<Company Name>

Calculator Project
Software Requirements Specifications

Version <1.0>

Revision History

Date	Version	Description	Author
10/08/2023	1.0	Specify Requirements for Calculator Project	Sean Brady, Chris Harvey, Kristin Boeckmann, Naran Bat, Peter Walsdorf, William Morris

Table of Contents

Software Requirements Specifications

1. Introduction

This SRS document outlines all of the requirements and specifications for creating, testing, and producing the calculator project. It will define the purpose, scope, definitions, and list out requirements needed for this application.

1.1 Purpose

The goal of this document is to lay out all the details for creating a calculator app in C++ that everyone can use easily. The SRS will provide a comprehensive description of the external behavior, design constraints, nonfunctional requirements, and other necessary factors for the development of our calculator application that is not only user-friendly but also meets the specified quality standards.

1.2 Scope

This SRS applies to a calculator program model in C++ that will handle functions and constraints that are explained below. The Use-Case model for this application is interaction between the user and program, which will mostly be providing math expressions, handling incorrect inputs, and returning solutions.

1.3 Definitions, Acronyms, and Abbreviations

- Equation: a statement that the values of two mathematical expressions are equal
- Expression: Numbers, symbols and operators grouped together that show the value of something.
- Operators: any symbol that indicates an operation to be performed (e.g. d/dx means to differentiate with respect to x)
- PEMDAS: parentheses, exponents, multiplication, division, addition, and subtraction. Indicates the order of operations (from left to right) of an expression.
- Order of operation: the rules that tell us the sequence in which we should solve an expression with multiple operations

1.4 References

N/A

1.5 Overview

The SRS will describe the overall system, users, hardware, software, and operational constraints and requirements of the calculator application.

2. Overall Description

Humans have for millenia needed to perform complex abstract object representation using mathematics in order to speed up and validate counting. Being able to figure out groups of groups quickly and accurately helps to solve everyday tasks such as baking all the way to expansive, large-scale engineering projects such as the large hadron collider. This application will help users to solve complex counting tasks such as addition, subtraction, multiplication, and division. Which are instrumental in human survival and flourishing.

2.1 Product perspective

The domain of the product will play an important role in defining the requirements. For example, if being used in a scientific context the answer may have to be much more precise whereas if being used at home to do everyday calculations, less decimal accuracy and rounding may be okay – or preferred.

2.1.1 System Interfaces

Standard libraries (for example, math.h and stdio.h) would provide the easiest development experience, as many computers come with these libraries already available for developers to use.

2.1.2 User Interfaces

Unless the user requires equations to be securely saved and available across devices, there will be no requirements related to storing user data, authentication, or authorization.

2.1.3 Hardware Interfaces

Depending on the domain, a keyboard or touch-interface may be appropriate. For desk work, a keyboard would be the most efficient for quickly inputting equations. For jobs where people work on their feet, a touch interface may be more convenient to carry around and use.

2.1.4 Software Interfaces

If users in this domain are used to command line interfaces, a C.L.I would provide users with a very quick input/output experience. However, as many people are not familiar with command line interfaces, a graphical user interface would provide users a very easy and intuitive experience, without having to first read a manual.

2.1.5 Communication Interfaces

Unless the user requires equations to be saved and available across devices, there will be no communication requirements.

2.1.6 Memory Constraints

Unless the program will need to be run on an embedded system with *extremely* low memory, there will be no memory constraints as nothing is being stored after the calculation completes, and the equations will not take up very much memory. If being run on an embedded system, memory may become a concern during development

2.1.7 Operations

How many functions the user needs to perform the necessary calculations will impact the requirements. Some domains require extensive libraries of mathematical functions such as sine, cosine and tangent, where others only require basic mathematical operations.

2.2 Product functions

- The program will take user input.

- The program will validate user input.
- The program will translate well-formed user input into an object that can be evaluated by the program.
- The program will accurately add, subtract, multiply, divide.
- The program will evaluate equations in the correct order according to mathematical order of operations.
- The program will solve exponential equations.
- The program will accept parentheses to customize the order that operations are performed.

2.3 User characteristics

- The user will know how to use a Command Line Interface.
- The user will not need to persist data after program execution.
- The user will not need to authenticate or authorize to use the program.

2.4 Constraints

- The application must be programmed in C plus plus.
- The application must run on GNU/Linux machines.

2.5 Assumptions and dependencies

- The system will be able to compile and execute C programs.
- The system will provide all essential libraries for development (such as stdio.h)

2.6 Requirements subsets

- Input requirements (how the user will be able to input data)
- Validation requirements (how the program will be able to insure valid input)
- Arithmetic requirements (what functions and operations the program will understand and evaluate)
- Output requirements (how the system will output the solved equation or validation errors)

3. Specific Requirements

- User should be able to input valid equation: the user should be able to input valid equations such as 2+2-(834*240)/((log(10^5)^(1/999999999)) and receive and output
- Equations should be validated: Users should **not** be able to input “invalid equations” such as ‘2+483_cat-dog#/0’.
- Program should be able to perform basic arithmetic (addition, multiplication, logarithms, etc…): The program should be able to perform arithmetic operations such as 2+2 and return 4, or 2^2 which would also return 4.
- Application should be launched from an executable file: The program served to users should be a .exe, .deb, or .tar file.
- User will interact with the program through a linux bash interface: The user should be able to access, use, input into, and receive output from the linux bash interface.
- Outputs will be returned and printed to the user’s screen: For any imputed equation, the answer to that equation should be outputted to the users screen
- Users should not be able to input equations past a certain length: Any equation with length greater than a predefined limit, x, should be considered invalid and not be allowed by the program.
- Program should run until user ends it: The program should continue to run until the user exits via some interaction with the process (i.e. the input of a keyword “exit”)
- Program should be able to handle parentheses and expressions inside them in order to determine order of operations: The program should be able to intake parenthesis and nested expressions, and evaluate them normally (e.g. ((2+2)*8)/4 should return 8).
- Program should be able to recognize and evaluate expressions with numeric constants (pi, tau,

square roots): the program should be capable of recognizing and using numeric constants (i.e. an input of "pi" should be valid and an approximation will be used such as 3.1415)

- Use C++ for programming: The C++ programming language should be used for the development of the calculator application
- Application must be easy to use and understand: The application should be simplistic and a user should be able to understand the purpose of and how to use it with minimal effort
- Program should output instructions on using the application upon starting: Upon starting the application, the user should see output instructing them on how to properly interact with the program.

3.1 Functionality

3.1.1 <Functional Requirement One: Arithmetic Operations>

The calculator program shall support the following basic arithmetic operations:

1. Addition: Users should be able to add two or more numbers.
2. Subtraction: Users should be able to subtract one number from another.
3. Multiplication: Users should be able to multiply two or more numbers.
4. Division: Users should be able to divide one number by another.
5. Exponentiation: Users should be able to calculate exponentials (e.g., x^y).
6. Square Root: Users should be able to calculate the square root of a number.
7. Parentheses: Users should be able to clarify the order of operations by using parentheses to bracket off expressions.
8. Variable Assignment: Users should be able to assign specific characters to numeric values.

3.1.2 <Functional Requirement One: Input Handling>

1. The program should allow users to input numbers using both the numeric keypad and the standard keypad
2. It shall support decimal numbers with proper decimal point handling
3. The app should handle invalid inputs gracefully and provide informative error messages

3.1.3 <Functional Requirement Three: Output>

The program will return output in the following ways:

1. Valid input: if the user inputted a valid equation, they should receive a clear and accurate output.
2. Invalid input: if the user did not input a valid equation, they should receive a clear error message (or error messages if applicable) explaining the error.
3. Mathematical errors: the program should be able to report common mathematical errors to the user, such as divide by zero error.

3.1.4 <Functional Requirement Four: Validation>

The program will perform the following validations, and output the following errors:

1. Syntax errors:
    a. The program should process syntax errors (for example, not including closing parenthesis, or an operator)
2. Math errors:
    a. Divide by zero should be able to be inputted and return a clear error message

3.2 Use-Case Specifications

Use case 1: User can input an expression through the CLI, after pressing enter the user will either be shown the answer for the solved expression, or will receive an error message expressing how the entered information was incorrectly formatted.

Use case 2: User can input "help" into the CLI, this will cause output to be shown to the user that gives

detailed instructions on how to use the program

Use case 3: User can input “exit” into the CLI, this will cause the program to terminate.

3.3 Supplementary Requirements

1. Scalability to make sure multiple users or multiple expressions could be run at one time
2. Making the program easy to run and use in the command line, and making it easy to get access to.
3. The program should be able to operate on any software (Mac, Linux, and Windows)
4. Outputs will be printed in the command line interface that is active from the user.
5. Application must be easy to use and understand.
6. User should not be able to input equations past a certain length
7. Program should output instructions on using the application upon starting.
8. Program should handle incorrect inputs (e.g. , ? @ divide by zero, etc.).
9. Program will use C++ for programming

4. Classification of Functional Requirements

Functionality	Type
User should be able to input valid equation	Essential
Equations should be validated	Essential
Program should be able to perform basic arithmetic (addition, multiplication, logarithms, etc…)	Essential
Application should be launched from an executable file	Desirable
User will interact with the program through a linux bash interface	Essential
Outputs of expressions will be returned to user	Essential
Program should run until user ends it	Essential
Program should be able to handle parentheses and expressions inside them in order to determine order of operations	Essential
Program should be able to recognize and evaluate expressions with numeric constants (pi, tau, square roots)	Desirable

5. Appendices

N/A