Team 1

Arithmetic Expression Evaluator
Software Architecture Document

Version 1.0

Revision History

Date	Version	Description	Author
30/10/2024	1.0	Started the document	Everyone

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

The purpose of this Software Architecture Document is to delineate the architectural overview of the system, using different architectural views to depict different aspects of the system. This document is meant to explain the important architectural decisions that our team made regarding the system. Beyond the introduction, this document will cover the system's architectural representation and models, architectural goals and constraints, use-case view, logical view, interface description, size and performance, and quality. This document is intended for the team members so they can understand the architectural details of the system.

1.2 Scope

This document applies to the arithmetic evaluator we are creating. It will affect the way that the program is created by outlining the different pieces of the program.

1.3 Definitions, Acronyms, and Abbreviations

+ (addition)

- (subtraction)

* (multiplication)

/ (division)

% (modulo)

** (exponentiation)

1.4 References

Copy of 348 Project Plan - Group 1.docx

1.5 Overview

The Software Architecture document is organized into 8 sections, with possible subsections included. The sections include:

- Introduction
    - Purpose
    - Scope
    - Definitions
    - References
    - Overview
- Architectural Representation
- Architectural Goals and Constraints
- Logical View
- Overview
- Architecturally Significant Design Modules or Packages
- Interface Description
- Quality

2. Architectural Representation

The project will be coded in C++, where there will be functions for every operator and for input handling.

3. Architectural Goals and Constraints

The software should be able to handle all inputs whether they are valid or not, and if valid, perform the calculations inputted by the user and return the correct output however many times the user wants. The program should only end if the user inputs so.

4. Use-Case View

4.1 Use-Case Realizations

5. Logical View

Input Module

Tokenizer Module

Parser Module

Evaluation Module

User Interface Module

Error Handling Module

5.1 Overview

This program will start with the user interface module. Next it will use the input module, then go to the tokenizer module followed by the parser module. Finally the program will use the evaluation module to finish the calculations. If at any point there is an error, the error handling module will be called.

5.2 Architecturally Significant Design Modules or Packages

Input Module: Handles getting the input from the user. This will allow input of a mathematical expression and then pass it on to the other modules for the next step

Tokenizer Module: This will break up the expression into separate parts that will be evaluated separately

Parser Module: This parses the equation that we broke up previously and determines the order that we will evaluate it in. This module will make sure every parenthesis has a closing parenthesis and evaluates operator precedence

Evaluation Module: This does the actual calculations for the evaluator. The previous module will send in pieces of the expression into this in the right order and then this module will use PEMDAS in order to actually evaluate the expression.

User Interface Module: This module provides the actual interface that the user will use to interact with the program. This should have clear choices for the user to make so that they can use the arithmetic evaluator effectively.

Error Handling Module: This module should handle the errors for the program. If the user inputs an invalid expression into the input module, the duty should be passed on to this module in order to handle that error properly.

6. Interface Description

The user interface for the system will be the command-line terminal on the user's computer. The user will be able to enter an arithmetic expression that contains numbers or the operators listed in **Section 1.3**. Other inputs will not be accepted. The system will output the calculated result of the expression, with adherence to proper operator and parantheses precedence rules.

7. Size and Performance

8. Quality

The evaluator should have an easy interface for the user along with being able to calculate the expression efficiently and quickly. It should also be reliable and able to handle all inputs from user, even if they are invalid.