# Boolean Logic Legends

# Boolean Logic Simulator in C++
# Software Requirements Specifications

**Version 2.0**

# Revision History

| Date | Version | Description | Author |
|---|---|---|---|
| 24/Mar/24 | 1.0 | Initial Software Requirements Specifications document | All team members |
| 01/May/24 | 2.0 | Updated requirements, dates to match those discussed in meetings | Sam Grimsley |
| | | | |
| | | | |

# Table of Contents

# Software Requirements Specifications

## 1. Introduction

The Software Requirements Specification (SRS) serves as a structured blueprint delineating the anticipated functionalities, requirements, and expectations of the software application. This document offers a meticulous overview of the program's scope and intricacies, facilitating a comprehensive understanding prior to the commencement of development activities.

### 1.1 Purpose

The purpose of the SRS is to comprehensively describe what the software is expected to do. It will also list the components required for software functionality. Additionally, the SRS will cover essential factors such as nonfunctional requirements and design constraints that are crucial for software development

### 1.2 Scope

The SRS applies to a boolean logic simulator that is written in C++. The program will simulate the behavior of logic circuits. Key features needed for this program include logical operator support, expression parsing, truth value input, evaluation and output, error handling, and parenthesis handling. Guidelines for this program include the use of OOP principles to structure code, comments, and documentation to explain the logic and functionality of the program, the development of unit tests to verify correctness and different expressions, clear and informative error messages, and a user-friendly interface (text-based or graphical). Use-Cases associated with this program are evaluation and output, truth value input, and error handling

### 1.3 Definitions, Acronyms, and Abbreviations

- SRS: Software Requirements Specification
- OOP: Object-Oriented Programming
- OOP Principles: Encapsulation, Abstraction, Inheritance, Polymorphism
- EECS: Electrical Engineering and Computer Science (University of Kansas)
- EECS 348: Software Engineering I
- Logical Operations: AND (&), OR (|), NOT (!), NAND (@), XOR ($)
- Expression Parsing: Understand user-written Boolean expressions, Order of Operations
- Truth Value Input: Allow users to define truth values using “T” and “F”
- Evaluation and Output: Display final truth value (True or False) of the entire expression
- Parenthesis Handling: Program correctly reads expressions enclosed within parentheses

### 1.4 References

| Title | Date | Source | Link |
|---|---|---|---|
| EECS348: Project Description | Feb 17, 2024 | EECS 348 Software Engineering I Lecture on Canvas | 2024-EECS348-project-description.pdf |

### 1.5 Overview

| | |
|---|---|
| Overall Description | Describes the general factors and requirements for the program as well as background for the requirements |
| Specific Requirements | Describes all software requirements including functional and supplementary requirements, and use-cases. |

| | |
| --- | --- |
| Classification of Functional Requirements | List of the functional requirements indicating their type (Essential, Desirable, Optional). |

# 2. Overall Description

## 2.1 Product perspective

### *2.1.1 System Interfaces*

The boolean logic simulator will use the C++ standard input and output streams to get input from the user and display output.

### *2.1.2 User Interfaces*

The simulator will have a user-friendly interface that allows for the creation and manipulation of boolean expressions. Users can input expressions through a command-line interface. The output, answer to the expression will be displayed in a clear manner. The user can easily exit the program. If the user inputs an invalid expression, then the program will display a message indicating an invalid input.

### *2.1.3 Software Interfaces*

The simulator is built in C++ and will utilize standard C++ libraries.

### *2.1.4 Memory Constraints*

The simulator is designed to be lightweight, with minimal memory footprint. It should efficiently manage memory to handle even complex logic expressions without significant resource consumption.

### *2.1.5 Operations*

The core operation of the simulator involves parsing and evaluating boolean logic expressions input by the user. This includes super for various logic operators and the ability to handle expressions involving both boolean values and variables. The application provides output in the form of truth values or comprehensive truth tables.

## 2.2 Product functions

The Boolean Logic Simulator will simulate various logic circuit behaviors. This includes any of the various logic operators, as well as parenthesis parsing in order to create more complex expressions. Additionally, the simulator should be able to accept multiple input types in order to give users more flexibility with the simulations. Input may include boolean values or variables, and the simulator should be able to handle both cases in a way that produces meaningful results.

The simulator itself will be parsing the user-provided logical expression in order to provide a resulting value or set of values. For instance, if the user were to input “True or False”, the simulator should generate an output of “True”, and if given a variable expression such as “A or B” it should generate a set of all possible outputs given the various variable possibilities.

Output for the simulator should include options such as truth values and truth tables. Where possible, it should also be possible for the simulator to load multiple tables side-by-side for comparison. Additionally, some users may benefit from being able to create their own tables for easy comparison between expressions and desired outputs.

The simulator should have a clear user-interface that allows the user to easily create whatever expression they desire with a variety of value and variable options. The output should similarly be displayed in a clear manner, using tables and labels where appropriate. This interface can be either graphical or text-based; the priority is making it easily readable and usable.

Any errors either in user input or simulator functionality should be handled and described by the application. Error messages should be clear and informative, while error handling should prevent the application from failing due to any possible errors.

### 2.3 User characteristics

Users may include anyone with an interest in circuits or digital logic. This could range from students in a digital logic or circuits class to professionals looking for a quick description of a given circuit. Students may look to use the simulator in order to check their understanding of expressions or to find ways to create a desired output. Professional users are more likely to be using this in order to simplify complex expressions or check that the desired outcome is what is being implemented.

### 2.4 Constraints

This application must be implemented with C++. It must be completed by May 1, 2024. Any communication between users and the application should be possible using a basic keyboard. The application must run consistently and runtimes should be quick enough so as not to disrupt the user experience.

### 2.5 Assumptions and dependencies

It is assumed that users have an understanding of how to read truth values and truth tables, as well as that they have a basic understanding of the fundamentals of logic expressions.
The project may depend on various C++ standard libraries. Other dependencies will be determined and described as the project is developed.

## 3. Specific Requirements

### 3.1 Functionality

The system should allow the user to be able to use Boolean logic and simulate True(T) and False(F) expressions, calculating the final value. It must be able to handle errors and provide an error message. Below is a breakdown of specific functional requirements:

#### *3.1.1 Expression Parsing:*

The project needs a function to tokenize an input expression from the user. Each input expression should be one input that needs to be broken down into subsequent parts by the program for processing.

#### *3.1.2 Operator Precedence:*

The program needs to define the precedence of the operators rules and evaluate the input expression while considering operator precedence.

#### *3.1.3 Parenthesis Handling:*

The program needs a mechanism to identify and evaluate expressions within parentheses. Parentheses should be paired, and if any parenthesis is not paired an error should be given.

#### *3.1.4 Boolean Recognition:*

The program should recognize True (T) and False (F) operators.

#### *3.1.5 User Interface:*

The program needs a user-friendly interface. The user interface should be clear and concise, allow users to enter expressions, and display the results of the calculation. The interface should have consistent readable formatting and help messages where necessary. The program needs to allow users to easily exit the program. A user should be able to access, use, and interpret the results of the program without any prior knowledge of the program or how it works.

#### *3.1.6 Error Handling:*

The program needs an error management system to manage scenarios like unpaired parentheses, division by zero, invalid expression, invalid operators, etc. The program should display the correct error message.
The following is an incomprehensive list of error messages that should be in the program and an example for each:

- Missing Operand: ! & T
- Unknown Operator:T ? T
- Mismatched Parentheses: (T |)

- Circular Logic: !( T & T )
- Empty Expression:
- Double Operator: T && & F
- Missing Truth Value: X | Y
- Inconsistent Characters: True | F
- Operators after Operand: True!
- Invalid characters: a & b

#### *3.1.7 Arithmetic Operations:*

The program needs to support the following boolean operators:

- AND (&)
- OR (|)
- NOT (!)
- NAND (@)
- XOR ($)

#### *3.1.8 Input Validation:*

The program needs to validate the input expression to ensure that it adheres to the specified format. The program needs to reject invalid expressions with error messages.

#### *3.1.9 Result Presentation:*

The program needs to display the calculated result in a clear message.

#### *3.1.10 Variable Operands:*

The program should allow for either True and False as input operands, or for variable operands such as X, Y, Z, etc. These inputs may be used individually or in combination with each other.

#### *3.1.11 Truth Tables:*

If using variables as input, the program should generate a truth table with all possible input combinations and their corresponding outputs.

#### *3.1.12 Comparisons:*

The program should allow for comparisons between different logic expressions, either by showing multiple truth tables or showing the corresponding truth values for each expression.

#### *3.1.13 Input Types:*

The program should allow for different input formats, such as variables instead of truth values or different ways of typing operators (AND vs &).

#### *3.1.14 Expression Generation:*

The program should provide the option to input a number of variables and truth values (T, F) in a truth table, then generate an example logic expression that would satisfy the table.

##### *3.1.14.1 Don’t Care’s:*

The program may also be expanded to allow for Don’t Care conditions to be added to the generation truth table, providing the user with more options for expression generation.

### 3.2 Use-Case Specifications

*Primary Actor:* User

*Goal:* The user wants to evaluate a boolean expression’s final truth value using the provided C++ program.

*Preconditions*: The user has access to the command line interface, and has the program installed.

*Exception conditions*: The user enters an invalid expression such as unmatched parenthesis, or unsupported characters. The program should display an informative error message defining the error and

prompting the user to enter a valid expression.
*Postconditions*: The user receives a calculated result as True or False. The program then remains available for further evaluation or interaction.

*Scenario*:

1. The user launches the boolean expression evaluator program.
2. The program displays a prompt indicating readiness to receive input. The program should display a list of what the program is able to do.
3. The user enters a boolean expression containing boolean variable and boolean operators. The expression may contain (T/F) operators, logic operators (AND, OR, NOT, NAND, and XOR), parentheses, and whitespace for readability.
4. The program parses the input expression, evaluates if it contains an error and evaluates the expression if valid.
5. The program displays the calculated result to the user or an error message.
6. Optionally, the user may choose to input another expression or exit the program.

*Additional Scenarios:*

- If the user decides not to enter an expression initially, they can exit the program without evaluating an expression.
- The user can choose to view help documentation regarding the program.

*Assumptions:*

- The user understands boolean operation syntax.

### 3.3 Supplementary Requirements

3.3.1 The project must be developed using C++ and any of its standard libraries.

3.3.2 The simulator should have a minimal memory footprint allowing it to run in a minimal time period (as not to disrupt user experience) on most machines.

3.3.3 Any input must be possible using basic ASCII characters.

3.3.4 The simulator should be able to run reliably without unforced errors.

3.3.5 The project must be completed by May 2.

3.3.6 The project development process and deliverable documents shall conform to UPEDU guidelines.

## 4. Classification of Functional Requirements

| Functionality | Type |
| --- | --- |
| Expression Parsing | Essential |
| Operator Precedence | Essential |
| Parenthesis Handling | Essential |
| Boolean Recognition | Essential |
| User/Command Line Interface | Essential |
| Error Handling | Essential |

| | |
| --- | --- |
| Arithmetic Operators | Essential |
| Input Validation | Essential |
| Presentation | Desirable |
| Variable Operands | Optional |
| Truth Tables | Optional |
| Comparisons | Optional |
| Input Types | Desirable |
| Expression Generator | Optional |
| Don’t Care’s | Optional |

## 5. Appendices

Any appendices used will be added here.