Calculon

C++ Calculator
Software Requirements Specifications

Version 1.0

Revision History

Date	Version	Description	Author
10/11/23	1.0	Creation and Declaration of Software Requirements Specifications	Nabeel Ahmad , Shero Baig, Arnav Jain, Zonaid Prithu, Omar Mohammed, Yaeesh Mukadam, Humza Qureshi

Table of Contents

Software Requirements Specifications

1. Introduction

1.1 Purpose

The program solves and returns the results of any given arithmetic expression.

Requirements:

- Parse arithmetic expressions input by users, ensuring operator precedence and parentheses are appropriately considered.
- Support essential arithmetic operations (and do them according to PEMDAS order), namely:
  - Addition (+)
  - Subtraction (-)
  - Multiplication (*)
  - Division (/)
  - Modulo (%)
  - Exponentiation (^)
- Detect and compute constants present within the expressions
- Support for Unary operations
- Offer a user-friendly command-line interface, ensuring a seamless experience for users to input expressions and retrieve results.
- Implement comprehensive error-handling mechanisms, especially for common pitfalls like division by zero and erroneous expression inputs.

Design Constraints:

- Development Time: As this is a term project, we are constrained by the timeline of the semester. This limits the extent of features and optimizations we can introduce.
- Software Complexity: Given the scope of the project and our academic commitments, there is a limit to the software's complexity. While we aim for a robust tool, certain advanced features or optimizations might be deferred to future iterations.
- Tools and Libraries: We're working with standard C++ libraries and avoiding external dependencies to ensure portability and ease of grading.
- User Interface: Our evaluator will be command-line based. A graphical user interface (GUI) is outside the current project's scope.
- Memory and Performance: While our tool aims to evaluate standard arithmetic expressions efficiently, extremely long or complex expressions might impact performance due to memory and computational constraints.
- Future Enhancements: While we are open to embracing changes (such as the introduction of floating-point values or the '**' operator), the timeline for such enhancements is uncertain. They are contingent on project progress and feedback.

1.2 Scope

The C++ Command Line Calculator performs arithmetic operations given an arithmetic expression from the command line interface (CLI). It performs the basic operations, namely addition, subtraction, multiplication, and division, as well as more complex ones such as modulus and exponentiation and will perform operations with parentheses to dictate grouping and arithmetic precedence. Unary operations will also be supported. The program will also be able to handle errors in the expression entry such as incomplete operations, or division by zero. The program is associated with the C++ Calculon Use Case Model.

1.3 Definitions, Acronyms, and Abbreviations

Definitions:

- **Arithmetic Expression**: A mathematical notation involving numbers and operation symbols that denotes a single mathematical operation.
- **Operator Precedence**: The rule dictating the order in which operations in arithmetic expressions are carried out.
- **Tokenization**: The process of converting an input string into discrete units, called tokens, for easier parsing and processing.
- **Data Structure:** Data structures are specialized formats, such as stacks, lists, heaps, and trees, used to organize, process, retrieve, and store data efficiently.

Acronyms:

- **UPEDU**: Unified Process for Education - A simplified version of the Rational Unified Process (RUP) tailored for educational projects.
- **RUP**: Rational Unified Process - A software development process framework.
- **PEMDAS**: Parentheses, Exponents, Multiplication and Division (from left to right), Addition and Subtraction (from left to right) - A mnemonic for the order of operations in arithmetic.
- **CLI**: Command-Line Interface - A user interface where users interact with software by typing commands.
- **GUI**: Graphical User Interface - A visual interface where users interact with software using graphical elements

1.4 References

1. *Kruchten, P. (2003). The Rational Unified Process: An Introduction (3rd ed.). Addison-Wesley.*
2. *Stroustrup, B. (2013). The C++ Programming Language (4th ed.). Addison-Wesley Professional.*

1.5 Overview

The rest of the document is split into 4 different subtopics: Overall Description, Specific Requirements, Classification of Functional Requirements, and Appendices. Overall Description describes the general factors that affect the product and its requirements. Specific Requirements describes all software requirements to a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements. Classification of Functional Requirements describes all functional requirements and orders them by type or appearance. Finally, Appendices has extra nonessential information.

2. Overall Description

2.1 Product perspective

2.1.1 System Interfaces

The system does not interface with any external systems directly. Its main interaction is with the user through the command-line interface.

2.1.2 User Interfaces

The primary user interface is the command-line interface.

2.1.3 Hardware Interfaces

The primary user hardware interface is the keyboard and the monitor.

2.1.4 Software Interfaces

The Arithmetic Expression Evaluator is built using C++. It doesn't rely on external software interfaces but might be expanded in the future to integrate with other software platforms or libraries.

2.1.5 Communication Interfaces

The program does not have any communication interfaces, as it is not connected to any external system.

2.1.6 Memory Constraints

While the system is designed to be lightweight and efficient, complex and long arithmetic expressions might require significant memory for tokenization and parsing. However, standard computer systems should handle the memory requirements with ease.

2.1.7 Operations

The system operates by tokenizing the input, parsing the tokenized input to understand its structure, and then evaluating the arithmetic expression based on the rules of arithmetic and operator precedence.

2.2 Product functions

The primary function of the product is to evaluate arithmetic expressions. It handles various operators (+, -, *, /, %, ^) recognizes numeric constants, considers operator precedence, and manages expressions within parentheses.

2.3 User characteristics

Users are expected to have basic computer literacy and an understanding of arithmetic operations. The target user base ranges from students to professionals who need a reliable tool to evaluate arithmetic expressions.

2.4 Constraints

The entire program has to be written in C++ and there needs to be at least one data structure used.

2.5 Assumptions and dependencies

Assumes the baseline functionalities of the C++ coding language, as well as the stdio.h and math.h library.

2.6 Requirements subsets

Currently, there are no subsets of requirements. All requirements are treated with equal priority, except for potential future features like floating-point evaluation, which might be considered in subsequent versions.

3. Specific Requirements

3.1 Functionality

3.1.1 Functional Requirement One: Expression Parsing

The system will tokenize user-inputted arithmetic expressions, breaking them into individual elements. Once tokenized, these elements, including numbers, operators, and parentheses, will be parsed to understand their hierarchical relationship.

3.1.2 Functional Requirement Two: Operator Precedence and Correct Calculations

The system will adhere to the PEMDAS rule for evaluating expressions. This ensures operations are performed in the correct mathematical order, from parentheses to addition and subtraction in order to get the correct answer.

3.1.3 Functional Requirement Three: Parenthesis Handling

Parentheses in expressions will be identified and prioritized by the system. Operations enclosed within them will be evaluated first, in line with the PEMDAS rule.

3.1.4 Functional Requirement Four: Numeric Constants

The system will initially process integer values within expressions. It will be designed with flexibility, allowing potential inclusion of floating-point values in the future.

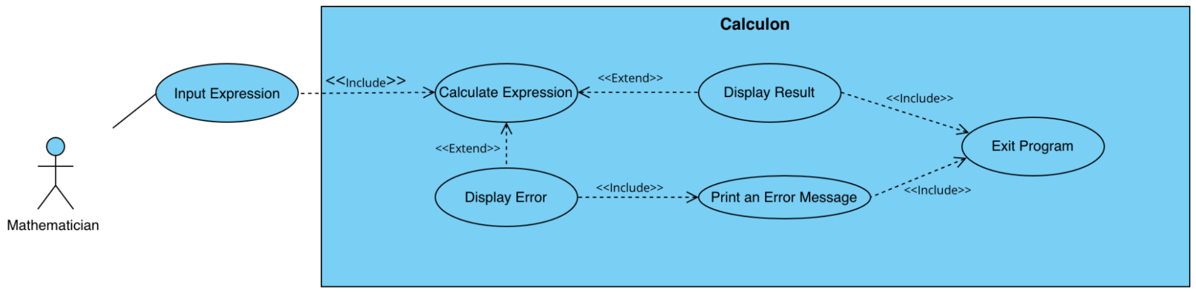
3.1.5 Functional Requirement Five: Command-Line User Interface

A user-friendly command-line interface will be provided by the system. Through this interface, users can seamlessly input expressions and obtain calculated results.

3.1.6 Functional Requirement Six: Error Handling

The system is equipped to identify invalid arithmetic expressions and potential pitfalls, like division by zero. Upon detection, users will be informed through clear and concise error messages.

3.2 Use-Case Specifications



3.3 Supplementary Requirements

All requirements have already been listed.

4. Classification of Functional Requirements

Functionality	**Type**
Expression Parsing	Essential
Operator Precedence	Essential
Parenthesis Handling	Essential

Numeric Constants	Essential
User Interface	Essential
Error Handling	Essential
Floating Point Handling	Desirable
Handling of “**” as an exponentiation	Optional

5. Appendices

N/A