**Team Project 1**

Project 2A – Infix Expression Parser

**Luke Janis**

**Montana Shaw**

**Nunzio Lamartina**

**Robert Blocker**

**February 25th, 2022**

**Table of Contents**

[1.](#_gjdgxs) System Design 2

[2.](#_30j0zll) UML Diagram 2

[3.](#_1fob9te) Test Cases 2

[4.](#_2et92p0) Team Member Contribution 2

[5.](#_tyjcwt) Possible Improvements 2

# System Design

The program on the whole consists of four classes all serving vital roles.

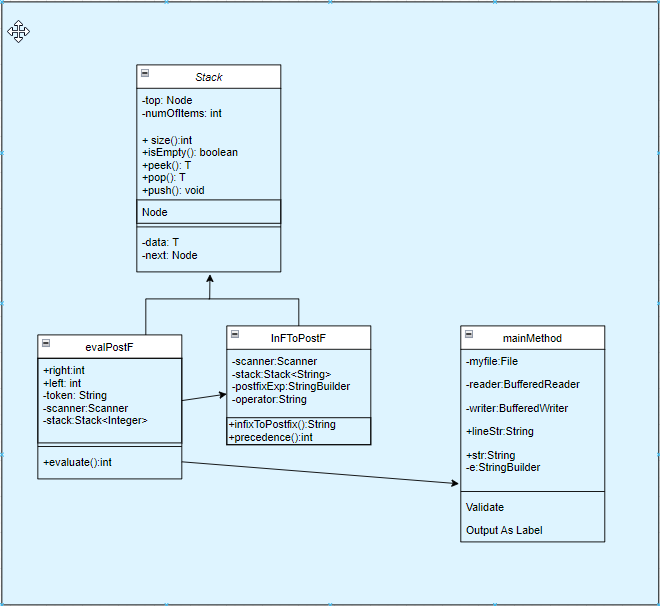
**mainMethod -** mainMethod is the entry point of the program. The main function of this class is where test expressions are written to a run-time generated file which will later be parsed. This design aspect aided in testing code more rapidly as well as ensuring an understanding of what was already functional during commits. After writing, the method parses it line by line, making the necessary calls to other classes and logging the work and results in a user-friendly fashion.

**Stack -** The class Stack serves as the primary data structure for storage. It is a singly-linked list operating under FIFO principles. Stack is used in evalPostF as an integer stack and in inFToPostF as a String stack.

**inFToPostF -** This class makes up the bulk of the program and contains the method infixToPostFix which parses infix expressions and converts them to postfix. A string stack is used to store tokens indexed by a scanner which parses the argument passed. The class contains only two loops, non-nested, making it of time complexity O(n).

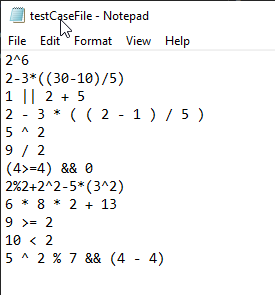
**evalPostF -**  evalPostF does the work of evaluating the postfix expression returned by the aforementioned method. In its method evaluate(), an integer stack is used to hold the numbers parsed from postfix form. When an operator is found instead, the operands are popped from the stack, after identifying the sign and performing it, the result is pushed back onto the stack.

# UML Diagram



# Test Cases

Here is our test file for our inputs. Let’s look at these test cases and compare our inputs. For starters, we mixed spacing to show the program does not consider white space.



For the infix expression, **2^6** we expect an evaluation of **64**.

For the infix expression, **2-3\*((30-10)/5)** we expect an evaluation of (**-10**).

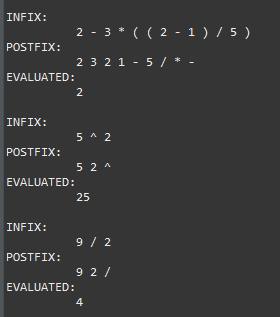
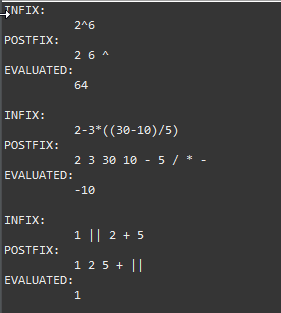
For the infix expression, **1 || 2 + 5** we expect a boolean evaluation of **1** as it is *true*.

For the infix expression, **2-3\*((2-1)/5)** we expect an evaluation of **2**,

while **(3\*((2-1)/5)) = 0.6 → 0** as we convert to the whole integer 0, therefore **2 - 0 = 2**.

For the infix expression, **5^2** we expect an evaluation of **25**.

For the infix expression, **9/2** we expect the integer value of **4**, as we don’t want 4.5 evaluated.



After our first 6 test cases, we can see the program gave the exact output that we expected.

For the infix expression, **(4>=4) && 0** we expect a boolean evaluation of **0** as it is *false*.

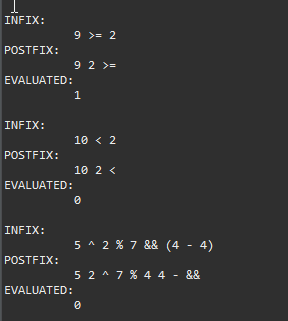
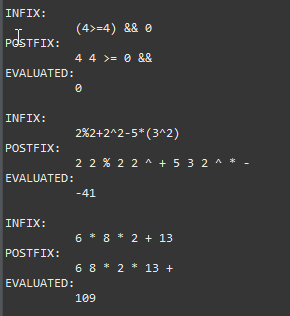
For the infix expression, **2%2 + 2^2 - 5\*(3^2)** we expect an evaluation of (**-41**).

For the infix expression, **6 \* 8 \* 2 + 13** we expect an evaluation of **109**.

For the infix expression, **9 >= 2** we expect a boolean evaluation of **1** as it is *true*.

For the infix expression, **10 < 2** we expect a boolean evaluation of **0** as it is *false*.

For the infix expression, **5 ^ 2 % 7 && (4 - 4),** expect a boolean evaluation of **0** as it is *false*.



After testing the final cases, we see that our program gave the expected output for each of the 12 cases while ignoring white space. Our program gives the infix expression from the input file, outputs the postfix expression, and then evaluates and outputs the evaluation.

# Team Member Contribution

Luke Janis – Luke worked on the development of the code for the project. Through logic and testing, Luke helped make sure the code functioned properly. Luke helped develop code in the main class along with the two additional classes the group created. Luke helped with the logic behind creating an efficient algorithm and program.

Montana Shaw – Montana worked on the development of the code for the project. He helped develop the logic while working on the code for the main class, the classes parsing the string, and helped with setting precedences. Also developed code to create a working, efficient program.

Nunzio Lamartina – Nunzio worked on the development of the code for the project and also collaborated with Robert to document the project. Nunzio joined the group in the middle of the project and worked to integrate himself with the group to help with the code development. He contributed to the logic of the string parser and the code of the postfix evaluation. Nunzio helped to document the project logic during the group meetings.

Robert Blocker – Robert worked on the development of the code for the project and also collaborated with Nunzio to document the project. Worked on the code development for the parsing of the given expression and the logic and development of the code to evaluate the postfix expression for the correct output. Worked on logic for an efficient algorithm. Robert helped to document the project logic during the group meetings.

# Possible Improvements

Our group took note of the possible improvements from our first project to make sure we had a program that was more conscious of the time complexities.

However, there could be some improvements to the simplicity of the program. Some of the code could be updated to create more efficient lines of code.