資料結構
Data Structure

HW02

Group Members

學號 1:113AB0014 姓名 1:曾致嘉

學號 2:113310223 姓名 2:張智凡

HW02-Q1. Introduction

The goal of this projects is to design and implement a fully functional calculator using Stack and Linked List, to perform arithmetic operations such as addition, subtraction, multiplication and division. The calculator take inputs in infix notation, convert the inputs to post-infix notation using a stack based algorithm and then evaluate the operation. Using data structures in this project makes it difficult-less to handle data. Stacks manage operators and parentheses, convert and evaluate expressions, while Linked lists gives dynamic memory allocation and performance efficient element to customize of a stack implementation. So, this project aims to increase the understanding of some data structures concepts like stacks, linked list and infix to post-fix algorithm etc...

Hw02-Q2. Design and Implementation

Data Structures Used

The different data structures used inside the code are:

- **Linked list:** This allowed dynamic resizing and efficient memory usage.
- **Stack:** Supports operations such as push(), pop(), peek(), isempty().
- We also used an array to store postfix expressions while implementing the conversion of infix to postfix algorithm.

Algorithm Description

- **Infix to Postfix conversion algorithm:** First, we create a stack. For each element from the input, if it is an operand, output it. If it is a right parentheses, pop and outputs stack elements until we pop a left parenthesis. Otherwise, if it is an operator or left parenthesis, pop and output stack elements until we find either an element with lower priority or a left parenthesis or the stack is empty.

- **Postfix evaluation algorithm:** First initialize an empty stack. For each input, if it is a digital number, push it to the stack. Instead, if it is an operator, pop 2 numbers from the stack and then perform the operation for these 2 numbers and store it in the stack. We continue the process until we get only one element inside the stack. Finally, we return the top of the stack as result.

- **Input parsing approach:** In this approach, the string is taken as an input and parsed character by character. Multi-digit numbers and decimals are processed by checking for digit characters and forming tokens.

Key features of the calculator

The calculator supports basics operations such as Addition (+), Subtraction (-), Multiplication (*), Division (/), Exponentiation (^) and modulus (%). It also uses stack to handle terms with parentheses properly during the conversion of infix to postfix. It can also handle basic errors such as division by zero and unmatched parentheses. For additional feature, we used Qt software to design a simple user interface for better interaction.

HW02-Q3. Analysis

Complexity Analysis

- **Conversion algorithm:** During the conversion each character is processed once. For a string of length n, the outer loop iterates n times. Inside this loop, for each stack operations, they processed n times because the function traverses nearly the entire linked list to find the second to last node. The worst case is when we have to perform O(n) stack operations. So, the time complexity for this algorithm is is O(n*n) = $O(n^2)$. (inefficient, we know).

- **Evaluation algorithm:** If n is the number of characters in the postfix string, each character is processed once, So for processing n characters in the outer loop we took n times. For stack operations such as push, pop, top, they are constant time, i-e O(1). So, the complexity time is O(n).

- Performance considerations: Actually, we implement a less inefficient stack with operation runtime n. each operations should have been processed once, i.e O(1). That causes the infix to postfix algorithm take more runtime operations, n^2 times instead of n times. We will discuss a potential amelioration in the possible improvement part.

Challenges and Solutions

The challenging part was to evaluate unary operations such as -5+3. So we use a boolean flag (**expectUnary**) to track when a unary operator is expected. Every time the flag expect a unary operator, we insert an implicit zero in front of it. So if the expectUnary flag expect -5 it will be like 0 – 5 = -5.

Testing Strategy

We tested the project using expressions with various logic and combinations of operators, numbers and parentheses.

Input Expression and **Expected Output:**

3+4-2	5.0
2*3+4	10.0
2*(3+4)	14.0
10/2+3	8.0
2^3+1	9.0
25%7+10	14.0
(2+3)*(4-1)	15.0
2.5+3.5*2	9.5

Edges cases considered

Edge cases	**results**
Empty input	0.0
5	0
5/0	inf
((2-3)*(5+4))	-9.0

The calculator handles empty input, single number input, division by zero and nested parentheses properly. The program does not crashed in neither of these test.

HW02-Q4. Results and Discussion

Even though the use of a stack with runtime $O(n^2)$ the program executes instantly. But, we noted the calculator handles some invalid input like 4+*3 and output zero, but an error message could have been better. This is not really a limitation, but it is not common to have to evaluate this kind of operation.

Possible Improvements

Some possible improvements or features that can be added are:

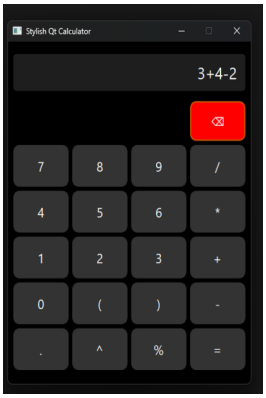
1. Handling errors more properly. Instead of output 0, an error message could have tell the user to check the expression.
2. Implementing stacks for running constant time operations instead of n times.
3. Additional functions such as Logarithms and Trigonometric for more specific tasks.

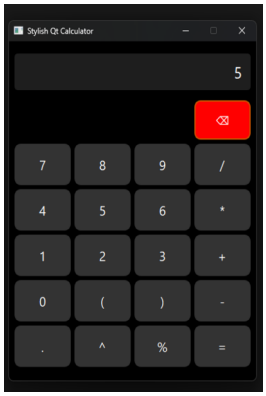
Conclusion

The goal of this project was to build a calculator that parses and evaluate arithmetic expressions entered in an infix form. Implementing this calculator increases our understanding of fundamentals data structures such as Stack, Linked list and their supporting operations such as insert and delete. In summary, The project demonstrates how a stack using linked list can be used to convert infix expression to postfix and then evaluate the expression.

References

Sample Output





We used Qt software to design a simple user interface for better interaction. So User need to install QT software. Follow this step to run the project:

1. Launch Qt Creator.

2. Click on “Open Project” from the welcome screen or go to File → Open File or Project….
3. Navigate to your project folder.
4. Select: -CmakeLists.txt
5. Click Open.

You should have the header mainwindow.cpp, Mainwindow.h, expression_evaluator.cpp, expression_evaluator.h, main.cpp automatically

Now you can run the project.

NB: User may also create a new project and the create the header file, and paste the codes.

Try the given expressions found on **Testing Strategy** part.

Tools Used:

1. https://chatgpt.com/share/680e0256-9060-8010-a59b-c43f7040ab52
2. https://chatgpt.com/share/68207a8a-cf28-8002-96e1-b3e359f1fe21
3. Data Structures And Algorithms Made Easy By Narasimha Karumanchi

Git link: https://github.com/extrateres260/1132_DataStructure-Lab01-Lab01_Q1.cpp/tree/main