Team 2

Calculatorator
Software Requirements Specifications

Version 1.0

Revision History

Date	Version	Description	Author
10/19/2024	0.5	Did section 1	Riley
10/20/2024	0.8	Compiled sections 2 and 3 from Ruben and Sean	Kaia
10/20/2024	1.0	Classified functional requirements	Sam

Table of Contents

Software Requirements Specifications

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification document is to clarify the requirements of our previously outlined software. Herein, we include all functional and nonfunctional requirements, design constraints, and all other factors necessary to provide a complete and comprehensive description of the software to be developed.

1.2 Scope

This Software Requirements Specification describes the requirements to be met by the Calculatorator project. The details of the individual iterations will be described in the Iteration Plans.

The plans as outlined in this document are based upon the product requirements as defined in the Project Plan, and include the program properly handling +, -, *, /, %, and **, processing numeric constants, managing parentheses, and handling errors like division by zero.

1.3 Definitions, Acronyms, and Abbreviations

OOP: Object oriented programming, a computer programming model that organizes software design around data, or objects, rather than functions and logic.
QA: Quality Assurance

1.4 References

-Iteration Plans: Not yet created

-Vision: Create a fully functioning, robust calculator developed entirely within C++. The program must properly handle +, -, *, /, %, and **, process numeric constants, manage parentheses, and handle errors like division by zero.

1.5 Overview

This Software requirements specification contains the following information:

-An overview section to outline all important aspects of the software, including it’s dependencies and broad specifications

-A description of the functionality of the software, including functional and nonfunctional requirements, constraints, and optional functions.

2. Overall Description

2.1 Product perspective

2.1.1 System Interfaces

2.1.2 User Interfaces

Development will focus around users interacting with the program through the command line. Equations will be input in a single line of text and the output will be displayed in another line. If extra development time allows, a desktop application with buttons and a text output will be created.

2.1.3 Hardware Interfaces

2.1.4 Software Interfaces

The software interfaces that will be used are the C++ programming language and a GCC compiler to create an executable file that can be run in the terminal. It will use C++'s standard input and output library <cstdio> to read and display information, and the math library <cmath> to aid in calculations.

2.1.5 Communication Interfaces

2.1.6 Memory Constraints

Because of the fact that the program will be relatively simple and run on machines very well equipped to handle basic programs, memory will likely not be a major constraint on development. Input storage, code size, and recursion still should be minimized to free up more memory on the system for other tasks.

2.1.7 Operations

2.2 Product functions

The program will be able to take in a string of numerical digits and operators such as {+, -, /, *, ^, //} and will be able to process the input as a mathematical expression and compute the results. The program will be able to compute with the correct order of pemdas, including performing operations within parentheses first. The program will also be able to tell the user if the expression they have input is invalid if it's operators are not set up correctly or if it otherwise would not result in an answer

2.3 User characteristics

An average user of the program will know the definitions of the operators and will have basic knowledge of arithmetic.They will have familiarity with setting up equations, and will likely be using the product to speed up some of the busy work associated with higher level math such as algebra or calculus. They will also have a working knowledge of how to use programs run in the command line.

2.4 Constraints

The program's constraints will include limiting input size, error handling, and optimization. The input string will be limited to 255 characters (or another arbitrarily picked number) to avoid large expressions that may cause issues with memory. If there is incorrect syntax, the program should be able to process it without crashing and describe to the user what went wrong. Development should focus on creating a lightweight program that minimizes operations and is as optimized as possible within the development timeframe.

2.5 Assumptions and dependencies

For the purposes of this program, it will be assumed that all users have access to a will have the capability to open and navigate a command line interface. It will also be assumed that users will

be using machines that meet minimal hardware requirements, which include adequate RAM (at least 500Mb) and a processor capable of handling basic computational tasks. Most if not all modern computers will easily meet these requirements. The program will be dependent on a compiler (only used by the developers, code will be shipped as an executable file) and the <cmath> and <cstdio> libraries.

2.6 Requirements subsets

3. Specific Requirements

-**Expression Parsing**: The ability to parse arithmetic expressions, considering operator precedence and parentheses.

-**Operator Support**: Support for addition, subtraction, multiplication, division, modulo, and exponentiation.

-**Parenthesis Handling**: Ensuring proper evaluation of expressions with parentheses.

-**Numeric Constants**: Recognition and computation of numbers in expressions.

-**User Interface**: A user-friendly command-line interface to input expressions and display results.

-**Error Handling**: Robust handling of errors like division by zero or invalid inputs.

3.1 Functionality

This section outlines the functional requirements for the calculator. It's organized by the features your project needs to implement.

3.1.1 Expression Parsing

Requirement Description: The system must tokenize user-input arithmetic expressions, respecting operator precedence and parentheses to prepare the expression for evaluation.

3.1.2 Operator Precedence

Requirement Description: The system must follow the PEMDAS rules to determine the correct evaluation order of operations within an arithmetic expression.

3.1.3 Parenthesis Handling

Requirement Description: The system must evaluate expressions with parentheses by identifying and solving the innermost expressions first before moving outward.

3.1.4 Numeric Constants

Requirement Description: The system must recognize numeric constants within the input and compute their values correctly.

3.1.5 Error Handling

Requirement Description: The system must handle errors such as division by zero, invalid numeric input, and malformed expressions. It should display appropriate error messages and terminate the expression evaluation safely.

3.1.6 User Interface

Requirement Description: The system must provide a clear and user-friendly command-line interface where users can input arithmetic expressions and view the calculated results.

3.2 Use-Case Specifications

Use Case 1: Addition

Description: The user inputs two numbers and selects the addition operator (+). The system adds the numbers and displays the result.

- Precondition: The user has entered two valid numeric inputs.
- Postcondition: The calculator displays the sum of the two numbers.
- Main Flow:
  - User inputs two valid numbers.
  - User selects the addition operator (+).
  - System computes the sum.
  - Result is displayed to the user.
- Exception Flow:
  - Invalid input is provided (handled by Error Handling Use Case).

Use Case 2: Subtraction

- Description: The user inputs two numbers and selects the subtraction operator (-). The system subtracts the second number from the first and displays the result.
- Precondition: The user has entered two valid numeric inputs.
- Postcondition: The calculator displays the result of the subtraction.
- Main Flow:
  - User inputs two valid numbers.
  - User selects the subtraction operator (-).
  - System computes the difference.
  - Result is displayed to the user.
- Exception Flow:
  - Invalid input is provided (handled by Error Handling Use Case).

Use Case 3: Multiplication

- Description: The user inputs two numbers and selects the multiplication operator (*). The system multiplies the two numbers and displays the result.
- Precondition: The user has entered two valid numeric inputs.
- Postcondition: The calculator displays the product of the two numbers.
- Main Flow:
  - User inputs two valid numbers.
  - User selects the multiplication operator (*).
  - System computes the product.
  - Result is displayed to the user.
- Exception Flow:
  - Invalid input is provided (handled by Error Handling Use Case).

Use Case 4: Division

- Description: The user inputs two numbers and selects the division operator (/). The system divides the first number by the second and displays the result.
- Precondition: The user has entered two valid numeric inputs and the second number is not zero.

- Postcondition: The calculator displays the quotient.
- Main Flow:
    - User inputs two valid numbers.
    - User selects the division operator (/).
    - System computes the quotient.
    - Result is displayed to the user.
- Exception Flow:
    - Second number is zero (handled by Error Handling Use Case).
    - Invalid input is provided (handled by Error Handling Use Case).

Use Case 5: Modulo

- Description: The user inputs two numbers and selects the modulo operator (%). The system computes the remainder of the division of the first number by the second.
- Precondition: The user has entered two valid numeric inputs.
- Postcondition: The calculator displays the remainder.
- Main Flow:
    - User inputs two valid numbers.
    - User selects the modulo operator (%).
    - System computes the remainder.
    - Result is displayed to the user.
- Exception Flow:
    - Invalid input is provided (handled by Error Handling Use Case).

3.3 Supplementary Requirements

Non-Functional Requirements:

1. **Performance:** The calculator must return a result for any valid expression quickly.
2. **Usability:** The user interface must be intuitive, ensuring that users can easily input expressions and interpret results.
3. **Reliability:** The system must handle all edge cases and invalid inputs without crashing or producing undefined behavior.
4. **Error Handling:** Clear and informative error messages must be provided for invalid operations (e.g., division by zero, incorrect syntax) and invalid inputs (e.g., non-numeric characters).

Development Constraints:

1. **Technology Stack:** The calculator must be implemented in C++
2. **Coding Standards:** All code must adhere to standard C++ coding conventions, including proper documentation, naming conventions, and structured error handling.
3. **Testing:** The system must include unit tests for all core functionality, ensuring that the calculator handles arithmetic operations and edge cases correctly.

4. Classification of Functional Requirements

Functionality	***Type***
Expression Parsing	*Essential*
Operator Precedence	*Essential*
Parenthesis Handling	*Essential*
Numeric Constants	*Essential*
Error Handling	*Essential*
User Interface	*Essential*

5. Appendices