LogicSim Labs

Logic Gate Simulator
Software Requirements Specifications

Version 1.1

Revision History

Date	Version	Description	Author
03/20/24	1.0	Document detailing the requirements specifications for the Logic Gate Simulator Project with use-case modeling.	Lily Gray, Fatima Avila-Cobian, Daniel Bobadilla, David Sutherland, Gaby Kill
03/24/24	1.1	Finalized document detailing the requirements specifications for the Logic Gate Simulator Project with use-case modeling.	Lily Gray, Fatima Avila-Cobian, Daniel Bobadilla, David Sutherland, Gaby Kill

Table of Contents

Software Requirements Specifications

1. Introduction

The Boolean Logic Simulator Software Requirements Specifications (SRS) covers the comprehensive requirements to complete the given task of evaluating Boolean logic expressions. These requirements will be essential for the system's development and will guide it as well. Covering phases of its development like design, implementation, and testing. This will help align the needs of the stakeholder's expectations and the outcome of the product.

1.1 Purpose

The Software Requirements Specifications Document describes how the software should interact with the user in ways that satisfy shareholder preferences. Software Requirements encompass functional requirements, non-functional requirements, and constraints placed on the software by other programs and/or existing applications. These requirements will be laid out in detail in Section 3 to provide a comprehensive view of exactly what the Logic Gate Simulator project should be capable of. This document is related to the Software Development Plan for the EECS 348 Logic Gate Simulator Project[1], which lays out plans for the development of the entire project, including software requirements.

1.2 Scope

This Boolean logic simulator software is meant to be capable of evaluating AND, OR, NOT, NAND, and XOR expressions. All expressions are based on Boolean values and generate a single Boolean answer with expression variations. The simulator serves to quickly solve many problems such as digital circuit design and computer architecture and processor design to name a few. This software is not meant to provide truth tables or detailed explanations of the evaluation.

1.3 Definitions, Acronyms, and Abbreviations

Operator precedence: this refers to the order in which operators should be calculated, like PEMDAS in math, above all else the NOT operator must be performed first (besides parenthesis which will be described below)

Parenthesis calculation: the first thing the program must do is locate the innermost matching pairs of parentheses and simplify everything inside it into a single true or false. Once all parenthesis in the input is simplified it can continue as normal

1.4 References

See the Appendix at the end of the document.

1.5 Overview

This Software Requirement Specification for the Boolean Logic Simulator is meant to provide a path for the development of any logic gate-related software. This document outlines its requirements and constraints for its overall development which is structured throughout the document with description of how it's meant to work and what kinds of tasks it is meant to handle along with the expected output format.

2. Overall Description

Section 2 provides a brief introduction to factors that are influential in determining software requirements. This section will give descriptions of each factor in the context of software engineering so the reader can have an adequate background for Section 3. The functional and non-functional requirements and design constraints are listed in Section 3.

2.1 Product perspective

The perspective is for the program to take in reasonably complex user inputs and output one single answer, either true or false. It should be able to calculate operators in order and prioritize parenthesis above all else, treating the parenthesis as its own smaller calculation.

2.1.1 User Interfaces

The user interface will start with a simple message asking to input the problem, if the input is incomplete or incorrect then it will give an error message and prompt again. Otherwise, once it calculates the answer it should display true or false and prompt the user for another input.

2.1.2 Software Interfaces

In the code, it will be clearly indicated what is what with comments, for example the stack class we use, all the functions and their purpose, and printing messages based on what happened (if an “and” was performed).

2.1.3 Memory Constraints

The program will use memory with the stack where it must store the input one item at a time. The exact limit depends on the computer it's run on.

2.2 Product functions

Basic functions: there will be a function for calculating each individual operator, so 5 of those in total.
Stack functions: for the stack there will be an add, pop, peek, and return length function.

2.3 User characteristics

The users for this program will most likely be students or teachers who are looking for a quick way to solve a logic problem.

2.4 Constraints

The program can't make a truth table, its only output is one true or false. There is also most likely an unknown limit as to how large and complex the input can be for the computer to handle.

2.5 Assumptions and dependencies

This program assumes that the user understands that the only input for true or false is T and F, no 1s or 0s and no spelling out the words, only the letters. It's assumed the user won't put in any spaces, and that they know all the symbols for the logic operators.

3. Specific Requirements

The Specific Requirements section of the SRS explains the function, non-functional, and constraint requirements necessary for the Logic Gate Simulator project. The purpose of documenting these requirements with a high degree of detail is to ensure that the project can be designed and tested to the exact desired specifications. Use-case modeling will be used to describe different ways users can interact with the program. The use cases will be outlined in section 3.2.

3.1 Functionality

This section will list the functional requirements of the Logic Gate Simulator project. The functionality of the software will be described using natural language and use-case modeling. The functional requirements encompass the features of the program that require direct user interaction and yield a tangible result, such as returning a value that the user types. Non-functional requirements in section 3.3 will cover factors that affect the performance and robustness of the code. Section 3.3 will also list the constraints that influence the design process.

3.1.1 Prompt User Input

Upon execution, the program will prompt the user for input. The program will expect the input as a Boolean expression in the form of: ‘operand’ [operator] ‘operand.’ Once the input is evaluated, the program will continue to prompt the user until the user exits the program.

3.1.2 Parse User Input

Once user input is supplied, the program will parse through the input character by character and will handle each token accordingly. Once parsed, the program will either evaluate the input or return an error.

3.1.3 Evaluate User Input

The program must be able to parse a user’s expression typed in the terminal and store it as an array. Through a series of steps, the program should return the evaluation of the logical expression.

3.1.3 Recognize Invalid Input

The program must be able to reject a user's input if there is an invalid expression such as an operator other than &, |, @, $, !. Other invalid expressions include mismatched parentheses and an incorrect number of variables. The program should evaluate expressions with extraneous parentheses but be able to sort through them correctly.

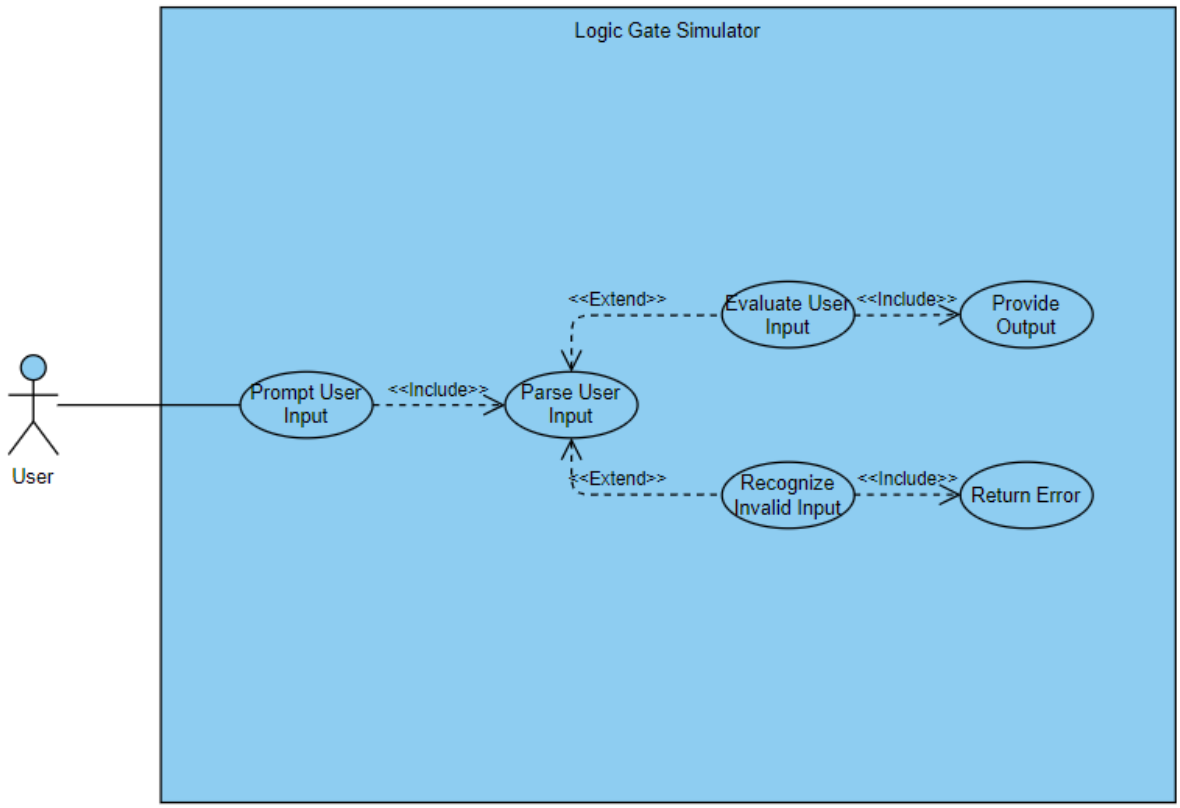
3.1.4 Return Error for Invalid Input

Ideally, the program will print an error to the screen if the input was invalid.

3.1.5 Give Output in Readable Format

The output for this program should be a simple T or F. Ideally, the output will print the original expression to the screen along with its logical evaluation in the form of "original expression = logical evaluation".

3.2 Use-Case Specifications



3.3 Supplementary Requirements

The Logic Gate Simulator has several non-functional requirements and design constraints. PowerPoint slides from EECS 348 were consulted for guidance on considering non-functional requirements[2].

Non-Functional Requirements:

- Readability- the code must be commented thoroughly and have a logical flow.
- Concision- the code must be concise, e.g. extraneous and verbose code should be reworded for ease of understanding.
- Portability- the program should be able to run on any terminal or IDE that handles C++.
- Performance- the software should not be weighed down by heavy memory usage or extraneous lines of code to ensure an immediate return of results to the user.
- Usability- the program should be easy to understand and interact with, especially from the user's perspective.
- Reliability- given the nature of the project, the program should not experience any system failures.

Constraints:

- Programming Language- the project must be completed in C++.
- Design Team- the team is limited to 5 people.
- Environment- the team is using the Virtual Studio Code IDE.
- Knowledge- only one team member has prior experience with C++.
- Project Description- the code must be object oriented.

4. Classification of Functional Requirements

Functionality	Type
Prompt user input	Essential
Parse user input	Essential
Evaluate user input	Essential
Recognize invalid input	Essential
Return error for invalid input	Desirable
Give output in a readable format	Desirable

5. Appendices

Appendices are considered part of the requirements for this document.

1. Gray, L., Bobadilla, E., Avila Cobian, F., Sutherland, D., Kill, G. “EECS 348 Logic Gate Simulator Project Software Development Plan,” last modified February 25, 2024, https://github.com/gmkill/348_Group_Project/blob/main/01-Project-Plan-marked.pdf.

2. Saiedian, H. *Software requirements engineering* [PowerPoint Slides]. Department of Electrical Engineering and Computer Science. University of Kansas.