**CPT-287 Group Project Report**

**By: Sam Atienza, Ryan Schoonover, Seth Wolf**

**Project 2: Infix Expression Parser**

**System Design**

This program runs through a very basic main program, which only serves to read through a text file of expressions, titled “expression.txt”. From this point, the system runs through a while loop forcing it to continue until no expressions remain unsolved. This means that while the scanner has an expression that needs to be resolved, spaces are added first (if need be) to normalize a format for the program to solve. From this point, the expression is converted from Infix to Postfix as the team decided postfix would be the easiest format to evaluate. Finally, the Postfix expression is evaluated, resulting in the solution. This system mainly utilizes stacked array lists to solve an expression, while also occasionally using arrays. One stacked array list of strings is used in the InfixToPostfix class, titled S. This was used in union with the string builder postfix to assist in the conversion of an expression from infix form to postfix form. Also in the InfixToPostfix class is a string array titled tokens. This is what our team used to keep track of the tokens involved with the performance of an operation to solve an expression. In the postfixEval class, our team used another string array called tokens. Another stacked array list, this time comprising of doubles, is also used in the postfixEval class. This stacked array list is titled oppStack, and was used in tandem with if statements to solve the postfix expression. The if else statements were used to identify what operations must be performed, and when complete the result was pushed to oppStack. Finally, the solution to the expression was pulled from oppStack, and printed to the console.

**Graphical user interface, application, Teams

Description automatically generatedUML Diagram**

**Test Cases #1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Step No.** | **Operation Performed** | **Expected Output** | **Actual Output** |
| **1** | Power (precedence 7) | 2+2^2\*3 | 2+2^2\*3 |
| **2** | Arithmetic (Multiplication, precedence 6) | 2+4\*3 | 2+4\*3 |
| **3** | Arithmetic (Addition, precedence 5) | 2+12 | 2+12 |
| **4** | **SOLVED** | 14 | 14 |

**Test Case #2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Step No.** | **Operation Performed** | **Expected Output** | **Actual Output** |
| **1** | Power (precedence 7) | 2%2+2^2-5\*3^2) | 2%2+2^2-5\*3^2) |
| **2** | Arithmetic (Multiplication, Modulo, precedence 6) | 2%2+4-5\*(9) | 2%2+4-5\*(9) |
| **3** | Arithmetic (Addition, Subtraction, precedence 6) | 0+4-45 | 0+4-45 |
| **4** | **SOLVED** | -41 | -41 |

**Contributions and Future Improvements**

**Sam Atienza –** InfixToPostfix class, Project Repository

**Ryan Schoonover –** PostfixEvaluation class, StackedArrayList class, Precedence Class

**Seth Wolf –** Main class, AddSpaces class, PostfixEvaluation class

While the system was built to be as efficient as possible, room for improvement always exists in systems like these. One way our team could have improved this program would be through the avoidance of brute force if-else statements. Our team had trouble brainstorming around this issue, thus the result being brute force. In future versions of this program, our team could improve this program by utilizing a scanner and a loop to detect for more operations to be performed, hopefully reducing lines of code needed and resulting in a more efficient system. Despite this, our team still built the program to run as efficiently as possible, while also still functioning properly.