Tech Titans

Boolean Logic Simulator
Software Requirements Specifications

Version 1.2

Revision History

Date	Version	Description	Author
19/03/2024	1.0	Initial work on the document.	Ceres Botkin, Cole Charpentier, Caden LeCluyse
22/03/2024	1.1	Working on use case section of document	Ceres Botkin
23/03/2024	1.2	Functionality requirements	Caden LeCluyse

Table of Contents

Software Requirements Specifications

1. Introduction

1.1 Purpose

The purpose of this SRS is to fully describe the external behavior of the Boolean Logic Simulator. This includes the functional requirements, the nonfunctional requirements, as well the design constraints. This SRS should be used by the developers as a reference when implementing the desired functionality.

1.2 Scope

This SRS applies to the Boolean Logic Simulator, a program written in C++. The program will simulate the behavior of logic circuits including operations such as AND, OR, NOT, NAND, and XOR. The features are grouped into functional requirements, nonfunctional requirements, and constraints. It is associated with the UML use-case model.

1.3 Definitions, Acronyms, and Abbreviations

SRS: Software Requirements Specifications

UML: Unified Modeling Language

C++: A high-level, general-purpose programming language. This is the programming language our program will be written in.

AND (&): A logical operator which returns true if both operands are true.

OR (|): A logical operator which returns true if at least one operator is true.

NOT (!): A logical operator which inverts the truth value of its operand.

NAND (@): A logical operator which returns true if both operands are false.

XOR ($): A logical operator which returns true if exactly one operand is true.

See the project glossary.

The program/our program/this program: A complex boolean expression evaluator/simulator. The code and documentation is on github, at https://github.com/YAKU-Student/EECS-348---Group-Project

1.4 References

1.5 Overview

The rest of the SRS contains the overall description, specific requirements, a table containing the classification of functional requirements, and appendices. The SRS is segmented into sections 1-5 with sections 1-3 containing subsections (1.1, 1.2, etc.). Earlier sections provide a general overview of the SRS while later sections become more specific.

2. Overall Description

2.1 Product perspective

2.1.1 System Interfaces

The program will not have a system interface

2.1.2 User Interfaces

The program will use a simple command line interface. It will accept an expression from the user and output the result to the command line.

2.1.3 Hardware Interfaces

This program will use C++ I/O functions to take user input from the keyboard, using the operating system's input mechanisms.

2.1.4 Software Interfaces

The boolean expression simulator will not use a software interface.

2.1.5 Communication Interfaces

The program will not use a communication interface.

2.1.6 Memory Constraints

The program has very minimal memory constraints. It just needs to be able to store the expression, result, and potentially the truth table.

2.1.7 Operations

The program will be able to perform the following operations:

- Parse boolean expressions entered by the user
- Evaluate the boolean expression
- Print the result back to the user

2.2 Product functions

The program will be able to parse and evaluate boolean expressions that contain the following operations: **AND**, **OR**, **NOT**, **NAND**, and **XOR**. Operation precedence will be able to be defined using parentheses. The program will be able to identify errors and notify the user. The program will print the result back to the user if there are no errors.

2.3 User characteristics

The program will be used by students, programmers, and engineers.

2.3.1 Students

The program will help students at all levels that wish to evaluate boolean expressions.

2.3.2 Programmers

The program is designed to help programmers of all types and at all levels that wish to see how a boolean expression would be evaluated in a programming context.

2.3.3 Engineers

The program will aid engineers that wish to see the result of a combination of logic gates.

2.4 Constraints

The program must be implemented in C++. The program must be able to run on Linux, macOS, and Windows.

2.5 Assumptions and dependencies

The following assumptions will be made:

- The user will enter any valid or invalid boolean expression
- The user may enter invalid operations
- The user may enter invalid variables

Dependencies:

- LLVM
- CMake

2.6 Requirements subsets

The requirements are one of two types: **Functional Requirements** and **Supplemental Requirements. Functional Requirements** are requirements that modify the behavior of the program itself. The **Functional Requirements** can either be deemed *essential, desirable, or optional.*

3. Specific Requirements

3.1 Functionality

The section below describes the **Functional Requirements** of the program.

3.1.1 Boolean Expression Evaluation

The program shall correctly evaluate boolean expressions composed of **AND**, **OR**, **NOT**, **NAND**, and **XOR** operators (&, |, !, @, and $). It will support grouping the expression with parentheses. The program will support single letter boolean variables (**T**/**F**).

3.1.2 Expression Parsing

This program must correctly take an input from the user, and parse the input into a representation that is suitable for evaluation.

3.1.3 Error Handling

The program must correctly identify if there is an error in the input it received from the user. It will promptly tell the user what the error was and not evaluate the expression. The error will not result in program termination, and will instead continue onto another expression.

3.1.4 Program Lifetime

The program will continue to run until **quit** or **exit** is entered by the user.

3.1.5 Truth Table Generation

The program will generate and print a truth table based on the boolean expression passed in.

3.1.6 Alternate Variable Representation

The program will also accept **True** or **False** in place of **T** and **F**.

3.2 Use-Case Specifications

The user will be provided with a terminal style (command line) interface where the user will be able to input a single boolean expression to be evaluated. A boolean expression is defined as a string of characters which include representations of the boolean true and false (T/F) and representations of the boolean operators AND, OR, NOT, NAND, and XOR (&, |, !, @, and $ respectively). The program will then parse the input and determine if it is a well formed boolean expression string.

If the boolean expression string is not well-formed, the program will implement error handling by returning an error to the user which will explain which part of the boolean expression string is ill-formed. If the expression is well-formed, then the program will utilize parentheses to mark sub-boolean expressions, proper operator precedence (the order in which the sub-expressions are evaluated based on operator), and internal truth tables for the operators, to correctly determine the result of the boolean expression. In addition, the user is able to set a variable T or F to be the result of a boolean sub-expression. The program will then print the result to the user, in which case the user can input another boolean string, or exit the program.

A well-formed boolean expression can be represented in context free grammar notation (discounting spaces and using \ to separate different expansions) as:

<Boolean Expression> → <Variable> = <Boolean Sub-Expression> \
    <Boolean Sub-Expression>

<Boolean Sub-Expression> → (<Boolean Sub-Expression>) \
    ! <Boolean Sub-Expression> \
    <Boolean Sub-Expression> <Operator> <Boolean Sub-Expression> \
    <Truth Value>

<Truth Value> → T \ F \ True \ False

<Operator> → & \ | \ @ \ $

In addition to the program itself, the program will ship with unit tests to make sure that the program is working as outlined in this section. The program will also contain a comprehensive README which will explain how to use the program itself and will contain multiple examples of well-formed statements.

3.3 Supplementary Requirements

The program itself will be written in the C++ language and will be compiled to an executable file which would be able to run on the KU EECS Lab computers. The code will contain detailed comments to aid reading during code reviews which will clearly explain how the code works.

The program will include a README document that will explain how to install and use the program.

4. Classification of Functional Requirements

Functionality	**Type**
Boolean Expression Evaluation	Essential
Expression Parsing	Essential
Error Handling	Essential
Program Lifetime	Essential
Truth Table Generation	Desirable
Alternate Variable Representation	Optional

5. Appendices

As of the current revision, there are no appendices.