<Company Name>

Boolean Logic Simulator
Software Requirements Specifications

Version <1.0>

Revision History

Date	Version	Description	Author
03/24/2024	<1.0>	Completed Section 4 Classification of Functional Requirements	Jack Gerety
03/24/2024	<1.0>	Completed Sections 1.4 References and 3.3 Supplementary Requirements	Kaleb Howard
03/24/2024	<1.0>	Completed Sections 2.1.2 User Interfaces and 2.1.4 Software Interfaces	Kyler Luong
03/24/2024	<1.0>	Completed Section 1.1, 1.2, 1.3	Malek Kchaou
03/24/2024	<1.0>	Completed Sections 3.1 Functionality, 3.2 Use case specification	Cyn Tran
03/24/2024	<1.0>	Completed Section 5 Appendices. 1.5	Jackson Holle

Table of Contents

Software Requirements Specifications

1. Introduction

The Software Requirements Specification (SRS) serves as a comprehensive document outlining the complete set of requirements for the development of the Boolean Logic Simulator software. This document captures both functional and non-functional requirements, design constraints, and other pertinent factors essential for understanding the software's scope and behavior.

1.1 Purpose

The purpose of this SRS is to provide a detailed description of the external behavior of the Boolean Logic Simulator software. It encompasses all necessary requirements to ensure the successful development, implementation, and operation of the software. Additionally, it delineates design constraints and non-functional requirements crucial for a thorough understanding of the software's specifications.

1.2 Scope

The Boolean Logic Simulator software is designed to facilitate the understanding and visualization of Boolean logic operations. It includes features such as inputting Boolean expressions, simulating their behavior, and visualizing the results through a user-friendly interface. This SRS applies to the entire software application, covering all subsystems and functionalities associated with the Boolean Logic Simulator.

1.3 Definitions, Acronyms, and Abbreviations

Definitions:

Boolean Logic: A branch of algebra that deals with variables that can have one of two possible values, typically true or false.

Logic Gate: An elementary building block of a digital circuit that performs a Boolean function.

Truth Table: A mathematical table used in logic to compute the functional values of logical expressions.

Expression: A mathematical statement that combines variables, constants, and operators.

AND Operator: A logical operator that returns true only if both operands are true; otherwise, it returns false.

OR Operator: A logical operator that returns true if at least one of the operands is true; it returns false if both operands are false.

NOT Operator: A unary operator that negates the value of its operand, i.e., it returns true if the operand is false, and false if the operand is true.

NAND Operator: A logical operator that returns false if both operands are true; otherwise, it returns true.

NOR Operator: A logical operator that returns true only if both operands are false; it returns false if at least one of the operands is true.

XOR Operator (Exclusive OR): A logical operator that returns true if exactly one of the operands is true; it returns false if both operands are true or both are false.

XNOR Operator (Exclusive NOR): A logical operator that returns true if both operands are true or both operands are false; it returns false otherwise.

Simulation: The imitation of the operation of a real-world process or system over time.

Input Interface: The means by which users can provide input to the software, such as through text fields or graphical elements.

Output Interface: The means by which the software presents results or output to the user, such as through

graphical displays or text output.

Syntax: The set of rules that defines the combinations of symbols that are considered to be correctly structured statements or expressions.

Evaluation: The process of determining the value or outcome of an expression or operation.

Error Handling: The process of detecting, reporting, and managing errors or exceptions that occur during program execution.

Acronyms and Abbreviations:

GUI: Graphical User Interface

API: Application Programming Interface

HTML: Hypertext Markup Language

CSS: Cascading Style Sheets

CPU: Central Processing Unit

RAM: Random Access Memory

UI: User Interface

UX: User Experience

QA: Quality Assurance

OS: Operating System

HTTP: Hypertext Transfer Protocol

HTTPS: Hypertext Transfer Protocol Secure

Git: Version Control System

Jira: Issue Tracking Tool

IDE: Integrated Development Environment

The above list should provide a solid foundation for understanding the terminology and abbreviations used in the context of this Boolean Logic Simulator project.

1.4 References

[IEEE830-1998]: IEEE Recommended Practice for Software Requirements Specifications, IEEE Computer Society, 1998.

EECS 348 Project Description, Dr Hossein Saiedian, 00-2024-EECS348-project-description.pdf

- Document exists in canvas projects files. Already pre-created and referenced in this document.

1.5 Overview

The Software Requirements Specification (SRS) provides a detailed blueprint for the development of the Boolean Logic Simulator software. It outlines the product's purpose, scope, and specific requirements, guiding the design, development, and testing phases. The document is structured to cover general factors influencing the product, detailed functional and non-functional requirements, use-case specifications, and supplementary requirements. By following the SRS, the development team can ensure the creation of a robust and user-friendly software solution that meets stakeholder expectations.

2. Overall Description

The Boolean Logic Simulator software aims to provide users with a versatile tool for exploring and understanding Boolean logic operations. Operating as a standalone application, it offers an intuitive interface for inputting Boolean expressions and visualizing their evaluations. The software is designed to cater to a diverse user base including students, educators, and professionals in various fields.

2.1 Product perspective

The Boolean Logic Simulator operates independently, requiring no integration with external systems. It serves as a self-contained solution for users to experiment with Boolean logic concepts.

2.1.2 User Interfaces

When the program is running, it will ask the user for a boolean expression, and when given the boolean expression, it will print the evaluated expression either as T or F.

2.1.4 Software Interfaces

Software interfaces give the ability to have different functions and methods to execute in order for the program to run. We will ask the user to give a boolean expression, and have functions ready to break down the expression, by looking for the order of operations first with parentheses, then following the boolean orders of operation of NOT, then ADD, then OR. After doing so it will divide and conquer based on the expression, and after completing the evaluation it will print T or F for True or False. It will also be able to handle errors from the user.

2.1.6 Memory Constraints

Memory constraints may impose limitations on the amount of data the Boolean Logic Simulator can process and store during operation. These constraints will be considered during the design and implementation phases to optimize memory usage and ensure efficient performance, especially for systems with limited memory resources.

2.2 Product functions

The core function of the software is to process Boolean expressions entered by users and to accurately evaluate them according to logical operations. Through algorithmic parsing and evaluation, the software generates truth values (either true or false) based on the provided expressions.

2.3 User characteristics

Target users encompass individuals with varying levels of expertise in Boolean logic. From novices seeking introductory knowledge to experienced practitioners requiring a practical tool, the software caters to a broad spectrum of users. Users are expected to possess basic familiarity with Boolean logic principles.

2.4 Constraints

The development process of the Boolean Logic Simulator is bound by constraints such as time, resources, and compatibility. Adherence to project timelines and budgetary limitations is imperative for successful delivery. Additionally, ensuring compatibility with different operating systems and hardware configurations is essential. The time requirement is the biggest source of limitation as the final program needs to be delivered by the end of spring 2024;

2.5 Assumptions and dependencies

Assumptions underpinning the software's development include the expectation of valid input from users and the software's ability to handle common syntax errors gracefully. Dependencies encompass access to requisite development tools, libraries, and programming languages necessary for implementation.

3. Specific Requirements

3.1 Functionality

This session describes in detail the expected and planned requirement for the program the team is building. In order to make sure the program works as intended and meet all of the requirements set out. This also works as a guide for the developers to easily identify all of the features and functions they need to design and implement for the program.

3.1.1 <Functional Requirement One>

The program can take user expressions to be calculated by the program.

<Functional Requirement Two>

The program can detect which expression is invalid and let the user know that their expression is an error.

<Functional Requirement Three>

The program can distinguish the different meanings of the expressions. They can tell which expression is to be evaluated and which expression affects the expression. EX: (1 + 2) * 3, the program should know the expression addition and multiplication is to be calculated and the parentheses dictate the order of the calculation.

<Functional Requirement Four>

The program prints out the expression the user inputted and the result of the input. The display of user's input allows users and developers to check if the program processes the expression correctly.

3.2 Use-Case Specifications

- The user is prompted to enter the expression they want evaluated. The user inputs this expression, which can consist of truth values (T or F), logical operators (AND, OR, NOT, NAND, NOR, XOR, XNOR), parenthesis, or spaces.
- The user submits their expression, error checking is then performed and displays an error if one is present. If no errors are present and the expression evaluates successfully, the program will print the final result.
- Scope: Calculate boolean logical expression prompted by user and output True/ False for such expression.
- Stakeholders:

User - anyone who wishes to use the program, which will enter their desired expression to be evaluated.

Developers - team members work together to design, implement, test and maintain the program to ensure the program works correctly and effectively.

- Preconditions:

  Users can access the program, therefore the program has to have a field in the program which allows the user to enter their expression to be evaluated.

  Users can get access to the program through permission of the group.

- Output: The result of the calculation from the user has to be evaluated correctly. To test this the team will create several test cases with proven results. If the result of the program match with the proven result the program works correctly else, they program need to be revised
- Input: User can enter the expression into the program. User's input has to be greater than 2 symbols, if the user inputs two values then the program will stop and inform the user that their input is false.
- Requirement: the only input that is acceptable are the AND, OR, NOT, NAND, XOR, NOR,

XNOR (boolean expression) and parentheses to determined the order of the expression, any other symbol is invalid for user to input to the program,

3.3 Supplementary Requirements

- We will implement our program in the C++ language to adhere to the project's language requirement. We will also compile it with a compiler suitable with the C++ language and ensure compatibility with the platforms Windows and Linux.
- As per the Project description, the program should be able to handle complex logic circuits with multiple gates and input/output signals. We will set testing values of varying amounts of gates and input and output signals, say 10 gates and 10 input signals, to ensure the program evaluates within a reasonable time frame of about a second. There will be at least one complete expression present in the input line.
- To ensure input validation, we will ensure we are testing for invalid and incomplete expressions. Our input signals will take in T or F, and our gates will be checked to ensure the program knows what it is and there is not an unknown character.
- As briefed above, we will ensure platform compatibility with Windows and Linux, testing on both to make sure our program works and can be successfully compiled on both our machines and the school lab machines.

4. Classification of Functional Requirements

Functionality	Type
The program will be able to parse and evaluate arithmetic expressions that contain these operators (T,F,|,$,@,!)	Essential
The code will be written in C++	Essential
The result of calculating the arithmetic expression will follow appropriate order of operations rules for logical expressions	Essential
The program will be able to handle parentheses	Essential
The program will handle errors, such as dividing by zero, and will display these errors to the user	Essential
The program's user interface will be user friendly	Essential
The program will allow users to interact with it through a GUI	Optional

5. Appendices

The project will follow the UPEDU process.

Other applicable process plans are listed in the references section, including Programming Guidelines.