Group 0

Arithmetic Expression Evaluator in C++
Software Requirements Specifications

Version <1.0>

Revision History

Date	Version	Description	Author
17/10/2024	1.0	Initial Meeting for part two	Evan Rogerson, Mason West, Nick Heyer, Rahul nesan, Ben Haney, Samantha Adorno, Zain Cheema

Table of Contents

Software Requirements Specifications

1. Introduction

1.1 Purpose

The purpose of this SRS document is to define the functional and non-functional requirements for the development of an Arithmetic Expression Evaluator in C++. This system will parse, evaluate, and return results for the arithmetic expressions containing operators such as addition, subtraction, multiplication, division, and parentheses.

1.2 Scope

The Project must be completed by the end of the semester. Additionally we must progressively work on the project in order to not fall behind and meet smaller deadlines (such as our initial plan on Sept 29). We must have a fully completed project submitted by the deadline in such a way that it doesn't have errors.

This Software Development Plan applies to the development of the Arithmetic Expression Parser project. It covers all aspects of the project, including design, implementation, testing, and error handling. Each phase of the project will be influenced by this plan, ensuring that all requirements are met, and the final product is delivered on time and operates as expected.

1.3 Definitions, Acronyms, and Abbreviations

Software Requirements Specification: (SRS)

Use Case Modeling: The process of coming up with requirements and laying out the plan for the project. This entails interactions, dependencies, and other relationships between different actors.

See the Project Glossary

1.4 References

This document may make reference to our Software Development Plan, which can be found in the Github repository for our project.

1.5 Overview

This document is structured into sections that describe the overall product perspective, specific functional requirements, use-case specifications, and supplementary requirements. The document is intended for developers, testers, and any who are interested in the development of the Arithmetic Expression Evaluator.

2. Overall Description

The Arithmetic Expression Evaluator is a C++ tool designed to make it easy to parse and calculate arithmetic expressions for a larger compiler project. It supports key operations like addition, subtraction, multiplication, division, modulo, and exponentiation, all while following standard math rules for order of operations. With a user-friendly command-line interface and solid error handling for common issues like division by zero or mismatched parentheses, the evaluator aims to provide a smooth experience for users. Though it starts with integer arithmetic, it's built to evolve, allowing for future enhancements like support for floating-point numbers and more operations.

2.1 Product perspective

2.1.1 System Interfaces

The system interface includes a working computer to run the program and allow for command line input from the user and output for the user to read.

2.1.2 User Interfaces:

Command-line input and output

2.1.3 Hardware Interfaces

The keyboard and monitor is the primary user hardware interface.

2.1.4 Software Interfaces

The user will interface with our program through the command line. The program is self contained and does not interface with any other software.

2.1.5 Communication Interfaces

2.1.6 Memory Constraints

The system has an efficient design. The long and complex expressions may require a significant amount of memory for parsing. Regular computer systems today should handle the memory conditions with no difficulty. We will use double data types which store 8 bytes of data to ensure that we are able to store even the largest of numbers.

2.1.7 Operations

2.2 Product functions

Our calculator will be able to take inputs from users in string format and return the evaluated result abiding by the rules of PEMDAS. It should be able to handle any inputs from users including invalid expressions. If the expression is invalid such as dividing by 0, it will let the user know. The error handling will be robust to handle any scenario.

2.3 User characteristics

The User is expected to have a basic knowledge of using the command-line interface and understanding of the mathematical operations

2.4 Constraints

The evaluator will only support basic arithmetic operations and parentheses.
Floating- point precision will be limited to system capabilities.

2.5 Assumptions and dependencies

We assume that the user is capable of running the program and interacting with the program through the command line, inputting arithmetic expressions that are properly formatted so that our calculator can parse and evaluate them.

2.6 Requirements subsets

3. Specific Requirements

The Arithmetic Expression Evaluator is designed with several specific requirements to make it user-friendly and effective. It should accept a string input for arithmetic expressions and break it down into numeric constants and operators, supporting addition, subtraction, multiplication, division, modulo, and exponentiation. The evaluator must follow standard math rules for operator precedence and correctly handle nested parentheses to ensure accurate calculations. Error handling is crucial, so it should provide clear messages for issues like division by zero, unmatched parentheses, invalid operators, and missing operands. A straightforward command-line interface will allow users to input expressions easily and view results clearly, with helpful prompts guiding them along the way. The system should process valid expressions quickly—within two seconds—and achieve 100% accuracy in evaluations. It needs to be intuitive and reliable, avoiding crashes from incorrect inputs. Additionally, the code should be modular for easy updates and enhancements, supported by thorough documentation. Use cases include successfully evaluating valid expressions and managing invalid input gracefully, ensuring a smooth experience for all users.

3.1 Functionality

3.1.1 Input Handling

The system must accept mathematical expressions from the user in the form of strings. The input string may contain numbers, operators (e.g., +, -, *, /, %, **), and parentheses.

3.1.2 Expression Evaluation

The system must support the evaluation of arithmetic expressions using the operators addition (+), subtraction (-), multiplication (*), division (/), modulus (%), and exponentiation (**).

The system must evaluate expressions based on the correct order of operations: parentheses, exponents, multiplication and division (left to right), and addition and subtraction (left to right).

3.1.3 Detect Invalid Expressions

The system must detect and handle invalid expressions, such as unmatched parentheses, division by zero, or invalid characters, by returning appropriate error messages to the user.

3.1.4 Output Results

The system must display the result of the evaluated expression to the user after processing the input. The result should be printed clearly.

3.2 Use-Case Specifications

We will create a use case modeling diagram of our calculator. This will include all of the actors within our program. The use case specifications will also detail all of the relationships and dependencies between actors.

3.3 Supplementary Requirements

In addition to the core function of the calculator being able to effectively parse information, we have a few supplementary requirements that we want to implement into the program. First of all, the program must have a user- friendly graphical interface. This should make it easy for users to be able to input and receive

information.

4. Classification of Functional Requirements

Functionality	Type
Receive arithmetic expressions from the user. (Input handling)	Essential
Following proper order of operations, accurately evaluate the expression. Should be able to handle addition, subtraction, multiplication, division, floor division, exponentials, and parenthesis. (Expression evaluation)	Essential
Find and handle invalid expressions such as division by zero, and sending an error message (Detect invalid expressions)	Essential
Print result to user. (Output result)	Essential

5. Appendices

The following appendices provide supplementary information to support the development process. These appendices are not considered part of the formal requirements but serve as guidelines to ensure consistency and quality throughout the project lifecycle.

A.1 Development Process

- The project will follow the UPEDU process. Other applicable process plans are listed in the references section, including Programming Guidelines.

A.2 Programming Guidelines

- Follow Google's C++ Style Guide for consistent, readable code.
- Use proper error handling and add comments where needed.

A.3 Design Guidelines

- Stick to object-oriented design and modular structure.
- Use UML diagrams as per standard practices.

A.4 Testing Guidelines

- Use Google Test (GTest) for unit tests.
- Cover all cases, especially operator precedence and parentheses.