**CPT-287 Group Project Report**

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

**Project 3A: Binary Tree Infix Expression Parser**

**System Design**

This program was created to take a user inputted infix expression, create a binary expression tree from that expression, and print the result to the console. Due to similarities between this project and our last one, our team was able to make use of a few pre-created classes discussed in the last project report, including AddSpaces, PostFixEvaluation, Precedence andStackedArrayList. While this did save time working on the project, a few of these classes had to be modified to fit the needs of this program. Along with the classes previously mentioned, our team created the Main, Conversion, StackC, StackN, Tree, and treeNode classes. This program utilizes a few key data structures, including stacks to help solve the expression. Each stack, (C and N) contain both a top and max int data fields, while also containing a character array. Along with this, both stacks also feature the push, pop, and isEmpty methods. A data structure in this program is the Tree class, responsible for the construction of our tree. Using the Tree class, our tree is constructed through use of stack N and conversion, which helps the tree decide where to push/pop the nodes depending on the data contained in them. Finally, our tree is made up of the class treeNode, which features a left and right treeNode child, and char data. The Main method in our program takes in user input, converts it to a binary expression tree, then outputs the solution to the console for the user to see.

**UML Diagram**

**Graphical user interface, application

Description automatically generated**

**Test Case #1**

|  |  |  |  |
| --- | --- | --- | --- |
| **Step No.** | **Equation Steps** | **Steps Taken** | **Output** |
| **1** | Construct expression tree | Get user input | Construction of expression tree |
| **2** | 2+2^2\*3 | Gets user input | 2+2^2\*3 |
| **3** | 2+4\*3 | Evaluates 2^2 (4) | 2+4\*3 |
| **4** | 2+12 | Evaluates 4 \* 3 (12) | 2+12 |
| **5** | **14** | Evaluates 2+12 (14) | 14 |
| **6** | **SOLVED** | Done | 14 |

**Test Case #2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Step No.** | **Equation Steps** | **Steps Taken** | **Output** |
| **1** | Construct expression tree | Get user input | Construction of expression tree |
| **1** | 2%2+2^2-5\*3^2) | Evaluates 3^2 (9), and 2^2 (4) | 2%2+2^2-5\*3^2) |
| **2** | 2%2+4-5\*(9) | Evaluates 2%2 (0), and 5\*9 (45) | 2%2+4-5\*(9) |
| **3** | 0+4-45 | Evaluates 0+4-45 | 0+4-45 |
| **4** | **-41** | Evaluates 4-45 | -41 |
| **5** | **SOLVED** | Done | -41 |

**Contributions and Future Improvements**

**Sam Atienza –** Main class, treeNode class, Project Repository

**Ryan Schoonover –** Trees class, Conversion class, Precedence Class

**Seth Wolf –** Main class, AddSpaces class, PostfixEvaluation class, StackC/StackN class, Conversion class

While the system was built to be as efficient as possible, room for improvement always exists in systems like these. Due to the similarities of our last project, our program suffers the same effect as the program utilizes brute force if statements in one of the classes. In future versions of this system, we could also implement a loop to continue asking users to input expressions until they exit the loop. This would not have been a difficult task and has already been created as we utilized a similar feature in Project 1. Another way we could have improved this system would have been to add some form of error checking. While this would have made the program more complex and most likely slowed it down just a tad, it would ensure that our program would always spit out correct answers to the console. An error detection method we could have used would be to sole the expression in a different manner, not using trees, and verify that the solutions we received through both ways of solving was the same.