Group 22

Arithmetic Parser
Software Architecture Document

Version 1.1

Revision History

Date	Version	Description	Author
11/03/24	1.0	Initial document creation & drafting Section 1	Janna Dungao
11/08/24	1.1	In-person meeting and second drafting.	Yoseph Ephrem, Mahdi Essawi, Beckett Malinowski, Shane Huang

Table of Contents

Software Architecture Document

1. Introduction

The introduction of this Software Architecture Document contains the purpose, scope, definitions, acronyms, abbreviations, references, and overview of the document.

1.1 Purpose

The purpose of this document is to provide a comprehensive overview of the system from an architectural standpoint. As such, different architectural views will be utilized to depict the different parts of the system. It should provide our group with a reference to which we can use during implementation in order to ensure a uniform structure throughout the program.

In the overall project documentation, we will utilize previous documents and build upon them to prepare for implementation. This document is structured so the user may be introduced to necessary terms and an outline of the architectural structure of the project. The specific audiences for this document are our team members and future users of our program.

1.2 Scope

This Software Architecture Document applies to the structure and design of the future implementation of the arithmetic parser project. The project implementation, user manual, and test cases will be affected and influenced by this document.

1.3 Definitions, Acronyms, and Abbreviations

Definitions

- Implementation – the actual code and files for the project

Acronyms

- Interface – the component that the user interacts with; the interaction between a user and software and/or hardware

Abbreviations

- PEGTL (Parsing Expression Grammar Template Library)
- METL (Math Expression Template Library)

1.4 References

GitHub Link – contains all created documentation for this project including:

- Meeting Log
- 01-Project-Plan.pdf
- 02-Software-Requirements-Spec.pdf
- README.md

1.5 Overview

The rest of this Software Architecture Document contains the architectural representation, goals and constraints, the use-case view, the logical view, an interface description, the project's size and performance, and quality.

This document is organized in a way such that the user may be introduced to an overview of the project, then learns about how the software may be used and how it will work/be implemented. Next, the user will learn how the project will architecturally be designed as well as a brief description of the user interface. Finally, the quality of the project will discuss how this document contributes to all the capabilities of the system.

2. Architectural Representation

We plan on utilizing C++ classes that are already created in the libraries that we use to construct this project. These classes will be implemented inside our main class that we will create, called Parser. We will add additional classes for functionality that is not covered in the library. Our project will be compatible to run inside of the terminal by retrieving input for specific step-by-step from options provided, as well as visually in a window that we will create. Our goal is to emphasize the flexibility making this easy to use for various people.

3. Architectural Goals and Constraints

Safety – in the case of the parser, we can ensure validity of input and output through the GUI. The user will not be allowed to enter a string, for example, since all valid inputs will be available to the user via buttons on the GUI. Buttons will be available for numbers 0-9, trigonometric functions, operators and parentheses and the equals to submit the input. All classes used will use private variables to ensure any other programs interfacing with the parser do not read or write on variables when they shouldn't.

Portability, Distribution, Reuse – the product should be relatively easy for a developer to use. The download from GitHub is fairly easy to complete. It will be available to any and all developers to use, but any additional features desired by anyone will have to be forked and implemented.

Development Tools – The team may utilize the C++ qt module and accompanying software to create a GUI interface for users to interact with.

Design and implementation strategy – The main Parser class will have its behavior defined largely using modules defined in Part 5. Another class may be defined for the GUI Application using the qt module.

Legacy code – the code will not need to be maintained after the release date due to the small scope of the product.

4. Use-Case View

4.1 Use-Case Realizations

5. Logical View

This project will be working on multiple levels like parsing, evaluating, function management etc. We will be using the following packages for their respective purposes.

Cmath:
Cmath is a very simple package and is primarily going to be used for some of the advanced mathematical functions like the trigonometry, exponential, logarithmic, power, and root functions. This package will save us a significant amount of time in the coding aspect leaving a lot of the tedious functions up to the library.

PETGL Package:

This package will be responsible for the central class used for parsing input from the user and converts that input to the output for the operation. It will follow the basic arithmetic laws of mathematics like PEMDAS in its functionality. This class will interact with the PETGL package and its contents as the "caller" leading the parsers by taking commands and directing them to get the result and then returning that it to the user.

METL Package:

This package will primarily work on evaluating the mathematical expressions created inside of the main parser class that will be used with our PETGL package. It will handle operations by computing the numbers, variables, and entire functions while following PEMDAS as well to correctly and efficiently compute the values asked of it. This is also the library where most of our mathematic functions/methods will be pulled from. This library contains a lot of advanced and simple functions that will be used to make the engineering process for some of the methods faster allowing us to make the parser as good as it can be following our goals for its capabilities.

5.1 Overview

The design model is structured into packages that separate concerns such as parsing, expression evaluation, and user interface components. The use of header-only libraries simplifies integration and enhances portability across different platforms.

The architecture comprises several layers:

1. **User Interface Layer**: Built using the Qt framework to provide a graphical interface for user interaction.
2. **Parsing Layer**: Utilizes PEGTL and METL to parse mathematical expressions based on parsing expression grammar.
3. **Mathematical Evaluation Layer**: Employs the cmath library to handle various mathematical functions and calculations.
4. **Core Logic Layer**: Manages the interaction between the UI and the parsing/evaluation processes.

5.2 Architecturally Significant Design Modules or Packages

User Interface Package:

This package contains the classes necessary for building the graphical user interface using the Q**t framework**, allowing users to input mathematical expressions and view results.

Parsing Package:

This package contains the parsing logic utilizing the **PEGTL** library for expression parsing and the **METL** library for mathematical expressions.

Mathematical Functions Package:

This package leverages the **cmath** library to provide a range of mathematical and Trigonometric functions for use in expression evaluation.

6. Interface Description

Instead of having users take the inputs directly from a keyboard, we proposed to take an interface that will be comprised of buttons that will convert their input onto a display of which the output can be seen. This will eliminate any outside exceptions needing to be handled such as characters or string being input instead of numeric values. Valid inputs of which the program will entail include operators (to determine what function is arithmetically put to use), numbers to see how they're changed, and parsers to isolate which expressions are going to be prioritized.

7. Size and Performance

8. Quality

Using the previously mentioned architecture, the extensibility can branch out through the use of the libraries we're able to import. That way, we won't have to compute any specific methods such as trigonometry functions from scratch. In theory, our parser should be as reliable as any other calculator, so long as exceptions and the execution of code runs accordingly. Any machine that will be able to compile using C++ should be able to have our program transferred over without any issues.