ByteMath

Arithmetic Expression Evaluator Software Requirements Specifications

Version 1.0

Revision History

Date	Version	Description	Author
<dd/mmm/yy>	<x.x>	<details>	<name>
10/19/24	1.0	Add to sections 2.1.1-2.1.6	Ellie Thach
10/20/24	1.0	Finished introduction	Trey Baptista

Table of Contents

Software Requirements Specifications

1. Introduction

The **SRS** outlines the requirements for the **Arithmetic Expression Evaluator.** It details both functional and nonfunctional requirements, ensuring the system meets user expectations and project goals.

1.1 Purpose

The SRS describes the functionality, constraints, and behavior of the **Arithmetic Expression Evaluator**, defining how it should parse, evaluate, and compute arithmetic expressions. It also covers design constraints and testing requirements.

1.2 Scope

This document applies to the **Arithmetic Expression Evaluator**, which handles operations like +, -, *, /, %, **, and parentheses. It ensures correct order of operations (PEMDAS) and numeric constant handling. It also addresses error handling and user interaction.

1.3 Definitions, Acronyms, and Abbreviations

- **SRS**: Software Requirements Specification
- **PEMDAS**: Order of operations (Parentheses, Exponents, Multiplication, Division, Addition, Subtraction)
- **C++**: The programming language used for development

1.4 References

- **Project Glossary**: Defines the key terms and acronyms used in this document (SRS, PEMDAS, C++).
- **UML Use-Case Model**: Provides the use-case diagram that outlines the functionality required for the arithmetic expression evaluator, including interaction with the terminal user interface and expression processing.

1.5 Overview

The **Software Requirements Specification (SRS)** is organized as follows:

1. **Introduction**: Provides the purpose, scope, definitions, and references.
2. **Overall Description**: Describes product perspective, interfaces, functions, user characteristics, and constraints.
3. **Specific Requirements**: Details the functional and non-functional requirements.
4. **Classification of Functional Requirements**: Categorizes functional requirements.
5. **Appendices**: Includes additional references and supporting information.

2. Overall Description

2.1 Product perspective

2.1.1 System Interfaces

The arithmetic expression evaluator will not have system interfaces.

2.1.2 User Interfaces

The arithmetic expression evaluator will have a simple command-line user interface. The user will be able to enter in an arithmetic expression as input, and the program will output the result of the evaluation.

2.1.3 Hardware Interfaces

The arithmetic expression evaluator will not have hardware interfaces.

2.1.4 Software Interfaces

The arithmetic expression evaluator will be developed as a component for a larger compiler product for language L.

2.1.5 Communication Interfaces

The arithmetic expression evaluator will not have communication interfaces.

2.1.6 Memory Constraints

The arithmetic expression evaluator will have minimal memory constraints. The memory constraints would be that the program needs to contain the expression that is being evaluated, as well as the result of the evaluated expression.

2.2 Product functions

Operations:

The calculator will allow users to perform basic operations such as Addition, Subtraction, Multiplication and Division. Exponentiation, Trig and integration can be included as part of a later version.

I/O:
Users will enter their expression on a command line interface using the keyboard. The results will be displayed in the same window.

Error handling:
Common errors faced will be appropriately sent to the right error handler.
Unrecognized characters
Division by zero
Incorrect number of brackets

2.3 User characteristics

The users of the calculator can range from students to professionals who need it for everyday tasks. Students can use it for math homework, whereas professionals would expect advanced features trying to evaluate trig functions and integrals. Users' experience level can vary but all assume a basic knowledge of working on a computer and order of operations.

2.4 Constraints

Platform Compatibility:
The calculator must be compatible with specific operating systems (e.g., Windows, macOS, Linux Android, iOS)
Performance Limitations:
Must be able to run within milliseconds and memory usage must be less than 100MB.

2.5 Assumptions and dependencies

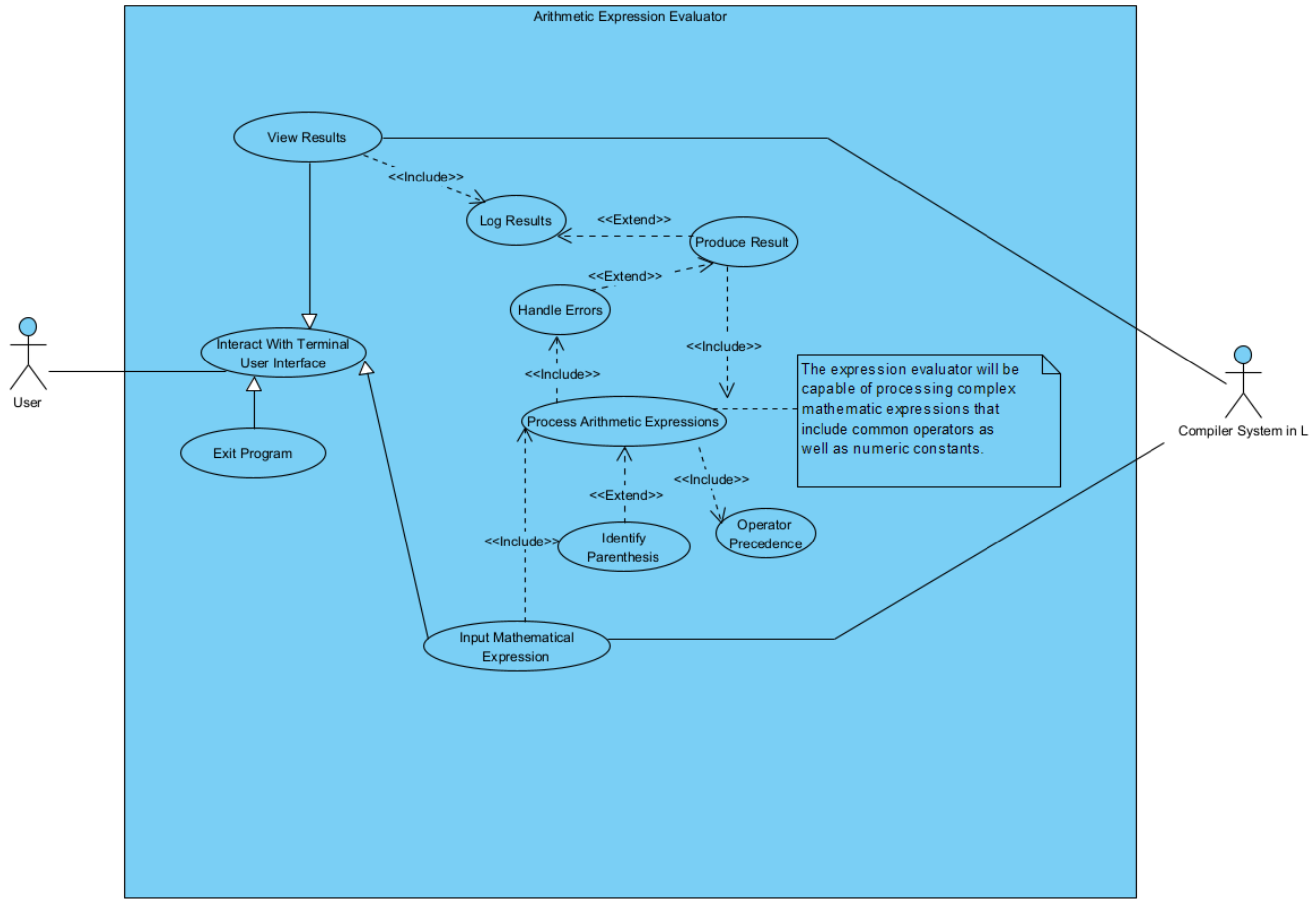
Assumptions:
Basic understanding of order of operations and left to right precedence.
The program is run on a computer with minimum resources.
Dependencies:
Third party libraries must be stable and regularly checked for updates.
Developemnt tools such as IDEs like Visual studio for testing the application.
User feedback will direct the future direction of the application.

3. Specific Requirements

The arithmetic expression evaluator must be capable of performing complex mathematic expressions, following a handful of restrictions requiring a diverse range of compatibility for given expression inputs. The evaluator must produce an output and maintain functionality that can allow for ease of implementation into a larger product, programmed in a programming language *L*. A collection of the most common operators, being addition, subtraction, multiplication, division, modular division, and exponentiation must be capable of producing an integer output when given numeric constants.

3.1 Functionality

The necessary functions for the arithmetic expression evaluator are included within the provided UML use-case model.


Arithmetic Expression Evaluator
View Results
<<Include>>
Log Results
<<Extend>>
Produce Result
<<Extend>>
Handle Errors
<<Include>>
<<Include>>
User
Interact With Terminal User Interface
Exit Program
Process Arithmetic Expressions
The expression evaluator will be capable of processing complex mathematic expressions that include common operators as well as numeric constants.
Compiler System in L
<<Include>>
<<Extend>>
<<Include>>
Identify Parenthesis
Operator Precedence
Input Mathematical Expression

3.2 Use-Case Specifications

The use-case diagram contains the requirements and use-case specifications required for the development of the arithmetic expression evaluator. The user's interaction with the program is captured by the "Interact With Terminal User Interface" use case and related cases. Inputting an expression can be completed by either a user or external program. The most complex aspect of the diagram is the "Process Arithmetic Expressions" case, as it requires error checking, parenthesis handling, and operator precedence to function properly. Within the actual process, the evaluator will handle numeric constants as well as several mathematic operators, as listed in section 3.0.

3.3 Supplementary Requirements

Aside from the main functionalities of the expression evaluator, the program must be expected to run within a reasonable time frame, currently anticipated to be a few seconds. The fast response times will provide results quickly for either the user or the larger compiler product.

Additionally, the evaluator will have alternative functionality that can allow processing and returning of floating-point values. The implementation of floating-point compatibility should be simple, with the calculations simply using float variable typing compared to integer variables.

4. Classification of Functional Requirements

Functionality	Type
Get input from the user and store it.	Essential
Check the input for correctness and raise an error if it is not a valid expression.	Essential
Parse the expression and store it in a tree or similar data structure as numbers and operations according to the PEMDAS order of operations.	Essential
Implement addition functionality (+) between two numbers.	Essential
Implement subtraction functionality (-) between two numbers.	Essential
Implement multiplication functionality (*) between two numbers.	Essential
Implement division functionality (/) between two numbers.	Essential
Implement modulus functionality (%) between two numbers.	Essential
Implement exponentiation functionality (**) between two numbers.	Essential
Implement error handling to tell the user what's wrong with the expression if anything is.	Essential
Implement a user interface where the user can input expressions and see the results.	Essential
Handle square brackets ([]) in addition to parenthesis.	Optional
Handle floating point values in addition to integers.	Optional

5. Appendices

Appendices will either be specified as part of the SRS or not. Currently, there are no appendices.

https://github.com/aidanp12/EECS348PROJ_GROUP7