MacroHard

Boolean Expression Evaluator
Software Requirements Specifications

Version <1.0>

Revision History

Date	Version	Description	Author
<24/03/24>	<1.0>	<details>	<name>

Table of Contents

Software Requirements Specifications

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) document is to outline the functional and nonfunctional requirements for an arithmetic expression evaluator written in C++. This software will parse and evaluate arithmetic expressions containing operators such as +, -, *, /, %, and ^, as well as numeric constants. It aims to handle expressions with parentheses for precedence and grouping, ensuring accurate calculations according to the PEMDAS rule.

1.2 Scope

The software application under development is an arithmetic expression evaluator. It is associated with the core functionality of parsing arithmetic expressions, evaluating them according to operator precedence (PEMDAS), and handling parentheses for correct expression grouping. This document applies to the version of the software that supports the specified operators and may be updated to accommodate future expansions, such as additional operators or functionalities.

1.3 Definitions, Acronyms, and Abbreviations

PEMDAS: Parentheses, Exponents, Multiplication and Division (from left to right), Addition and Subtraction (from left to right). This rule determines the order of operations in arithmetic expressions.

SRS: Software Requirements Specification.

1.4 References

IEEE830-1998: IEEE Recommended Practice for Software Requirements Specifications. A guideline that provides the recommended practices for software requirements specifications.

C++ Standard Library: Documentation on the C++ programming language's standard library, used for this project's development.

1.5 Overview

This document is organized as follows: Following the Introduction, there is an Overall Description section dedicated to detailing the general factors that affect the product and its requirements. Following the Overall Description section there is a Specific Requirements section describing the specific behavior and functionalities expected from the arithmetic expression evaluator, including expression parsing, operator support, parenthesis handling, and numeric constant recognition. It also contains a section on the nonfunctional requirements (i.e. performance, usability, and maintainability requirements). To supplement the clarity of the requirements specified, the next section is the Classification of Functional Requirements. This section details the functional requirements of the software in tabular form and includes their functional relationship to the overall product (essential, desirable, or optional). Additional sections may include Design Constraints, System Features, and Other Requirements as necessary to provide a comprehensive specification for the software project.

2. Overall Description

2.1 Product perspective

The product is within the domain of boolean algebra and digital logic. We need to understand logic gates, truth tables, circuit design, and how to interact with boolean algebra equations. Additionally, we need to understand where this product will be used. It could be used similarly to a calculator online, for quick computation, or in an educational context where students are learning within the domain of the product, meaning that they need explanations and some form of guidance within the product.

2.1.1 User Interfaces

One of the factors that could affect the building of the user interface is the choice of using a graphical versus textual UI. Using a textual interface requires use of the interface and designing a terminal friendly format. This can be done in a few functions. On the other hand, if we do a graphical interface, we will need to learn about GUIS and how to apply them. For user input, we have to address differences between users when inputting the expression. lowercase letters, spacing, and more.

2.1.2 Software Interfaces

We need to define and organize the different components of the software in a clear manner, because it promotes reusability of code. Could be commented, and clean. There should be file organization in the github, so team members can pull and push without having merge errors.

2.1.3 Memory Constraints

We need to choose the right data structures for the boolean expressions and values, to ensure that we don't consume excessive memory. The software needs to be able to handle large expressions that could put stress on the computer. We need to verify the express input to prevent excessive memory consumption and limit the size of input buffers to prevent memory exhaustion with large inputs. We need to test different expressions and inputs of different memory usage, to make sure that it runs smoothly with any expression given.

2.2 Product functions

There are few things that could affect product functions. First is the technical restraints the user has to face such as limitation of computational resources such as the memory or CPU. Additionally, the length and complexity of the expression can cause the product to take a longer execution time, and could cause issues if the proper data structures aren't used and memory isn't used efficiently. The external libraries used need to be compatible with other machines running the software. Finally, the way the user inputs the expression and the way the computer outputs the necessary data will be different based on whether we use a graphical interface or a textual interface for our UI.

2.3 User characteristics

Users may vary in technical experience, from people with years of coding experience to those who haven't interacted with a computer as much. Additionally, even though the software is the same for everyone, each person has their own use for it. Some may use it to learn boolean algebra, while others use it for designing circuits or something more practical. Because some users have limited knowledge on boolean algebra, the product should have clear explanations, or at least a step by step/truth table on where it got the answer from. The product needs to be able to provide use to each of these types of users, without having to sacrifice one over the other.

2.4 Constraints

The time constraint in finishing the project can result in a product that isn't completely efficient and is forced to cut some corners. This could be in the case of memory allocation and efficiency, organization of the UI for inputs and outputs, and maybe even incorrect values in the output. The time constraint forces us to prioritize certain tasks over others, and this could affect the scope in terms of the features implemented. There are also constraints in terms of our team, because there will be times where one team member is unable to complete their task, which means that another member must take it up along with their own work, which could lead to sloppy or inefficient code. Additionally, there are technical constraints. Each team member's device is different meaning a different rate of development depending on the hardware and software within each device.

2.5 Assumptions and dependencies

We have to note user assumptions and how users will interact with the system. This will help determine the design of the UI. The product needs to be able to handle dependency management on external systems. This ensures compatibility. Additionally, we have resource dependencies man power, time, and hardware infrastructure. We have a deadline and certain tasks take priority over others, so building a minimal viable product requires the right tasks to be finished first. There are also sponsor dependencies. We must implement the feedback given from our grader and make sure it aligns with the requirements and expected behavior.

3. Specific Requirements

3.1 Functionality

3.1.1 <Functional Requirement One>

Title: Operator Support
Description: The program will have logical operations for standard logic gates using the appropriate symbol.

- AND: &
- OR: |
- NOT: !
- NAND: @
- XOR: $

Reason: To simulate gate logic with a Boolean input for each side of the expression. For example, T & F will return F.

3.1.2 <Functional Requirement Two>

Title: Expression Parsing
Description: The program will parse user-input Boolean expressions in infix notation, with correct grouping operations (parentheses) and operator precedence.
Operator precedence:

- NOT: !
- NAND: @
- AND: &
- XOR: $
- OR: |

Reason: To allow multi variable expressions with correct output.
Dependencies: FR1, FR3, FR6

3.1.3 <Functional Requirement Three>

Title: Truth Value Input
Description: User input will define truth values for variables.

- True: T
- False: F

Reason: This will be the input expression used by FR1 and FR2.

3.1.4 <Functional Requirement Four>

Title: Evaluation and Output
Description: The program will calculate the final truth value for the entire expression and display the result.
Reason: Output the result of gate logic from user input.
Dependencies: FR1, FR2, FR3

3.1.5 <Functional Requirement Five>

Title: Error Handling
Description: Handle invalid expressions, missing parentheses, unknown operators, and any other issues.
Reason: To output an error message to the user for any incorrect use of the program.

will return F.

3.1.6 *<Functional Requirement Six>*

Title: Parenthesis Handling
Description: Logic for handling expressions enclosed within parentheses.
Reason: Allow grouping of certain sections of an expression with parentheses.

3.2 Use-Case Specifications

3.2.1 *<Use Case One>*

Title: User Interface
Description: A user will see an interface upon starting the program.

- Shows name of program being run.
- Informs user to check the README file to understand how to use the program.
- Asks user for a valid Boolean expression.
- Displays results of valid expression.
- If not valid, displays error message.
- Prompts user to continue or quit.

Note: There is seemingly one use case for this program, a single user enters one expression at a time and receives some output based on this input.

3.3 Supplementary Requirements

3.3.1 *<Supplementary Requirement One>*

Title: Programming Language
Description: The program will be implemented using the C++ language.

3.3.2 *<Supplementary Requirement Two>*

Title: README file
Description: A user manual explaining how to use the program with examples of valid expressions and the expected output.

3.3.3 *<Supplementary Requirement Three>*

Title: Documented Code
Description: The code should be commented for easy readability of each code block.

4. Classification of Functional Requirements

Functionality	Type
Operator Support: Implement logical operations for: • AND (&): Returns True if both operands are True • OR (|): Returns True if at least one operand is True • NOT (!): Inverts the truth value of its operand • NAND (@): Returns True only if both operands are False (opposite of AND) • XOR ($): Returns True if exactly one operand is True	Essential
Expression Parsing: Develop a mechanism to parse user-provided Boolean expressions in infix notation, respecting operator precedence and parentheses.	Essential
Truth Value Input: Allow users to define truth values (True/False) for each variable represented by T and F.	Essential
Evaluation and Output: Calculate the final truth value of the entire expression and present it clearly (True or False).	Essential
Error Handling: Implement robust error handling for invalid expressions, missing parentheses, unknown operators, or other potential issues, and provide informative error messages.	Essential
Parenthesis Handling: Ensure that your program can handle expressions enclosed within parentheses (including seemingly excessive but correctly included pairs of parentheses) to determine the order of evaluation.	Essential