# Taus

# Arithmetic Expression Evaluator
# Software Requirements Specifications

**Version 1.0**

# Revision History

| Date | Version | Description | Author |
|---|---|---|---|
| 10/20/24 | 1.0 | Completed filling out the requirement specifications and solidified plans for programming the project. | Charlie Doherty, Jonathan Gott, Josh Dwoskin, Calder Kamman |
| | | | |
| | | | |
| | | | |

# 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 <Functional Requirement One> 6
   - 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 SRS is to detail the requirements for the Arithmetic Expression Evaluator which is our C++ program designed to evaluate arithmetic expressions using certain operators. This document provides a comprehensive outline of functional requirements, non-functional requirements, and constraints on the project. This ensures that the program meets the expected behaviors and performance standards.

### 1.2 Scope

The scope of this SRS is to handle expressions with basic arithmetic operators, including addition (+), subtraction (-), multiplication (*), division (/), modulo (%), and exponentiation (**). It supports parsing of expressions that contain parentheses to ensure correct evaluation according to the order of operations. This program will serve as a core component of a broader compiler product for language L.

### 1.3 Definitions, Acronyms, and Abbreviations

PEMDAS: Parentheses, Exponents, Multiplication and Division, Addition and Subtraction

UI/UX: User Interface/User Experience

C++: A programming language used for the development of the evaluator.

### 1.4 References

EECS 348 Term Project Specifications (2024)

Project Plan for Arithmetic Expression Evaluator, Version 1.2, September 25, 2024.

### 1.5 Overview

This SRS document includes a detailed description of the software's functional and non-functional requirements. It outlines the overall system architecture, use-case specifications, and supplementary requirements. The document aims to ensure that the software's design aligns with its intended functionality, providing a basis for testing and validation.

# 2. Overall Description

### 2.1 Product Perspective

The Arithmetic Expression Evaluator is designed as a standalone utility but is intended for integration into a larger compiler framework for language L. The evaluator acts as a critical component in interpreting arithmetic expressions and evaluating them correctly, adhering to established precedence rules.

- **2.1.1 System Interfaces**: The evaluator interacts primarily with the user through a command-line interface (CLI), accepting input directly from the user and returning output to the same interface. When integrated into the larger compiler framework, the evaluator will interface with other modules of the compiler through predefined function calls.
- **2.1.2 User Interfaces**: The command-line interface prompts users to enter arithmetic expressions, displaying calculated results or error messages when applicable. The interface is designed for ease of use, requiring minimal user instructions while providing clear feedback.
- **2.1.3 Hardware Interfaces**: The evaluator does not require direct interaction with hardware, as it operates entirely within a software environment on any standard computing device capable of running C++ programs.
- **2.1.4 Software Interfaces**: The evaluator depends on the C++ standard library for string manipulation, input/output operations, and handling basic data structures like stacks. It also requires a compiler that supports C++17 or newer.
- **2.1.5 Communication Interfaces**: Communication is limited to user input and output through the command line. For integration purposes, a simple API can be developed to allow other software components to invoke the evaluator's functionality.
- **2.1.6 Memory Constraints**: The software is designed to run efficiently on systems with at least 1 GB of RAM. Memory consumption is mainly determined by the size of the input expressions and the depth of nested operations. For typical arithmetic expressions, the memory requirements are minimal.
- **2.1.7 Operations**: The evaluator is executed as a command-line program, taking a single expression as input, parsing it, evaluating it, and outputting the result. Users can repeatedly enter expressions without restarting the program.

### 2.2 Product Functions

The primary functions of the Arithmetic Expression Evaluator include:

- Expression Parsing: Breaks down user input into tokens that represent numbers, operators, and parentheses.
- Precedence Handling: Evaluates expressions according to the rules of operator precedence and parentheses.
- Arithmetic Operations: Supports basic operations like addition, subtraction, multiplication, division, modulo, and exponentiation.
- Error Detection: Identifies and provides feedback on issues such as division by zero, mismatched parentheses, and invalid input.

### 2.3 User Characteristics

Users of this software are expected to have basic familiarity with:

- Command-line interfaces (CLI) to interact with the evaluator.
- Arithmetic expressions and the standard order of operations.

Users include:

- Developers integrate the evaluator into a compiler project.
- Students learning about expression parsing and evaluation.

### 2.4 Constraints

The development and operation of the Arithmetic Expression Evaluator include the following constraints:

- Development Timeline: Limited time is allocated for project completion, requiring efficient use of resources.
- Initial Input Scope: The initial version supports only integer inputs, with potential future updates to include floating-point support.
- Resource Availability: Testing and debugging resources are limited, influencing the scope of testing during development.

### 2.5 Assumptions and Dependencies

- The evaluator assumes integer-only inputs for the initial implementation, with plans to accommodate floating-point inputs in future iterations.
- It depends on the availability of a C++ compiler supporting C++17 or newer for building and running the software.
- The evaluator will be integrated with other compiler components, making it dependent on predefined interfaces for smooth interaction.

## 3. Specific Requirements

### 3.1 Functionality

The Arithmetic Expression Evaluator must fulfill the following functional requirements: Expression Parsing, Precedence Handling, Arithmetic Operations, Error Detection

#### 3.1.1 Expression Parsing

Breaks down user input into tokens that represent numbers, operators, and parentheses.

#### 3.1.2 Precedence Handling

Evaluates expressions according to the rules of operator precedence and parentheses.

#### 3.1.3 Arithmetic Operations

Supports basic operations like addition, subtraction, multiplication, division, modulo, and exponentiation.

#### 3.1.4 Error Detection

Identifies and provides feedback on issues such as division by zero, mismatched parentheses, and invalid input.

### 3.2 Use-Case Specifications

Use Case 1: Evaluate Expression

- Actor: User
- Precondition: User has entered a valid arithmetic expression.
- Main Flow: The user inputs an arithmetic expression, the evaluator processes the input, computes the result, and displays the output.
- Postcondition: The result of the evaluated expression is displayed or an error message is shown if the input is invalid.

### 3.3 Supplementary Requirements

Performance: The evaluator should process expressions and return results within a second for typical inputs.
Usability: The command-line interface should provide clear instructions and error messages.
Reliability: The software must correctly evaluate expressions and handle common edge cases such as division by zero.

## 4. Classification of Functional Requirements

The following table categorizes the functional requirements based on their importance:
- Essential: Must-have features for the evaluator to function correctly.
- Desirable: Features that improve user experience but are not critical for basic functionality.
- Optional: Nice-to-have features that add extra functionality.

| Functionality | Type |
| --- | --- |
| Parsing arithmetic expressions | Essential |
| Evaluating expressions with operator precedence | Essential |
| Handling parentheses | Essential |
| Providing user-friendly error messages | Desirable |
| Support for floating-point numbers | Essential |
| Customizable command-line prompts | Optional |

## 5. Appendices

Appendix A: Glossary

- SRS: Software Requirements Specification
- PEMDAS: An abbreviation for remembering the order of operations: Parentheses, Exponents, Multiplication and Division, Addition and Subtraction.
- CLI: Command-Line Interface, a text-based interface used to interact with software by typing commands.
- C++: A programming language that supports object-oriented, procedural, and generic programming.

Appendix B: References

- Project Plan for Arithmetic Expression Evaluator, Version 1.2, September 25, 2024.