Kansas Instruments

Compiler Expression Parser

Software Requirements Specifications

Version 1.0

Revision History

Date	Version	Description	Author
10/09/24	1.0	Created and finished SRS	All

Table of Contents

Software Requirements Specifications

1. Introduction

This Software Requirements Specifications document lists and describes the functional and non-functional requirements for our arithmetic expression parser project. Additionally, this document lists the constraints and provides a use-case model to express the purpose of out project.

1.1 Purpose

The purpose of the Software Requirements Specification is to fully describe the external behavior of the Compiler Expression Parser. It also describes nonfunctional requirements, design constraints, and other factors necessary to provide a complete and comprehensive description of the requirements for the software.

1.2 Scope

The Software Requirements Specification (SRS) applies to the "Compiler Expression Parser" a software application aimed at parsing and evaluating arithmetic expressions in language *L*. The application will support operators including addition, subtraction, multiplication, division, modulo, and exponentiation, while recognizing numeric constants and handling parentheses for operator precedence. The system ensures adherence to mathematical rules, error handling, and expression evaluation correctness. This document is linked to the project's influencing requirements specification, software design, development, and testing. The SRS aims to guide developers in implementing and verifying the system's functional and non-functional requirements.

1.3 Definitions, Acronyms, and Abbreviations

- SRS: Software Requirements Engineering
- UI: User Interface
- OO Programing: Object Oriented Programing
-

1.4 References

N/A

1.5 Overview

This Software Requirements Specification contains the following information:

Overall Description	Outlines the interfaces, functionalities, and constraints of the arithmetic expression parser. It also describes the assumptions and dependencies for the project.
Specific Requirements	Describes the functional and non-functional requirements of the expression parser and the behavior of use cases within the system.
Classification of Functional Requirements	Lists all of the functional requirements and organizes them by priority: essential, desirable, and optional.

2. Overall Description

2.1 Product perspective

2.1.1 System Interfaces

N/A

2.1.2 User Interfaces

The product is expected to provide a user interface which allows human users to input arithmetic expressions and receive the calculated results.

2.1.3 Hardware Interfaces

N/A

2.1.4 Software Interfaces

The expression parser is being developed as a component to be included in a larger product, the compiler for language *L*.

2.1.5 Communication Interfaces

N/A

2.1.6 Memory Constraints

Expression parsing is a ubiquitous part of any programming language, so the parser should use memory sparingly while still delivering the versatility expected in *L*.

2.1.7 Operations

N/A

2.2 Product functions

The program must parse an expression while obeying operator precedence and parenthesis. It should be able to evaluate any expression using the common arithmetic operators and recognize when numeric values, floats, and variables are inputted. The program must handle any error that could result from the user’s expression.

2.3 User characteristics

The UI must allow the user to enter an expression and display the evaluated expression. If the program encounters an error, it should display an understandable error message.

2.4 Constraints

The program should be compatible with C++ compilers on any operating system and use OO principles.

2.5 Assumptions and dependencies

It should also be able to evaluate the expression quickly and handle large amounts of operators within an expression.

2.6 Requirements subsets

N/A

3. Specific Requirements

- The program should not consume more than 50 MB of memory when evaluating standard arithmetic expressions.
- The system shall parse arithmetic expressions provided as input.

- The system shall support parsing of numeric constants and operators (+, -, *, /, %, **).
- The system shall evaluate expressions following the standard order of operations (PEMDAS):
    - Parentheses ()
    - Exponentiation **
    - Multiplication *, Division /, Modulo %
    - Addition +, Subtraction -
- The system shall handle left-to-right or right-to-left associativity as applicable:
    - Exponentiation (**) shall be evaluated right-to-left.
    - All other operators (+, -, *, /, %) shall be evaluated left-to-right.
- The system shall correctly evaluate expressions with nested or multiple sets of parentheses.
- The system shall detect and report the following errors:
    - Unmatched parentheses.
    - Division by zero.
    - Invalid characters in the input expression.
    - Incorrect usage of operators (e.g., missing operands).
- The error messages should clearly describe the issue to the user.
- The system shall provide a command-line interface for inputting expressions and displaying results.

3.1 Functionality

3.1.1 Expression Parsing

This program should be able to parse arithmetic expressions entered by the user, considering operator precedence and parentheses.

3.1.2 *Arithmetic Operators*

Addition (+): The program should add values together when encountering a plus sign.

Subtraction (-): The program should subtract values when encountering a subtraction sign.

Multiplication (*): The program should multiply values when encountering an asterisk.

Division (/): The program should divide values when encountering a forward slash.

Modulo (%): The program should perform the modulo function when encountering a percentage sign.

Exponentiation (**): The program should raise the value to the power when encountering double asterisks.

3.1.3 Numeric Constants

Numeric constants within expressions should be recognized and calculated.

3.1.4 Friendly User Interface (UI):

This program should create a user-friendly and legible command-line interface that allows users to enter expressions and displays the calculated results.

3.1.5 Error Handling

This program should implement robust error handling to manage scenarios like division by zero or other invalid expressions. The error messages displayed should be clear and informative.

3.1.6 Floating-point Value Support

This program will support the request to accommodate floating point numbers should the user desire it.

3.1.7 Variable support

This program will be adaptable to accept and handle variables.

3.2 Use-Case Specifications

- Adding numbers together
- Subtracting two numbers
- Multiplying numbers
- Dividing numbers
- Finding the modulo of two numbers
- Find the value of a number raised to an exponent
- Finding the solution to more complex problems that use a combination of these use cases.

3.3 Supplementary Requirements

- The program shall evaluate an expression and display the result within 0.5 seconds for typical cases.
- The program shall efficiently handle expressions with up to 50 operations.
- The command-line interface shall be user-friendly, with prompts and clear error messages.
- The program shall follow object-oriented principles to allow for future extensions, such as supporting floating-point numbers.
- The program shall compile and run on standard C++ compilers on Windows, macOS, and Linux platforms.
- The program shall maintain consistent behavior for all valid inputs.
- The program shall handle unexpected or invalid input without crashing.

3.3.1 *Code*

The code will implement OO programming for better efficiency and functionality.

The code will include comments and documentation to explain the logic and functionality of the program.

3.3.2 Security

This system will ensure users are unable to tamper with and manipulate code.

4. Classification of Functional Requirements

Functionality	**Type**
Expression Parsing	Essential
Evaluate Expressions Following PEMDAS	Essential
Handle Left-to-Right or Right-to-Left Associativity	Essential
Handle Parenthesis and Evaluate Based on Parenthesis	Essential

Addition Operation	Essential
Subtraction Operation	Essential
Multiplication Operation	Essential
Division Operation	Essential
Modulo Operation	Essential
Exponential Operation	Essential
Numeric Constant Recognition and Calculation	Essential
Friendly User Interface for Inputting Expressions and Displaying Results	Essential
Error Handling (Unmatched Parenthesis, Division by Zero, Invalid Characters, Incorrect Usage of Operators)	Essential
Clear Error Messages (Gracefully Handle Unexpected or Invalid Input Without Crashing)	Essential
Floating-point Value Support	Desirable
Variable Support	Optional
Efficient Evaluation and Answer Output (within 0.5 seconds)	Essential
Efficiently Handle Expressions with up to 50 Operations	Essential
Implement OO Principles to Allow for Future Extensions	Essential
Compile and Run on Standard C++ Compilers on Windows, macOS, and Linux platforms	Essential
Maintain Consistent Behavior for All Valid Inputs	Essential

5. Appendices

Any other important documents for the project can be found in the GitHub repository
https://github.com/NevanSnider/EECS348-Team-Project

SRS: Software Requirements Engineering
UI: User Interface
OO Programing: Object Oriented Programing