Group 4

Arithmetic Expression Evaluator
Software Architecture Design

Version 1.0

Revision History

Date	Version	Description	Author
9/25/2024	0.0	Initial meeting, discussion of meeting times, and assigning roles for the project	Alexander Carrillo
10/16/2024	1.0	Software requirements discussion, reviewing rough code, talking about next steps	Alexander Carrillo
11/4/2024	**1.0**	Software architecture design discussion and writing of document	Alexander Carrillo

Table of Contents

Software Architecture Document

1. Introduction

This document serves to show an overview of Group 4's Arithmetic Expression Evaluator. The introduction will show the purpose, scope, definitions, references, and then an overview for not only the document but also the entire project.

1.1 Purpose

This document aims to deliver a clear architectural outline of the Simple Calculator system, implemented in C++. It conveys the core architectural decisions to assist developers, system designers, and stakeholders in understanding the system's structure and design. The document is intended to guide development, support system evolution, and maintain alignment with the project's goals.

1.2 Scope

This Software Architecture Document applies to the Simple Calculator project, which is designed to handle basic arithmetic operations: addition, subtraction, multiplication, division, modulus, exponentiation, and complex calculations adhering to the PEMDAS order of operations, including parentheses.

1.3 Definitions, Acronyms, and Abbreviations

PEMDAS: describes the process of the order of operations, parentheses, exponents, multiplication, division, addition, and subtraction. in that order

Operators: the symbols that the program will read from the user input, +, -, *, **, /, %

C++: Chosen coding language

1.4 References

This document might refer to other documents such as document 1 or 2, this refers to the following

01-Project-Plan

02-Software-Requirements-Specifications

1.5 Overview

The rest of the document will go over, 2. Architectural Representation, 3. Architectural Goals and Constraints, 4. Logical View, 5. Interface Description, and 6. Quality

2. Architectural Representation

The architecture in this project is almost completely made out of the functions we will be using to evaluate the operations, each operator will have a function,

- Addition
- Subtraction
- Multiplication
- Division
- Modulus
- Exponents

We will also have a class structure for initially processing the arithmetic function and then converting integers that are multiple digits into more readable single integers (1|2|3 -> 123), also checking for negative numbers, and anything else that would be taking up multiple indexes in the initial raw user input.

3. Architectural Goals and Constraints

Our goal is to have easily readable and traversable code to make sure any issues can be found and dealt with quickly. One of the constraints we have is that we are limited to the programming language C++ and the boundaries of the language. Some other more user-oriented constraints will be the limited operators, (+, -, /, *, **, %), no trig calculations, and no calculus calculations.

4. Logical View

This section will go over the function and class structure of the program, consisting mostly of functions for the operators and a class structure for parsing and PEMDAS

4.1 Overview

Because of the simplicity of the program, there aren't any instances of hierarchy, there will only be functions for the operators (+,-,*,/,%,**) which interact directly with the main and the user input.

We will be using a class for parsing and cleaning up the raw user input into something easier to iterate over and operate with

4.2 Architecturally Significant Design Modules or Packages

Operator Functions:

- Addition
  - Takes the two numbers on either side of the '+' symbol and returns the sum
- Subtraction
  - Takes the two numbers on either side of the '-' symbol and returns the difference
- Multiplication
  - Takes the two numbers on either side of the '*' symbol and returns the product
- Division
  - Takes the two numbers on either side of the '/' symbol and returns the quotient
- Modulus
  - Takes the two numbers on either side of the '%' symbol and returns the remainder
- Exponents
  - Takes the two numbers on either side of the '**' symbol and returns the exponential

Class:

- The class will be used for two main functions
  - First, making the arithmetic function into a more parsable element
    - turning multi-digit numbers into one number (1|2 > 12)
    - turning negative numbers into individual numbers (-|1|2 > -12)
    - checking for exponents (*|* > **, or some other symbol)
  - Second, we will use it to iterate over the new array, following PEMDAS and eventually returning the correct output for the given function
    - separate loops for:
      - Parentheses
      - Exponents
      - Multiplication, division, modulation
      - Addition subtraction
    - creating a new list after each operation and then feeding it pack into the loops

5. Interface Description

Valid interfaces need to be checked to make sure the file does not crash, this will be done by thoroughly checking the user input to make sure that the program can return a value. These

inputs will be cases such as unmatched parentheses, undefined spaces, and repeat operators (1 + + 2).

The user interface will simply be the input through the terminal, this will help us limit the mistakes the user can make with input so we can more effectively create fail states that can handle errors.

6. Quality

We want to be able to break down the program into functions and classes for readability and reliability. With a more modular design, we'll be able to quickly fix or adjust any issues that come up during the coding process. This also makes picking up development after stopping working on it at any given time.