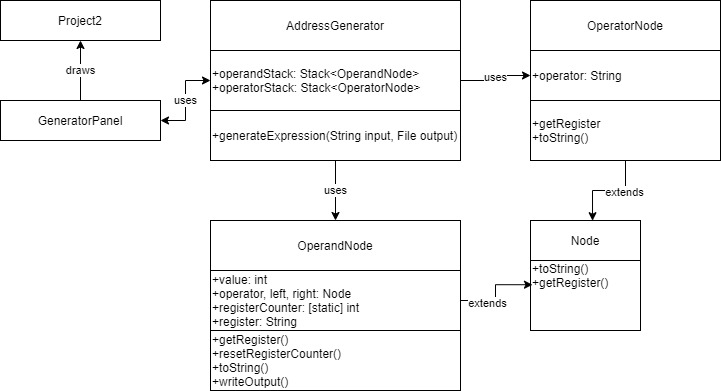
# CMSC350 Project 2 Documentation

By Brian Yu

## Goal

The goal of this project was to develop an in-fix expression generator that would read in a post-fix expression and convert it into an in-fix expression using an expression tree. The generator will also generate an output file with the three address instructions needed to calculate the generated in-fix expression.

## UML Diagram



## Test Cases

I performed basic functional testing to make sure my program could handle the expected use cases of the in-fix expression generator. Specifically I made sure that the generator could handle post-fix expressions with or without spaces between operators and detect invalid tokens.

### Test Case 1

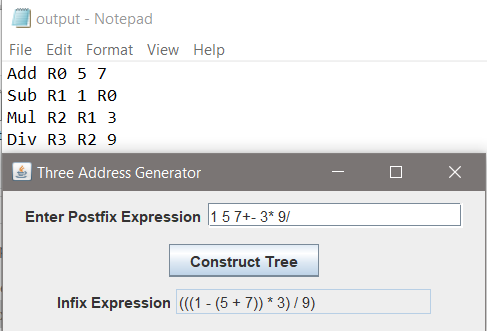
This test case inputted a post-fix expression containing no spaces and all the arithmetic operators. I inputted “1 5 7+- 3\* 9/”. The output was “(((1 - (5 + 7)) \* 3) / 9)” as expected. The outputted instructions were:

Add R0 5 7

Sub R1 1 R0

Mul R2 R1 3

Div R3 R2 9



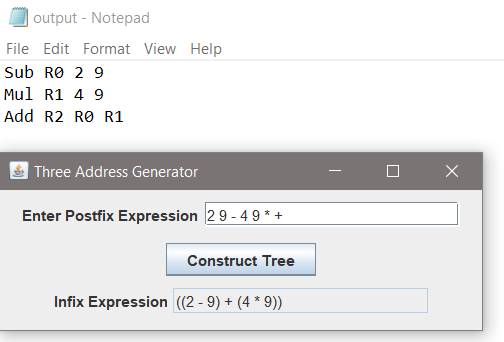
### Test Case 2

This test case I inputted a post-fix expression containing spaces in between the operators. I inputted “2 9 – 4 9 \* +”. The output was “((2 - 9) + (4 \* 9))” as expected. The outputted instructions were:

Sub R0 2 9

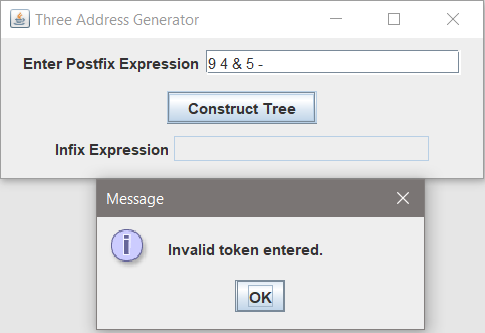
Mul R1 4 9

Add R2 R0 R1



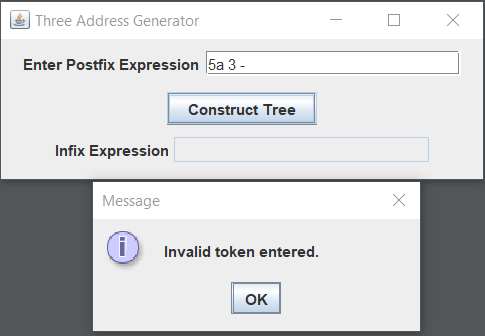
### Test Case 3

This test case I inputted an invalid operator in my expression. I inputted “9 4 & 5 –“ with an expected error pop-up. No output file is expected to be generated. This test case passed.



### Test Case 4

This test case I inputted an invalid token directly after a digit to ensure an error was thrown. No output file is expected to be generated. This test case passed.



## Lessons Learned

I wasn’t sure about one of the requirements for the project so I wanted to defend my decision here in the lessons learned section. I saw that a requirement was to be able to parse the post-fix expression correctly without spaces, but I realize that it puts me in a difficult spot if there are no spaces between integers. I would not be able to decipher without knowing the user’s intent whether or not 1234 was supposed to be one thousand two hundred thirty-four or 1, 2, 3, and 4. So I only accounted for no spaces in-between operands and operators and between operators and operators.

One of the other challenges I encountered from this project compared to last project was throwing a visible error for invalid tokens. In the previous project, I simply ignored any non-valid characters rather than actively checking for them. My solution wasn’t very elegant and involved having an empty if-statement to catch and ignore the spaces in my string array generated from the input.

Finally, I recognize that my method of creating the three-address instruction output file was very hacky and inefficient. I wasn’t able to think of a more efficient solution at this time and resorted to writing out all 4 cases of whether or not use the actual value or the register in the instructions to represent the node. If I were to do this again, I would try to build in a way to differentiate between leaf nodes and internal nodes in order to have more efficient code.