Doing Math With C++

Software Requirements Specifications

Version 3.0

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 03/10/23 | 1.0 | Initial ideas for the scope, functional requirements, purpose, and definitions | William Hedges, Eric Loseke, Achinth Ulagapperoli |
| 10/10/23 | 2.0 | Ideas for the overall description | William Hedges, Eric Loseke |
| 13/10/23 | 3.0 | Final edits on the file | Manish Singh, Eric Loseke |

Table of Contents

1. Introduction 4

1.1 Purpose 4

1.2 Scope 4

1.3 Definitions, Acronyms, and Abbreviations 4

1.4 References 4

1.5 Overview 4

2. Overall Description 5

2.1 Product perspective 5

2.1.1 System Interfaces 5

2.1.2 User Interfaces 5

2.1.3 Memory Constraints 5

2.1.4 Operations 5

2.2 Product functions 5

2.3 User characteristics 5

2.4 Constraints 5

2.5 Assumptions and dependencies 5

2.6 Requirements subsets 5

3. Specific Requirements 5

3.1 Functionality 5

3.2 Use-Case Specifications 6

3.3 Supplementary Requirements 6

4. Classification of Functional Requirements 6

5. Appendices [6](#_heading=h.qsh70q)

Software Requirements Specifications

# Introduction

This document examines the purpose, scope, definitions, acronyms, abbreviations, references, and overview of the Software Requirements Specifications. It explains the requirements for building a calculator in C++. We go into how we will use a Stack to act as an order of operations of addition, subtraction, multiplication, division, parenthesis, modulus, and exponents. We will prompt the user for a mathematical expression, iterate through the expression, looking for operators like +-\*, etc., and then push that to the stack.

## Purpose

The purpose of this document is to outline the requirements to build a calculator using C++. We need this document because it is the blueprint to build our code. The SRS fully describes the external behavior of the application or subsystem identified. It also describes nonfunctional requirements, design constraints, and other factors necessary to provide a complete and comprehensive description of the requirements for the software.

## Scope

This applies to the software application of our calculator using C++. It is associated with the use case of a user wanting to enter math and have the solution printed.

## Definitions, Acronyms, and Abbreviations

* Stack: a FILO structure that stores values. Typically has pop(), push(), peek(), and isEmpty() methods.
* FILO: first in last out
* %: Operator for modulus: the remainder of the division operation.
* /: Operator for division
* \*: Operator for multiplication
* ^: Operator for exponentiation
* (: Opening parentheses
* ): Closing parentheses
* PEMDAS: Acronym for the order of operations: Parentheses, Exponents, Multiplication, Division, Addition, Subtraction.
* Stack overflow: Error called when attempting to push onto a full stack
* Stack underflow: Error called when attempting to pop from an empty stack

## References

We do not reference another document.

## Overview

The rest of the SRS contains descriptions of the Requirements and goes into details, such as the functional requirements and use-case specifications.

# Overall Description

**Product perspective: The user can enter a mathematical expression and have the result printed.**

### System Interfaces: There will be one interface where the user will interface with the prompt that asks the user to enter math, and the user enters an expression.

### User Interfaces: Same as the system interface.

### Memory Constraints: We will use doubles with 64-bit floating precision.

### Operations: addition, subtraction, multiplication, division, modulo, exponentiation

## Product functions: To evaluate the result of basic mathematical expressions that the user inputs

## User characteristics: A person who needs the solution to a mathematical expression

## Constraints: Must be implemented with C++, must use a data structure like a stack.

## Assumptions and dependencies:

## Assume the user will enter a mathematical expression.

## We depend on each other to collaborate and finalize the code.

## We assume that we will complete the project by the allotted time, which is the deadline set by Prof. Saiedian.

## Requirements subsets:

* Arithmetic Operations: Define the mathematical operations the calculator should support, such as addition, subtraction, multiplication, and division.
* User Input: Specify how users can input numbers and operators
* Calculation Execution: Detail the calculator performs calculations, including the order of operations and error handling

# Specific Requirements

## Functionality

**Input Processing**

* The calculator will accept user input as a string containing an arithmetic expression containing operators +, -, \*, /, %, ^, and numeric constants. The calculator will be able to process inputs grouped in parentheses by the user if needed to specify the order of operations.
* The calculator will check if the user input is a valid arithmetic expression and throw an exception if a user inputs an invalid expression.

**Expression Evaluation**

* The calculator will evaluate mathematical expressions based on the order of operations according to the PEMDAS rules and print the result.
* The calculator will support addition, subtraction, multiplication, division, modulo, and exponentiation.
* The calculator will evaluate multiplication and division from left to right.
* The calculator will evaluate addition and subtraction from left to right.
* The calculator will evaluate expressions and operators using a stack.

**Stack Operations**

* The calculator will use stack operations (including push, pop, peek, and isEmpty) to handle the order of operations to evaluate the arithmetic expression.
* The calculator will be able to process stack overflow and stack underflow.

**User Interface**

* The calculator will have a user-friendly interface for input and output on the command line that displays the result of the expression, allows the user to make a new calculation, and allows the user to exit the calculator.

**Error Handling**

* The calculator will throw exceptions when given errors by the user, like division by zero or invalid input, and display appropriate error messages that help the user fix the error.

## Use-Case Specifications

Prompt user to input text

Numeric Constants: Recognize and calculate numeric constants within the expression.

Requires valid expression from the user, but we will have error handling such as divide by zero.

Can handle addition and subtraction

Can handle multiplication and division

Can handle modulus and exponentiation

Knows how to group expressions by parentheses

Prompt user for input

Invalid input alert

Division by zero warning

## Supplementary Requirements

* We must use C++ and use a data structure like a Stack.
* Must be posted on GitHub and be shared from GitHub.
* Use object-oriented programming principles to structure the code.
* Include comments and documentation to explain the logic and functionality of the calculator.
* Develop unit tests to verify the correctness of the calculator.
* Ensure the program provides clear and informative error messages for invalid input.

# Classification of Functional Requirements

|  |  |
| --- | --- |
| **Functionality** | **Type** |
| Numeric Constants: Recognize and calculate numeric constants within the expression. | Essential |
| Can Handle Add and Subtract | Essential |
| Can Handle Multiply and Divide | Essential |
| Can Handle Modulus and Exponents | Essential |
| Knows how to group by Parentheses | Essential |
| Prompt user for input | Essential |
| Invalid input alert | Essential |
| Division by zero warning | Essential |

# Appendices

# No appendices are needed