**­­**

**Infix Expression Parser**

Adam Carmichael

Evan Coyler

Lindsey Erwin

The program starts by reading a list of infix expressions from an input file. The input file is formatted so that each infix expression is on a single line. The program reads one line at a time from the input file and then two functions are called to evaluate/calculate the expression.

The first function is used to convert the infix expression from the input file into a postfix expression. This function creates a stack, and a string builder to create the postfix expression. While the infix string is being parsed each “token” is checked to see whether it goes to the postfix string or the stack. The stack is used to store values inside parentheses to ensure those values inside the parentheses get evaluated first, and it is used to store the operators. The function also checks the precedence of operators to make sure they are in the correct order so that the postfix expression can be calculated correctly. The precedence of operators is determined by another function that sets operators with the highest precedence with a higher integer number (The higher the precedence the higher the integer number). Once the entire infix expression has been parsed the function returns the postfix expression, and the second function is called to evaluate the postfix expression.

The second function also creates a stack to assist with the calculation of the postfix expressions. The function uses a for loop to iterate through the characters of the postfix expression. If the character is an operand, it gets pushed to the stack, and if the character is an operator the next two values in the stack get popped from the stack and evaluated with whatever the operator may be. In our function we use a switch case to determine how said operator is evaluated. Once the two values are evaluated then the result of that evaluation get pushed back to the stack. This process is repeated until the end of the postfix expression is reached, and it returns the calculated number of the postfix expression.

Stacks are used in these functions to ensure the order of the operands and operators so that they can be calculated correctly and calculated with the correct precedence.

An improvement on this program could be to have the methods that convert the expression to postfix and evaluate the postfix expression in a separate class to reduce the amount of code in the main program.

|  |  |
| --- | --- |
| contributions |  |
| project report | Adam Carmichael |
| Precedence function / main(driver) code | Adam Carmichael |
| infixtopostfix function | Lindsey Erwin |
| evalualtepostfix function | Evan Colyer |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case** | **Description** | **Input** | **Expected Result** | **Actual Result** | **Pass/Fail** |
| **1** | Check result of the infix expression parser | 1+2\*3 | 7 | 7 | Pass |
| **2** | Check result of the infix expression parser | 2+2^2\*3 | 14 | 14 | Pass |
| **3** | Check result of the infix expression parser | (1+2)\*3 | 9 | 9 | Pass |

Diagram

Description automatically generated