HL Group

Boolean Logic Simulator in C++
Software Architecture Document

Version 1.1

Revision History

Date	Version	Description	Author
04/08/2024	1.0	Initial Template Filled Out	Kemar Wilson
04/08/2024	1.0	Initial Template Filled Out	Abdulahi Mohamed
04/08/2024	1.0	Initial Template Filled Out	Riley England
04/14/2024	1.1	Reviewed and Corrected Information	Kemar Wilson, Riley England

Table of Contents

Software Architecture Document

1. Introduction

HL Group aims to develop a C++ Boolean Logic Simulator software application that enables users to perform logical operations based on Boolean algebra. This Software Architecture Document provides a comprehensive overview of the architectural design and structure of the Boolean Logic Calculator software. The document outlines the purpose, scope, definitions, acronyms, and abbreviations used throughout the software development process. By detailing the architecture and design principles, this document serves as a reference guide for developers, stakeholders, and users involved in the creation and utilization of the Boolean Logic Calculator software.

1.1 Purpose

The Software Architecture Document for the Boolean Logic Simulator in C++ serves as a fundamental guide that provides a comprehensive architectural overview of the system. It is designed to capture and convey the significant architectural decisions made during the design and development of the Boolean Logic Simulator. This document utilizes various architectural views to depict different aspects of the system, ensuring a holistic understanding of the architecture.

Role and Structure of the Document: The primary role of this Software Architecture Document is to outline the architectural design and decisions behind the Boolean Logic Simulator project. It serves as a roadmap for developers, architects, and stakeholders to understand the system's design principles, components, interactions, and constraints. The document is structured to present clear and detailed insights into the system's architecture, emphasizing key design choices and their implications.

Audience and Usage: The document is intended for a diverse set of audiences involved in the project, including software architects, developers, project managers, quality assurance teams, and stakeholders. Software architects will use this document to define the system's high-level structure, interfaces, and behavior, guiding development efforts. Developers will refer to it for detailed technical information on the system's components and interactions. Project managers can leverage the document to assess project progress and ensure alignment with architectural guidelines. Quality assurance teams will utilize it to define test strategies aligned with the system's architecture. Stakeholders will gain insights into the system's architecture and understand how it aligns with business objectives.

1.2 Scope

The Software Architecture Document for the Boolean Logic Simulator in C++ provides a comprehensive overview of the system's architecture, capturing significant design decisions. It influences system design, development efforts, interactions, scalability, performance, maintenance, and evolution. The document is intended for architects, developers, project managers, quality assurance teams, and stakeholders, guiding them in understanding and utilizing the system's architecture to ensure successful project outcomes.

1.3 Definitions, Acronyms, and Abbreviations

Project/Program: Boolean Logic Simulator in C++ as described by the SDS. See *Section 1.4 References for SDS*
SAD/This document: Software Architecture Document. See *Section 1.4 References for SAD*
SDP: Software Development Plan. See *Section 1.4 References for SDP*
SRS: Software Requirements Specifications. See *Section 1.4 References for SRS*

1.4 References

SRS: Accessible at https://github.com/KNEternity/348HL/tree/main/docs

SDP: Accessible at https://github.com/KNEternity/348HL/tree/main/docs

SAD: Accessible at https://github.com/KNEternity/348HL/tree/main/docs

1.5 Overview

The **Software Architecture Document** for the Boolean Logic Simulator in C++ serves as a comprehensive guide that delves into the intricate details of the system's architectural design and structure. This document encapsulates a wealth of information, including architectural representation, goals, constraints, use cases, logical views, interface descriptions, size and performance considerations, and quality attributes.

By exploring the **architectural representation**, the document sheds light on what software architecture entails for the current system and how it is visually depicted. It enumerates the necessary views and elucidates the types of model elements contained within each view, providing a holistic understanding of the system's architecture.

The **architectural goals and constraints** section dives into the software requirements and objectives that significantly impact the architecture. It delves into various aspects such as safety, security, privacy, portability, and distribution, capturing the unique constraints that may apply, including design strategies, development tools, team structures, schedules, and legacy code considerations.

In the **use-case view**, significant use cases or scenarios are outlined to highlight central system functionalities or architectural coverage that stress specific architectural points or delicate aspects of the architecture. This section serves as a pivotal point in understanding the system's core functionalities and architectural implications.

Moving on to the **logical view**, the document provides insights into the architecturally significant parts of the design model, emphasizing its decomposition into subsystems and packages. It introduces key classes, their responsibilities, relationships, operations, and attributes, offering a detailed perspective on the system's structural design.

The **interface description** section elaborates on the major entity interfaces, encompassing screen formats, valid inputs, and resulting outputs. It serves as a bridge between the software system and its users, providing clarity on how users interact with the system and what outputs they can expect.

Size and performance considerations are meticulously outlined, detailing the major dimensioning characteristics of the software that impact the architecture, along with target performance constraints. This section is crucial in understanding the system's scalability, efficiency, and resource requirements.

Lastly, the **quality attributes** section delves into how the software architecture contributes to the system's capabilities beyond functionality. It emphasizes extensibility, reliability, portability, and other quality attributes, ensuring that the system meets the highest standards in terms of safety, security, and privacy implications.

In essence, the **Software Architecture Document** for the Boolean Logic Simulator in C++ is a comprehensive repository of architectural insights that empowers stakeholders, architects, developers, and project managers to make informed decisions, align with architectural guidelines, and drive successful project outcomes.

2. Architectural Representation

The architectural representation section provides a comprehensive insight into the software architecture of the current system, elucidating its essence and visual depiction. It serves as a foundational guide to understanding the architectural design and structure that underpins the Boolean Logic Simulator in C++.

Software Architecture Overview: Software architecture for the current system encapsulates the foundational design principles, structural components, and behavioral aspects that define the system's functionality and interactions. It encompasses the high-level blueprint that guides the development and implementation of the Boolean Logic Simulator, outlining the system's key architectural elements and their relationships.

Representation and Views: The software architecture is visually represented through a series of architectural views, each offering a unique perspective on the system's design and functionality. These views act as lenses through which different aspects of the system are analyzed and understood.

- **Functional View:** This view focuses on the functional aspects of the system, detailing the primary functions, operations, and interactions that drive the system's behavior. It contains model elements such as use cases, scenarios, and functional requirements, providing a clear insight into the system's functional capabilities.
- **Structural View:** The structural view delves into the system's structural composition, highlighting the components, subsystems, and their relationships. It includes model elements like classes, interfaces, packages, and their dependencies, offering a deep dive into the system's structural organization.
- **Behavioral View:** The behavioral view explores the dynamic aspects of the system, elucidating how components interact and behave at runtime. It contains elements such as sequence diagrams, state machines, and activity diagrams, showcasing the system's behavioral patterns and interactions.

Model Elements in Each View:

- **Functional View:** Contains elements such as use cases, scenarios, functional requirements, user stories, and business processes.
- **Structural View:** Encompasses elements like classes, interfaces, packages, modules, relationships, dependencies, and architectural styles.
- **Behavioral View:** Includes elements such as sequence diagrams, state diagrams, activity diagrams, interaction diagrams, and event traces.

By enumerating the necessary views and detailing the types of model elements within each view, the architectural representation section provides a holistic understanding of the system's architecture. It serves as a foundational guide for stakeholders, architects, and developers to comprehend the architectural design and structural nuances of the Boolean Logic Simulator system.

3. Architectural Goals and Constraints

The software requirements and objectives that significantly influence the architecture of the Boolean Logic Simulator in C++ are detailed. This section also encompasses the special constraints that impact the architectural decisions and design considerations of the system.

Software Requirements and Objectives:

- **Safety:** Ensuring the system operates reliably and securely without risks to users' data or system integrity.
- **Security:** Implementing robust security measures to protect sensitive information and prevent unauthorized access.
- **Privacy:** Safeguarding user privacy and data confidentiality through compliance with privacy regulations.
- **Portability:** Designing the software to be easily portable across different platforms and environments.
- **Distribution:** Facilitating efficient distribution and deployment of the system to end-users.
- **Reuse:** Promoting code reusability to reduce redundancy and enhance maintainability.

Special Constraints:

- **Design and Implementation Strategy:** Adhering to specific design patterns and architectural styles to meet system requirements. The project must be implemented using the C++ programming language. It should also be able to run on popular operating systems, including but not limited to: Linux and Windows.

- **Development Tools:** Utilizing specific development tools and technologies to support the system's architecture.
- **Team Structure:** Aligning team roles and responsibilities to ensure effective collaboration and communication.
- **Schedule:** Meeting project timelines and milestones to ensure timely delivery of the system.

4. Use-Case View

The use-case view of the Boolean Logic Simulator in C++ outlines the essential functionalities and interactions that the software will support. It encompasses the following key use cases:

1)Accept User Inputs	2)Parse Boolean Expressions	3)Evaluate Boolean Expressions	4)Output Truth Value Results
Users can input Boolean expressions along with truth values for variables using the command line interface.	Software parses the input from the users Boolean expressions, recognizing operators (&,\,!,@,$) also handles parenthesis for precedence.	Software evaluates the entire expression, considering the truth values that were provided by the user for each of the variables.	Software outputs the final truth value result of the evaluated Boolean expression for the user to view

These use cases illustrate the core functionality of the Boolean Logic Simulator, guiding the development efforts to ensure that the software effectively handles user inputs, parses expressions, evaluates them accurately, and provides the results as expected.

4.1 Use-Case Realizations

Expression	Scenario	Functionality	Contribution
1)User Inputs Boolean Expression	“A&B|!C”	Parse Expression and recognizes a, b, and c	Now prepares for the evaluation of A and B or Not C
2)Parsing Boolean Expression	((A&B)|(C&D))	Handles nested expression and prioritizes what’s inside parenthesis	Ensures correct order from parsing taking precedence rules into account
3)Evaluating Boolean Expression	Provided with truth values (e.g. A=True, B=False, C=True, D=False)	Computes final truth value of expression based on the values that were e given	Applies Boolean algebra rules to calculate truth value of expression
4)Outputting Result	Since Evaluation is complete the software outputs Boolean expression	Displays the final result (True or False) to user	Presents the output in a clear and understandable manner to user

This section provides detailed scenarios or realizations of how users interact with the Boolean Logic Simulator to achieve specific tasks. Each use-case realization demonstrates the actual functioning of the software and how different design elements contribute to its functionality. Here are a few selected use-case realizations:

These use-case realizations illustrate how the software processes user inputs, parses expressions, evaluates them, and presents the results, showcasing the functionality and effectiveness of the Boolean Logic Simulator in C++.

5. Logical View

The Logical View section of the Software Architecture Document for the Boolean Logic Simulator in C++ focuses on the system's structural design. It breaks down the design model into subsystems, packages, and classes, highlighting key components and their roles.

Overview: This part provides a high-level summary of the system's structure, showing how components are organized into subsystems and packages.

Architecturally Significant Packages: For each important package, details are provided on its composition, including the classes and utilities it contains. The responsibilities, relationships, operations, and attributes of key classes are explained to give a clear picture of how they contribute to the system's functionality.

By presenting this Logical View, the document simplifies the understanding of the system's design, outlining the essential components and their interactions for stakeholders, architects, and developers.

5.1 Overview

This software is organized into a layered architecture in order to facilitate separation based on purpose:

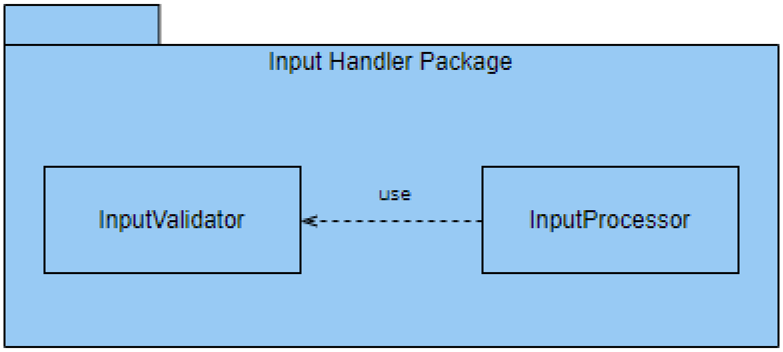
- **Input Layer:** Manages all user interactions, ensuring that all inputs are collected, validated, and prepared for processing.
- **Processing Layer:** Handles the core functionality of parsing and evaluating the inputted Boolean expressions.
- **Output Layer:** Responsible for formatting and delivering the evaluation results back to the user.

This layered approach promotes scalability and maintainability, allowing for individual aspects of the application to be updated independently.

5.2 Architecturally Significant Design Modules or Packages

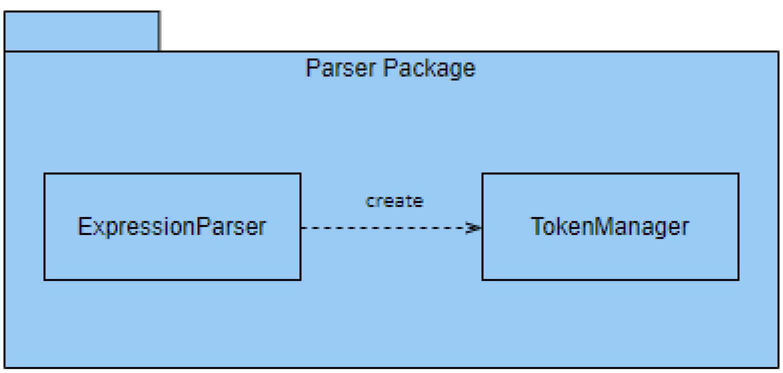
Input Handler Package:

- **Description:** Manages the reception and initial processing of user-inputted logic expressions via command line.
- **Key Classes:**
  - **InputValidator:** Validates the syntax and semantics of user-inputted expressions. Ensures input contains only valid operators/characters and that parentheses are structured correctly.
  - **InputProcessor:** Transforms the raw input data into a structured format for further processing.



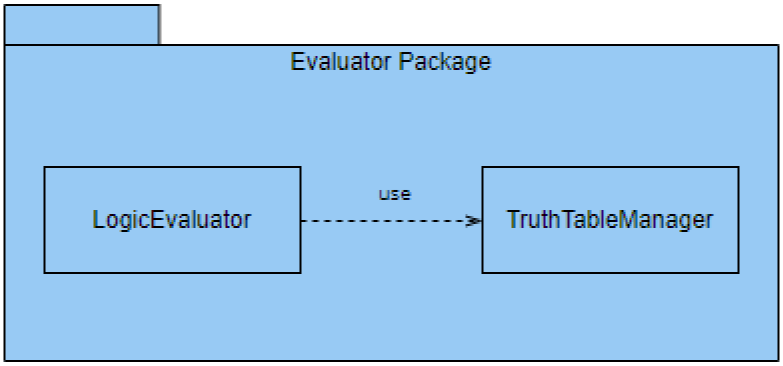
Parser Package:

- **Description:** Responsible for parsing validated logical expression input into a format that can be evaluated by the logic processor.
- **Key Classes:**
  - **ExpressionParser:** Parses the input strings to validate the syntax and semantics of user-inputted expressions. Checks for input errors.
  - **TokenManager:** Manages tokens and assists in data transfer to the evaluation phase. Ensures the order of operations according to parentheses and logical rules are preserved.



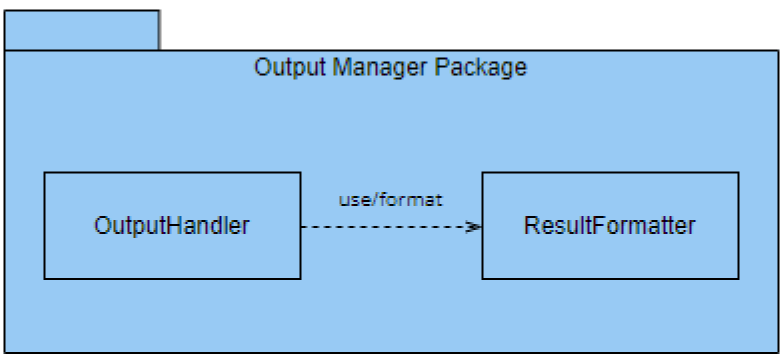
Evaluator Package:

- **Description:** Contains the computational logic for evaluating Boolean expressions based on the parsed input.
- **Key Classes:**
  - **LogicEvaluator:** Computes the Boolean result using a stack-based evaluation algorithm to handle precedence and ensure correct logical outcomes.
  - **TruthTableManager:** Manages a truth table when complex expressions with multiple variables are evaluated, supporting detailed traceability of logic evaluation.



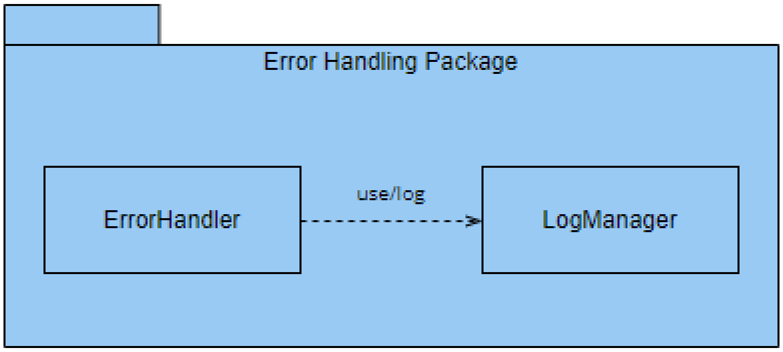
Output Manager Package:

- **Description:** Formats and displays the results of the Boolean evaluations to the user.
- **Key Classes:**
  - **ResultFormatter:** Formats the evaluation result in a user-friendly manner, providing clarity and immediate feedback on the validity or result of the Boolean expression.
  - **OutputHandler:** Manages the console output, ensuring that results are displayed promptly and clearly.



Error Handling Package:

- **Description:** Responds to raised errors throughout the input, parsing, and evaluation stages.
- **Key Classes:**
  - **ErrorHandler:** Specializes in capturing and responding to errors throughout the input, parsing, and evaluation stages.
  - **LogManager:** Optionally logs errors and user inputs for debugging and maintenance purposes.



6. Interface Description

This project utilizes a simple command-line interface:

Input Interface: Users type Boolean expressions directly through the command line. The input is validated for syntax and structural correctness before being processed.

Output Interface: The results, either the resulting evaluation or error messages, are displayed on the command line. The software then evaluates the inputted expressions and outputs the result as either ‘True’ or ‘False’. The output is clear and formatted to be easily understood.

This format keeps the user interface simple and focused, aligning with the project’s focus on efficiency and usability.

7. Size and Performance

Size: This project is designed to be compact and requires minimal resources.

Performance: This project is optimized for fast response times. This ensures a responsive experience for users interacting through the command-line interface.

8. Quality

Extensibility: The system is built to easily incorporate additional features, such as new types of logical evaluations or enhanced parsing capabilities, without significant restructuring.

Reliability: Error handling is robust, with clear feedback provided to users on syntax and processing errors. This ensures that the system behaves reliably under a variety of input conditions.

Usability: The command-line interface is designed to be user-friendly, with clear instructions and error messages that make the system accessible to anyone.

Maintainability: The modular design allows for easy updates and maintenance. Changes in one module, such as adding new logical operators or enhancing the parser, can be made independently of others.

Portability: As the program is developed in C++, the application can be compiled and run on all major operating systems.

Performance: The system meets defined performance criteria, ensuring quick response times and handling of expressions efficiently to maintain user satisfaction.