<The Booleanators>

<Boolean Logic Simulator in C++>
Software Architecture Document

Version <1.0>

Revision History

Date	Version	Description	Author
<dd/mmm/yy>	<x.x>	<details>	<name>

Table of Contents

Software Architecture Document

1. Introduction

1.1 Purpose

The goal of this lab is to use what we know about software architecture and apply it to a real project by creating a Software Architecture Document (SAD). This document will help us organize and design our software project efficiently. We'll pick a suitable software design and pinpoint its key parts, like the input class, tokenizer, parser, evaluation class, user interface, and error handler. Through this project, we aim to learn how different design choices affect how well the software works and performs.

1.2 Scope

This Software Architecture Document (SAD) applies to our entire software project. It affects how we plan, design, and build our software. This document guides us in organizing the different parts of our project, like how data is entered, processed, and displayed to users. It also influences decisions about the software's layout and helps ensure our project works well and can grow or change as needed. By following this document, we can make sure our software is easy to use, efficient, and ready to handle any updates or improvements in the future.

1.3 Definitions, Acronyms, and Abbreviations

AND (&): Returns True if both operands are True

OR (|): Returns True if at least one operand is True

NOT (!): Inverts the truth value of its operand

NAND (@): Returns True only if both operands are False (opposite of AND)

XOR ($): Returns True if exactly one operand is True

T: True Statement

F: False Statement

1.4 Overview

This Software Architecture Document features architectural representation of our project software as well as featuring the architectural goals and architectural constraints of the team project. The document expands on the Logical View of the software and gives examples of architectural significant classes for our team project software. Finally, this document features example inputs and resulting outputs as a guide for the final operational ability of the Boolean Logic Simulator.

2. Architectural Representation

The architecture of our software is depicted using the Logical View. This view focuses on the functional aspects of the system, showcasing how the software components interact with one another to process data and perform tasks.
Input Class: Manages user inputs and prepares data for processing.

Tokenizer: Breaks down the input into manageable tokens, which are the basic elements like operands and operators.

Parser: Analyzes the tokens according to the rules of grammar to understand the structure and relationship between tokens.

Evaluation Class: Evaluates the parsed data, applying logical operations to compute results.

User Interface: Provides the graphical interface for users to interact with the software, enter data, and view results.

Error Handler: Manages and responds to errors within the software, ensuring stability and providing

feedback for troubleshooting.

Each of these components is crucial for the operation of our Boolean Logic Simulator, a key project for our team. This simulator uses logical operations such as AND, OR, NOT, NAND, and XOR to compute and display results based on user inputs. This document not only explains the roles of these components but also demonstrates how they contribute to the overall functionality of the system. By understanding and implementing this architecture, we aim to create a software tool that is efficient, easy to use, and capable of evolving with future needs.

3. Architectural Goals and Constraints

The architectural goals of our software program are extremely limited. Safety, security, and privacy are of low focus to our software. The main architectural goal of our project is to effectively perform the Boolean logic simulation. The project will need to be usable in Windows, Mac, and Linux environments. The project code will also be easily maintained and easily modified and expanded if new features are desired. Additionally, the code of the software will be structured so as to be easily reused if necessary for future projects.

The architectural constraints of the project include the relatively limited knowledge of the chosen programming language (C++) by the members of the team. Additionally, the team is limited in its ability to meet and construct the project for long periods of time.

4. Logical View

4.1 Overview

The major layers of the software are the data layer, the service layer, the logic layer and finally the presentation layer.

The data layer will contain the Boolean logic gates and other expressions given. This layer will interact with the logic layer and the service layer.

The service layer will ensure that the logic layer can perform the correct evaluation. This includes parsing the input expression through the input operands, operators and parentheses. The service layer will also handle any errors arising from incorrect or invalid inputs. The service layer will interact with both the data layer and the logic layer to provide the transition of the input.

The logic layer will actually handle the Boolean logic and give the final result of the expression. This layer will interact with the data layer to properly parse the expression and do the correct evaluations of the Boolean expressions. The logic layer will interact with all other layers and is the most important.

The presentation layer will be the layer visible to the user and will deal with user interface, input and providing clear output, all in a user-friendly way. This layer will interact with the logic layer and process the given output into a clear result for the user.

4.2 Architecturally Significant Design Modules or Packages

Input Class - This class will deal with taking in the user's input and storing it

Tokenizer Class - This class will manipulate the input into a format that will allow the parser and evaluation classes to properly evaluate the correct input

Parser Class - This class will determine the order of operations based on given operator precedence and handle parentheses to ensure the correct evaluation of sub-expressions

Evaluation Class - This class will take the result from the parser class and complete the evaluation of the Boolean expression following the given Boolean Rules

User Interface Class - This class will be a user-friendly text-based interface that will allow the user to easily enter inputs and view outputs of Boolean expression

Error Handling Class - This class will oversee multiple parts of the expression process such as ensuring that

the user's input is valid and handling any other errors that arise during the evaluation process

5. Interface Description

The major interface will be a user-friendly text-based input interface that takes in a Boolean expression from the user and the output will be the final truth value for the input expression.

Valid Inputs	Resulting Outputs
T&F	F
T|F	T
T&!F	T
F@F	T
T$T	F
(T&F)@(T|F)	F

6. Quality

The software architecture's main focus will be its extensive ability to handle complex Boolean expressions and reliably output the correct result of those expressions. The program will also be usable in Windows, Mac, and Linux environments. Other characteristics such as safety, security, and privacy are not a focus of the software architecture.