Project Report:

Critical Thinking 4 – Infix Calculator

Alejandro Ricciardi

Colorado State University Global

CSC400: Data Structures and Algorithms

Professor: Hubert Pensado

September 9, 2024

Project Report:

Critical Thinking 4 – Infix Calculator

This documentation is part of the Critical Thinking 4 Assignment from CSC400: Data Structures and Algorithms at Colorado State University Global. This Project Report is an overview of the program's functionality and testing scenarios including console output screenshots. The program is coded in Java JDK-22 and named Critical Thinking 4 (Infix Calculator).

The Assignment Direction:

Create an Infix Calculator

Implement an infix calculator in Java that evaluates arithmetic expressions in infix notation. The program should support the basic arithmetic operations:

- addition (+)
- subtraction (-)
- multiplication (*)
- two division operations: (/ and %)

Additionally, the program should handle operands and display the final result.

Requirements:

1. Your java code.
2. Screenshots showing the test of your code, where the following should be tested:
    1. The program should handle both single-digit and multi-digit operands.
    2. The program should handle valid postfix expressions.
    3. Display an error message for invalid expressions.
    4. Display the result for valid expressions.

Example:

```
public class InfixCalculator {
    public int evaluateInfix(String infixExpression) {
        // Your implementation here
        // ...
        return 0; // Placeholder
    }

    public static void main(String[] args) {
        InfixCalculator calculator = new InfixCalculator();

        // Example 1: Valid Expression
        String expression1 = "(4+2)*3";
        System.out.println("Result 1: " + calculator.evaluateInfix(expression1));
```

```
        // Example 2: Valid Expression
        String expression2 = "5+(3*7)";
        System.out.println("Result 2: " + calculator.evaluateInfix(expression2));

        // Example 3: Invalid Expression
        String expression3 = "4+2*3"; // Missing parentheses
        System.out.println("Result 3: " + calculator.evaluateInfix(expression3));
    }
}
```

Sample Output:

```
Result 1: 18
Result 2: 26
Error: Invalid infix expression
```

Submit your completed assignment as a .java source code file.

⚠️ **My notes:**

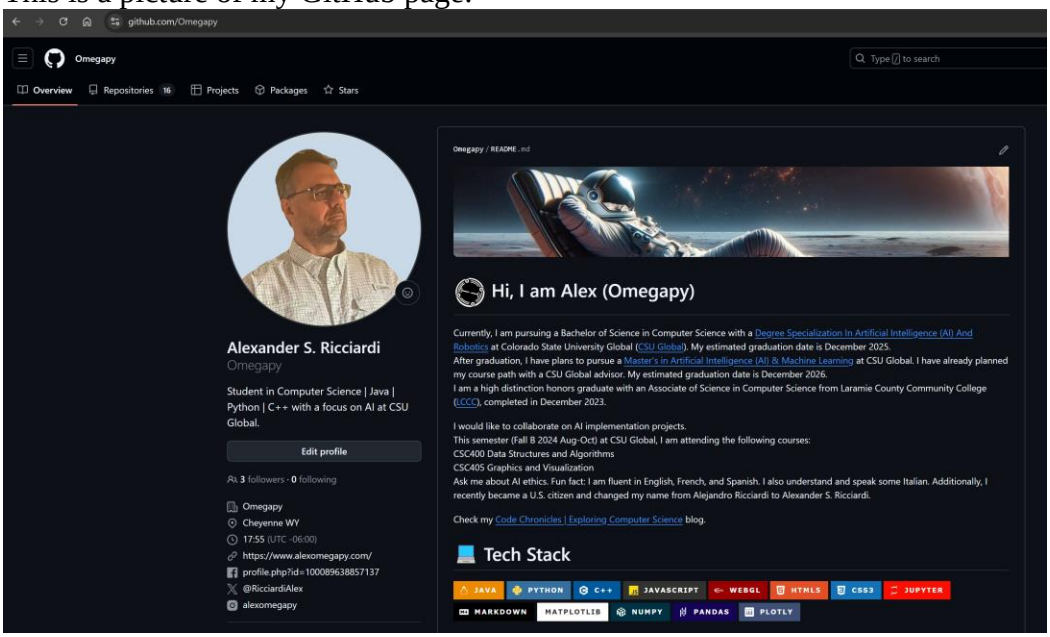
- In addition to the required arithmetic operations it handles exponents (^), decimal numbers, and parentheses.
- The program implements my Linked Stack ADT, a stack implementation that uses a linked list structure, a chain.

My Program Description:

- The program is an implementation of an Infix calculator that evaluates arithmetic expressions in infix notation.
- The program converts Infix expressions stored in a text file into Postfix expressions, then computes the Postfix expressions and displays the computation results.
- The program utilizes a Stack Abstract Data Structure (Stack ADT) to manage operators and operands when converting Infix expressions to Postfix form and during the evaluation of Postfix expressions.
- The Stack ADT is a linked list structure or chain using generic types.
  [element | next] -> [element | next] -> [element | next] -> null.

Git Repository

This is a picture of my GitHub page:



I use GitHub as my Distributed Version Control System (DVCS), the following is a link to my GitHub, Omegapy.
My GitHub repository that is used to store this assignment is named My-Academics-Portfolio and the link to this specific assignment is: https://github.com/Omegapy/My-Academics-Portfolio/tree/main/Data-Structures-and-Algorithms-CSC400/Critical-Thinking-4

Classes Description:

- **LinkedStack Class**
  A Stack ADT class that implements a stack using a linked list structure, a chain, to store elements. Each node contains data and a reference to the next node in the chain.
  [element | next] -> [element | next] -> [element | next] -> null.

- **InfixToPostfix Class**
  The InfixToPostfix class implements methods that convert infix arithmetic expressions into postfix expressions for easier evaluation. It supports multi-digit numbers, negative numbers, operators, and parentheses.

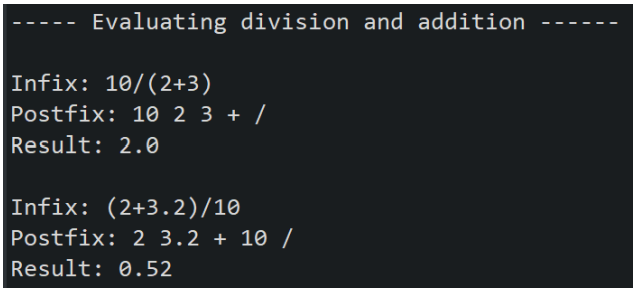
- **PostfixEvaluator Class**
  The PostfixEvaluator class implements methods that evaluate arithmetic expressions written in postfix notation. Performs postfix computations.

- **InfixCalculator Class**
  The InfixCalculator class tests and evaluates infix arithmetic expressions by converting them to postfix notation and then computing the results.

Screenshots

Figure 1

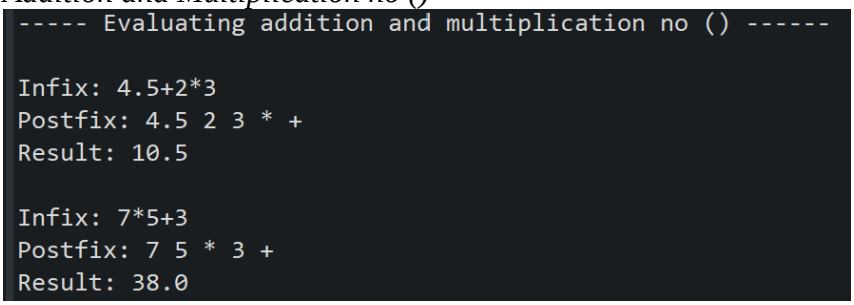
Division and Addition



Note: This demonstrates that parentheses have the highest precedence.

Figure 2

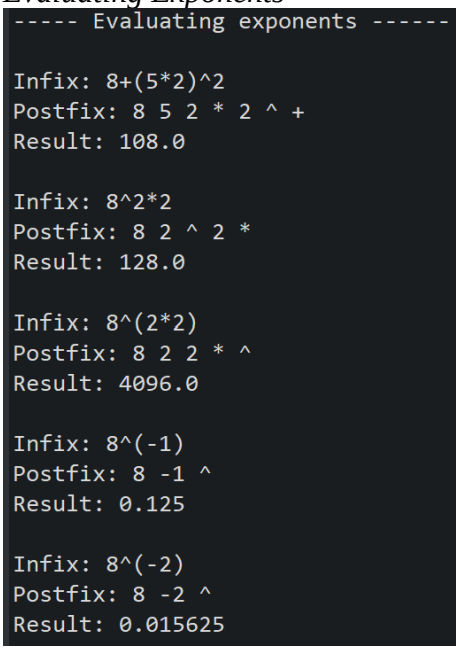
Addition and Multiplication no ()



Note: This demonstrates the precedence of multiplication over addition.

Figure 3

Evaluating Exponents



Note: This demonstrates that exponents have precedence over addition and multiplication, it also demonstrates the use of negative exponents.

Figure 4

Multiplication, Addition, Subtraction, and Negative Numbers

```
----- Evaluating multiplication, addition, subtraction, and negative numbers ------

Infix: 6*(4+5)-2
Postfix: 6 4 5 + * 2 -
Result: 52.0

Infix: (-6)*(4+(-5))-2
Postfix: -6 4 -5 + * 2 -
Result: 4.0
```

Note: This demonstrates the use of parentheses when using multiplication, addition, subtraction, and negative numbers.

Figure 5

Final Count Check

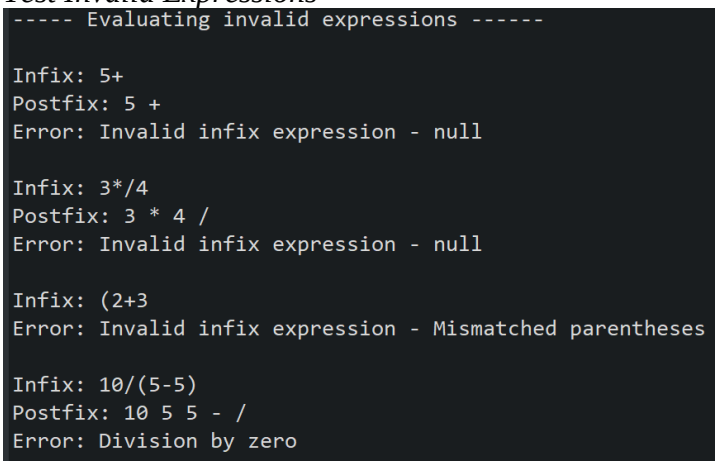
```
----- Evaluating modulus -----

Infix: 10%3+2^3
Postfix: 10 3 % 2 3 ^ +
Result: 9.0

Infix: 2^3+10%3
Postfix: 2 3 ^ 10 3 % +
Result: 9.0

Note that Modulus occupies the same place in the order of operations as multiplication and division

Infix: 2^3*(10%3)
Postfix: 2 3 ^ 10 3 % *
Result: 8.0

Infix: 10%3*2^3
Postfix: 10 3 % 2 3 ^ *
Result: 8.0

Infix: (10%3)*2^3
Postfix: 10 3 % 2 3 ^ *
Result: 8.0

Infix: 2^3*10%3
Postfix: 2 3 ^ 10 * 3 %
Result: 2.0
```

Note: This demonstrates the use of modulus and that it occupies the same place in the order of operations as multiplication and division. The postfix expression ‘2 3 ^ 10 * 3 %’ is the same as ‘((2^8)*10)%3’ infix expression.

Continue next page

Figure 6

Test Invalid Expressions



Note: This demonstrates invalid expressions.

As shown in Figure 1 through Figure 6 the program runs without any issues displaying the correct outputs as expected.