RAND Corporated

GUI Boolean Calculator
Software Requirements Specifications

Version 3.0

Revision History

Date	Version	Description	Author
16/03/2024	1.0	Overall Description	Shivansh Shrivas
19/03/2024	2.0	Section 1 and Section 3	Jahnvi Maddila
21/03/2024	3.0	final fixings of the document	Jahnvi Maddila

Table of Contents

Software Requirements Specifications

1. Introduction

This document examines the purpose, scope, definitions, acronyms, abbreviations, references, and overview of the Software Requirements Specifications. It explains the requirements for building a boolean calculator in C++. We go into how we will use basic commands in C++ to code various functionalities and the GUI for the project.

1.1 Purpose

The purpose of this document is to outline the requirements to build a boolean calculator using C++. We need this document because it is the blueprint to build our code. The SRS fully describes the external behavior of the application or subsystem identified. 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

This applies to the software application of our boolean calculator using C++. It is associated with the use case of a user wanting to enter math and have the solution printed.

1.3 Definitions, Acronyms, and Abbreviations

a. calculator – GUI Boolean Calculator
b. AND (&): Returns True if both operands are True
c. OR (|): Returns True if at least one operand is True
d. NOT (!): Inverts the truth value of its operand
e. NAND (@): Returns True only if both operands are False (opposite of AND)
f. XOR ($): Returns True if an odd number of inputs are true.

1.4 References

We do not reference another document.

1.5 Overview

The rest of the SRS contains descriptions of the Requirements and goes into details, such as the functional requirements and use-case specifications.

2. Overall Description

2.1 Product Perspective

2.1.1 System Interfaces

The calculator will run on various operating systems such as Windows, MacOS, and Linux, utilizing standard system libraries and resources.

Minimum Operating System Requirements:

Windows: 10 (64-bit) or later versions

MacOS: 10.14 (Mojave) or later versions
Linux: Ubuntu 18.04 LTS (or equivalent) or later versions

2.1.2 User Interfaces

The user interface will consist of input fields for entering Boolean expressions and buttons for performing operations like AND, OR, NOT etc. (Refer 2.1.7). It will also be equipped with parenthesis support for more specific and complex Boolean expressions. There will also be output fields to display the results of the operations.

2.1.3 Hardware Interfaces

a. **Input Devices:** The calculator will interact with standard input devices such as keyboard and mice (or similar compatible pointing devices).
b. **Output Devices:** The calculator will display the results of the Boolean expressions on standard output devices such as monitors, external displays, etc.

Minimum Hardware Requirements:
Processor: Intel Core i3 (or equivalent) or higher.
Memory (RAM): 4 GB RAM or higher.
Storage: 100 MB of available disk space for installation.
Display: Minimum resolution of 1280x800 pixels.
Graphics: Integrated graphics or discrete graphics card with DirectX 10 support.
Input Devices: Keyboard and mouse or compatible pointing device.

2.1.4 Software Interfaces

a. **CLI Compatibility:** The calculator should be able to accept input and provide output through a terminal interface.
b. **Cross-Platform Support:** By supporting both GUI and CLI modes of operation, the calculator demonstrates cross-platform compatibility, enabling users to run it on various operating systems without the need for significant modifications.
c. **Error Handling:** The calculator should handle errors gracefully and provide informative messages to users in case of invalid input or other issues. Error messages should be clear and actionable, guiding users on how to correct the problem.

2.1.5 Communication Interfaces

No specific communication interfaces are required as the calculator operates as a standalone application.

2.1.6 Memory Constraints

The application should be designed to be memory-efficient, considering the potential complexity of Boolean expressions that users may input. Memory usage should not exceed 100 MB during normal operation to ensure efficient resource utilization.

2.1.7 Operations

Supported Logical Operations:
a. **AND (&):** Returns True if both operands are True.
b. **OR (|):** Returns True if at least one operand is True.

c. **NOT (!):** Inverts the truth value of its operand.
d. **NAND (@):** Returns True only if both operands are False (opposite of AND).
e. **XOR ($):** Returns True if exactly one operand is True.
f. **Parenthesis [(,)] support:** For grouping expressions.

2.2 Product Functions

The calculator will allow users to:

a. Input Boolean expressions using a graphical user interface.
b. Perform Boolean operations (AND, OR, NOT, XOR, etc.) on the input expressions.
c. View the results of the operations in real time.
d. Clear/reset the input and output fields.

2.3 User Characteristics

The target users for the GUI Boolean Calculator are individuals who need to evaluate Boolean expressions but prefer a more user-friendly interface than command-line tools or programming environments. Users may vary in their familiarity with Boolean algebra and computer operations.

2.4 Constraints

a. Platform Compatibility:

The graphical user interface (GUI) should adapt seamlessly to different screen resolutions, including standard monitors and high-DPI displays.

The calculator must be compatible with Windows, macOS, and Linux.

b. Performance Requirements:

Input validation should occur instantaneously, providing immediate feedback to the user upon entering expressions.

Boolean operations must execute within milliseconds, even for complex expressions with multiple operators and variables.

Memory usage should not exceed 100 MB during normal operation to ensure efficient resource utilization.

c. Resource Constraints:

The calculator should be designed to run efficiently on low-resource devices such as budget laptops and mobile devices, with minimal impact on battery life.

Disk space usage should be minimized, with the application installation size not exceeding 50 MB.

d. User Experience Guidelines:

Error messages should be clear and actionable, guiding users on how to correct input errors or resolve issues.

2.5 Assumptions and Dependencies

Assumptions:

a. **User's Basic Understanding of Boolean Algebra:** It is assumed that users have a basic understanding of Boolean algebra concepts such as logical operators (AND, OR, NOT etc.) and Boolean expressions. The calculator's interface will provide tooltips or help documentation to assist users who may require additional guidance.

b. **Availability of System Resources:** It is assumed that the calculator will run on modern computing devices with sufficient resources (e.g., CPU, memory) to support its operation. Performance testing will be conducted to ensure optimal performance on a variety of hardware configurations.

c. **Stability of Software Libraries**: The calculator may rely on third-party libraries or frameworks for certain functionalities (e.g., graphical rendering, input handling). It is assumed that these libraries will remain stable and compatible with the target platforms throughout the development and deployment process.

Dependencies:

a. **Operating System Compatibility:** The calculator's compatibility with different operating systems (e.g., Windows, macOS, Linux) depends on the availability of system resources and support for required software libraries.

b. **Development Tools and Frameworks:** The development of the calculator depends on the availability and stability of development tools (e.g., integrated development environments, compilers) and frameworks (e.g., GUI libraries, programming languages) used in the development process.

c. **User Feedback and Testing:** The refinement and improvement of the calculator's features depend on user feedback and testing. Assumptions about user behavior and preferences may need to be validated through user testing and feedback collection, which could impact the project's direction and priorities.

2.6 Requirements Subsets

No specific subsets are identified for this project at this stage, as the requirements are relatively straightforward and cohesive. However, as development progresses, subsets may be defined to organize and manage more complex features or enhancements.

3. Specific Requirements

3.1 Functionality

3.1.1 Input Processing

- The boolean calculator will accept user input as a string containing a boolean expression consisting of logical operators such as "and", "or", "not", and boolean constants "True" and "False". Users can use parentheses to specify the grouping of operations if needed.
- The calculator will validate user input to ensure it forms a syntactically correct boolean expression and will raise an exception if an invalid expression is detected.

3.1.2 Expression Evaluation

- The boolean calculator will evaluate boolean expressions according to logical operations based on the precedence rules of logical operators.
- It will support logical operations including "and", "or", and "not".
- Evaluation will be performed from left to right for both "and" and "or" operations.
- The calculator will utilize a stack-based approach to evaluate expressions and operators.

3.1.3 User Interface

- The boolean calculator will have a user-friendly interface for input and output on the command line that displays the result of the expression, allows the user to make a new calculation and allows the user to exit the calculator.

3.1.4 Error Handling

- The boolean calculator will raise exceptions in response to errors made by the user, such as invalid input or unsupported operations. It will display appropriate error messages to assist the user in rectifying the error.

3.2 Use-Case Specifications

Expression Parsing:

- Define a use case where the program parses user-provided Boolean expressions in infix notation.
- Ensure proper handling of operator precedence and parentheses to represent the expression structure accurately.

Truth Value Input:

- Allow users to input truth values (True/False) for each variable represented by T and F.
- Ensure the program captures and utilizes these truth values in evaluating the Boolean expressions.

Evaluation and Output:

- Develop a use case for evaluating the final truth value of the entire expression.
- Present the result clearly to the user, indicating whether the expression evaluates to True or False.

Error Handling:

- Implement robust error handling mechanisms to address various scenarios such as:
  - Invalid expressions
  - Missing parentheses
  - Unknown operators
  - Other potential issues that may arise during expression evaluation.
- Provide informative error messages to guide users in resolving encountered issues.

Parenthesis Handling:

- Define a use case to handle expressions enclosed within parentheses.
- Ensure the program correctly interprets and evaluates nested expressions, respecting the order of evaluation determined by parentheses.

3.3 Supplementary Requirements

Performance:

- Specify non-functional requirements related to performance, such as response time and system scalability, to ensure efficient execution of the Boolean expression evaluator.

Usability:

- Define usability requirements focusing on user interaction and interface design.
- Ensure the program offers a user-friendly experience through clear prompts, intuitive input methods, and informative output messages.

Portability:

- Address requirements related to portability, specifying the platforms or environments on which the program should run seamlessly.

Reliability:

- Specify reliability requirements concerning the stability and correctness of the program's operation.
- Ensure the program behaves predictably and consistently under various conditions, minimizing the occurrence of errors or unexpected behavior.

Maintainability:

- Define requirements aimed at facilitating future maintenance and updates to the program.
- Ensure the codebase is well-structured, documented, and easily extensible to accommodate potential enhancements or modifications.

4. Classification of Functional Requirements

Functionality	Type
Recognize and Evaluate Boolean Constants: Recognize and evaluate boolean constants (True and False) within the expression	Essential
Can Handle Logical AND	Essential
Can Handle Logical OR	Essential
Can Handle Logical NOT	Essential
Can Handle Logical XOR	Essential
Can Handle Logical NAND	Essential
Prompt user for input	Essential
Invalid input Alert	Essential
GUI	Essential
History of Operations	Desirable
Aesthetic GUI	Desirable

5. Appendices

No appendices are needed