Group 22

Arithmetic Parser
Software Requirements Specifications

Version <1.5>

Revision History

Date	Version	Description	Author
09/30/24	1.0	Initial edit	Janna Dungao
10/16/24	1.1	Initial draft of sections 1 & 2	Janna Dungao
10/17/24	1.2	Updated sections 3 & 4	Yoseph Ephrem
10/18/24	1.3	Updated section 4 & 5	Dalen Journigan
10/19/24	1.4	Updated sections 3 & 4	Mahdi Essawi
10/20/24	1.5	Tweaked some of the punctuation to improve consistency	Beckett Malinowski

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 7

5. Appendices 7

Software Requirements Specifications

1. Introduction

This section details the purpose, scope, definitions, acronyms, abbreviations, references, and an overview of this SRS.

1.1 Purpose

The purpose of this SRS is to identify what functions we need to implement in order to fulfill all of the requirements for this class's term project. It will identify the systems, subsystems, and other environments involved and describe nonfunctional requirements, design constraints, and the external behavior of the application to provide a complete and unambiguous description of the software requirements.

1.2 Scope

This SRS applies to the scope of our entire arithmetic parser project as a whole. It will detail the required features (being the mathematical computation and parsing of input) and it is associated with the Use-Case model(s) of a typical calculator. Additionally, it will discuss any additional features not mentioned in the course's project requirements.

1.3 Definitions, Acronyms, and Abbreviations

Definitions

- Interface – defines interactions between a user, software, and/or hardware

Acronyms

- PEMDAS – Parenthesis, Exponents, Multiplication/Division, Addition/Subtraction (used for order of operations)

Abbreviations

- IDE – Integrated Development Environment
- SRS – Software Requirements Specification
- UI – User Interface

1.4 References

GitHub Link – contains all documents and code including:

- Project Management Plan
- Meeting Log

1.5 Overview

The rest of this SRS goes into much greater detail about the specifications of the features we expect the arithmetic parser to have. It will detail user, software, and memory interfaces. Overall, it will give a greater view of how the arithmetic parser will function and how we will create it.

2. Overall Description

This section contains the general factors involved with this project providing a solid background for its requirements. It includes, for example, User Interfaces, Software Interfaces, and Memory Constraints.

2.1 Product perspective

2.1.1 System Interfaces

2.1.2 User Interfaces

This project will have a user-friendly command-line interface that takes input entered by users and outputting/displaying the calculated result.

2.1.3 Hardware Interfaces

2.1.4 Software Interfaces

The software interface for this project will be any computer with the Linux OS and software that can run our code such as an IDE or terminal.

2.1.5 Communication Interfaces

2.1.6 Memory Constraints

We do not have specified memory constraints.

2.1.7 Operations

2.2 Product functions

Product functions include expression parsing, operator support (+, -, etc.), parenthesis handling, numeric constants recognition, friendly UI, and error handling.

2.3 User characteristics

This product will be functional for users of all demographics given that they can operate a basic scientific calculator and a command-line interface.

2.4 Constraints

This project must be completed in C++ and maintained using GitHub. Additionally, we have time constraints as we must meet certain deadlines for each deliverable. The UI must be a command-line interface.

2.5 Assumptions and dependencies

For this project, we assume that the user is familiar with basic algebra and using a command-line interface. Additionally, this document is a dependency on the previous Project Management Plan document. The Project Architecture and Design is a dependency of this document. The Project Implementation is a dependency of the architecture and design. Finally, the project test cases and user manual are dependencies of the project implementation.

2.6 Requirements subsets

3. Specific Requirements

This section contains every requirement for this software/project to such detail that we will be able to design a system satisfactory to said requirements using this document alone.

3.1 Functionality

The arithmetic parser program will be capable of executing the following functions:

3.1.1 <Arithmetic Operations>

The parser will be able to calculate mathematical expressions using addition, subtraction, multiplication, and division.

3.1.2 <Order of Operations>

The parser will be able to identify certain keys in the order of operations following the order of operations, following the (PEMDAS) rules:

- Parentheses
- Exponents
- Multiplication and Division (left to right)
- Addition and Subtraction (left to right)
- It will allow for nested parentheses and properly respect operator precedence

3.1.3 <Functions and Exponents Support>

The parser will be able to handle exponentiation using the ^ operator, in addition to supporting basic mathematical functions, such as:

- Square Root (√)
- Trigonometric functions (sin, cos, tan)
- Logarithmic functions (log, ln)

3.1.4 <History of Calculations>

The parser will maintain and store a history of calculations (up to 10 previous calculations), allowing users to view previous expressions.

3.1.5 <Responsive Errors>

The parser can detect errors in certain expressions entered in the log. Providing clear feedback to the user. Such examples include:

- Division by Zero
- Syntax Errors
- Undeclared Variables
- Invalid Operations

3.1.6 <Complex Expressions>

The parser will be able to solve equations using user stored variables inside of the expressions.
Solving undefined variables will also be a function of the parser.
It will handle:

- Variable Assignment: Users can assign value to variables (e.g., x = 10).
- Substituting Variables in Expressions: Users can input expressions using defined variables such as (2 * x +3), with the parser replacing variables with their assigned value during evaluation.

3.2 Use-Case Specifications

The parser will have various use case specifications. Going through various functionality one of the use cases will be to simply compute expressions (e.g. 2+2, 4/2, 2*2, 2-2). Another specification will be working with complex expressions involving multiple operators and parenthesis (e.g. (2+2)/1). Error detection will also give us a use case specification; this is from specifying the error in the expression.

3.3 Supplementary Requirements

In our supplementary requirements we include our non-functional requirements and constraints. One of

supplementary requirements is our platform availability. Our parser should be able to run on different operating systems such as Windows or macOS. The parser should also have a runtime requirement which will be very quick and only take seconds to compute the expressions.

4. Classification of Functional Requirements

Functionality	Type
Arithmetic Operations	Essential
Order of Operations	Essential
Responsive Error	Essential
Complex Expressions	Essential
Runtime Efficiency	Essential
Function and Exponent Support	Essential
History of Calculations	Desired
Cross-Platform	Desired

5. Appendices

Appendices are not to be considered part of the requirements but as an aide in understanding the requirements explained above.

5.1 Glossary of Terms

PEMDAS (Operator Precedence): Parentheses, Exponents, Multiplication and Division, Addition and Subtraction. Defines the order in which the operands should be evaluated.

Use-cases: A scenario in which a user interacts with the scythes to achieve a specific goal.

UI (User Interface): The visual part of software systems that allow users to interact with the system (for example mathematical expressions).

IDE (Integrated Development Environment: A software suite that provides facilities for all programmers for software development. Code editor, compiler, debugger, etc.

SRS (Software Requirements Specification): A document that outlines both the functional and the non-functional requirements of a system.

Tokenization/Lexing: The process of breaking down a string of text into smaller units called tokens.

Parsing: The process of analyzing a sequence of tokens to decide its grammatical structure according to predefined rules.

Test-cases: Structured like scenarios designed to confirm that a software system, like an arithmetic parser, functions as expected. Each test case specifies an input, the expected output, and the criteria for success or failure.

5.2 Syntax Error Handling

1. Detecting errors: During tokenization, the lexer checks whether each character is valid and for parsing the parser identifies if tokens follow the correct order, adhere to operator precedence rules, and maintain balanced parentheses and if an error is found then the parser generates a descriptive error message, stops further evaluation, and may or may not provide hints or details for correction depending on the programmer.

2. Other common syntax errors:

- Unbalance parentheses: This occurs when the number of opening and closing parentheses in the expression do not match. The error message should mention that a parenthesis is missing as well.
- Consecutive operators: This occurs when two or more operators are placed consecutively with any operand in between. The error message will say “Unexpected Operator”.
- Division by zero: This is not strictly a syntax error; it is a critical runtime error that the parser must handle.
- Invalid token: This occurs when a character is not recognized as a valid token. The error message will say “Invalid token”.