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

[This section describes the software requirements and objectives that have some significant impact on the architecture; for example, safety, security, privacy, use of an off-the-shelf product, portability, distribution, and reuse. It also captures the special constraints that may apply: design and implementation strategy, development tools, team structure, schedule, legacy code, and so on.] constraints: limits of c++, limit of operators, no trig calculations, no calculus calculationrs

4. Logical View

[This section describes the architecturally significant parts of the design model, such as its decomposition into subsystems and packages. And for each significant package, its decomposition into classes and class utilities. You should introduce architecturally significant classes and describe their responsibilities, as well as a few very important relationships, operations, and attributes.]

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.

4.2 Architecturally Significant Design Modules or Packages

[For each significant package, include a subsection with its name, its brief description, and a diagram with all significant classes and packages contained within the package.

For each significant class in the package, include its name, brief description, and, optionally, a description of some of its major responsibilities, operations, and attributes.]

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

[A description of how the software architecture contributes to all capabilities (other than functionality) of the system: extensibility, reliability, portability, and so on. If these characteristics have special significance, such as safety, security or privacy implications, they must be clearly delineated.]