Said Haji

University of Missouri - Kansas City

CS201 – Data Structures

Project Report

Title: Infix Expression Evaluator
Course: Data Structures – Project 1a
Instructor: Syed Jawad
Date Submitted: May 9, 25

Infix Expression Evaluator

Data Structures – Project 1A

Assumptions

This project assumes all expressions are valid C++-style infix expressions containing integers, logical, and arithmetic operators. The following assumptions were made for implementation:

- **Integer-Only Operands**: All input operands are assumed to be integers. Floating-point or alphanumeric inputs are not handled.
- **No Runtime Input**: Expressions are hardcoded within the main() function rather than accepted from the user at runtime.
- **Unary Operator Clarification**: Unary - (negative numbers) is internally replaced with the keyword "neg" to differentiate it from binary subtraction during parsing and evaluation.
- **Boolean Evaluation**: Logical operations and comparisons return integers—true is treated as 1 and false as 0—as per standard C++ implicit conversion rules.
- **Error Handling Strategy**: The program is designed to detect and report only the **first syntax or runtime error** encountered in an expression, providing an informative message and the character index (where possible).
- **Whitespace Tolerance**: Extra whitespace in the expression is ignored, and expressions with or without spaces are treated equally.

UML Class Diagram

```
+-------------------------------+
|      MathLogicEvaluator      |
+-------------------------------+
| + evaluate(string): int      |
|                              |
| - tokenizeExpression(string)  |
| - validateTokenSequence(vec)  |
| - convertToPostfix(vec)       |
| - evaluatePostfixExpression() |
| - calculateBinaryOperation()  |
| - calculateUnaryOperation()   |
| - isUnaryOperator(string): bool |
| - isRightAssociative(string): bool |
+-------------------------------+
```

- The class is designed for **modularity** and clarity, separating concerns such as parsing, validation, conversion, and evaluation.

- All helpers are encapsulated as private methods, while evaluate() is the only public entry point, simplifying usage in main().

Efficiency of Algorithms

The evaluator performs a sequence of linear-time operations, optimized using classical parsing techniques:

🔹 **Tokenization – O(n)**

- Each character of the expression is visited exactly once.
- Groups digits into multi-digit numbers during traversal.
- Whitespace is ignored.
- Operator lookup is constant-time using a unordered_map.

🔹 **Validation – O(n)**

- Token sequence is verified using a single forward pass.
- Checks for invalid sequences such as 3 4, ++ <, and repeated binary operators.
- Stops immediately after the first detected issue, optimizing for fast error detection.

🔹 **Infix to Postfix Conversion – O(n)**

- Utilizes the Shunting Yard algorithm.
- Each token is pushed and popped at most once.
- Operator precedence and associativity are handled with simple comparisons.

🔹 **Postfix Evaluation – O(n)**

- Standard postfix (RPN) evaluation using a stack.
- Each operand is pushed once and popped once per operation.

Can it be improved?

- These are already optimal for general-purpose infix evaluation.
- Further optimization would increase complexity without significant gains.

Individual Contributions

This project was completed individually by Said Haji

All components were solely developed, tested, and documented by the author:

- Design and implementation of the MathLogicEvaluator class
- Tokenizer, validator, infix-to-postfix converter, and evaluator logic
- Comprehensive error checking with character-index reporting
- Test case creation and debugging
- Full documentation and report writing

References

- Shunting Yard Algorithm: https://en.wikipedia.org/wiki/Shunting_yard_algorithm
- Infix to Postfix Conversion using Stack in C++: https://www.geeksforgeeks.org/infix-to-postfix-conversion-using-stack-in-cpp/
- https://en.cppreference.com/w/cpp/language/operator_precedence
- Expression evaluation guides and classroom materials provided by the course instructor

Final Notes

The MathLogicEvaluator class provides a clean and scalable solution for evaluating complex infix expressions in C++. With well-defined helper functions, error reporting, and support for a full range of operators, this implementation is both academically thorough and practically usable.

It is ideal for future extension into GUI calculators, script parsers, or embedded logic interpreters.