University of Missouri-Kansas City

CS 303 Data Structures

Section 2

Project 1

Infix Expression Evaluating

by

Tuyen Vu

1. **Acknowledgment**

This project is about evaluating an infix expression directly by combine the converting-infix-to-postfix algorithm and evaluating-postfix algorithm.

Firstly, to convert infix to postfix, I created a stack called operatorStack to hold the operators in the expression. Also, there is a variable named tempPosfixString to store the postfix expression. Here are the rules when converting infix to postfix of my program:

-If they are operands, add them to tempPosfixString

-If the top of operatorStack is \* or /, add it to tempPosfixString then pop it.

Ex: 2\*3-3 becomes 2 3 \* 3 –

Ex: 2\*3/2 becomes 2 3 \* 2 /

-If the close parenthesis arrives, add all the operator before the open parenthesis to tempPosfixString while popping them.

-If the top of operatorStack is \* or /, add it to tempPosfixString then pop it.

Ex: 3+10-(2+6/2) becomes 3 10 2 6 2 / + - +

Ex: 10+2\*(3+4-1)+5 becomes 10 2 3 4 1 - + \* 5 + +

-Else, push operators or left parenthesis to operatorStack

-Before pushing, after the odd number of operator –, all the operators + and – will be changed to – and +. Operators – outside the parenthesis only affect outside operators, and operators – inside the parenthesis only affect inside operators.

Ex: 10-3+4-2\*(3+4-1)+4 becomes 10 3 4 2 3 4 1 - + \* 4 - - - -

Ex: 10-3+4-2\*(3-4+1)+4 becomes 10 3 4 2 3 4 1 - - \* 4 - - - -

-If the operatorStack is not empty in the end, add all the remaining operators to tempPosfixString while popping them.

Secondly, to evaluate the posfix expression, I created a stack called operandStack to store the operands. All the operands are pushed to the stack as they arrive. When an operator arrives, operandStack will pop the top two operands and perform the arithmetic calculation. The result is push to the stack as a new operand.

1. **UML Diagram**

|  |
| --- |
| Calculator |
| -tempPosfixString: string |
| **+**Calculator  +pop\_operator(): void  +infix\_to\_posfix(input:string): void  +get\_posfix\_string(): string  +pos\_calculator(): double |

|  |
| --- |
| Stack<char> |

1

operatorStack

|  |
| --- |
| Stack<double> |

1

operandStack

1. **Time Complexity**

The Big O for infix-to-posfix function is the same as evaluating-posfix fuction, which is O(n) where n is the length of the arithmetic expression.

1. **Output**

