MacroHard

Boolean Expression Evaluator
Software Architecture Document

Version <1.0>

Revision History

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

Table of Contents

1. Introduction 4
   - 1.1 Purpose 4
   - 1.2 Scope 4
   - 1.3 Definitions, Acronyms, and Abbreviations 4
   - 1.4 References 4
   - 1.5 Overview 4
2. Architectural Representation 4
3. Architectural Goals and Constraints 4
4. Use-Case View 4
   - 4.1 Use-Case Realizations 5
5. Logical View 5
   - 5.1 Overview 5
   - 5.2 Architecturally Significant Design Packages 5
6. Interface Description 5
7. Size and Performance 5
8. Quality 5

Software Architecture Document

1. Introduction

This Software Architecture Document (SAD) serves to detail the architectural framework of the Boolean Expression Evaluator system. It delineates the system's design using different architectural views to convey the key architectural decisions that have been made. This document is fundamental to ensuring a cohesive understanding of the system's architecture among all project stakeholders. The general purpose of this document is to guide the development process and support the system's scalability and maintenance.

1.1 Purpose

This SAD provides an architectural overview of the Boolean Expression Evaluator, illustrating the system's structure through different architectural perspectives. It is with this document that we hope to illustrate the significant architectural decisions made during the system's design and development. The document serves multiple audiences, including the current project team and the project's stakeholders, granting them insights into the system's architectural rationale and structure. It is intended for use as a reference for ongoing development efforts, and for informing decision-making processes related to system construction.

1.2 Scope

The scope of this SAD encompasses the architectural design and structure of the Boolean Expression Evaluator system. It details the system components, their interactions, and the architectural decisions that shape the system's ultimate development and construction.

1.3 Definitions, Acronyms, and Abbreviations

SAD: Software Architecture Document

KU: University of Kansas

EECS: Electrical Engineering and Computer Science

UML: Unified Modeling Language

PEMDAS: Parentheses, Exponents, Multiplication and Division, Addition and Subtraction

1.4 References

IEEE830-1998: IEEE Recommended Practice for Software Requirements Specifications

C++ Standard Library documentation

Project Plan document

Software Requirements Specification (SRS) document

1.5 Overview

Following this introduction, the document is organized into several sections, each addressing different aspects of the software architecture:

- Architectural Representation: Describes what architecture was chosen for the system and how it is represented.
- Architectural Goals and Constraints: Outlines the factors that influence the architecture of this system. This includes any requirements, objectives, and constraints.
- Logical View: Details the design model's decomposition into subsystems, packages, and classes.
- Interface Description: Describes the system's interfaces, including user interaction mechanisms.
- Quality: Explains how the architecture supports system qualities such as extensibility, reliability, and portability.

2. Architectural Representation

Architecture: Main program and subroutines

- There will be a main program to get input from the user until the user quits
- When the main program receives input it will pass it to the first component
- The first component will return the data it receives after passing the input it received from the main program to another component. This continues until the final component is reached

Components:

- user interface (command line) - formatted output and instructions for use
- input module - get input
- tokenizer - break input into operands, operators and parentheses
- parser - apply order of operations and handle parentheses and sub expressions
- evaluation module - return the truth value of a boolean expression

3. Architectural Goals and Constraints

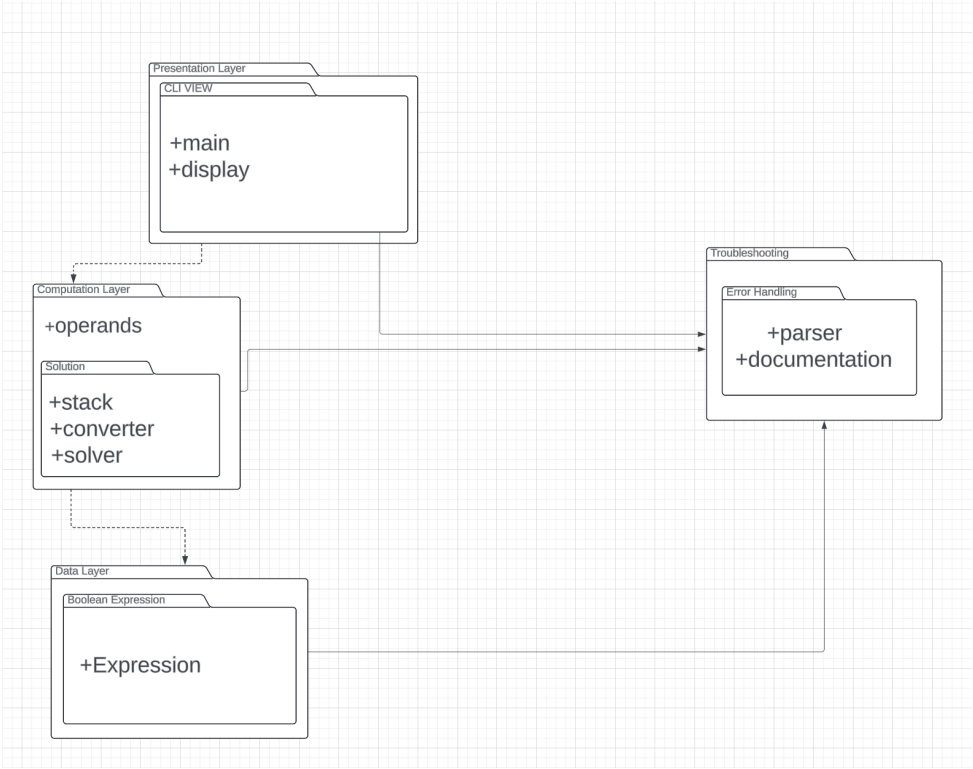
At this moment there are no safety, security, or privacy concerns/constraints for our end goal. As for portability, it will require either a laptop or desktop with the ability to run C++, distribution will be through GitHub via downloads. As for constraints brought on via our design strategy, development tools, and or team structure, the only constraint to be considered is that we must use C++ as it is required for this project.

4. Logical View

4.1 Overview

There are 4 main layers in the model. First is the presentation layer which handles UI and displaying/inputs. The second is the Computational Layer which handles the computing of the boolean expression into a True of False value after the parsing is done. The third layer is the Troubleshooting layer which contains the documentation on how to use the product and the parser which converts the boolean expression into something the code can read. This will check for errors before computation and pushing onto stack. The fourth layer is the Data layer which holds the boolean expression and properties along with it.

4.2 Architecturally Significant Design Modules or Packages



Presentation Layer:

CLI VIEW: The first class is main which runs the entire program. The best place to put it is the Presentation layer because it is the top layer in the scope of the model. The other class is display which takes input of a boolean expression and outputs the steps and final answer.

Computation Layer:

This layer also contains the properties of each operand and the effect it will have on the equation during the conversion. This is just a set of operands and their conversions to the usable symbols in c++.

SOLUTION: There are 3 classes in the solution layer. THe first is the stack class. This is a traditional stack data structure with a push and pop. This will store the order of operations, and the order of which parenthesis affect this order. The second class is the converter which takes a parsed equation and converts it into data structure and boolean algebra that can be computed in c++. The third class is the solver. This class will solve the boolean algebra equation using the stack and converted equation.

Troubleshooting Layer:

ERROR HANDLING: The first class is the parser which takes in a boolean expression and checks for errors such as missing parentheses, unknown operators, etc. The second class is documentation which contains strings of all the information needed to learn and troubleshoot using the CLI and product.

Data Layer:

BOOLEAN EXPRESSION: There is only one class in this package and it is the boolean expression. This would contain the expression, and the properties along with it, such as a parsed version of the equation, converted equation, and the stack of operations. This class interacts with the other classes in the Computation layer and stores all the results of each class and package within itself as an array or string.

5. Interface Description

Input will be taken through the command line, valid input will be checked in the parser.

Examples of Valid Expressions:

Expression: (T | F) $ F

Evaluation: True

Expression: ! (T & T)

Evaluation: False

Expression: (F @ T) | (T @ F)

Evaluation: False

Expression: (T $ T) & F

Evaluation: False

Expression: ! F | ! T

Evaluation: True

Expression: (((((T | F) & F) | (T & (T | F))) @ (T @ T)) $ (! (T | F)))

Evaluation: True

Expression: ((F $ ((T | F) & (F @ (T | F)))) | (T $ (T & F)))

Evaluation: False

Expression: (((! (T $ F)) & (T @ T)) | ((F | T) & (T $ T)))

Evaluation: False

Expression: (((T @ T) $ (F @ T)) | ((!T) & (T | (!T))))

Evaluation: True

Expression: ((F @ T) $ (T | (F & F))) & (T & (T @ (!T)))

Evaluation: False

Examples of Invalid Expressions:

1. Missing operand: ! & T [Reason: Missing operand after NOT]
2. Unknown operator: T ? T [Reason: Unrecognized operator symbol]
3. Mismatched parentheses: (T |) False [Reason: Missing closing parenthesis]
4. Circular logic: T = !(T & T) [Reason: Variable defined in terms of itself]
5. Empty expression: [Reason: No operands or operators present]
6. Double operator: T && & F [Reason: Two consecutive AND operators]
7. Missing truth values: X | Y [Reason: Variables without assigned truth values]
8. Inconsistent characters: True | F [Reason: Using both "T" and "F" for variables]
9. Operator after operand: True! [Reason: NOT applied after a value, not before]
10. Invalid characters: a & b [Using lowercase letters instead of "T" and "F"]

6. Quality

1. The software will be Windows 10/11 and Ubuntu compatible.
2. The user interface will be easy to use by referencing the README file for formatting.
3. Each feature will be discussed in the README file.
4. The interface will protect the user from errors by identifying mistakes and informing the user of the mistake or correcting the input if possible.
5. Incorrect solutions will be fixed as bugs are tested and output will always be correct.
6. The software will operate in a timely manner with efficient calculations.