Boolean Logic Legends

Boolean Logic Simulator in C++

User’s Manual

Version 1.0

Revision History

Date	Version	Description	Author
02/May/2024	1.0	Completed User Manual	Entire Group

Table of Contents

User Manual

1. Purpose

The user manual for the Boolean Logic Simulator serves as a guide on installing, running, and using the program. The user manual contains an introduction to the program, how to get started, advanced features, troubleshooting, examples, a glossary, and frequently asked questions.

User Manual Overview:

- **Introduction**: A brief overview of the software, including its purpose, key features, and how to install and run it
- **Getting started**: Instructions on using the program, including entering expressions, utilizing operators and functions, and interpreting results
- **Advanced features**: Lists and describes any advanced features of the software
- **Troubleshooting**: Common issues users may encounter and how to resolve them
- **Examples**: Lists valid inputs and their evaluations, and invalid inputs and their errors
- **Glossary of Terms**: Defines all technical terms used in the user manual
- **FAQ**: Answers to frequently asked questions about the software

2. Introduction

The Boolean Logic Simulator simulates the behavior of logical circuits. It receives a boolean expression from the user, calculates it, then returns true or false. The program supports logical operators, expression parsing, expression evaluation, parenthesis handling, and error handling.

There are two methods to install and run the program:
Method 1: EXE Download (Preferred):

1. Download 'BooleanLogicSimulator.exe' from GitHub repository into a designated folder
2. Run the program by opening the file or opening a terminal in the folder and running './BooleanLogicSimulator.exe'

Method 2: Manual Compilation:
Installation:

1. Download all program files and place them into a designated folder
2. Install a C++ compiler onto your system

Compilation:

1. Open Terminal
2. Navigate to the program folder using the 'cd' command
3. Execute the command: 'g++ -o (or the corresponding command for your compiler) BooleanLogicSimulator interface.cpp'

Running the Program:

1. Double-click 'BooleanLogicSimulator.exe' located in the program folder
   or enter './BooleanLogicSimulator.exe' into your terminal

This method allows for more control over the files if desired, allowing for additional functionality to be manually added by modifying the program files before compiling. However, for simply running the program as-is, method 1 should be preferred.

3. Getting started

If you choose to compile the code yourself, a C++ compiler like GCC or Clang will be needed. Additionally, all three of the program files: interface.cpp, isValidExpression.h, and evaluator.h must be downloaded into a singular folder. Whether through your machine's terminal or a C++ IDE, navigate to the folder that holds those three files. Compile the interface.cpp program and run the executable code. Alternatively, the executable code is available in the Github project folder. You can execute that file in your device's terminal without compilation if desired. You are now ready to start using the program.

When starting the program, you'll see a welcome message and a small amount of information regarding the program. You can immediately begin entering boolean expressions to evaluate. You can also exit the program at any time by entering "exit" into the terminal. The program accepts operands: 'T' and 'F'. The T and F characters represent true or false boolean values. The program also accepts operators: '&' (AND), '|' (OR), '!', (NOT), '@' (NAND), '$' (XOR). The parentheses characters: '(', and ')' are also accepted inputs. Any other inputs into the program will result in an error message.

All of the operators other than '! (NOT) are binary and require two operands on either side of the operator. '!' (NOT) is unary, meaning that it only requires one operand on the right side to work. Using parentheses in your expression is not required, but is recommended. The program has a built in precedence for the operators, which goes NOT => NAND => AND => OR => XOR. However, for increased brevity and easier readability parentheses usage is encouraged. Parentheses must be paired. If an unmatched parentheses is detected an error message will be thrown.

When a valid boolean expression is given to the program, the expression will be evaluated and the result will be given to you as "True" or "False" in the terminal. You can immediately evaluate another boolean expression or exit the program with the "exit" command.

4. Advanced features

No advanced features are present in the current iteration of the Boolean Logic Simulator.

5. Troubleshooting

Virtually every possible input should provide either an output or an error message. There are two possible issues that could be encountered: Errant truth value outputs, incorrect error messages.

5.1 Errant Truth Values

While all inputs should result in a correct output, it is possible that an expression may not give the intended output due to factors such as operator precedence. If an expression is giving an unexpected value, try the following:

1. Add parenthesis
    a. If not done already, try adding parentheses around each sub-expression in the logic equation. This allows the user to have complete control over the operator order, which should improve the output.
2. Remove extraneous parentheses
    a. While this should not affect the output, having extraneous parentheses can cause an error in that it can be harder to read, which could then result in parenthesis grouping the wrong segments together. As such, it is smart to remove any extraneous parentheses and double-check that the input is exactly what is wanted.
3. Write it out by hand
    a. If nothing else seems to be working, it is always worth double-checking the expression itself. The simulator should always be providing the correct output for whatever the expression is as it reads

it, so if parenthesis and operator order have been made clear then there should be no issues with the simulator. As such, if the output is still unexpected, double-check what the output of the expression should be.

5.2 Errant Error Messages

If you are facing repeated error messages or error messages that seem not to make sense, there are several ways to determine what the actual error is. To begin with, check that all characters are valid per the simulator's requirements and that parentheses are balanced. Then, check that all operators and operands are in the correct order and not doubled up anywhere. Lastly, try breaking apart the expression. As shown in section 6.2.1, when multiple errors are present only one error message will appear, which can make fixing the expression take longer. To fix this, break the expression into smaller pieces and make sure each part is written correctly. Once you verify that each segment is error-free, carefully put them back to evaluate the expression as a whole.

6. Examples

6.1 Valid Input

The following lists several example inputs and their corresponding outputs:

- 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: True
- 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

6.2 Errant Inputs

Additionally, the simulator will provide error messages if the input is incorrect or formatted wrong:

- Missing operand: ! & T
  - Error: Missing operand after NOT.
- Unknown operator: T ? T
  - Error: Unrecognized symbol.
- Mismatched parentheses: (T |) False
  - Error: missing operand after operator.
- Empty expression:
  - Error: No operands or operators present.
- Double operator: T && & F
  - Error: Two consecutive operators.
- Missing truth values: X | Y
  - Error: Unrecognized symbol.
- Inconsistent characters: True | F

    - Error: Unrecognized symbol.
- Operator after operand: T!
    - Error: Expression ends with an operator.
- Invalid characters: a & b
    - Error: Unrecognized symbol.

6.2.1 Multiple errors

Please note, if multiple errors are present then the simulator will only list one of the errors. For instance:

- Many errors: T! && F) $ a
    - Error: Missing operand after NOT.

Additionally, as can be seen by comparing this error with the operator after the operand error above, the simulator may not always describe the error the same way, depending on if an error falls into multiple categories. So while T! is missing an operand after NOT, it also ends with an operator, so this error gets thrown first. Even so, the error messages should always be accurate even if they don't always appear exactly the same depending on the case.

7. Glossary of terms

AND - A logical operation that returns true if both inputs are true, otherwise false.

Binary - A numbering system using two symbols, typically 1 and 0. Base 2.

GCC - GNU Compiler Collection, a suite of programming tools including compilers for notable languages like C++.

IDE - Integrated Development Environment, a software application allowing users to edit and run programming languages.

NAND - A logical operation that returns false if both inputs are true, otherwise true.

NOT - A unary logical operation that inverts the input, returning true if the input is false and false if the input is true.

Operand - A variable or value upon which an operation is performed in a mathematical or logical expression.

Operator - A symbol or function representing a mathematical or logical operation to be performed on one or more operands.

OR - A logical operation that returns true if at least one of the inputs is true, otherwise false.

Unary - A mathematical/logical operation involving only one operand. Example: NOT

XOR - A logical operation that returns true if exactly one of the inputs is true, otherwise false.

8. FAQ

1. What is a Boolean Logic Simulator?
    a. A Boolean Logic Simulator evaluates and simulates logic circuits, processing expressions with logical operators and returning the result.
2. How do I input an expression in the simulator?
    a. Input the expression as a boolean expression. For example, to simulate "T AND F OR T," write "(T & F) | T". Must use the defined truth values: T (true) and F (false).
3. What are the supported operators and their symbols?
    a. AND: &
    b. OR: |

  c. NOT: !
  d. NAND: @
  e. XOR: $

4. What do I do if I get an error message?
   a. Read the error message for insight into the potential issues with your expression. Adjust the input based on the guidance provided in the user manual.
5. Can the simulator evaluate complex logic expressions?
   a. Yes, the simulator is capable of evaluating complex logic expressions.
6. Are there any limitations to expressions?
   a. Yes, the expressions must be syntactically correct and use only the logic operators that are supported by the simulator.