EECS348: Term Project in C++

Project Title: Boolean Logic Simulator in C++ (Simplified as Boolean Expression Evaluator)

Spring 2024

Project Objective

This project delves into the world of **digital logic**. You will develop a C++ program acting as a simplified **Boolean logic simulator**. The aim of this project is to develop a program that simulates the behavior of logic circuits, including operations such as AND, OR, NOT, NAND, and XOR. The program should be able to handle complex logic circuits with multiple gates and input/output signals. The project will provide students with comprehensive hands-on experience in software engineering, emphasizing the development process from project planning to fully realized product. While aiming for the functionality of a full-fledged circuit simulator, we'll focus on evaluating **Boolean expressions** for an introductory learning experience.

This project provides an opportunity to explore the fascinating concepts of **logic gates**, **truth tables**, and **expression evaluation**. You'll gain valuable skills in parsing, data structures, algorithm design, and software engineering principles.

Key Features

1. **Operator Support:** Implement logical operations for:

    - 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` if one or both operands are `False` (opposite of AND)
    - XOR ($): Returns `True` if exactly one operand is `True`

2. **Expression Parsing:** Develop a mechanism to parse user-provided Boolean expressions in infix notation, respecting operator precedence and parentheses.
3. **Truth Value Input:** Allow users to define truth values (True/False) for each variable represented by T and F.
4. **Evaluation and Output:** Calculate the final truth value of the entire expression and present it clearly (True or False).
5. **Error Handling:** Implement robust error handling for invalid expressions, missing parentheses, unknown operators, or other potential issues, and provide informative error messages.
6. **Parenthesis Handling**: Ensure that your program can handle expressions enclosed within parentheses (including seemingly excessive but correctly included pairs of parentheses) to determine the order of evaluation.

Project Guidelines

- Use object-oriented programming principles to structure your code using C++.
- Include comments and documentation to explain the logic and functionality of your program.
- Develop unit tests to verify the correctness of different operator evaluations and complex expressions.
- Ensure that your program provides clear and informative error messages for invalid input or situations.
- Design a user-friendly interface (text-based or graphical) for interacting with the Boolean expression evaluator.

Deliverables

1. The common software engineering artifacts, such as a project management plan, a requirements document, a design document, and test cases. These will be described in separate announcements.
2. A well-documented C++ program that functions as a Boolean expression evaluator with the specified features.
3. A user manual or README file explaining how to use your program, including examples of valid expressions and their expected outputs.

Grading Criteria

Your final product will be evaluated based on the following criteria (a total of 120 points):

- Correctness of expression evaluation (handling of operators, precedence, parentheses) [60 points]
- Robustness and error handling (handling invalid input, syntax errors, etc.) [20 points]
- Code quality, including structure, readability, and comments [20 points]
- Documentation and user manual quality (clarity, comprehensiveness, ease of use) [20 points]

Note: Feel free to explore additional features or optimizations beyond the specified requirements to enhance your project and demonstrate your creativity and engineering skills.

Examples of Valid Expressions

Expression: (T | F) $ F
Evaluation: True

Expression: ! (T & T)
Evaluation: False

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

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"]