Huang and Co.

Boolean Expression Evaluator Software Architecture Document

Version 1.0

Revision History

Date	Version	Description	Author
06/04/2024	1.0	Split tasks	Andrew Huang, Elizabeth Miller, Hunter Long, Katherine Swann, Keaton Xu
06/04/2014	1.1	Completed tasks and reviewed	Andrew Huang, Elizabeth Miller, Hunter Long, Katherine Swann, Keaton Xu

Table of Contents

Software Architecture Document

1. Introduction

This Software Architecture Document specifies how the Boolean expression evaluator (B.E.E.) will be implemented by providing an overview on the architecture of the system, i.e. the design of the software.

1.1 Purpose

The purpose of this document is to specify how the requirements for the B.E.E. will be implemented; i.e., the comprehensive architectural overview of the system. It will provide an overview of the structure of the software and any constraints in this implementation.

1.2 Scope

This document outlines all the components, interfaces, and interactions between modules used to implement the B.E.E. Details of the program code (such as classes and methods) are described.

1.3 Definitions, Acronyms, and Abbreviations

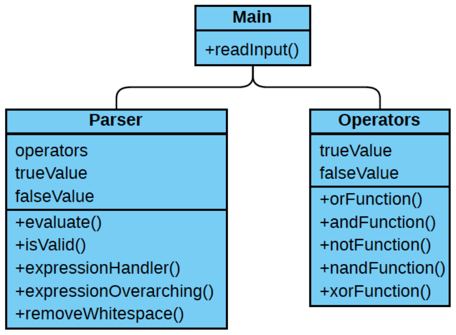
- **Operators:** logical functions through which two Booleans are evaluated (e.g. AND, OR, NOT, etc.)
- **Parsing:** process by which user input is dissected and evaluated.
- **Tokenization:** process by which an input is prepared for a module, with the goal being to remove extraneous characters such as spaces and punctuation

1.4 References

N/A

1.5 Overview

Class diagram:



- **Section 2:** Enumerates elements of software architecture
- **Section 3:** Describes various constraints on the architecture, such as security and portability and how it affects the development of the architecture
- **Section 4:** Lists use cases that significantly affect the system architecture
    - Section 4.1: Gives description of how software elements function in specific use cases
- **Section 5:** Provides breakdown of architectural elements into subsystems, packages, and classes.
    - Section 5.1: Provides overall decomposition of system into packages/layers

  - Section 5.2: Provides description and diagram of each package, as well as name and description of each significant class in the package
- **Section 6:** Describes interfaces through which user interacts with program
- **Section 7:** Describes constraints for dimensions and performance of program, and how this affects its architecture
- **Section 8:** Describes how architecture contributes to non-functional characteristics of program

2. Architectural Representation

The architecture in the system is represented in a class diagram, and the interaction between these parts through methods.

- **User view:** Sees the user interface side of the program. Has capabilities to have their input data interpreted.
- **System view:** Takes in the client's input from the user interface, and then uses a tokenizer, parser, evaluation algorithm, and output formatting to return a comprehensible output to the user.

client → user interface → tokenizer → parser → evaluation algorithm → output formatting → client

Output data of specific functions being sent to separate classes is shown through arrows above.

3. Architectural Goals and Constraints

Software requirements:

- Ability to run/compile C++ programs on the machine. (i.e. GNU g++ 11.4)
- User interface interacting with modules of different functionality suggests that object oriented programming is crucial from the architecture (classes are required, and the design should involve modeling them).

Objectives:

- I/O Parsing - Module that uses C++ input and output streams to take user input into the program. Converts the raw string data into syntax that the program can use.
- Evaluation - Module to take the input that was parsed through the parsing module, and have a favorable answer.
- Output - Module that handles how the system formats and outputs the evaluation obtained from the evaluation module.

4. Use-Case View

N/A

4.1 Use-Case Realizations

N/A

5. Logical View

5.1 Overview

This project's design model is relatively simplistic and can be generally considered as a singular package with a class(es) containing the key features described in the project description document. The key features of this project will include operator support for implementing logical operations, expression parsing, truth value input, evaluation and output, and error/parenthesis handling. In order to implement these key features, this project will require an input module, a tokenizer module, a parser module, an error/parenthesis handler module, an evaluation module, and a user interface module that will allow for a friendly interface to provide expressions and view results. All of these tools for the project design will include error handling and parenthesis handling as necessary within their function body.

5.2 Architecturally Significant Design Modules or Packages

5.2.1 Boolean Expression Evaluator

Class name: Main

Description: Runs the B.E.E. program and begins by asking a user for an input

Class name: Parser

Description: Cleans inputs given to Main by removing any extra spaces, while also evaluating if the input is valid. If valid, run through an evaluate function. The expressionsHandler will run until a singular value True denoted by T or False denoted by F is the final output. This is controlled by another function called expressionOverarching.

Class name: Operators

Description: contains the functions for the OR, AND, NOT, NAND, and XOR operators.

6. Interface Description

The program will utilize the built-in terminal interface of the given operating system or integrated development environment. As such, inputs will be entered via the command line, with the resulting output also being displayed in the terminal. Statements directing users how to input data will be used to facilitate a straightforward and user-friendly interface. Valid inputs must follow given syntax and parentheses requirements such that it can be read by the Main class.

7. Size and Performance

N/A

8. Quality

The software architecture supports the quality requirements:

- The B.E.E. will be Linux compliant and thus can be deployed onto any EECS cycle server.
- The user interface of the B.E.E. will be designed to be appropriate for a computer-literate user community.
- The program will handle possible errors and output relevant and descriptive error messages.

- The program will be well-documented and formatted to consider future maintenance or addition of new features.