HEAAP Software

# Arithmetic Expression Evaluator in C++
# Software Requirements Specifications

**Version 1.1**

# Revision History

| Date | Version | Description | Author |
|---|---|---|---|
| 10/08/2024 | 1.0 | All team members discussed and were assigned sections.<br>Peter – 3.1, 3.2, cleanup<br>Hart – 2.1.2, 2.1.4, 3.3<br>Emily – 2.2-2.4<br>Arpa – 1.1-1.5, 5<br>Addison – 2.1.6, 2.5, 4<br>Location: LEEP | Addie, Peter, Arpa, Emily, Hart |
| 10/20/2024 | 1.1 | Completed Software Requirements Specifications | Addie, Peter, Arpa, Emily, Hart |
| | | | |
| | | | |

# Table of Contents

# Software Requirements Specifications

## 1. Introduction

### 1.1 Purpose

A Software Requirements Specification (SRS) document fully defines the external behavior of an application. It outlines what the software is supposed to accomplish without going into how it will be implemented. The document not only details functional requirements like what the system should do, but also specifies nonfunctional requirements, which includes performance, security, usability, and scalability standards that the software must meet. The document addresses design constraints, such as technology limitations or compliance with industry standards, ensuring that the system is built within specific standards. Overall, the Software Requirements Specification document helps align stakeholders and ensure that the final product meets expectations and lays the groundwork for efficient development and testing.

### 1.2 Scope

This Software Requirements Specification document applies to Visual Studio Code (VS Code). This application is a versatile and lightweight code editor. The software is designed for code editing and development, supporting multiple programming languages and offering features like debugging, syntax highlighting, and version control integration. The primary feature grouping includes the core editor, extensions management, Git integration, debugging tools, and terminal access. This SRS is associated with key use-case models, like writing and editing code, creating, modifying, and saving files in many programming languages. It also debugs code and handles Git repository tasks like committing, pushing, pulling, and managing within the editor.

### 1.3 Definitions, Acronyms, and Abbreviations

*See Glossary in Appendix*

### 1.4 References

*See References in Appendix*

### 1.5 Overview

The rest of the Software Requirements Specification (SRS) document outlines the overall description and specific functional requirements for a command-line calculator. It is organized into three main sections. The first section covers the Overall description. This section is about the general factors that affect the product and its requirements, including the different types of interfaces, operations and functions. The next section is the Specific Requirements which discusses the required functionality, which includes supporting basic arithmetic operations such as addition, subtraction, multiplication, division, modulo, and exponentiation. Also includes following mathematical rules like PEMDAS, and handling parentheses. The last main section is Classification of Functional Requirements. This specifies the functionality and their types. For example, that user input is an essential function. The last part of this document is the Appendices which includes the references and glossary.

# 2. Overall Description

## 2.1 Product perspective

### *2.1.1 System Interfaces*

### *2.1.2 User Interfaces*

- The user interface will be a command line interface. It accepts input in the form of arithmetic expressions from the user. The calculator will use this input to create the output and print it in the terminal.

### *2.1.3 Hardware Interfaces*

### *2.1.4 Software Interfaces*

- Environment: the program will run as a standalone executable file in the command-line environment.
- Input: the input will be accepted as a string containing arithmetic expressions through the command-line.
- Output: the output will be returned and printed as a number. If an error occurs, an error message will be printed.
- Libraries: no external libraries will be used. The only libraries used will be for math (cmath) and I/O (iostream).

### *2.1.5 Communication Interfaces*

### *2.1.6 Memory Constraints*

There are no significant memory constraints given the scope of the project and execution target of a modern general-purpose computer.

### *2.1.7 Operations*

## 2.2 Product functions

The system will provide a command-line interface for parsing and evaluating arithmetic expressions. It will support basic operations like addition, subtraction, multiplication, division, modulo, and exponentiation, while ensuring expressions follow mathematical conventions such as operator precedence (PEMDAS) and correct use of parentheses.

## 2.3 User characteristics

Users are expected to have basic familiarity with arithmetic and algebraic expressions but do not need to be technically advanced. The system will guide users with clear error messages and an intuitive command-line interface, ensuring ease of use even for those with limited technical experience.

## 2.4 Constraints

The system must respect mathematical rules, handle edge cases like division by zero, and scale efficiently with more complex expressions. It will need to provide meaningful error feedback and a user-friendly interface, while ensuring robust performance and accurate parsing.

## 2.5 Assumptions and dependencies

There are no dependencies beyond the stock libraries included with C++. The assumptions will largely reflect the constraints, but some are as follows: 1) The project will be made with very limited monetary and time resources. 2)

The finished software will run on the EECS Linux machines.

### 2.6 Requirements subsets

## 3. Specific Requirements

### 3.1 Functionality

#### *3.1.1 User Input*

The program must be able to ask for and store an expression from the user. This expression should be checked for errors before it is passed off to the parser to ensure that it is a valid expression. If it is not, the program should display an error message and allow user to enter a new expression.

#### *3.1.2 Expression Parsing with Operator Support*

The program must parse through the valid expression entered by the user with support for the following operators: + (addition), - (subtraction), * (multiplication), / (division), % (modulo), ** (exponentiation). Parsing must happen from left to right while respecting PEMDAS and recognizing or calculating numeric constants as necessary.

#### *3.1.3 Parentheses Handling*

The program must be able to parse through expressions that use parentheses. The program should display an error message if parentheses are used incorrectly, otherwise it should correctly parse the expression with respect to the parentheses.

#### *3.1.4 User Interface*

The program must have a user-friendly interface that works with the command line to get user input and show results as output.

#### *3.1.5 Error Handling*

The program must display descriptive error messages when the expression is not valid for any reason or gives an undefined result. The program then must allow the user to input a new expression.

### 3.2 Use-Case Specifications

There is only one use-case path where the user is involved because all the user does is enter inputs, but actions done by the program will be considered as well.

User Input: The user is the actor and enters an expression in which the post condition is error-checking and then storage in the program. The path will then go to parsing.

Re-enter User Input: If an error is raised by the user input, the program will handle the error and present a message to the user in the interface. The precondition is an invalid expression entered by the user. The user is the actor and will then be able to input a new expression and the post condition is a valid expression being stored by the program and sent to the parser which is next in the use-case path.

Expression Parsing: The precondition is correctly error-checked user input of an expression and there are no outside actors meaning it is all done within the computer. The math library may possibly be considered an actor. The expression will be evaluated, and errors will be handled if they come up while calculating the simplification of the expression. The post condition for this use case is that a resulting value will be returned which then goes to the next use case in the path which is the output in the interface.

Output in User Interface: The computer is the only actor here and the precondition is a value returned from the parser function. The results of the expressions evaluator must be printed in a user-friendly format in the user interface which is the command line. The post condition is that the details of this use-case do indeed occur and that the user actor reads the result. This is the end of the use-case path.

### 3.3 Supplementary Requirements

- Efficiency: The program should handle typical arithmetic expressions efficiently, even for complex nested expressions.
- Scalability: The system should handle expressions with increasing complexity (i.e., multiple operators, parentheses) without significant performance degradation.
- Compatibility: The program should be portable across platforms that support C++ (e.g., Windows, Linux, macOS). It must compile and run on any standard C++ environment.
- Usability: A README file will be provided, explaining how to use the program, with examples of input expressions and outputs. The error messages should be descriptive enough to guide the user toward correcting invalid input.

## 4. Classification of Functional Requirements

| Functionality | Type |
| --- | --- |
| ...User Input | Essential |
| ...Expression Parsing with Operator Support | Essential |
| ...Parentheses Handling | Essential |
| ...User Interface | Essential |
| ...Error Handling | Essential |

## 5. Appendices

Appendices are not part of the requirements but are included to supplement the document.

### 5.1 References

"IEEE Recommended Practice for Software Requirements Specifications," in IEEE Std 830-1998 , vol., no., pp.1-40, 20 Oct. 1998, doi: 10.1109/IEEESTD.1998.88286.

### 5.2 Glossary

Software Requirements Specification (SRS) – A document that describes the system's requirements.

Command-line Interface (CLI) – A text-based interface used to interact with software by typing commands.

Arithmetic Expression – A combination of numbers and operators (such as +, -, *, /) used in mathematical calculations.

Standalone Executable – A program that runs independently without needing other software to function.

PEMDAS – An acronym representing the order of operations in arithmetic: Parentheses, Exponents, Multiplication, Division, Addition, Subtraction.

Parsing – The process of analyzing a string of symbols.

Error Handling – The process of responding to and managing errors in a program.

Modulo – A mathematical operation that finds the remainder when one integer is divided by another.

Exponentiation – A mathematical operation involving raising a number to the power of another number.

Parentheses Handling – Managing and correctly interpreting expressions that use parentheses in arithmetic.

User-Friendly – Software designed to be easy to use for users.

Edge Case – A problem or situation that occurs only at an extreme (maximum or minimum) operating parameter.

Library – A collection of precompiled routines that a program can use.
Efficiency – The ability of the system to perform tasks.

Scalability – The capability of a system to handle a growing amount of work or its potential to accommodate growth.

Reliability – The system's ability to perform its required functions under stated conditions for a specified period.

Usability – The ease with which a user can interact with a system to achieve desired goals.

Division by Zero – An undefined mathematical operation where a number is divided by zero, which must be handled carefully in programming.

Linux – An open-source operating system commonly used in academic and engineering environments.