RAND Corporated

GUI Boolean Calculator
Software Architecture Document

Version 3.0

Revision History

Date	Version	Description	Author
09/04/2024	1.0	beginning draft	Jahnvi Maddila
13/04/2024	2.0	Explaining interface description.	Jay Patel, Vamsi Doddapaneni
14/04/2024	3.0	Finishing Touches	Jahnvi Maddila, Tej Gumaste

Table of Contents

Software Architecture Document

1. Introduction

This document sheds light on the purpose, scope, definitions, acronyms, abbreviations, references, and overview of the Software Architecture. It describes the overall purpose of the software architecture, its scope, necessary definitions, and abbreviations to convey the overlying concept with the architecture.

1.1 Purpose

This document provides a comprehensive architectural overview of the system, using several different architectural views to depict different aspects of the system. It is intended to capture and convey the significant architectural decisions that have been made on the system. This document will come in handy to anyone who wants to know about our software architecture, its components, and the assembly of those components.

1.2 Scope

This document has its scope limited to decisions taken to design and the overall final structure of the software architecture for our Boolean Calculator project. All the explanations and the terms defined within this document have been designed keeping our Boolean Calculator software at the center.

1.3 Definitions, Acronyms, and Abbreviations

Software Architecture: The set of principal design decisions made about a system to be developed.
Object-oriented Design: Software architectural design where core functionality is stored in separate modules and each part has one specific responsibility.
Stack: Stack is a linear data structure that follows a particular order in which the operations are performed.
Postfix Notation: A mathematical notation where every operator follows all of its operands, allowing it to be used without parentheses as the order of operations is completely determined by the position of the operator and not the rules of precedence.

1.4 References

We do not reference another document.

1.5 Overview

The document has five sub-sections namely architectural representation, Architectural goal and constraints, Logical view, interface description, and quality.

1. Architectural representation:

This section focuses on explaining the software architecture for the current system. It details the object-oriented classes used to break down expressions and calculate the desired results.

2. Architectural goals and constraints:

This section discusses the modules used in the system and the reasons behind their selection. It provides insights into the design decisions and constraints that influenced the architecture.

3. Logical view:

This section delves into the classes and packages used for this product. It talks about their usage and functionality.It is further divided into 2 subheadings.

***3.1**- Overview:*

It describes the overall surface interaction of each class with one another. Details about cohesion and coupling can be found in this section..

***3.2**-Architecturally significant design modules or packages:*

A more detailed overview of classes and packages that play a huge role in the product.

4. Interface Descriptions:

This section describes the interface the user will be working with while using our software.

5. Quality

This section briefly summarizes the safety precautions and reliability of the components used to design the product.

2. Architectural Representation

There will be three main classes:

BooleanCalculatorApp: Which serves as the base frame for the calculator with entry and exit functionality, and a file icon, and will be where the calculator GUI gets initialized.

MainFrame: Contains the bulk of the UI for the calculator such as the input fields, buttons for boolean operators, and output display.

BooleanOperations: Contains the logic for the boolean calculator with expression parsing and expression validity.

3. Architectural Goals and Constraints

There are no obvious constraints on the architecture of the calculator. The only possible constraints are the limitations posed by WxWidgets (if any)

4. Logical View

1. Application Entry (App Subsystem)

This subsystem is responsible for initializing the application.

Classes: BooleanCalculatorApp: Inherits from wxApp. It's the application's starting point, responsible for setting up the main window.

2. Main Window (UI Subsystem)

This subsystem handles the main user interface elements.

Classes: MainFrame: Inherits from wxFrame. This class is the main window of the application, containing all the UI components like buttons, text controls, and menus.

3. ***Boolean Logic (Logic Subsystem)***

This subsystem is responsible for the core logic for Boolean operations.

BooleanOperations: This class contains static methods for performing Boolean operations. The UI subsystem utilizes these methods to process user inputs and produce outputs.

4.1 Overview

App Layer: Contains the application's entry point and setup logic.

Classes: BooleanCalculatorApp

Main Window Layer: Manages the main window and its components like menus and status bars.

Classes: MainFrame

Controls Layer: Includes all the GUI controls such as buttons and text controls that are used within the main window.

Classes: Managed by MainFrame but includes individual control classes like wxButton, wxTextCtrl, and potentially wxMenu.

Boolean Operations Layer: Contains all the logic needed to execute Boolean operations. These are utility functions that are stateless and designed to be called statically.

Classes: BooleanOperations

4.2 Architecturally Significant Design Modules or Packages

This will be the built-in C++ module containing the BooleanCalculator class.

BooleanCalculator Class: This is our core calculator functionality dedicated to Boolean logic operations. The primary methods in this class are PerformLogicalOperation() and Evaluate() which can be utilized in a main file to evaluate Boolean expressions and interact with the Stack.

Stack Class: This is a specialized Stack Class that leverages the C++ standard library to build the stack data structure, featuring push/pop methods crucial for managing the order and precedence of logical operations.

5. Interface Description

Screen Formats: On the user’s first glance at the page, they will see a welcome statement saying, “Welcome to the Boolean Calculator”. At the mid-section of the web page, users will have a rectangular input box that holds, “Enter your expression here!”, where they will present their boolean expression. Underneath the input box, there will be a set of buttons corresponding to some of the inputs like AND, OR, NAND, and XOR which will reduce the user's workload of typing it. Alongside these buttons, there will be an ‘Enter’ button. The result for the expressions will be presented below the initial screen that was shown. The user will have to scroll down the page to see their results. The prototype will have a simple linear to gradient background in color black. The input boxes will hold their original white color.

Valid Inputs: The expected input would have a basic boolean expression with a choice of operators. The user will have a choice to use parentheses to part the expression in a kind that the user would prefer. If the error holds an incomplete expression or any alien operator, an error message will be displayed on the top of the box.

Resulting Outputs: The essential result output consists only of True(T) or False(F) as per the expression given. The result will be shown after the initial screen, which is the input box and buttons. It will showcase the given expression and will have the answer to that expression displayed below.

6. Quality

The architecture effectively manages logical operator precedence and the proper handling of parentheses, ensuring precise evaluation of Boolean expressions. By compartmentalizing functionality within the BooleanCalculator class, the system facilitates the systematic processing of complex logical expressions, adhering strictly to the rules governing Boolean operations. This separation enhances the portability of the module, enabling the potential reuse of the BooleanCalculator class in other projects by simply importing it.

The utilization of well-established stack data structures within the BooleanCalculator Class significantly reduces the potential for errors in expression parsing and evaluation. This strategic choice contributes to the overall reliability and robustness of the system, ensuring consistent and error-free logical calculations.