Freaky Inc.

Freaky Calculator
Software Requirements Specifications

Version <1.0>

Revision History

Date	Version	Description	Author
04/10/2024	<1.0>	Initial Software Requirements Specifications document	All Team Members

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 5

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 6
   - 2.5 Assumptions and dependencies 5
   - 2.6 Requirements subsets 5

3. Specific Requirements 6
   - 3.1 Functionality 6
     - 3.1.1 <Functional Requirement One> 6
   - 3.2 Use-Case Specifications 8
   - 3.3 Supplementary Requirements 9

4. Classification of Functional Requirements 9

5. Appendices 10

Software Requirements Specifications

1. Introduction

The Software Requirements Specification (SRS) serves as a formal blueprint outlining the expected functionalities, requirements, and objectives of the software application. This document provides a detailed overview of the project's scope and complexities, ensuring a clear and thorough understanding before development activities begin.

1.1 Purpose

The purpose of the Software Requirements Specification (SRS) is to thoroughly define the expected functionality of the software. It will also identify the components necessary to ensure proper operation. Furthermore, the SRS will address critical aspects such as non-functional requirements and design constraints, which are essential to the software's development process.

1.2 Scope

This Software Requirements Specification (SRS) outlines the development of an arithmetic expression parser for a compiler in language L, using C++. The parser will handle and evaluate arithmetic expressions involving the operators +, -, *, /, %, and **, while effectively managing numeric constants and parentheses to ensure correct operator precedence. Key features include expression parsing, operator support, parentheses handling, and robust error management for issues like division by zero or invalid input. The project will follow object-oriented programming principles to ensure modularity and maintainability, with comprehensive documentation, unit tests, and a user-friendly interface that provides clear error messages. The parser is a vital component of the compiler, enabling expression evaluation and input validation.

1.3 Definitions, Acronyms, and Abbreviations

- **SRS**: Software Requirements Specification
- **OOP**: Object-Oriented Programming
- **OOP Principles**: Encapsulation, Abstraction, Inheritance, Polymorphism
- **EECS**: Electrical Engineering and Computer Science (University of Kansas)
- **EECS 348**: Software Engineering I
- **Arithmetic Expression Parsing**: The process of analyzing and interpreting arithmetic expressions to understand their structure and evaluate their result.
- **PEMDAS**: Order of operations in arithmetic expressions (Parentheses, Exponents, Multiplication/Division, Addition/Subtraction).
- **Operator Support**: Handling of arithmetic operators including +, -, *, /, %, and ** (exponentiation).
- **Parenthesis Handling:** Proper evaluation of expressions enclosed in parentheses to maintain correct operator precedence.
- **Numeric Constants**: Numbers used in expressions, initially limited to integers but may include floating-point numbers in future iterations.
- **Error Handling:** Detecting and managing errors like division by zero, malformed expressions, and invalid inputs.
- **Command-Line Interface (CLI)**: A text-based interface that allows users to input expressions and receive output or error messages.

1.4 References

TITLE	DATE	SOURCE	LINK
EECS348: Term-Project	October 4, 2024	EECS 348 Software Engineering I Lecture on Canvas	Files (ku.edu)

1.5 Overview

Overall Description: Describes the general factors and requirements for the program as well as background for the requirements.

Specific Requirements: Describes all software requirements including functional and supplementary requirements, and use-cases.

2. Overall Description

2.1 Product perspective

2.1.1 System Interfaces

- The arithmetic expression parser will utilize C++ standard input and output streams to receive input from the user and display results. The system will process arithmetic expressions entered via the command line and return the evaluated output based on operator precedence and grouping.

2.1.2 User Interfaces

- The parser will feature a user-friendly command-line interface (CLI) that allows users to input arithmetic expressions consisting of operators (+, -, *, /, %, **), numeric constants, and parentheses. The output will display the calculated result of the expression or an error message if invalid input is detected. Users can exit the program easily. If the input is malformed or contains errors such as division by zero, the system will notify the user with clear, descriptive error messages.

2.1.3 Software Interfaces

- The arithmetic expression parser will be developed in C++ and make use of standard C++ libraries for input handling, expression parsing, and error management.

2.1.4 Memory Constraints

- The parser is designed to be lightweight, with minimal memory usage, ensuring efficient handling of even complex arithmetic expressions without consuming excessive resources.

2.1.5 Operations

- The core functionality of the parser includes reading, parsing, and evaluating arithmetic expressions based on operator precedence and proper handling of parentheses. The program supports the six operators (+, -, *, /, %, and **), and evaluates expressions to produce a numeric result. It also handles errors, such as division by zero or invalid expressions, ensuring that the system remains stable and responsive during operation.

2.2 Product functions

- The arithmetic expression parser will evaluate various arithmetic operations, including addition, subtraction, multiplication, division, modulus, and exponentiation. It will also correctly interpret parentheses for grouping operations, ensuring adherence to PEMDAS rules. The parser accepts numeric inputs and will initially handle integer values, with potential future support for floating-point values. It will output either the evaluated result of the expression or an informative error message. The system will provide users with a straightforward way to input expressions and view results, focusing on accuracy and clarity.

2.3 User characteristics

- **Easy to Use:** The application is designed to be straightforward, with a command-line interface that allows users to input expressions and quickly receive output. No prior advanced programming knowledge is required, making it accessible to students and professionals alike.

- **Reliable Compilation:** The application will compile seamlessly on most C++ development environments, ensuring compatibility and ease of use during setup.
- **Robust Evaluation:** The parser is designed to handle a wide range of arithmetic expressions without errors, providing accurate results or informative error messages for invalid inputs.
- **Quick Learning Curve:** Users familiar with basic arithmetic operations and order of precedence (PEMDAS) will be able to use the application effectively without extensive training.

2.4 Constraints

- **Programming Language:** The application must be implemented in C++.
- **User Interaction:** All interaction between the user and the application will occur through a basic keyboard, using a command-line interface.
- **Memory Usage:** The parser must be lightweight, using minimal memory to handle even complex expressions efficiently.
- **Error Handling:** The application should provide clear, descriptive error messages for issues such as division by zero, malformed expressions, or other invalid inputs to ensure a stable and responsive user experience.
- **Compatibility:** The application will use standard C++ libraries to ensure cross-platform compatibility and ease of deployment in different development environments.
- **Future Scalability:** Initially, the parser will handle integer values but should be designed to accommodate future support for floating-point numbers.
- **Runtime Performance:** The application must run consistently with minimal runtimes, ensuring it provides an uninterrupted user experience.

2.5 Assumptions and dependencies

- The application assumes the user will input mathematical expressions in valid format using integer values initially. It is dependent on a command line interface and standard C++ libraries for cross-platform compatibility. The parser will assume that the system has a basic keyboard for input.

3. Specific Requirements

3.1 Functionality

The system should allow users to simulate and evaluate arithmetic expressions using various operators. It must handle errors gracefully and provide clear error messages when invalid input is detected. The following is a breakdown of specific functional requirements:

3.1.1 Expression Parsing

The project needs a function to tokenize an input expression from the user. Each input expression should be a single string input that the program will break down into individual components (numbers, operators, and parentheses) for processing.

3.1.2 Operator Precedence

The program must define the rules of operator precedence (following PEMDAS) and evaluate input expressions accordingly, ensuring that operations such as multiplication, division, and exponentiation are correctly prioritized over addition and subtraction.

3.1.3 Parenthesis Handling

The program needs a mechanism to identify and evaluate expressions within parentheses. Parentheses should be correctly paired, and if any parenthesis is missing or not paired, the program should provide a descriptive error message.

3.1.4 Numeric Recognition

The program should recognize numeric constants (integers) as valid inputs and support basic arithmetic operators: addition (+), subtraction (-), multiplication (*), division (/), modulus (%), and exponentiation (**).

3.1.5 User Interface

The program must feature a user-friendly command-line interface (CLI) that allows users to input arithmetic expressions. It should display the results of calculations in a clear, concise format and provide consistent feedback, including error messages when invalid input is detected. The interface should be straightforward, allowing users to access, use, and interpret results without requiring prior knowledge of how the program works. Additionally, the interface should offer an effortless way for users to exit the program.

3.1.6 Error Handling

The program requires an error management system to handle scenarios such as unpaired parentheses, division by zero, invalid expressions, and unsupported operators. It should display appropriate error messages in response to specific errors. Below is a list of error messages that the program should handle, along with examples:

Missing Operand: 3 +
Unknown Operator: 4 ? 5
Mismatched Parentheses: (3 + 4))
Empty Expression: ( )
Double Operator: 5 ++ 3
Division by Zero: 10 / 0
Invalid Characters: a + b
Incorrect Order of Operators: 5 *
Invalid Syntax: 2 + (5 *)

3.1.7 Arithmetic Operations

The program should support the following arithmetic operators:

Addition: +

Subtraction: -
Multiplication: *
Division: /
Modulus: %
Exponentiation: **

3.1.8 Input Validation

The program must validate input expressions to ensure they conform to the specified format. It should reject invalid expressions and display clear error messages to guide the user toward correcting their input.

3.1.9 Result Presentation

The program should display the evaluated result of the arithmetic expression in a clear, concise message. If the expression is invalid, the program should provide a detailed error message to help the user identify the issue.

3.1.10 Expression Evaluation

If using variables in expressions (in future iterations), the program should allow users to input values for these variables and evaluate the expression accordingly. The parser should correctly implement operator precedence and the order of operations (PEMDAS).

3.1.11 Expression Comparison

The program should enable users to compare different arithmetic expressions by displaying the evaluated results side-by-side or highlighting differences in the computed outcomes.

3.2 Use-Case Specifications

Primary Actor:

User

Goal:

The user wants to evaluate an arithmetic expression using the provided C++ program.

Preconditions:

The user has access to the command-line interface and has the program installed.

Exception Conditions:

The user enters an invalid expression, such as unmatched parentheses, division by zero, double operators, or unsupported characters. The program should display an informative error message indicating the nature of the error and prompt the user to enter a valid expression.

Postconditions:

The user receives a calculated result for the entered arithmetic expression. The program remains available for further evaluation or interaction.

Scenario:

- The user launches the arithmetic expression parser program.

- The program displays a prompt indicating readiness to receive input. It also displays a list of available features, such as supported operators and input format guidelines.
- The user enters an arithmetic expression containing numeric values, arithmetic operators (+, -, *, /, %, **), and parentheses, with optional whitespace for readability.
- The program parses the input expression, checks for any errors (e.g., unpaired parentheses, invalid operators), and evaluates the expression if it is valid.
- The program displays the calculated result to the user or an error message if an issue is detected.
- Optionally, the user may choose to input another expression or exit the program.

Additional Scenarios:

If the user decides not to enter an expression initially, they can exit the program without performing any evaluation.

The user can choose to view help documentation explaining how to use the program, including syntax rules, available operators, and error-handling guidelines.

Assumptions:

The user understands basic arithmetic operation syntax and the order of operations (PEMDAS).

3.3 Supplementary Requirements

3.3.1 The project must be developed using C++ and make use of its standard libraries.

3.3.2 The parser should have a minimal memory footprint, allowing it to run quickly and efficiently, ensuring a smooth user experience on most machines.

3.3.3 All user input must be possible using basic ASCII characters to maintain simplicity and compatibility.

3.3.4 The parser should run reliably without unexpected errors, providing robust error handling and clear messages for invalid inputs.

3.3.5 The project must be completed by May 1, 2024.

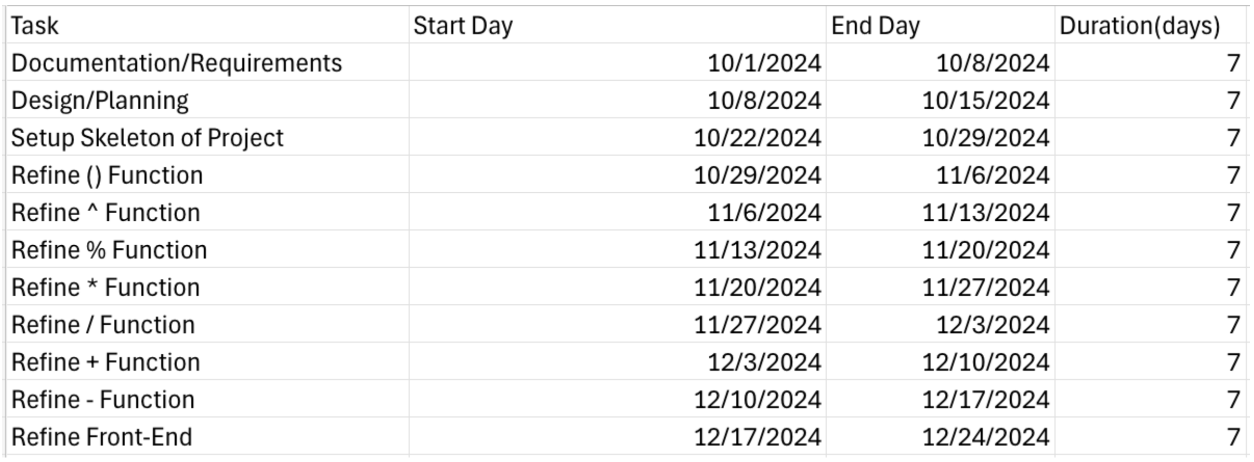
3.3.6 The project development process and all deliverable documents shall conform to the UPEDU (Unified Process for Education) guidelines.

4. Classification of Functional Requirements

Functionality	**Type**

Main File Structure	Essential
Basic Starting Design	Essential
Parenthesis Function ()	Essential
Exponential Function ^	Essential
Modulo Function %	Essential
Multiplication Function *	Essential
Division Function /	Essential
Addition Function +	Essential
Subtraction Function -	Essential
Implement Final Design	Essential
Fix Errors/Bugs	Essential
Perfect UX/UI Elements	Desirable
Add Easter Eggs to Software	Optional

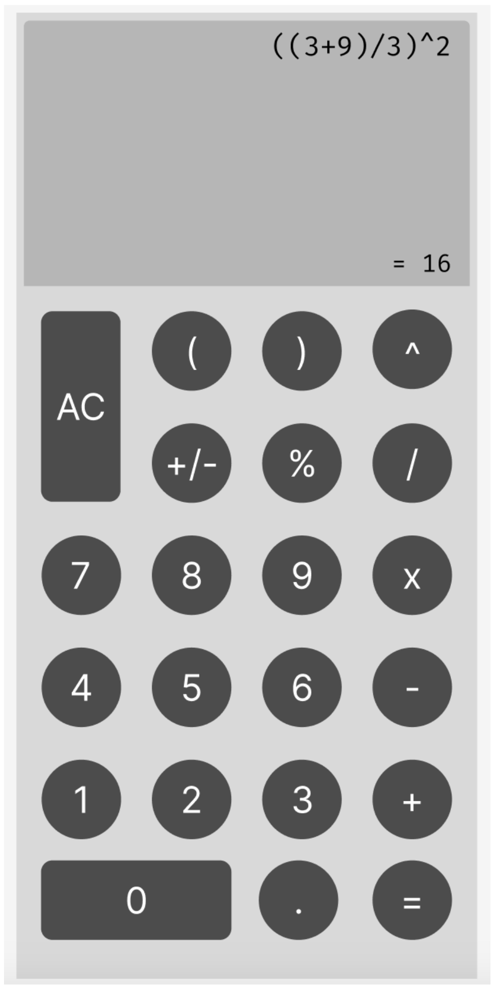
5. Appendices



Task	Start Day	End Day	Duration(days)
Documentation/Requirements	10/1/2024	10/8/2024	7
Design/Planning	10/8/2024	10/15/2024	7
Setup Skeleton of Project	10/22/2024	10/29/2024	7
Refine () Function	10/29/2024	11/6/2024	7
Refine ^ Function	11/6/2024	11/13/2024	7
Refine % Function	11/13/2024	11/20/2024	7
Refine * Function	11/20/2024	11/27/2024	7
Refine / Function	11/27/2024	12/3/2024	7
Refine + Function	12/3/2024	12/10/2024	7
Refine - Function	12/10/2024	12/17/2024	7
Refine Front-End	12/17/2024	12/24/2024	7

Project Plan 01: 01-Project-Plan.docx

Current Basic UX/UI Design:


((3+9)/3)^2
= 16
AC
(
)
^
+/-
%
/
7
8
9
x
4
5
6
-
1
2
3
+
0
.
=