Truthy

Boolean Logic Simulator in C++
Software Requirements Specifications

Version 1.0

Revision History

Date	Version	Description	Authors
7/3/24	1.0	Continued working on requirements	Evan, Nathan, Mitchel, Jay
21/3/24	1.0	Initial Submission Finalized	Evan, Nathan, Darshil, Mitchel, Jay

Table of Contents

Software Requirements Specifications

1. Introduction

This Software Requirements Specification (SRS) document pertains to the Boolean Logic Simulator project. It outlines the external behavior of software designed to process and return results from boolean expression input by users, incorporating operators such as AND, OR, NOT, NAND, XOR. In cases of invalid input, the software will return the type of error. The project must be developed in C++, be object-oriented, and be completed by April 30th.

The scope of this project includes evaluating boolean expressions, error handling, variable declaration and input, and displaying the solution along with a truth table. Details on the implementation of the project are covered in the project Architecture and Design document, not in this SRS.

1.1 Purpose

The purpose of the software is to handle every boolean expression inputted by a user that includes the following operands: And (&), Or(|), Not(!), Nand(@), Xor($). If a user inputs an invalid boolean expression, the type of error is returned back to the user instead of an output. This software must use parsing to break apart the string inputted by a user, and must be able to handle parenthesis. The software must allow the user to define variables with the correct truth values. Our design constraints consist of: the language has to be c++, it must be tested, it must be completed by April 30th, and it must be object oriented. Our non-functional requirements include a response time of under 5 seconds and a file size of less than 200 megabytes.

1.2 Scope

The project will include the following use cases:

- The ability to evaluate boolean expressions using AND, OR, NOT, NOR, NAND, XOR.
- Error handling, that will catch any invalid boolean expressions or any other errors.
- The ability to declare variables and/or input T or F values
- The solution to the boolean equation will be displayed, along with a truth table.
- A text-based interface that allows any valid user-inputs and displays any useful information that the program may compute.

The project will not include:

- Anything not mentioned in section 3.1 (Functionality). However, new features or functionality may be changed anytime at the discretion of the team members with approval of the project leader.

1.3 Definitions, Acronyms, and Abbreviations

Boolean Logic: A branch of mathematics that deals only in boolean values: True and False

Command Line Interface(CLI): A way of interacting with a computer or computer program using input text, usually from a keyboard

Graphical User Interface(GUI): The visual output of a computer program.

Software Requirement Specification(SRS): A document that outlines the requirements of a software engineering project

Infix Notation: Operand in between an operator e.g. (a + b)

1.4 References

Title	Source	Publishing Organization	Last Updated Date
Vision Document	Has not been created	Truthy	Has not been created
EECS 348 Project Description	Has not been created	KU	2/17/2024
Software Architecture Document	Has not been created	Truthy	Has not been created
User Manual	Has not been created	Truthy	Has not been created
Project Architecture and Design	Has not been created	Truthy	Has not been created
Project Plans	Project Plan	Truthy	2/25/2024

1.5 Overview

The *Software Requirements Specifications* document contains information about the requirements for the Boolean Logic Simulator project. This document provides requirements that describe how a part of the project should behave, but it does not provide specific information about how each part should be implemented. Information about the implementation of the Boolean Logic Simulator is described in the *Project architecture and design* document.

There are four main sections in the Software *Requirements Specifications* document: the overall description, specific requirements, classification of requirements, and appendices. The overall description constraints background related to the requirements including a description of different sections of the project, users who may use the project, and constraints the project must be completed under. The specific requirements contains details about requirements for the Boolean Logic Simulator project and references information in the overall description section. The requirements include functional requirements which describe how the application is expected to behave and non-functional requirements which describe constraints around how the application functions and how the application is developed. The classification of requirements section sorts each requirement into one of the requirement classes to summarize and prioritize each requirement. Finally, the appendices contain any additional references and data to supplement the requirements.

2. Overall Description

This section provides an overarching view of the Boolean Logic Simulator project, focusing on the environmental and contextual aspects that shape the software requirements. It's designed to give stakeholders a comprehensive understanding of the project's scope without delving into the specificities detailed in the subsequent section. This holistic approach aids in grasping the project's objectives, its interaction with users and other systems, and the constraints under which it operates.

2.1 Product perspective

The Boolean Logic Simulator is a self-contained product aimed at offering educational insights into digital logic through a hands-on approach. It serves as a bridge between theoretical knowledge and practical application, facilitating a deeper understanding of Boolean logic operations in digital circuits.

2.1.1 System Interfaces

As a standalone application, the simulator interfaces with the user's operating system to receive inputs and display outputs. It's designed to run on standard PC hardware without requiring specialized equipment.

2.1.2 User Interfaces

The simulator will feature a command line interface (CLI) that is intuitive and easy to use. Users can input Boolean expressions, evaluate expressions, and generate truth tables from Boolean expressions. This design emphasizes a streamlined, efficient experience for users who prefer scriptable or terminal-based

interactions, focusing on the core functionalities of expression evaluation and truth table generation within a CLI environment.

2.1.3 Hardware Interfaces

There are no specific hardware interfaces, as the simulator is designed to be hardware-agnostic. It requires only a standard computer with sufficient processing power to handle complex logic simulations.

2.1.4 Software Interfaces

The simulator is developed in C++ and will rely on standard libraries for its operations. It may interface with standard input/output libraries for the CLI.

2.1.5 Communication Interfaces

This application does not require network communication for its core functionality. All interactions are local to the device it runs on.

2.1.6 Memory Constraints

Given the educational nature of the simulator, it is optimized for efficiency and is designed to run effectively on computers with average memory capabilities.

2.1.7 Operations

The simulator supports various operations, including the creation and evaluation of Boolean expressions, error handling, and user interaction through both graphical and command-line interfaces.

2.2 Product functions

The product will offer functions for the evaluation of Boolean expressions, including support for logical operators like AND, OR, NOT, NAND, and XOR. It will parse expressions in infix notation, allow for variable truth value assignments, and provide clear output and error messaging.

2.3 User characteristics

The primary users of the simulator are students and educators in the fields of computer science and electrical engineering. Users are expected to have a basic understanding of Boolean logic but do not need advanced knowledge of software development or digital circuit design.

2.4 Constraints

The simulator is constrained by the capabilities of the hardware it runs on and the efficiency of its codebase. It must operate within the memory and processing limitations of average personal computers (8GB of ram and a 4 core processor).

2.5 Assumptions and dependencies

It's assumed that users have access to a computer capable of running C++ applications and that they possess a basic understanding of Boolean logic. The project's success is also dependent on the availability of development tools and libraries for C++ programming.

2.6 Requirements subsets

The project will be divided into subsets focusing on specific features like operator support, expression parsing, truth value input, evaluation and output accuracy, error handling, and user interface design. This modular approach facilitates efficient development and testing of the simulator's components.

3. Specific Requirements

Functional Requirements:

- The program should be designed to evaluate expression of the logical operators AND, OR, NOT, NAND, and XOR.
- It should be able to parse user-input boolean expressions.
- The program must correctly handle all errors.

- Users should be able to define True and False values, represented by T and F.

3.1 Functionality

3.1.1 Operator Support

1. The program must be able to perform AND operations. If given two true statements, the operation evaluates to true. Otherwise, it's false.
2. The program must be able to perform OR operations. If given two statements, the expression is true if either statements are true. Otherwise, it's false.
3. The program must be able to perform NOT operations. If given a statement, the expression is the conjugate statement.
4. The program must be able to perform NAND statements. If given two statements, the expression is true, if both statements are false. Otherwise, it's true.
5. The program must be able to perform NOR statements. If given two statements, the expression is false when both statements are the same.Otherwise, it's true.
6. The program must be able to perform XOR statements. If given two statements, the expression is only true when one statement is true. Otherwise, it's false.

3.1.2 Expression Parsing

- Expressions are parsed from user input
- Expressions may include literal boolean values, operators, parentheses, variables and whitespace
- The AND, NOT, OR, NAND, NOR, and XOR operators in expressions are parsed infix between two operands
- The NOT operator is parsed before the expression it negates
- Expressions can be nested within each other
- Expressions can be grouped with parentheses. The grouped expressions will be evaluated first before evaluating any outer operators
- Expressions are parsed in a way that respects operator precedence and expression grouping with parentheses

3.1.3 Truth Value Input

- Expressions may include literal values that evaluate to true or false
- A boolean true value is non case sensitive. It is represented as T or t
- A boolean false value is non case sensitive. It is represented as F or f

3.1.4 Evaluation and Output

- After expressions are parsed, they are evaluated and produce a single boolean value
- The boolean value output is displayed in the terminal (eg. True or False)
- If an invalid expression is given to the program, an error message is displayed as described in the Error Handling feature

3.1.5 Error Handling

- The program will be able to properly handle any errors including,
    - Mismatching or missing parenthesis
    - Missing or invalid operands
    - Missing or invalid operators
    - Any other problems that can be found
- The error handling will provide a detailed description of any errors.

3.1.6 Parenthesis Handling

- Expressions may include parentheses that group an inner expression
- During evaluation, these parentheses will be evaluated first before evaluating any neighboring operators
- For example in the expression (1 + 2) / 3 the expression 1 + 2 is evaluated before the / 3 expression

3.1.7 Truth Table Output

- The program may support truth table generation
- Expressions may include single letter variables like “x”, or “y” and must not be T or F to avoid conflicts with true and false values

- The truth table contains all possible combinations of input variables along with the output of the expression
- For example, in the expression “x|y”, the truth table will display x|y for the values of x=false with y=false, x=false with y=true, x=true with y=false, and x=true with y=true

3.2 Use-Case Specifications

- Input: The program must take in input from the user and store it to be processed.
- Operator support: The program should be able to read AND, NOT, OR, NAND, and XOR logical operators.
- Expression Parsing: The user input should be put in infix notation while adhering to precedent and parenthesis.
- Input storage: The tokenized input is stored in a data structure.
- Truth value input: The user can define truth values for the variable represented by T and F.
- Calculation: The program should calculate the inputted logical expression and provide a final true or false result.
- Error Handling: The program should check for invalid expressions and other issues. The program will provide an error message detailing the issue(s).
- Print: The program will display the final result of the given logical expression after a successful calculation.

3.3 Supplementary Requirements

Non-Functional Requirements:

- Response time of under 5 seconds for expressions less than a 100 characters on a M2 macbook
- File size under 200 megabytes

Constraints:

- Must be coded in C++
- Must be object oriented
- Must be completed by April 30th
- The program must be sufficiently tested

4. Classification of Functional Requirements

Functionality	Type
Operator Support	Essential
Expression Parsing	Essential
Truth Value Input	Essential
Evaluation and Output	Essential
Error Handling	Essential
Parenthesis Handling	Essential
Truth table	Desirable

5. Appendices

a. Reference Documents - This includes a list of materials or technical documents that were referred to in the preparation of this document.
b. Graphic Materials - This includes system architecture diagrams, data flow diagrams, ER diagrams, and other graphic materials that can help readers understand the structure and operation of the system.
c. User Manual - This provides user instructions and examples.
d. Code Documentation - This includes code annotations for instructions.