**Programming Project Report**

Name: Logan Campbell

Date: 7/20/17

**Academic Integrity Statement:** I pledge that I have neither given nor received unauthorized help on this programming assignment.

**Problem Statement:**

The main goal for homework 4 was to create a program that transformed a mathematical expression into an infix expression to a postfix expression and then evaluate the postfix expression. Using the user’s expression as input and the stack data structure to construct different parts of the mathematical expression(s). The output of the program would be the evaluation of the postfix. The error handling comes into play when there is too many or too less of the operators in the expression and also if there were mismatching parenthesis in the expression.

**Design:**

The design was mostly using functions to tell whether a certain string was an operator or if it had a higher precedence then a previous operator. Knowing the previous operator requires a different data structure, which would be the stack data structure. The stack data structure is used to gather information that can either be pushed into, popped out of, or looked at the top of the stack and from there do as you wish. This data structure is very useful in this program because we are using it to take off, set, or obtain information from a string created stack. Shunting yard and evaluate use this data structure to arrange data in a suffice way. Pro: of this data structure is that it is very effective however the con is that can be easy to assume that this is a data structure like an array or linked list but it is not the case especially with the different operations the stack class has.

**Implementation:**

I started out with the shunting yard algorithm. Most because I thought I would be able to construct results from it before evaluation function. Which I was wrong, of course. Regardless, I started off developing a stack: into a for loop that was able to read each part of the string: determine what each string was for the stack (with precedence order): set it into a string/stack for it to be obtain by evaluate function. From here the evaluate would take the stack and pop off two values and perform an operation on it depending on what the stack had behind such a mathematical symbol. This would happen until there is no more stacks to read. This would be sent to my driver and back to the user.

**Testing:**

Testing my program required myself to test the different examples from the set of examples from example.txt (was that too many examples?). Furthermore, using these different cases for my program, I was able to determine was worked and what didn’t work. Using expression that tested: precedence, too many/too less operators, incomplete expressions, characters, special characters, and missing parenthesis. So far everything worked as planned as shown in this report.

**Conclusions:**

Overall the assignment was somewhat exhausting. I tried to tackle this program without additional help until I got close to ‘finishing it’ and then to find out that my logic was right but my implementation was janky. I was devastated. Nonetheless, with a little bit of intuition and some prayers, I was able to create this functional program. I spent quite some time on it, dealing with my logic to function and vice versa, I was able to finish this program about 10 hours. What would I’ve done differently, develop a broken algorithm that wasn’t functional in the first place.