Orange

Arithmetic Expression Evaluator
Software Architecture Document

Version 1.0

Revision History

Date	Version	Description	Author
11/12/23	1.0	Initial document creation	Mikaela Navarro

Table of Contents

1. Introduction 4
   - 1.1 Purpose 4
   - 1.2 Scope 4
   - 1.3 Definitions, Acronyms, and Abbreviations 4
   - 1.4 References 4
   - 1.5 Overview 4

2. Architectural Representation 4

3. Architectural Goals and Constraints 5

4. Use-Case View 6
   - 4.1 Use-Case Realizations 6

5. Logical View 6
   - 5.1 Overview 6
   - 5.2 Architecturally Significant Design Packages 6

6. Interface Description 8

7. Size and Performance 8

8. Quality 8

Software Architecture Document

1. Introduction

The Software Architecture Document (SAD) serves as a comprehensive guide to the architectural aspects of the Arithmetic Expression Evaluator program. This document outlines the fundamental structures, components, and design principles that govern the program's software architecture. The information will provide developers and architects a clear understanding of the system's design decisions and how the components interact.

1.1 Purpose

The purpose of this Software Architecture Document is to articulate the architectural vision for the program. It details the high-level structures that shape the system, addressing both functional and non-functional requirements. By providing this architectural overview, the document will guide the development team in implementing a robust and scalable solution that aligns with the specified requirements.

1.2 Scope

The scope of this document encompasses the architectural considerations and decisions involved in the development of the program. Information on the modularization strategy, external interfaces, internal layout, and key components that constitute the software architecture will be provided. The document focuses on guiding principles, patterns, and practices that drive the design and construction of the program's system.

1.3 Definitions, Acronyms, and Abbreviations

For a comprehensive understanding of the architectural elements and design decisions, refer to the Project Glossary provided in the Software Requirements Specifications (SRS) document.

1.4 References

This document refers to the Shunting Yard Algorithm (wikipedia.org/wiki/Shunting_yard_algorithm) as a key resource influencing the program's design and implementation.

1.5 Overview

The Software Architecture Document is organized into sections that delve into the external and internal aspects of the program. The external interface design, including user interaction and input formats, is detailed first. The internal layout is explored, providing insights on the main modules, their interactions, and the overall flow of the program.

2. Architectural Representation

The software architecture for the Arithmetic Expression Evaluator program is a crucial aspect that governs the system's design and behavior. This section describes the software architecture for the current system, outlining the views necessary for a comprehensive understanding of the program's structure and functionality.

The program's architecture is represented through the following key views:

Module View

- Provide an overview of the modularization strategy adopted in the program's system. It enumerates the key modules, their responsibilities, and relationships. The model elements in this view include modules, interfaces, and dependencies.

- Represent the major components of the program's system including the Main Module, Solver Module, Tokenizer, and components related to the Shunting Yard Algorithm.

External Interface View

- Focus on the interactions between the program and its external environment. It details how the system communicates with users through standard text input and output streams. Model elements in this view include user interfaces, input formats, and the flow of data between the system and external entities.
- User interfaces encompass the standard text input and out streams defining how users interact with the system. Input formats include the specifications for valid mathematical expressions, including supported operator and syntax rules.

Process View

- Covers the main processes such as input handling, tokenization, the application of the shunting yard algorithm, and the computation of the postfix stack. Model elements in this view include processes, interactions, and dependencies.
- Processes represent the steps involved in the evaluation of arithmetic expressions. Interactions illustrate the flow of information between processes, showcasing the dynamic aspects of the program.

3. Architectural Goals and Constraints

The architectural goals and constraints for the program are defined by the software requirements and objectives that impact the system's design and behavior. This section outlines key considerations, including safety, security, privacy, use of off-the-shelf products, portability, distribution, and reuse. Additionally, special constraints that influence the design and implementation strategy, development tools, team structure, schedule, and potential interactions with legacy code are mentioned below:

Architectural Goals

Safety and Reliability

- The program aims to ensure the safety and reliability of arithmetic expression evaluations. Robust error handling and graceful recovery mechanism will be implemented t enhance the overall stability of the system.

Performance

- The system should be capable of handling mathematical expressions with minimal latency and computational overhead providing an efficient and optimal performance.

Modularity and Maintainability

- Well-defined modules, such as the Main Module, Solver Module, and Tokenizer, contribute to a clear and maintainable codebase.

Extensibility

- Design decisions should allow for the seamless integration of additional features without compromising the core functionality.

User Interface Interactivity

- Architectural decisions should support efficient communication with users through standard text input and output streams providing an interactive and user-friendly experience.

Architectural Constraints

Design and Implementation Strategy

- The program adheres to a design strategy focused on simplicity and ease of understanding. The implementation follows a modular approach, aligning with the outlined modularization strategy.

Development Tools

- The program is developed exclusively using standard C++ runtime and libraries. The choice of tools is constrained by the requirement to operate using the standard text input and output streams.

Team Structure

- The development team structure should be organized to effectively collaborate on the implementation of the program, emphasizing communication and coordination among team members.

Schedule

- The development schedule imposes constraints on project timelines. The architectural decisions should align with the project schedule to ensure timely delivery.

Legacy Code Interaction

- The program does not explicitly involve legacy code, however, any potential interactions with external systems or libraires should be carefully considered to avoid compatibility issues.

4. Use-Case View

N/A

4.1 Use-Case Realizations

N/A

5. Logical View

The Logical View of the Arithmetic Expression Evaluator program provides insight into the architecturally parts of the design model, focusing on its decomposition into subsystems and packages. Each significant package is explored, detailing its contained classes and their respective responsibilities, operations, and attributes.

5.1 Overview

The overall decomposition of the design model is structured in terms of its package hierarchy and layers. The Logical View encompasses several architecturally design modules or packages, each contributing to the system's functionality and coherence.

5.2 Architecturally Significant Design Modules or Packages

Main Module

- Serves as the entry point to the program, orchestrating user interactions and overseeing the flow of the system.
- Name: main.cpp
- Brief Description: Responsible for handling user input, invoking the necessary modules, and presenting results to the user.

Solver Module

- Evaluating arithmetic expressions, implementing the shunting yard algorithm, and computing the results.
- Name: solver.hpp
- Brief Description: Defines functions for computing expressions, including the application of the shunting yard algorithm and exception handling for invalid expressions.

Tokenizer Module

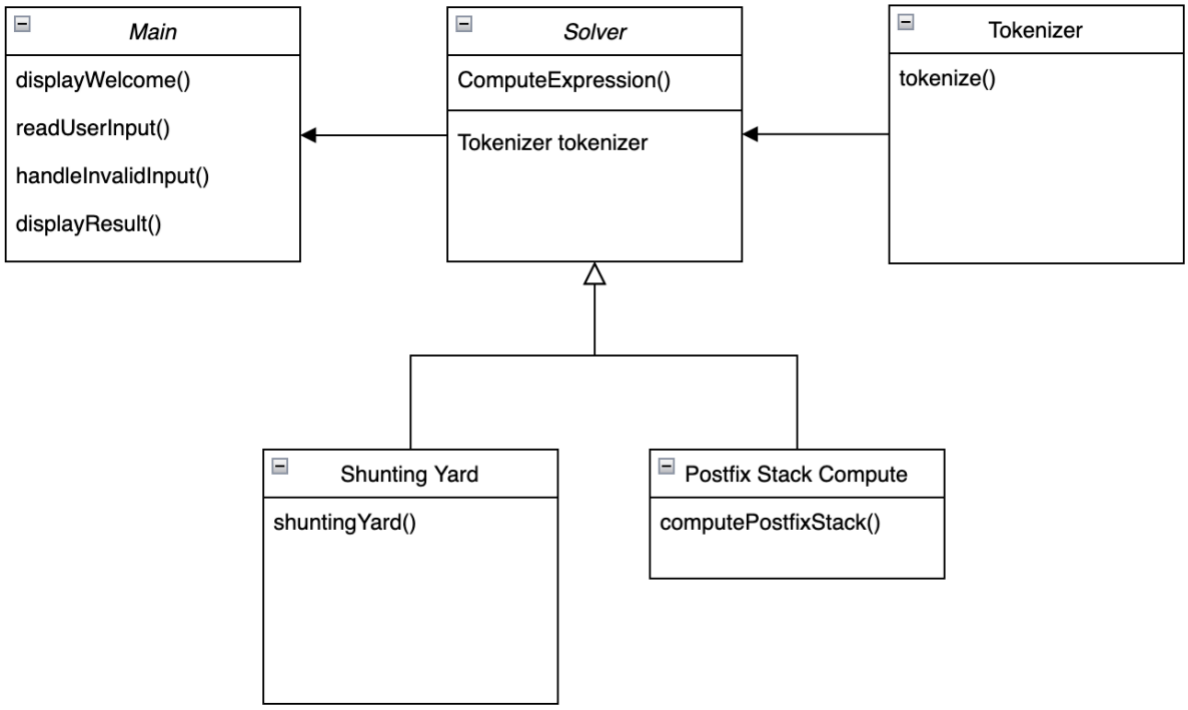
- Brief Description: Processes the input string and generates a sequence of tokens.

Shunting Yard Module

- Brief Description: Rearranges the sting of tokens using the Shunting Yard Algorithm

Postfix Stack Computation Module

- Brief Description: Computes the result from the resulting RPN queue.



Main Module Class

Name: main

Brief Description: Handles the main user interface logic.

Responsibilities: Displaying the welcome message, reading user input, handling invalid input exceptions, and displaying the results.

Operations: displayWelcomeMessage(), readUserInput(), handleInvalidInput(), displayResult()

Solver Module Class

Name: solver

Brief Description: Handles expression solving logic.

Responsibilities: Computing expressions

Operations: ComputeExpression

6. Interface Description

The program is a command-line interface (CLI) application that exclusively communicates using standard text input and output streams. It's designed as a Text-based User Interface (TUI) program for interactive use. The program follows a simple prompt-and-response interaction model.

Valid Inputs

- Users are prompted to enter mathematical expressions
- Expressions should adhere to the following format:
    - Whitespace is ignored, except for adjacent numbers separated only by whitespace, which is considered an invalid format
    - Valid infix notation expressions with supported operators: +, -, *, /, %, ^, **
    - Support for parentheses () and square brackets [] for controlling the order of operations
    - Decimal numbers with an optional fractional component, indicated by the '.'
    - Digit separators '_' and ',' are allowed within numbers

Resulting Outputs

- Program outputs the result of evaluating the entered mathematical expressions our an error message if the expression is invalid
- Output format is identical regardless of the source or destination of the input and output streams
- Error messages provide details about the nature of the error encountered

7. Size and Performance

N/A

8. Quality

The software architecture of the program is designed with extensibility in mind, allowing for easy integration of new features or modifications in the future. The modular structure separates the main functionality into distinct modules, such as the UI, Solver, Tokenizer, Shunting Yard, and Postfix Stack Computation. This modularization enhances maintainability and facilitates the addition of new functionalities or operators. For instance, introducing new mathematical operators or supporting different input formats can be achieved by extending or modifying specific modules without affecting the entire codebase.

The program exhibits a reliable behavior through error handling and exception management. The use of exception handling in both the 'main.cpp' and 'solver.hpp' files ensures that the program gracefully handles unexpected situations, providing informative error messages to users. The separation of concerns between the UI and Solver modules contributes to the reliability of the program, making it easier to identify and resolve issues. The input validation specified in the SRS document contributes to the program's reliability by ensuring that only valid mathematical expressions are processed.

The software architecture is designed to be platform-independent and relies solely on the standard C++ runtime and libraries. The use of standard input and output streams makes the program highly portable, allowing it to be executed on various operations systems with a command-line interface.