Kelly McCutchen
CMSC 350 6380 Data Structures and Analysis
June 17, 2018

Project 2: Expression Tree

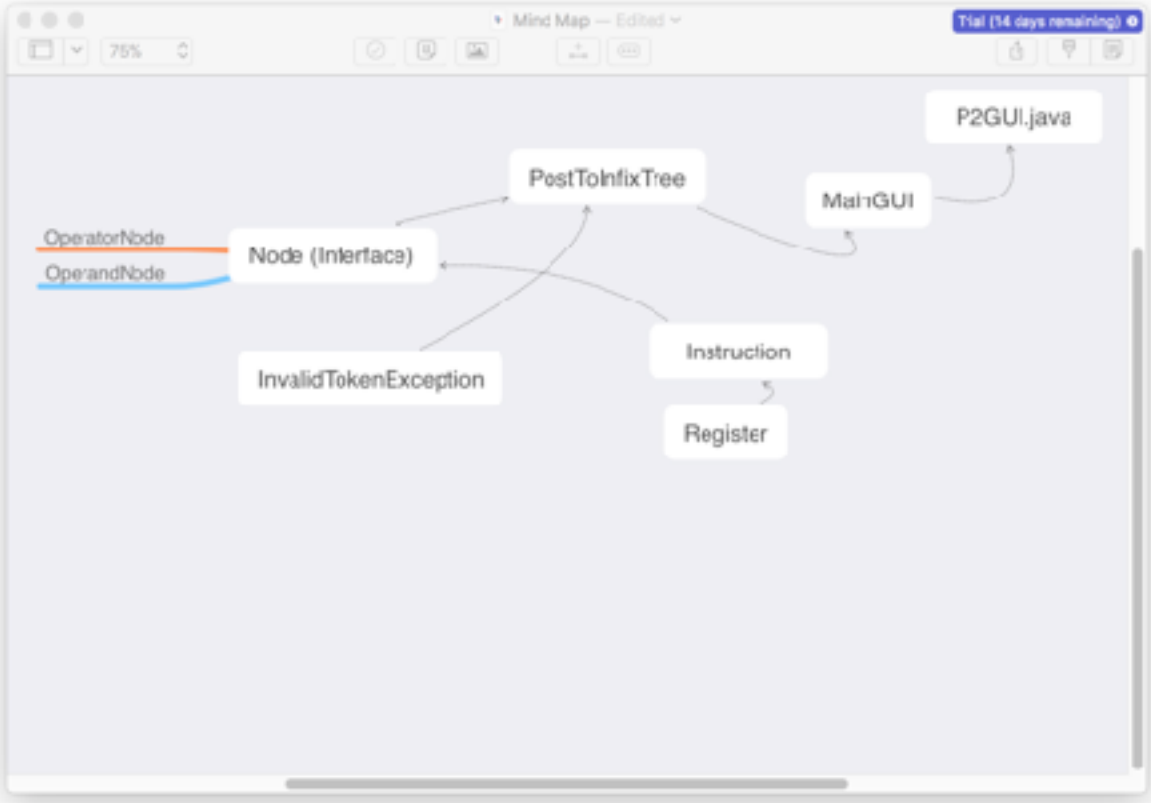
1.) Assumptions, Main Design Decisions, Error Handling

I assumed the input string would be formatted correctly aside from white space and incorrect tokens. This was a valuable assumption allowing the opportunity to focus on developing and implemented the postfix evaluator for the expression tree. Design decisions made were initially simple. I utilized the class structures given in the course content to develop the backbone of the Node architecture. This design was helpful (and informative) about the structure and layout of the program.

The program P2GUI initializes a subclass of JFrame, MainGUI. This GUI component is the main viewpoint into the program. MainGUI contains the textfields and buttons to interact with the program. The input text string is processed in an action handler managed through a JButton object. It is in this handler where I decided to create an instance of my tree constructor class PostToInfixTree.java. The class PostToInfix processes the input Postfix expression from the user after trimming white space and redefining as a character array list. A Node Stack for push/pop actions is implemented to create the nodal tree with a root node and leaf/children nodes of Operator and Operand nodes. This tree constructor class throws an InvalidTokenException, a custom exception class, in the event of incorrect tokens. Otherwise, it is fairly simple - it just pushes and pops nods from the stack based upon certain criteria. Operators are handled differently than operands which are simply pushed onto the stack. Operators pop two operands from the stack, create a new OperatorNode and then push this OperatorNode onto the stack.

In the implemented of the addressing portion of this program, I decided to implement a bubble-up algorithm. It is meant to process the tree from the bottom up with registers bubbling to the top. However, the design was overly complicated and while the code executed, it prints incorrect results to the output text file. This section of the code creates an Instruction and a Register class. Instructions are created based on whether there are two operator nodes in the current operator node, one of each, or two operands. In mapping out values, I determined it would be necessary to increment registers when OperatorNode children were each OperatorNodes, otherwise a single register would be sufficient to process these results. While I still think this is correct, the code inappropriately handles this bubble-up process for Register tracking.

2.) UML Diagram



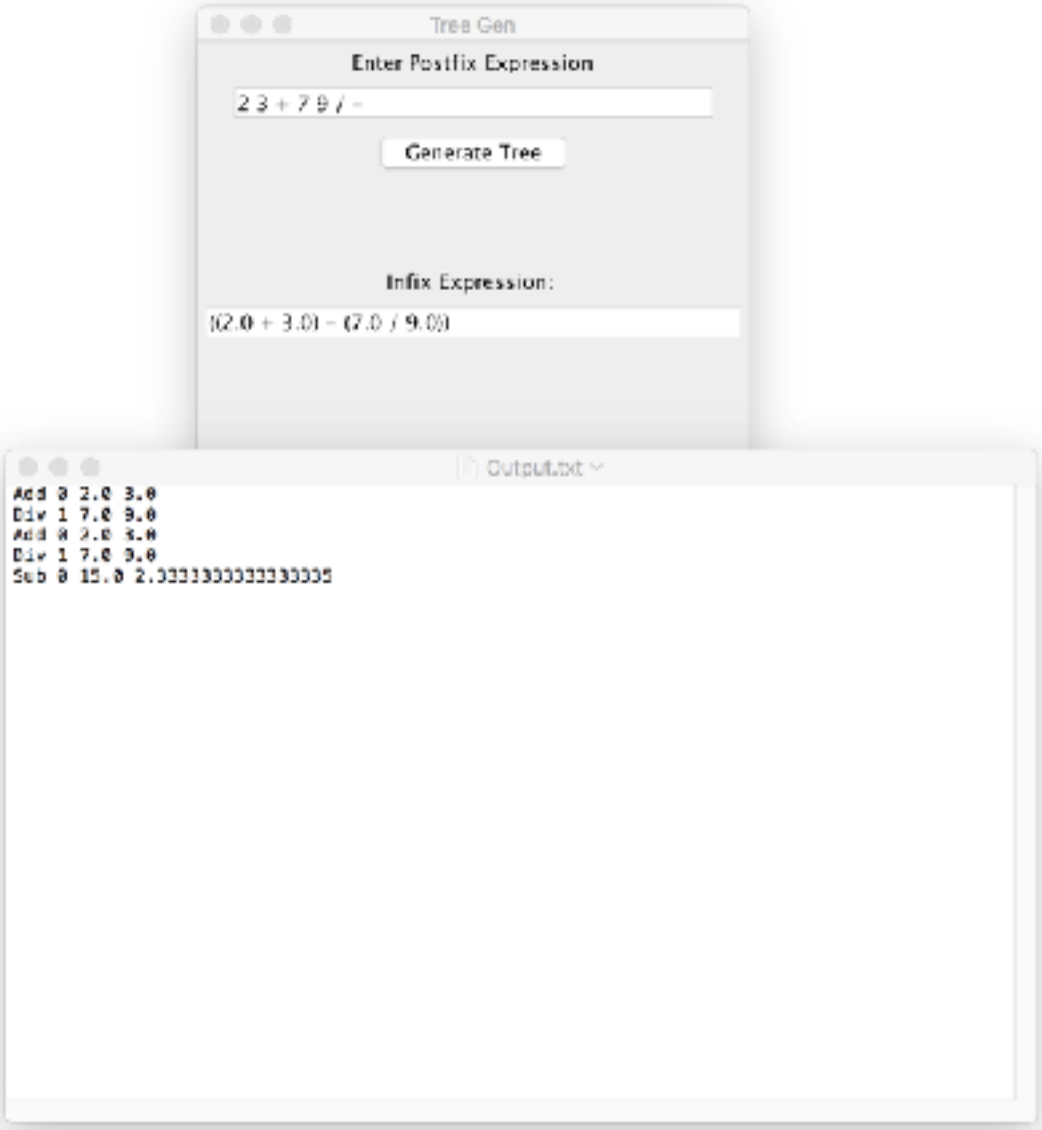
3.) Test Cases

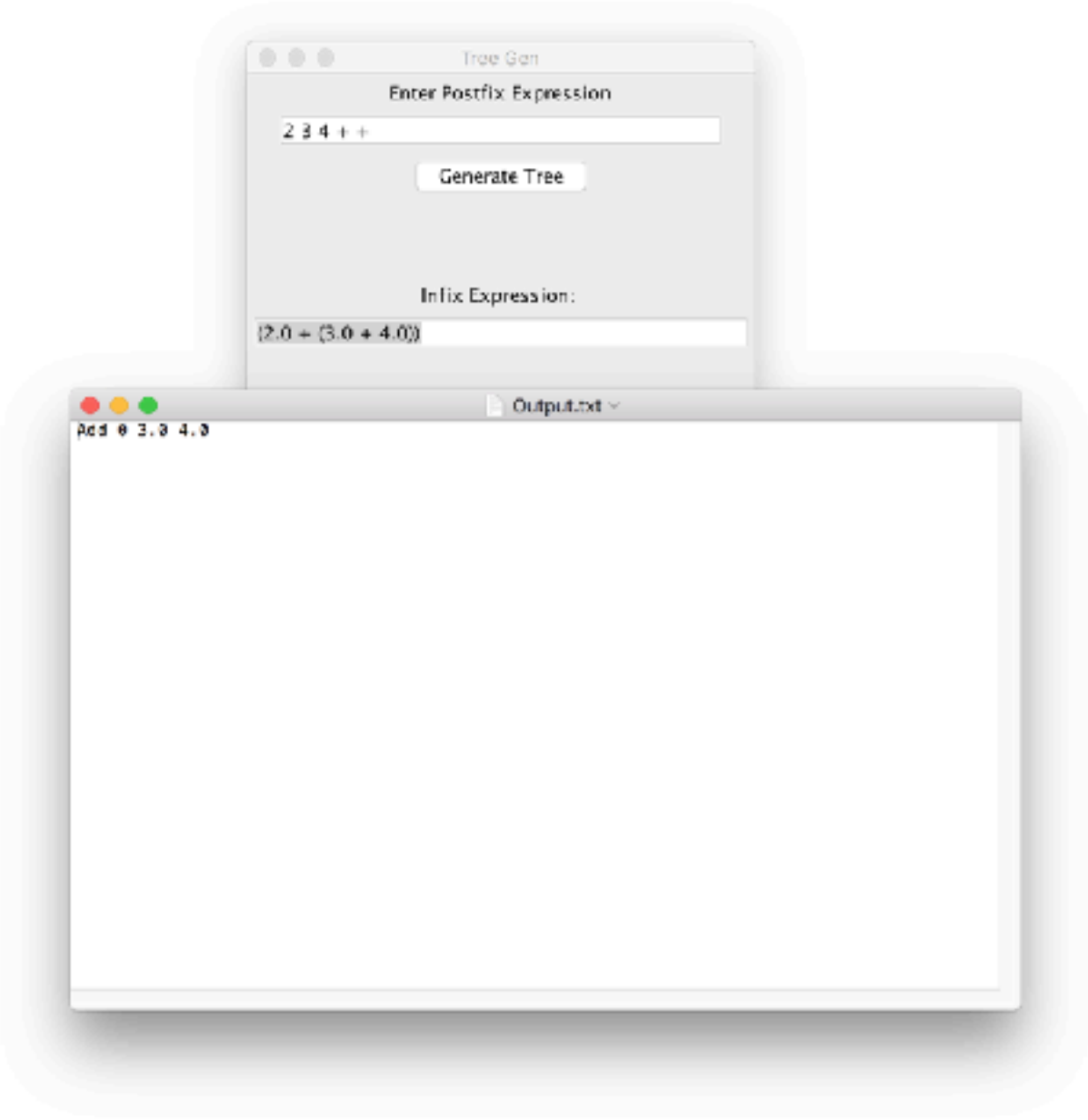
Test Aspect	Input Values	Expected Output	Actual Output	Pass/Fail
Accuracy	2 3 + 7 9 / -	((2.0 + 3.0) - (7.0 / 9.0))	((2.0 + 3.0) - (7.0 / 9.0))	Pass
Accuracy	2 3 4 + +	+ 2.0 + 3.0 4.0	+ 2.0 + 3.0 4.0	Pass
InvalidTokenException (begins with digit)	W 2e +	Invalid Character Output' Message	Invalid Character Output Messag	Pass
3-address format instructios	3 5 9 + - 2 3 * /	Add R0 5 9 Sub R1 3 R0 Mul R2 2 3 Div R3 R1 R2	Add 0 5.0 9.0 Mul 1 2.0 3.0 Sub 0 3.0 0 Mul 1 2.0 3.0 Div 0 -50.0 18.0	Fail

Test Aspect	Input Values	Expected Output	Actual Output	Pass/Fail
Accuracy	9 8 3 / +	(9.0 + (8.0 / 3.0))	(9.0 + (8.0 / 3.0))	Pass
White Space	2 3 + 7 9 /-	((2.0 + 3.0) - (7.0 / 9.0))	((2.0 + 3.0) - (7.0 / 9.0))	Pass
Test Invalid Operator	2 2 2 & &	Invalid Character Output' Message	Invalid Character Output Messag	Pass

4.) Screenshots

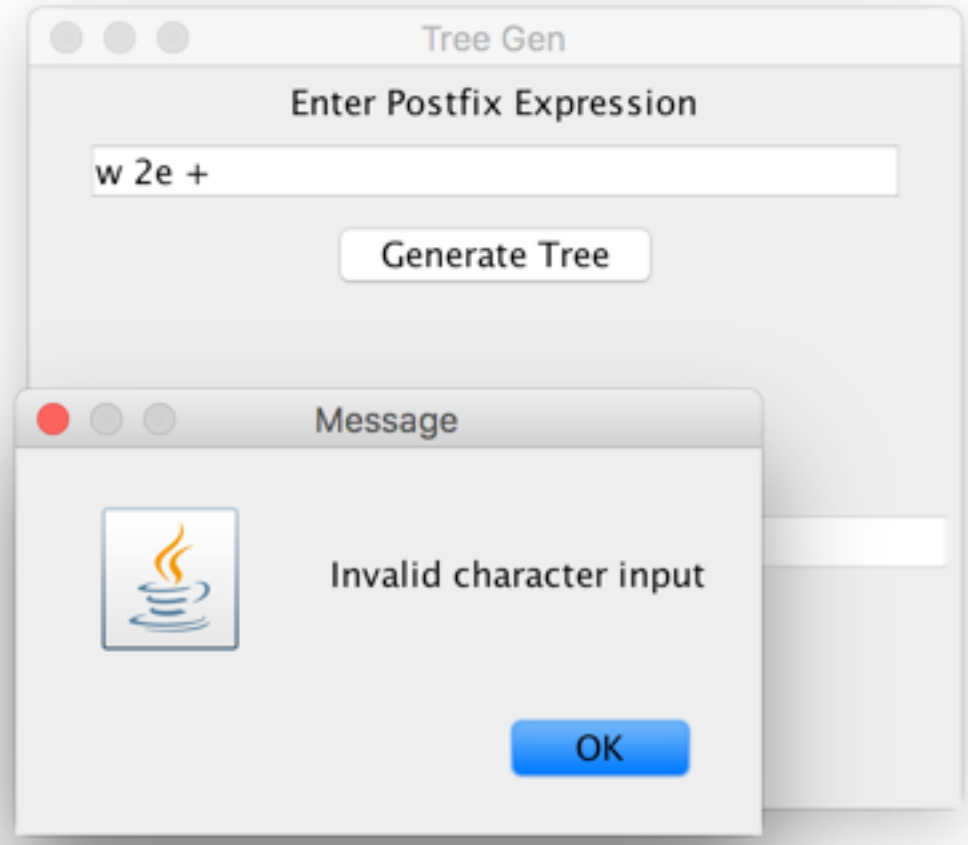
Case 1



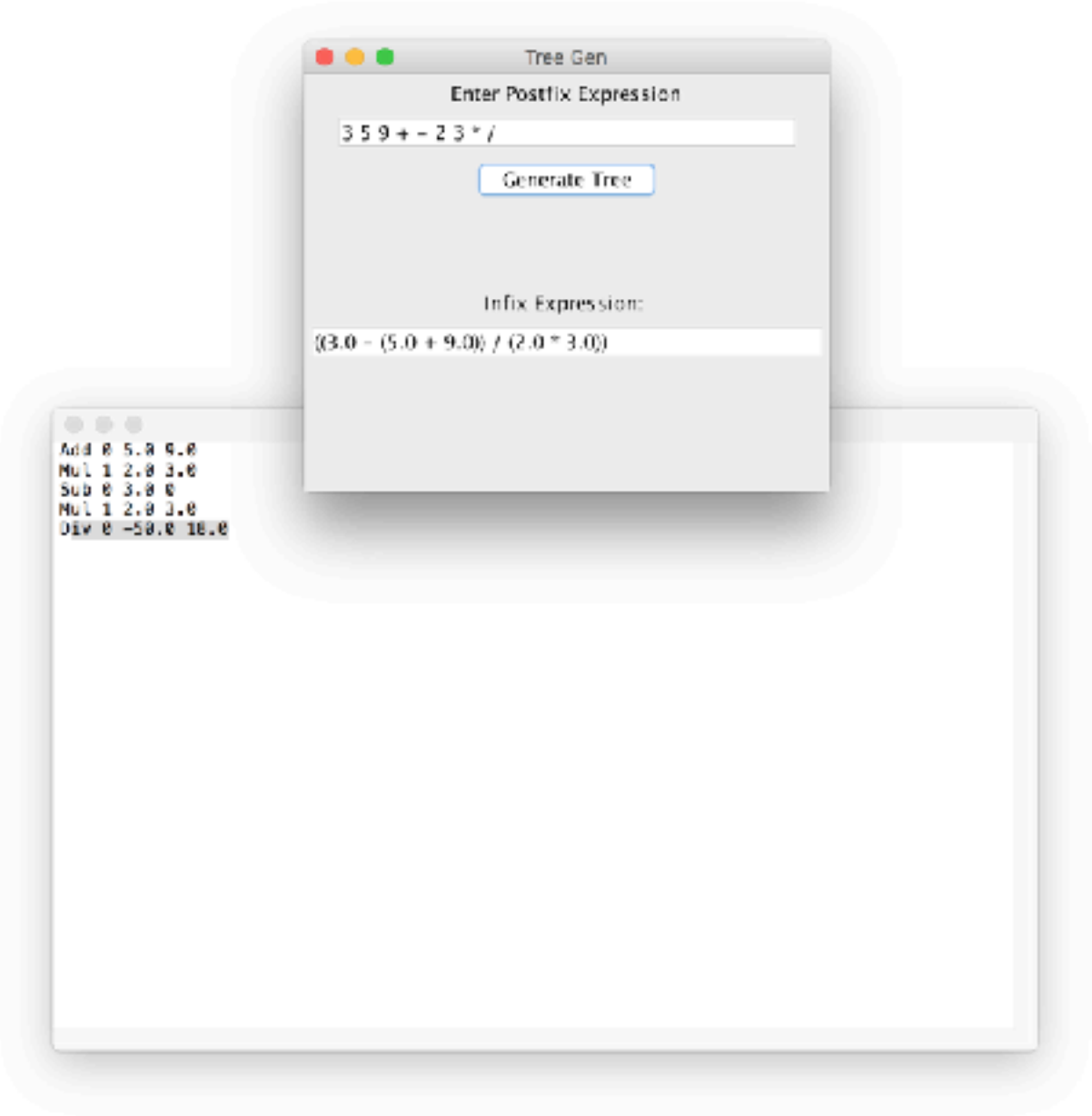

Tree Gen
Enter Postfix Expression
2 3 4 + +
Generate Tree
Infix Expression:
(2.0 + (3.0 + 4.0))
Output.txt
Add 0 3.0 4.0

Case 2

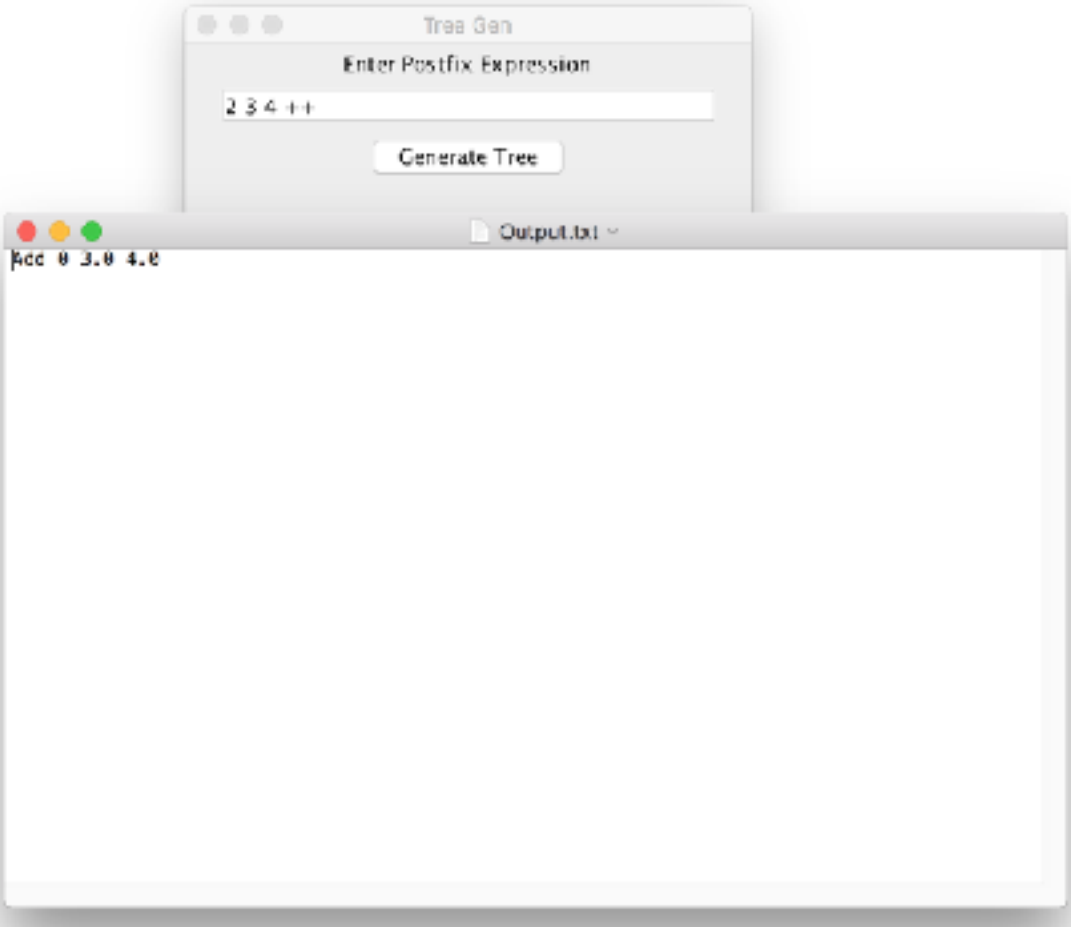
Case 3


Tree Gen
Enter Postfix Expression
w 2e +
Generate Tree
Message
Invalid character input
OK

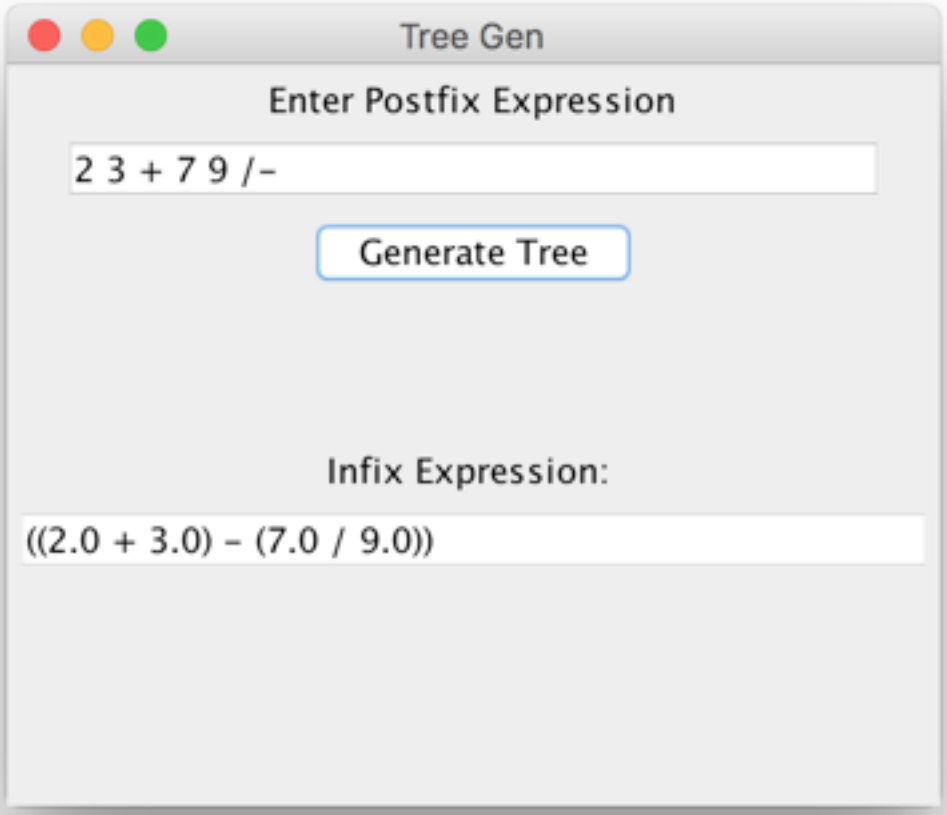
Case 4


Tree Gen
Enter Postfix Expression
3 5 9 + - 2 3 * /
Generate Tree
Infix Expression:
(3.0 - (5.0 + 9.0)) / (2.0 * 3.0)
Add 0 5.0 9.0
Mul 1 2.0 3.0
Sub 0 3.0 0
Mul 1 2.0 3.0
Div 0 -50.0 10.0

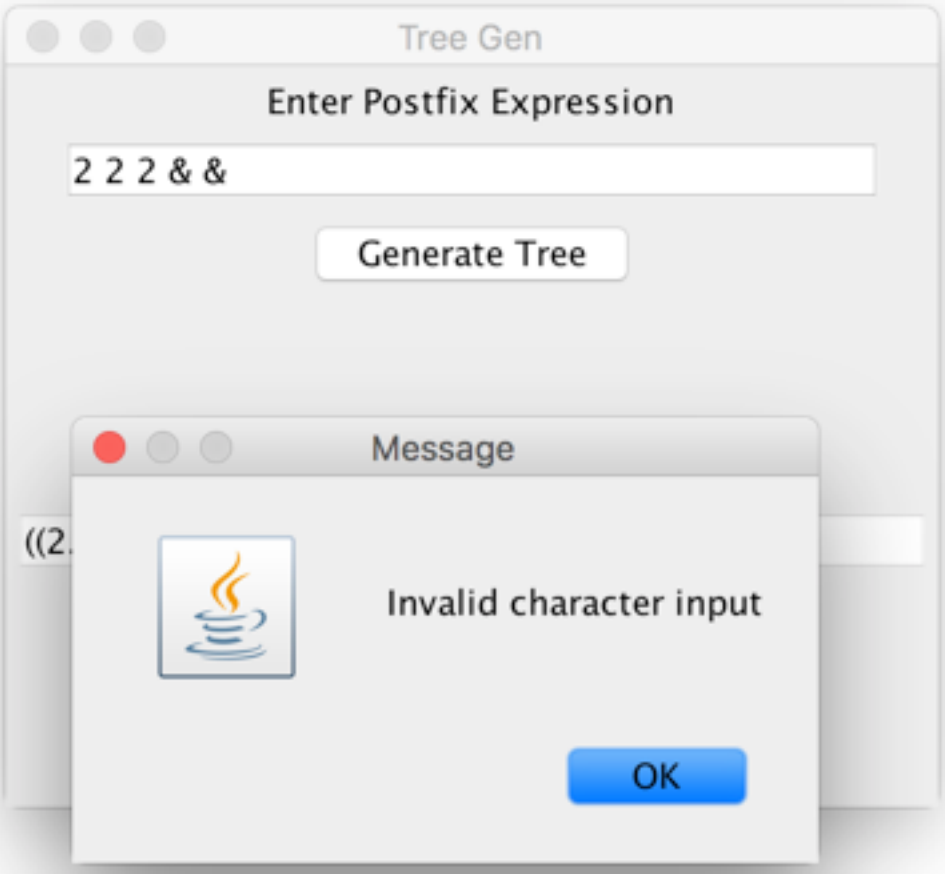
Case 5


Tree Gen
Enter Postfix Expression
2 3 4 ++
Generate Tree
Output.txt
Add 0 3.0 4.0

Case 6


Tree Gen
Enter Postfix Expression
2 3 + 7 9 /-
Generate Tree
Infix Expression:
((2.0 + 3.0) - (7.0 / 9.0))

Case 7


Tree Gen
Enter Postfix Expression
2 2 2 & &
Generate Tree
Message
Invalid character input
OK

5.) Lessons Learned

I learned to diagram trees during this project. In order to visualize the data structure and how its behavior should work, I had to draw out different versions of the tree, especially when trying to understand how to implement a bubble-up implementation for the register sorting. Though the algorithm is still faulty, I did learn to diagram and visualize the tree. Also, working with interfaces and abstract classes in this way was helpful and I now think I understand the inheritance relationships I did not fully understand before. When implementing the Node interface, I thought it was neat to use an interface to define the relationships of the OperatorNode and the OperandNodes without using a parent class. I did not understand interfaces could develop hierarchical relationships in this way. I thought they were simply functionally significant in order to define class behavior. Utilizing interfaces to design this tree will help me in further programming projects to identify constraints and develop data types.