<EECS 348 Term Project>

RRR<Arithmetic Expression Evaluator>
Software Architecture Document

Version <1.2>

Revision History

Date	Version	Description	Author
<01/NOV/2023>	<1.0>	<First version uploaded>	<Zack Sevart, Lingfeng Li, Asa Maker>
<05/NOV/2023>	<1.1>	<Fixed grammatical errors>	<Zack Sevart, Lingfeng Li, Asa Maker>
<11/NOV/2023>	<1.2>	<Final review of the document before submission>	<Zack Sevart, Lingfeng Li, Asa Maker>

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

This document will showcase the system's principal design decisions which will be the basis of our project. It will provide all components, connectors, and its configuration. Every detail of the specific software architecture used will be listed within this document

1.2 Scope

The architecture applies to a C++ project that functions as a calculator. The scope includes the design and organization of software components responsible for arithmetic calculations, mathematical operations, and user interface interactions.

1.3 Definitions, Acronyms, and Abbreviations

UI: The visual and interactive components that the user interacts with for the calculator

SAD: Software Architecture Document

1.4 References

Purpose: Lists sources and documents referred to in the SAD.

Details: Each reference includes its title, publication date, and where to find it.

Example: A book on software engineering practices or C++ language guidelines.

1.5 Overview

Function: Describes what the SAD contains and how it's organized.

Contents: Highlights main sections like Architectural Representation, Logical View, Process View, etc.

Use: Helps readers understand the document's layout and find specific information quickly.

2. Architectural Representation

a. Overall System Design

- Overview: Shows the main parts of the system and how they work together.
- Key Elements: User interface, expression parsing, calculation logic, and error handling.

b. Components and Structure

- Focus: Looks at the different parts of the system, like the parser and calculator.
- Key Elements: Components for parsing, calculating, handling inputs/outputs, and managing operator precedence.

c. How it Works in Action

- Focus: Describes how the system behaves when it's running.
- Key Elements: Process flow from taking user input to showing results, including error detection.

d. Setup and Environment

- Focus: Explains where and how the software runs.
- Key Elements: Details about the computer and software environment needed to run the application.

e. Data Management

- Focus: Concentrates on how data is used and managed in the system.
- Key Elements: Data structures for storing information and formats for inputs and outputs.

3. Architectural Goals and Constraints

a. Architectural Goals

- Accuracy: Ensure the program calculates expressions correctly.
- Speed and Efficiency: The program should run quickly and use resources effectively.
- Easy to Use: User-friendly interface for inputting expressions.
- Works on Many Systems: Should be able to run on different computers where C++ works.
- Easy to Update and Fix: Code should be simple to understand and modify.
- Handles any errors thrown its way to prevent any error during runtime.

b. Architectural Constraints

- Limited to C++: The entire project must be done in C++.
- Specific Tools and Environment: Restricted to tools that support C++ development.
- Deadline: There's a fixed timeline for completing the project.
- Team Collaboration: Work is divided among team members.
- Integrating Existing Code: If using previous code, it must fit well with the new project.
- Resource Limits: Depends on the available computer and software.
- Focus on Modular Design: Emphasis on breaking the project into smaller, manageable parts.

4. Use-Case View

4.1 Use-Case Realizations

5. Logical View

5.1 Overview

The project is structured into different functional layers:

- Input/Output Layer: Manages interactions with the user.
- Parsing Layer: Processes and interprets user inputs.
- Calculation Layer: Executes arithmetic calculations.
- Utility Layer: Provides supportive functionalities.

5.2 Architecturally Significant Design Modules or Packages

a. Input/Output Package

- Function: Handles user input and output.
- Main Class: ConsoleInterface - Reads user input and displays results or errors.

b.Parsing Package

- Function: Parses and organizes user input.
- Main Class: ExpressionParser - Converts input into tokens and identifies mathematical symbols.

c. Calculation Package

- Function: Carries out arithmetic operations.
- Main Class: Calculator - Performs calculations and handles math operations.

d. Utility Package

- Function: Offers additional helper tools.
- Main Class: ErrorHandler - Detects and reports input or calculation errors.

6. Interface Description

a. User Interface (UI):

- The UI consists of a graphical interface with input fields for operands, buttons for arithmetic operations, and a display area for results. It then consists of valid numeric digits 0-9, the default operators used for calculations, and float-point numbers. It then shows the result of the calculation requested by the user.

b. Command-Line User Interface(CLI):

- The CLI is where you input your arithmetic expressions, like "3 + 4". It's designed to be user-friendly: users enter an expression and it displays the result for the user. If there's an error in your input, the CLI promptly informs the user about the issue. This interface makes it easy for user to interact with the program and understand both the outcomes and any errors.

c. Expression Parsing Interface:

- This interface takes what users enter into the CLI and organizes it in a way that the program can process. It breaks down a user's expression into smaller parts, like separating numbers from operators (e.g., +, -). Then, it checks to make sure the user's expression is structured correctly, ensuring things like parentheses are used properly. This step is crucial for the program to accurately interpret and calculate your expression.

d. Calculation Interface:

- In this interface, the program performs the actual calculations on the user's expression. After the user's input is organized by the parsing interface, this part processes each part of the expression, executing operations like addition or multiplication. It also takes care of the order of operations, ensuring that calculations like multiplication are done before addition. This is essential for getting the correct answer to your expression.

e. Error Handling Interface:

- If there exist any issues during the parsing or calculation phases, this interface identifies them and communicates them back to the user. For instance, if a user attempts to divide by zero or if a user expression has a syntax error, it detects these problems and informs the user via the CLI. This feature helps users understand what went wrong and how to correct it, enhancing the overall usability and reliability of the program.

7. Size and Performance

8. Quality

a. Extensibility:

- Easy to add new features due to the modular design.
- Well-defined sections for easier updates.

b. Reliability:

- Strong error handling for stable performance.
- Testing each part separately to ensure it works well.

c. Portability:

- Uses C++, making it run on various systems.
- Minimal special requirements for running the software.

d. Safety and Security:

- Includes checks to prevent crashes and errors.

e. Performance:

- Focuses on fast and efficient calculations.

f. Usability:

- User-friendly interface for ease of use.