Whatever Co.

Whatever Calculator
Software Requirements Specifications

Version <1.0>

Revision History

Date	Version	Description	Author
<17/10/24>	<1.0>	<First version of the requirements spec>	<All team members>

Table of Contents

Software Requirements Specifications

1 Introduction

This Software Requirements Specification (**SRS)** document outlines the functional and non-functional requirements for the development of our arithmetic calculator, *Whatever Calculator*. The calculator's performance design can be located in *Section 1.2 Scope* and the specific description, timeline, and deadlines can be located in our *Software Development Plan* (**SDP**). This document will serve as a guide for the development team, stakeholders, and testers by providing a solid understanding of the system's expected performance and capabilities.

1.1 Purpose

The purpose of this **SRS** is to define the functionalities and the various characteristics of the software program. It provides detailed descriptions of how the system should operate and it outlines key features and user interactions. The intended audience includes the development team, quality assurance engineers, and stakeholders who are involved in the project.

1.2 Scope

Our arithmetic calculator software will be capable of handling basic arithmetic operations in the order of operations (PEMDAS). It will allow users to perform calculations involving operands with both integers and floats (decimal numbers). The system will include a user-friendly interface for a simple procession of inputs and outputs, and it will be designed to work on desktop platforms in a terminal. Our project is associated with the use-case model for an arithmetic expression evaluator that supports the following operations:

- Addition (+)
- Subtraction (-)
- Multiplication (*)
- Division (/)
- Modulus (%)
- Exponentiation (**)

1.3 Definitions, Acronyms, and Abbreviations

SRS: Software Requirements Specification. *See section 1.4 References - SRS*

SDP: Software Development Plan. *See section 1.4 References - SDP*

Use Case Specifications: Description of how systems interact with external users to accomplish a specific goal. *See section 3.2 for Use Case Diagram*

Functionality Requirements: The desired behaviors, operations, or functions a system must perform to meet user needs.

Supplementary Requirements: The non functional aspects of a system that is not already covered by use case specifications or already stated in functionality requirements

1.4 References

SRS: https://github.com/i662m589/EECS348TermProject

SDP: https://github.com/i662m589/EECS348TermProject

1.5 Overview

The rest of the document provides description on the product perspective, specific functionality including functionality and supplementary and proves a use case diagram to provide clarity for the overall product.

Overall Description

2. Product perspective

2.1.1 User Interfaces

If a user runs this program directly, then the user should be able to type expressions and have the evaluations of those expressions returned to them. If the user types an invalid expression, the program should inform the user that the expression is invalid as well as why said expression is invalid.

2.1.2 Software Interfaces

Per the project description, an external program should be able to pass an arithmetic expression into this one and receive the value of that expression in return in some form. If an invalid expression is entered, then the program should hand an error to whatever program called it.

2.1.3 Memory Constraints

The program should not consume a large amount of memory as all the program needs is a user interface and the means to evaluate functions. In addition, for the sake of stability and good practice, care should be taken to ensure no memory leaks exist.

2.2 Product functions

As the arithmetic parser, this interpreter's function is to be able to perform any necessary mathematical expression provided to it.

2.3 User characteristics

The user is assumed to have either read the documentation for this program or be familiar enough with arithmetic expressions to use this program.

2.4 Constraints

Ideally, this program should be able to work on any common operating system (Windows, Linux). Additionally, as established previously, other programs should be able to call this program to evaluate expressions.

2.5 Assumptions and dependencies

Since the program is stated to be a component for a larger program that is developed using C++, if the software is interacted with by other software, we can assume the other software is made using C++, and can communicate with that software in any way C++ allows. Beyond that, no assumptions or dependencies are to be made.

Specific Requirements

3. Functionality

3.1.1 Expression input via a command line interface:

The user should be able to enter an arithmetic expression via command line.

The system should support a number of arithmetic operators:

The system should support the following operators: addition, subtraction, multiplication, division, modulo, and exponentiation.

The system shall correctly evaluate expressions:

Different expressions should be parsed and handled correctly such as nested parentheses,

numeric constants, and follow PEMDAS rules. Inputs should also be tokenized and stored in a data structure to correctly represent the expression.

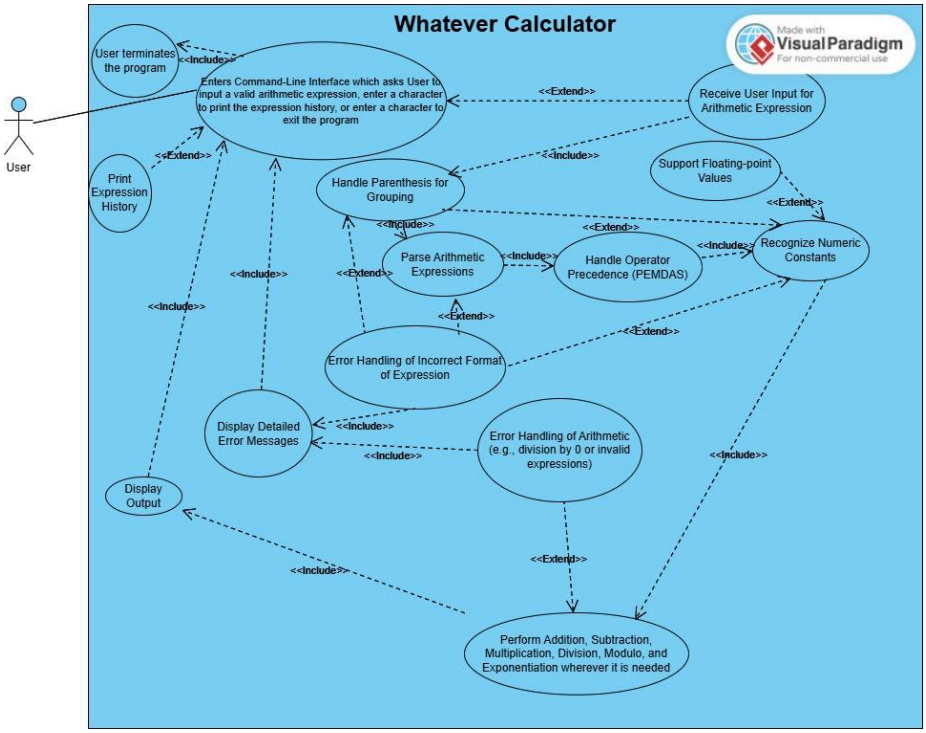
The system should display the result of a valid arithmetic expression from the user:

All valid inputs should be handled and displayed correctly as an output

Invalid inputs should be handled correctly:

Any invalid inputs from the user such as unmatched parenthesis or dividing by zero should be handled with a clear error message.

3.2 Use-Case Specifications



This use case shows the user interacting with our arithmetic calculator. First, when the user opens the program, they will be asked to either input a valid arithmetic expression, print any previous expressions that were inputted, or terminate the program (This satisfies the first functional requirement by allowing the user to input their expression via a command line interface). If the user inputs an expression, the program will receive the input before handling the grouping of parenthesis in the expression. An error can occur in this section if the user inputs an invalid grouping of parenthesis (I.E. parenthesis that were never closed). The system then parses the system for arithmetic expressions, where this can error if an invalid character was inputted into the system (This part of the system satisfies the second functional requirement by allowing the system to support the arithmetic operators that were stated in the requirement). The system then handles operator precedence for the expression using PEMDAS and recognizes the numeric constants that are associated with each arithmetic operation (it also recognizes here if the program should support floating-point values). These two parts of the system fulfill the third functional requirement by correctly evaluating the expressions using PEMDAS and handling the numeric constants so that the expression is represented correctly. The system then performs the arithmetic operators that were parsed in the expression, where the system will handle an error if an invalid arithmetic operation is performed (such as dividing by zero). After performing the operations, the system will print the solution to the user, which satisfies the fourth functional

requirement of being able to display the result to the user. The user is then asked the same three inputs that they were asked at the beginning. If the user asks to print the expression history, the program will print the previous solutions that were inputted into the system (or nothing if no expression has been processed). If the user asks to exit the program, the program will terminate. If an invalid input is entered in any part of the program, an error message will appear and ask the user to input another value or expression. The fifth functional requirement, which states that all invalid inputs will be handled with an error message, is therefore satisfied since all invalid inputs will be handled by displaying an error message to user before asking for their input once more.

3.3 Supplementary Requirements

- *The system should evaluate expressions with a time complexity of O(n) with n being the number of tokens in the input expression.*
- *A clear and user-friendly command line for easy use.*
- *The system must be implemented in C++ and follow object-oriented principles.*
- *Code should be structured for future adaptability and change such as handling float input values.*

4 Classification of Functional Requirements

Functionality		Type
Receive user input of arithmetic expression		Essential
Parse arithmetic expressions		Essential
Handle operator precedence (PEMDAS)		Essential
Support addition, subtraction, multiplication, division, modulo, exponentiation		Essential
Recognize numeric constants		Essential
Handle parentheses for grouping		Essential
Command line user interface		Essential
Error handling(e.g., division by 0 or invalid expressions)		Essential
Support floating-point values		Desirable
Detailed error messages		Desirable
Expression history		Optional

5 Appendices

Appendix A: Glossary of Terms

- **Expression**: A sequence of operands (numbers) and operators (e.g., +, -, *, /) that can be evaluated to produce a value.

- **Parse**:The process of analyzing an expression to determine its structure and meaning.
- **Operator Precedence**: The rules governing the order in which different operators are applied in an expression.
- **Integer values**:Whole numbers without any fractional or decimal component (e.g., 1, 42, -7). These values are usually processed directly as part of arithmetic operations.
- **float-point values**: Numbers that contain fractional parts, expressed with decimal points (e.g., 3.14, -0.5, 2.718). These are processed using floating-point arithmetic.
- **PEMDAS**: An acronym for the order of operations in arithmetic:
    - **P**: Parentheses — expressions inside parentheses are evaluated first
    - **E**: Exponents — exponents or powers are evaluated next.
    - **MD**: Multiplication and Division — evaluated from left to right.
    - **AS**: Addition and Subtraction — also evaluated from left to right.

Appendix B: Error Handling - Part of the requirements

This is a list of all of the possible errors, they will all be handled by returning an error message

- **Operator Without Operands**: * 5 + 2
- **Incorrect Operator Usage**: 4 / 0
- **Missing Operator**: 5 (2 + 3)
- **Invalid Character**: 7 & 3
- **Mismatched Parentheses**: ((3 + 2)

Appendix C: Operator Precedence and Associativity - Part of the requirements

Operator	Description	Precedence	Associativity
+, -	Addition, Subtraction	1	Left
*, /, %	Multiplication,Division, Modulo	2	Left
**	Exponentiation	3	Right

Appendix D: External References

- C++ Standard Library(<cmath>)
    - Provides mathematical functions such as pow(), fmod(), and sqrt() for handling operations like exponentiation, floating-point modulo, and square roots.