were to test each part of the code as I wrote it, I would have eliminated problems quicker and overall been more efficient.

Justification for Design Decisions

There are three classes in my program. The main entry point outlines the solution of converting prefix to postfix expressions without showing the details of the conversion. This allows for a cleaner code.

The Stack class is the data structure part of the assignment. I used array strings in order to make my own stack because we were not allowed to use the Java Stack library.

The PrefixToPostfix class is the conversion algorithm that I created in order to convert from prefix to postfix. Creating its own class is useful because it can be used at any time, no matter the case.