

Project 1

COP4530- Data Structures

Linked List Arithmetic

September 17, 2024

Instructions

This project implements a templated linked list data structure for-

Expression Validation: Before evaluating the expression, the program validates the input linked list to ensure it represents a valid arithmetic expression. It checks for proper alternation of digits, decimal points, and operations, as well as the presence of valid operation characters. Every single keystroke is considered one of the nodes for a linked list.

Arithmetic Evaluation: The core functionality of the project is the evaluation of the arithmetic expression represented by the linked list. The program traverses the linked list, reconstructs the float values from individual digits and decimal points, and performs the specified operations (+, -, *, /) on the operand (and returns a float, given that the expression is valid). The program will consider every single keystroke to be a new Node of an instance of the class `LinkedCalc`.

Example-

```
LinkedCalc<char> calc1;
calc1.insert('1');
calc1.insert('+');
calc1.insert('2');
calc1.validateExpression();//returns True
calc1.evaluateExpression();//returns 3.0f (float)
```

Methods

`validateExpression()`

The `validateExpression` method is a member function of the `LinkedList` class in the `Linked List Arithmetic` project. Its purpose is to validate the expression represented by the linked list to ensure that it follows the correct format and adheres to the rules of arithmetic expressions.

Steps-

Single pass through the list
Track state (last seen: digit, operator, or decimal point)
Ensure proper alternation and valid characters
Check decimal point usage
Verify expression starts and ends correctly

evaluateExpression()

- The evaluateExpression method is another member function of the LinkedList class in this project. Its purpose is to evaluate the arithmetic expression represented by the linked list and calculate the final result. Invalid expression conditions- 1. Consecutive decimal points (with no digits between them). 2. Consecutive operators (with no digits between them). 3. No digits following an operator.

Steps-

Two-pass approach:
a) First pass: Handle multiplication and division
b) Second pass: Handle addition and subtraction

Use helper function to convert digit sequences to floats

First pass (multiplication and division):

Traverse the list-

When encountering '*' or '/', perform the operation immediately

Replace the operator and second operand with the result

Second pass (addition and subtraction):

Traverse the modified list-

Perform additions and subtractions in order

Example interaction

See `tester.cpp` for details.

Assumptions

- A valid input will contain a series of floats or integers joined by the operators(+, -, *, and /).
- The output (for a valid expression) will always be a positive float.
- You can assume that the divisor will never be 0.

Order of operations

- a. Division, from left to right
- b. Multiplication, from left to right
- c. Addition, from left to right
- d. Subtraction, from left to right

Included Files

1. linked_calc.cpp
2. linked_calc.hpp
3. tester.cpp

How to handle segfaults?

1. Link 1
2. Link 2.

Implementation instructions

1. You are allowed to use `std::string` and `std::vector` for this project. You are not allowed to use `std::list`.
2. You can use as many helper functions as you see fit. The `isdigit()` function is already implemented for you.
3. Your code must be accompanied by a Makefile. A tutorial for Makefile's will be uploaded to Canvas.

Submission instructions

Compress all the files together and submit them to Canvas. Name the file- Project 1-F24.zip

Rubric

- 10 test cases*9 =90
- Proper documentation (comments explaining the logic/code)= 10