Team 14

Arithmetic Expression Evaluator in C++
Software Requirements Specifications

Version <1.0>

Revision History

Date	Version	Description	Author
<20/10/2024>	<1.0>	Initial Version	Team 14

Table of Contents

Software Requirements Specifications

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) is to provide a detailed description of the Arithmetic Expression Evaluator, its nonfunctional requirements, and its design constraints, and other necessary factors. This document serves as a complete and comprehensive guide for the requirements of the software. It aims to ensure that all stakeholders have a clear understanding of the system's requirements and functionalities.

1.2 Scope

We will be developing an arithmetic expression parser for a compiler L. The software will take an arithmetic operation from the user and return a correct answer.

Features: Our program will be capable of parsing through an expression the user gives on a command line. It will be capable of evaluating operators +, -, *, /, %, and ** as well as dealing with parentheses. Then the user will be given a final answer. Furthermore, errors will be handled and reported to the user elegantly.

Use Case Model: Get user input → parse through expression → calculate final answer → raise error?

1.3 Definitions, Acronyms, and Abbreviations

Arithmetic Expression: A combination of operators (+, -, *, /, %, and **) and numbers that can be mathematically calculated to give a final answer.

C++: A programming language that we will be using to develop this software.

IEEE (Institute of Electrical and Electronics Engineers): A large organization that, among other things, is known for developing professional standards for technology development

UPEDU (Unified Process for Education): A software development framework mostly used for learning

PEMDAS: The order of operations in math (Parentheses, Exponents, Multiplication, Division, Addition, and Subtraction). Used to make sure all expressions are solved in the same way.

SRS (Software Requirements Specification): The document you are reading. Lists the features a software product will have and expected behavior.

1.4 References

IEEE Standard for Software Requirements: Year: 1998, From: IEEE Xplore

UPEDU Guidelines: Year: 2001, From: UPEDU.org

C++ Programming Language, 4th Edition: Author: Bjarne Stroustrup, Year: 2013, From: Bookstore

PEMDAS:

Project Plan for Arithmetic Expression Evaluator: Year: Fall 2024, From: KU EECS348 materials

1.5 Overview

Section 1. Introductory material like the purpose, scope, and references.

Section 2. Describes the product in the overall description in general terms, like its interface, operations, and constraints, which helps to show the specific requirements.

Section 3. Talks about the specific functional and non-functional requirements of the system, with use cases and technical specifications.

Section 4. Classifies the requirements into sections such as (Essential, Desirable, and Optional) or by order of appearance in the document based on their importance.

Section 5. Contains any appendices or additional documentation relevant to the project.

2. Overall Description

2.1 Product perspective

2.1.1 User Interfaces

When the program is launched, the user will be prompted with a clear message, indicating that they may enter an arithmetic expression to be calculated.
The user will enter the expression directly into the terminal, following the prompt message. The program will exclusively accept the +,-,*,**,/, and % operators.
After the user has provided a valid expression, the program will return the result of the evaluated expression to the user.

Error messages will be used to handle invalid inputs

2.1.2 Software Interfaces

- C standard library, specifically <stdio.h> for the inputs and outputs of operations.
- Utilize <cmath> for handling exponentiation, as it is not a default C++ operator

- The program will work with operating systems by accessing and interacting with the terminal

- Software compatibility with modern compilers, such as GCC and MSVC.

- Separate modules will be used in the code to handle different expressions, errors, parsing, etc.
- Communication between these modules will be facilitated through function calls in the code.

2.1.3 Memory Constraints

Considering the low memory requirements of this program, it is expected that it will run on devices with at least 1 GB of RAM. This ensures compatibility with a wide variety of devices and systems.
Stress tests will be conducted to ensure the program can handle more complex expressions than we expect

2.2 Product functions

Expression Parsing: This program is able to parse arithmetic expressions entered by the user, taking into account operator precedence and parentheses

Operator Support: The program supports the following operators:

- + (addition)
- - (subtraction)
- * (multiplication)
- / (division)
- % (modulo)
- ** (exponentiation)

Parenthesis Handling: The program can handle expressions enclosed within parentheses to determine the order of evaluation

Numeric Constants: The program recognizes and calculates numeric constants within the expression.

2.3 User characteristics

The users of this program vary greatly in terms of educational level, experience, technical expertise, and disabilities. However, users of this program will typically fall under these characteristics:

- Capable of mentally computing basic algebra
- Has access to a PC with a basic dual-core processor, 1 GB of RAM, and a modern operating system with the capability to run a shell and terminal program.

2.4 Constraints

- Programming language: This software must be written in C++
- Supported operators: this software will only accept these operators +, -, *, /, %, and **
- Input: this software will only support the input of integers, not floating-point numbers
- Division by zero: this software must support division by zero by outputting helpful error messages leading to a smooth user experience and not crash when this occurs
- Compilers: this program must be compatible with modern compilers

2.5 Assumptions and dependencies

- The development team is familiar with C++ programming and OOP concepts
- The user will have the proper compiler installed to run the program
- The user will be able to do basic command-line actions like running the program and inputting their input on the command line
- User will i\put their expression on a single line rather than trying to do it on multiple lines on the command-line
- Basic terminal handling will be working properly as in terminating the program if needed, handling certain spaces, and typing in input/displaying the output

3. Specific Requirements

3.1 Functionality

3.1.1 Expression Parsing

The system must be able to parse arithmetic expressions entered by the user, identifying numeric constants and operators. This includes breaking the input down into tokens and organizing them into a data structure

such as a tree or stack.

3.1.2 Operator Precedence

The system must evaluate expressions based on the correct precedence of operators. The system must also apply PEMDAS rules during evaluation.

3.1.3 Operator Support

The system must be able to support the following operators: + (addition) , - (subtraction) , * (multiplication) , / (division) , % (modulo) , ** (exponentiation)

3.1.4 Parentheses Handling

The system must correctly interpret and evaluate expressions enclosed within parentheses, giving priority to operations inside parentheses.

3.1.5 Error Handling

The system must be able to provide error messages for invalid inputs, invalid operators, or division by zero.

3.1.6 User Interface

The system must have a command-line interface for users to input expressions. This will also be used to display results.

3.1.7 Extended Numeric Support

The system could allow for floating point number inputs in the future.

3.1.8 Error Recovery

The system could suggest corrections for minor user errors, such as missing operators or unmatched parentheses.

3.2 Use-Case Specifications

"Enter Arithmetic Expression" Use-Case

Actor: User

Preconditions: Inputted arithmetic expression must have correctly matched operators and operands and the expression must adhere to mathematical rules

Post Conditions: Output must be an integer that is equivalent to the inputted arithmetic expression

Success Scenario Flow Of Events:

- User inputs arithmetic expression
- Program determines order of evaluation based on parentheses and operator precedence
- Program recognizes and calculates numeric constants within the expression, doing so in order of evaluation
- Program then determines a final integer value based on the calculated numeric constants that is equivalent to the inputted arithmetic expression.
- This integer value is then printed to terminal

Alternative Flow:

- User inputs arithmetic expression that does not follow the stated preconditions
  - The program prints out a statement on the terminal explaining that the inputted arithmetic expression is invalid.
  - The program asks the user to enter another arithmetic expression

3.3 Supplementary Requirements

3.3.1 Non-Functional Requirements

3.3.1.1 Usability Requirements

The interface must be easy to use, with clear instructions for the user on how to input expressions. Error

messages also must be clear and concise

3.3.1.2 Reliability Requirements

The system needs to operate continuously without failure during normal use. All errors and invalid inputs must be handled properly without causing system crashes.

3.3.2 Development Constraints

3.3.2.1 Development Environment

The system must be developed using C++ as the programming language, adhering to object-oriented programming principles.

3.3.2.2 Coding Standards

The team must follow standard C++ coding conventions, including proper naming conventions, indentation, and commenting practices. Documentation is also needed.

3.3.2.3 Testing Constraints

The expression evaluator will be tested rigorously. Tests must cover all functional requirements and account for edge cases in expression parsing and evaluation.

4. Classification of Functional Requirements

Functionality	Type
Expression Parsing	Essential
Operator Precedence	Essential
Operator Support	Essential
Parentheses Handling	Essential
Error Handling	Essential
User Interface	Essential
Extended Numeric Support	Optional
Error Recovery	Optional

5. Appendices