**Team 19**

# EECS 348 Term Project
# Software Requirements Specifications

**Version <1.0>**

## Revision History

| Date | Version | Description | Author |
|---|---|---|---|
| <10/20/24> | <1.0> | Created and filled out the document | Team 19 |
| | | | |
| | | | |
| | | | |

# 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 Hardware Interfaces 5
     - 2.1.4 Software Interfaces 5
     - 2.1.5 Communication Interfaces 5
     - 2.1.6 Memory Constraints 5
     - 2.1.7 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
     - 3.1.1 Expression Parsing
     - 3.1.2 Evaluate Operator Precedence 7
     - 3.1.3 Handle Parentheses 7
     - 3.1.4 Numeric Constants 7
     - 3.1.5 Error-Handling 7
     - 3.1.6 Command-Line User Interface 8
   - 3.2 Use-Case Specifications 6
   - 3.3 Supplementary Requirements 6

4. Classification of Functional Requirements 6

5. Appendices 6

# Software Requirements Specifications

## 1. Introduction

### 1.1 Purpose

The purpose of this Software Requirements Specification (SRS) is to provide a comprehensive description of the arithmetic expression evaluator system, including its external behavior, functional requirements, and non-functional requirements. This SRS serves as the foundation for the design, development, and testing of the system by detailing the following:

- **External Behavior**: The main factor behind the external behavior of this system is regarding the system-user interaction through the CLI. The user shall input expressions, and in turn, the program will return either the evaluated result or display an error implying the format isn't suitable for solving.
- **Functional Requirements:** This particular system includes the functional requirements of expression parsing, operator handling, error management, and CLI interactions.
- **Nonfunctional Requirements:** In this particular system there are a few nonfunctional requirements, these include a response time of <100ms and a user-friendly design for the CLI.
- **Design Constraints:** This evaluator must be implemented in C++ using the standard libraries, it must also support error handling, the order precedence for PEMDAS, and finally it must operate within the standard command-line environment.

### 1.2 Scope

This Software Requirements Specification (SRS) applies to the development of an arithmetic expression evaluator, a command-line application designed to evaluate mathematical expressions based on operator precedence rules (PEMDAS). The system supports arithmetic operations, like addition, subtraction, multiplication, division, modulo, and exponentiation, while also handling parentheses and various error conditions.

Features and Subsystems:

- Expression Parsing: The program will parse and tokenize a user-input arithmetic expression, and recognize operators, constants, and parenthesis.
- Operator Precedence: This program will also follow the rules of PEMDAS to ensure proper order of operations on the expression.
- Error-Handling: The program will detect errors like division by zero, unmatched parenthesis, and some sort of invalid syntax.
- Command-Line Interface: In the text-based interface the user will enter their respective expression and receive feedback (either an error message or an evaluated result).

Use-Case Model:

Our use-case modeling is particularly focused on the interactions between the user

entering the expression and the evaluator itself.

- Valid Input: The user must input a correctly formatted arithmetic expression, in turn, the system returns the evaluated result.
- Invalid Input: The user inputs an expression with errors, in turn, the system returns a clear error message.

### 1.3 Definitions, Acronyms, and Abbreviations

- SRS:
  - Software Requirements specification, a document that describes the requirements in detail
- PEMDAS:

- The order of operations in arithmetic expressions - Parentheses, Exponents, Multiplication/Division, and Addition/Subtraction

- CLI:
  - Command-Line Interface, text-based interface where users input/receive information directly from a console
- C++:
  - A general-purpose programming language known for its efficiency and object-oriented programming
- Stack:
  - A  Data structure that follows Last-In-First-Out, think a stack of plates
- Parsing:
  - The process of analyzing a string to identify components and structure
- Tokenization:
  - Breaking down a string into manageable parts (tokens) such as numbers, operators, parentheses, etc
- Operator Precedence:
  - The rules that define the order in which different operations are performed in arithmetic
- Binary Tree:
  - A data structure in which each node has at most two children
- Modulo (%):
  - An operator that returns the remainder of a division operation
- Error Handling:
  - Mechanics in the program that detect and manage issues such as invalid syntax, unmatched parenthesis, or division by zero

## 1.4 References

- C++ Standard Library Documentation
  - Available from https://en.cppreference.com/w/
- Project Introduction Document
  - 00-2024-EECS348-Term-Project.pdf
- Project Plan Document
  - 01-Project-Plan.docx

## 1.5 Overview

- Section 1
  - Introduces the scope, purpose, and background of the project, along with defining references and terms used within the rest of the document
- Section 2
  - Gives a description of the system including user interfaces, software interfaces, product functions, and assumptions
- Section 3
  - Gives details to specific functional requirements such as - expression parsing, parentheses handling, operator precedence, error detection, and the CLI
- Section 4
  - Gives classification of the functional requirements based on priority and type
- Section 5
  - The appendices that offer additional reference material and documentation

# 2. Overall Description

The Expression Evaluator is a tool that will process and solve math expressions entered by users. it will work together with many other modules and functions

## 2.1 Product perspective

The Arithmetic Expression Evaluator is a small part of a bigger program called a compiler for the programming language C. This project's job is to read math problems provided by a user, it figures out what the problem is asking and gives out the correct answer. Its goal is to work on its own but is designed to easily work with other components of the program.

### 2.1.1 User Interfaces

*the product will use a Command line interface where the user can input the arithmetic expression directly. The interface will provide a prompt for the user and take in their input then print the output of the expression.*

### 2.1.2 Software Interfaces

The software will be developed using C++ and use standard libraries for input/output operations, error handling, and data structure management, such as stacks or binary trees for storing and processing expressions. all the functions in the code will need to interact with each other to ensure the proper output is given.

### 2.1.3 Memory Constraints

We want the evaluator to be designed to be as memory-efficient as possible. the system will primarily process integer inputs and in the future will need to process floats

## 2.2 Product functions

The Expression evaluator will need to Parse through the arithmetic expressions and identify the different components of the expression it will then need to either evaluate the expression based on operator precedence (PEMDAS) or handle an error that was inputted and provide the proper error message for the detected error. and lastly it needs to be intractable with a Command Line Interface that prompts the user for an expression and provides the output.

## 2.3 User characteristics

The evaluator is designed for users who have an understanding of arithmetic expressions and a basic understanding of command-line operations.

## 2.4 Constraints

Some of the constraints we have are the error handling because the evaluator must be able to handle different error cases such as division by zero, invalid syntax, and unmatched parenthesis. Another constraint is the language we need to use. We are constrained to using C++ and not any other language. the product must operate in the standard command line environments.

## 2.5 Assumptions and dependencies

Assumption:

We assume that the user has access to a command-line tool

We assume that the evaluator will be used as part of a bigger program that processes programming language code

Dependencies:

The tool relies on C++ libraries that handle the inputs that are given.

## 3. Specific Requirements

The Specific Requirements for the Arithmetic Expression Evaluator describe the essential functions, performance standards, interfaces, limitations, and dependencies needed for the system to work as intended. The requirements are broken down as follows:

### 3.1. Functionality

Expression Parsing: we want the program to be able to parse arithmetic expressions that are entered by the user

Operator support: the system needs to be able to support all the different operators (Ex. +, -, *, /, %, and **)

Parenthesis Handling: program needs to be able to handle parenthesis which ensures that the correct operation is precedence.

Numeric constants handling: Should be able to handle integers and floats in later iterations

Error Handling: like every good code our program needs to handle errors such as division by zero or unfinished parenthesis and other errors that we come across and provide a clear error message to describe the issue.

User interface: the user interface should be simple and easy to use and be able to display clear results and take in clear inputs

#### *3.1.1. Expression Parsing*

The program should parse the arithmetic expressions provided by the user. This must identify and tokenize components of the expression, this includes numeric constants, all the different operators, and also the parenthesis. These components will be stored in a data structure such as a stack to assure the evaluation is done properly based on the operator precedence.

#### *3.1.2. Evaluate Operator Precedence*

The system shall evaluate arithmetic expressions following the standard operator precedence rules (PEMDAS). The system will first evaluate expressions inside parentheses, followed by exponentiation (**), multiplication/division/modulo (*, /, %), and finally addition/subtraction (+, -). The evaluation will be performed from left to right for operators of equal precedence.

#### *3.1.3. Handle Parentheses*

The program should correctly handle expressions with parentheses to ensure proper grouping and precedence. Parentheses will be evaluated from the innermost to the outermost, ensuring that expressions enclosed in parentheses are computed before any surrounding operations.

#### *3.1.4. Numeric Constants*

The program should recognize and evaluate integer numeric constants in expressions. Future iterations of the system should include support for floating-point numbers, with appropriate changes to the parser and evaluator to handle decimal values.

#### *3.1.5. Error Handling*

*The program should provide error handling for invalid input expressions. This includes:*

- Detecting unmatched parentheses.
- Handling division by zero.
- Flagging the use of unsupported characters or operators.

- Returning error messages to the user when an invalid expression is detected.

*3.1.6. Command-Line User Interface*

The system shall provide an interface that allows users to input arithmetic expressions and receive evaluated results. The interface will ask users to enter an expression, display the calculated result, and provide clear error messages.

### 3.2. Use-Case Specifications

Example use-case: Invalid / Valid user input

- The user opens the program by running the file or executable in a terminal.
- The user is prompted to enter their mathematical expression.
- Upon entering a mathematical expression, the program does the following:
  - Handles parenthesis to ensure proper grouping and precedence.
  - Recognizes valid constants included in the expression.
  - Parses and tokenizes user input to be processed by the program.
  - Identifies proper edge cases such as dividing by zero or invalid input.
- If the user's input is valid, then the resulting number is displayed to the user.
- If the user's input is invalid, then the error is displayed to the user.
- The program then exits.

### 3.3. Supplementary Requirements

- The program should execute and return either a number or error, depending on user input, within a reasonable time (< 100ms).
- The program should only exit after the resulting number or an error is displayed to the user, and should not crash or malfunction with any user input.
- The expression should be evaluated in PEMDAS order, and operator precedence should also be considered.
- The interface should be user-friendly, and simple for one to input expressions.

## 4. Classification of Functional Requirements

| Functionality | Type |
|---|---|
| Expression Parsing | Essential |
| Evaluate Operator Precedence | Essential |
| Handle Parentheses | Essential |
| Numeric Constants | Essential |
| Error Handling | Essential |
| Command-Line User Interface | Essential |

## 5. Appendices

**N/A**