Connor clawson, ryan prinster , and JAMES SMOTHERMAn

Team 1

  CPT-287-80-20/FA

Project 2:

Infix expression parser

***The Breakdown***

For this project, Team 1 worked methodically. That is, the project was small enough that we worked within the same file and divided each task into methods. Connor took the lead by creating both the group GitHub and the first draft of the readExpression() method. After an inspired night, Ryan worked through the logic and implantation of solving an infix expression by converting it into a postfix expression. The methods used for solving include: solve(), which takes an infix string and calls the other methods before neatly displaying the information; infixToPostfix(), which converted an infix equation into a postfix equation; postfixEval(), which evaluates a postfix equation; precedence(), which is a boolean value comparing if an operator has higher precedence; and checkPrecedence(), which finds the precedence value of an operator. Once the general math was figured out, Connor handled dividing by zero so that it does not cause a crash. After that, James rewrote the readExpression() method to return expressions in the format the expression solver now requested. Finally, Connor created main(), which looped trough an input file and fed each non-blank line into the expression reader and then into the problem solver. After the project was completed, James used the chat and change logs to finish the documentation: group report, git wiki, and git readme.

***Possible Improvements***

A separate main file could be used by instantiating ExpressionSolver in that main file for those who prefer to keep classes separate. While unsure if it is faster, the if statements checking the precedence could be replaced with a switch statement using breaks after each different precedence level.

***The Class(s)***

![Text

Description automatically generated]()

Diagram created using: <https://app.diagrams.net/>

***Test Cases***

1) In the first test case “23>=22+1” was read

Which output:

![A picture containing text

Description automatically generated]()

Output is as expected.

2) In the second test case “( 2 > 3 ) - 2” was read

Which output:

![Text, letter

Description automatically generated]()

Output is as expected.

3) In the third test case “3 / ( 6 \* 5 - 30 )” was read

Which output:



Output is as expected.

4) In the fourth test case “7 == 6” was read

Which output:

![A picture containing table

Description automatically generated]()

Output is as expected.

5) In the fifth test case “( 3 + 4 ) || 1” was read

Which output:

![A picture containing text

Description automatically generated]()

Output is as expected.