348 Industries

EECS 348 Fall 2023 Term Project: Arithmetic Evaluation Program

Software Architecture Document

Version <1.0>

Revision History

Date	Version	Description	Author
11/09/2023	1.0	First Iteration of the Software Architecture Document for the program to be made for EECS 348 Fall 2023 Term Project. This program will employ specific types of architecture to implement the software which can take in numerous inputs and conduct the respective operations to produce the appropriate outputs.	Dev Patel Jay Chung Ruth Higgason Colin Marett

Table of Contents

Software Architecture Document

1. Introduction

This development plan outlines the overall development direction for the "Arithmetic Expression Evaluator in C++" project of EECS 348: Software Engineering course in the Fall semester of 2023. This plan serves as a key guide under the guidance of Professor Hossein for the successful execution of the project.

1.1 Purpose

The purpose of this plan is to ensure the successful progress of the project by managing the overall flow of the project and ensuring each team member clearly understands their roles and responsibilities. It also clearly outlines the project's core values and strategic approach, providing important reference material to the project manager.

1.2 Scope

This plan focuses on the overall management of the "Arithmetic Expression Evaluator in C++" project. It includes all stages of the project's life cycle, from the initial stage to deployment. This plan contains content consistent with the product requirements specified in the Vision Document.

1.3 Definitions, Acronyms, and Abbreviations

For definitions of terms, acronyms, and abbreviations related to the project, please refer to the separately provided project glossary.

1.4 References

This development plan refers to the following documents:

- Iteration Plans
- Development Case
- Vision Document
- Glossary
- Other related plans or documents, etc.

1.5 Overview

This development plan includes the following contents:

- Project Overview — Detailed description of the project's purpose, scope, and objectives. It clearly defines the deliverables that the project should provide.
- Project Organization — Describes the organizational structure of the project team and the roles of each team member.
- Management Process — Defines the estimated cost and schedule, the major phases, and milestones of the project, and describes the project's monitoring methods.
- Applicable Plans and Guidelines — Provides an overview of the software development process, and presents the methods, tools, and techniques to be followed.

These sections guide and support the development effort for the "Arithmetic Expression Evaluator in C++" project.

2. Architectural Representation

The user inputs a math equation using +, -, *, /, parentheses (), and exponents ^. If the user enters an invalid expression an error will occur. If the user enters extensive parentheses or uneven parentheses, the expression will be fixed to get rid of extra parentheses and even the parentheses. Each part of the expression will then be broken up and evaluated based on PENDAS. After the expression is fully evaluated, the answer will be printed out for the user to see.

3. Architectural Goals and Constraints

The software will be a stand-alone calculator where the user will only have access to the input they give the calculator and the output that calculator gives them which will be the answer to their math expression input. This project will be able to run only any device that can compile and run C++ code since the entire project will be coded in C++. The calculator will only have access to the user input and the code required to evaluate and output the answer to the user's math input. The calculator is a group project that has 5 group members. We have a Project Manager who identifies what needs to be done and organizes when it needs to be done to keep progressing with the project. There is Project Lead 1 who overlooks the coding part of the project and makes sure that everything to correctly coded with no bugs. There is Project Lead 2 who also overlooks the code part of the project but makes sure that what needs to be added or fixed with the code is getting added or fixed. There is a Team Admin who makes sure the team stays on schedule and that the team works together to get the project done on time. There is a Data Admin/Quality Assurance who tests the product to see if there are any bugs or features that should be added or fixed. The calculator will be made in

a semester, and it will be fully coded by this group, there will be no legacy code used to make this project.

4. Logical View

4.1 Overview

The logical view of the Arithmetic Expression Evaluator design focuses on the decomposition of the system into several architecturally significant packages. These packages represent the major functionalities and components required for the successful implementation of the arithmetic expression evaluator.

4.2 Architecturally Significant Design Modules or Packages

4.2.1 Tokenization Package

Description:

This package is responsible for tokenizing the input arithmetic expression. It includes classes and utilities to break down the input expressions into meaningful tokens, such as operators, operands, and parentheses.

```
------------------------------
Tokenization Package
------------------------------
+ Tokenizer
   - tokenize()
   - getNextToken()
------------------------------
```

Tokenizer Class:

Responsibilities:

- Tokenize the input arithmetic expression.
- Provides a method to retrieve the next token.

4.2.2 Parsing Package

Description:

The parsing package is responsible for creating a structured representation of the arithmetic expression. It includes classes and utilities to build a parse tree or use a stack-based approach for expression parsing.

```
----------------------
```

Parsing Package

+ Parser

  - parse()

  - evaluate()

Parser Class:

Responsibilities:

- Parse the tokenized expression.
- Build a parse tree or use a stack-based approach.
- Evaluate the parsed expression.

4.2.3 Operator Handling Package

Description:

This package is responsible for handling operators in the arithmetic expression. It includes classes and utilities to define operator precedence and evaluate expressions based on PEMDAS rules.

Operator Handling Package

+ OperatorHandler

  - handleOperator()

  - getPrecedence()

OperatorHandler Class:

Responsibilities:

- Handles the operators in the expression.
- Defines the operator precedence.
- Facilitates the evaluation of expressions based on operator rules.

4.2.3 User Interface Package

Description:

The user interface package is responsible for interacting with the user, accepting input expressions, and displaying the calculated results.

User Interface Package

+ UserInterface

- getUserInput()

- displayResult()

UserInterface Class:

Responsibilities:

- Accepts the user input for arithmetic expressions.
- Displays the calculated results to the user.

4.2.4 Error Handling Package

Description:

The error handling package is responsible for managing and reporting errors that may occur during expression parsing and evaluation.

Error Handling Package

+ ErrorHandler

- handleError()

ErrorHandler Class:

Responsibilities:

- Handles and reports the errors during expression parsing and evaluation.

5. Interface Description

The interface for this project will be a terminal input and output. Some valid inputs and their outputs are listed below

- Input: 2+5
- Output: 7

- Input: 10*3/5
- Output: 15

- Input: 4-(1-2)
- Output: 5

- Input: 3^3
- Output: 27

- Input: -(+1) + (+2)
- Output: 1

- Input: (((2 ^ (1 + 1)) + ((3 - 1) ^ 2)) / ((4 / 2) % 3))
- Output: 4

There are many other options which will provide valid outputs, which deal with extraneous parentheses for every operator, as well as other problems that don't invalidate an expression.

6. Size and Performance

The major dimensioning characteristics of the software that impact the architecture are scalability and maintainability. The software could have some usability issues, for example if someone keeps entering an expression and it doesn't evaluate but also doesn't tell them why it doesn't evaluate. However, that would be easy to fix, and thus shouldn't be a major dimensioning characteristic. Therefore, the maintainability and scalability of the code will be focused on. For maintenance, there will be thorough documentation and comments explaining why all parts of the code work, as well as how they affect other parts of the code. The scalability aspect will have us segment our code, most likely into modules or functions, which will then interact with each other so that adding new parts to the program is easy and efficient.

The software is expected to be able to parse certain expressions in well under a second, as it is only made to do quick calculations with basic arithmetic operators. Also, it should be able to parse all valid expressions that are handed to it.

7. Quality

The 'Arithmetic Evaluation Program' will be designed with a robust software architecture that not only ensures functionality but also emphasizes key characteristics such as extensibility, reliability, and portability. Each of these aspects contributes to the overall performance and usability of the system.

1. Extensibility:

- Modular Design: The program will be built with a modular architecture, dividing the functionality into distinct modules. This design facilitates easy addition of new features or operators without disrupting the existing codebase. Developers can extend the program by simply adding new modules or modifying existing ones, making it highly adaptable to future requirements.

2. Reliability:

- Error Handling Mechanisms: The architecture includes robust error-handling mechanisms to ensure the program gracefully handles unexpected inputs, preventing crashes and providing meaningful error messages to users. This reliability feature contributes to a stable and predictable user experience.
- Testing and Quality Assurance: The development process incorporates thorough testing procedures, including unit tests, integration tests, and system tests. Automated testing tools are employed to identify and rectify issues early in the development cycle, ensuring a reliable and bug-free software release.

3. Portability:

- Cross-Platform Compatibility: The program is designed to be platform-independent, ensuring it can run seamlessly on various operating systems. This is achieved by using programming languages and frameworks that support cross-platform development. This portability allows users to access the Arithmetic Evaluation Program on different devices without any compatibility issues.
- Containerization: The use of containerization technologies like Docker enhances portability by encapsulating the program and its dependencies. This ensures consistent behavior across different environments, making it easy to deploy and run the application on diverse platforms. However, this will not be employed in this program yet.

4. Scalability:

- Efficient Algorithms: The program utilizes efficient algorithms for arithmetic operations, ensuring optimal performance even with complex calculations. This scalability feature enables the program to handle a large volume of mathematical computations without compromising responsiveness.

In conclusion, the 'Arithmetic Evaluation Program' is not only a powerful calculator in terms of functionality but also excels in extensibility, reliability, portability, and security. These architectural considerations collectively contribute to a versatile and dependable software application, meeting the diverse needs of users while prioritizing their data security and privacy.