




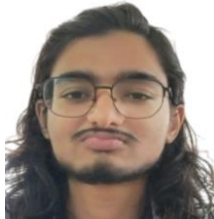


Graphic Era
Hill University
 DEHRADUN • BHIMTAL • HALDWANI

PROJECT AND TEAM INFORMATION

Project Title

Logic Expression Evaluator & Truth Table Generator

Student / Team Information

Team Name:	Logic Masters
Team member 1 (Team Lead) Name: Lakshya Singhal University Roll no.: 2418617 Student ID: 24011865 Email: lsinghal.graphichill@gmail.com Course: BTECH COMPUTER SCIENCE ENGINEERING (CSE)	
Team member 2 Name: Rakshit Tyagi University Roll no.: 2419499 Student ID: 24011760 Email: rakshitt680@gmail.com Course: BTECH COMPUTER SCIENCE ENGINEERING (CSE)	
Team member 3 Name: Antriksh Rana University Roll no.: 2418308 Student ID: 240111774 Email: antrikshrana187@gmail.com Course: BTECH COMPUTER SCIENCE ENGINEERING (CSE)	
Team member 4 Name: Kanika Rawat University Roll no.: 2418572 Student ID: 240121635 Email: kanikarawat.ddn@gmail.com Course: BTECH COMPUTER SCIENCE ENGINEERING (CSE)	

PROJECT PROGRESS DESCRIPTION

Project Abstract

This project aims to design and implement a robust Logic Expression Evaluator. The primary goal is to parse, validate, compute the truth table, minimise the Boolean expressions and implement a circuit diagram for the expression provided by a user. The core of this project relies heavily on data structures. We will implement algorithms to convert infix logical expressions (e.g., 'A AND (B OR NOT C)') into a postfix (Reverse Polish Notation) format, which is more suitable for machine evaluation. This conversion will be managed using the a **stack** to handle operator precedence and parentheses. Following conversion, a second **stack-based algorithm** will be employed to evaluate the postfix expression, efficiently determining the final true/false result. The project will support standard logical operators (AND, OR, NOT) and handle grouping with parentheses. The final deliverable will be a full stack application where a user can input a logical expression and variable assignments to receive its evaluated truth table, minimised expression and a circuit diagram as output.

Updated Project Approach and Architecture

Our approach is a multi-phase process built on a full-stack, hybrid architecture.

1. **Parsing and Evaluation Core:** The core logic is a standalone **C executable (bool_eval.exe)**. This program, built from `boolean_parser.c`, `truth_table.c`, and `stack.c`, is responsible for the high-performance computation: parsing the infix expression, converting it to postfix, generating the truth table, and (eventually) minimization.
2. **Web API Wrapper:** A **Node.js server (server.js)** acts as the web server and REST API. It receives HTTP requests from the frontend.
3. **Process Communication:** When the Node.js server receives a request, it uses a child process (`child_process.spawn` or `exec`) to run the compiled `bool_eval.exe` program. It passes the user's expression to the C program's standard input (`stdin`).
4. **Data Exchange:** The C program prints its result (e.g., a JSON string of the truth table) to standard output (`stdout`). The Node.js server captures this output, parses it, and sends it back to the React frontend as a JSON API response.
5. **Frontend:** A **React** single-page application (`index.html` is the entry point) provides the user interface. It communicates exclusively with the Node.js API.

Tasks Completed

Task Completed	Team Member
<ul style="list-style-type: none"> Implemented core stack data structure (stack.c, stack.h). Set up the Node.js project (package.json) and the web server API wrapper (server.js). Developed initial module for expression tokenizing/parsing (boolean_parser.c). Developed initial module for truth table generation (truth_table.c). Created the main entry point for the C logic program (main.c). Successfully compiled C modules into a single executable (bool_eval.exe). Developed the front-end using HTML & CSS 	<ul style="list-style-type: none"> Antriksh Rana / Kanika Rawat Antriksh Rana Kanika Rawat Lakshya Singhal Lakshya Singhal Rakshit Tyagi Rakshit Tyagi

Challenges/Roadblocks

The challenge is the **inter-process communication (IPC)** between the Node.js server and the C executable.

We must design a robust and efficient communication channel. This includes:

1. **Data Serialization:** Defining a strict data format for stdin and stdout. The C program must output data (like the truth table) in a format that Node.js can easily parse, such as a JSON string.
2. **Error Handling:** If the C executable crashes (e.g., from a segmentation fault or bad input), the Node.js server must catch this crash gracefully and send a proper HTTP 500 (Internal Server Error) response to the user, rather than crashing itself.
3. **Security/Efficiency:** We must ensure that the way we call the C executable (using `child_process`) is secure and does not allow for command injection. We also need to manage the overhead of spawning a new process for every request.

Tasks Pending

Task Pending	Team Member (to complete the task)
<ul style="list-style-type: none"> Implement the Quine-McCluskey minimization algorithm in C. Implement the circuit diagram generation logic (outputting JSON) in C. Finalize the data contract (JSON format) between the C program and Node.js. Implement robust error handling in server.js for C process failures. Develop the frontend UI with React (input forms, output display areas). Implement the circuit diagram visualization component (React Flow) on the frontend. Integrate the React frontend with the Node.js API. 	<ul style="list-style-type: none"> Lakshya Singhal Lakshya Singhal Antriksh Rana Kanika Rawat Rakshit Tyagi Rakshit Tyagi Antriksh Rana

Project Outcome/Deliverables

The project's primary **outcome** is to provide users (students, engineers, hobbyists) with a powerful and accessible web-based tool to instantly analyze, minimize, and visualize complex Boolean logic.

The key **deliverables** required to achieve this outcome are:

- **1. High-Performance C Logic Engine (`bool_eval.exe`):** A compiled executable program that serves as the backend's "brain." This program accepts an infix expression as input and outputs a structured JSON object containing:
 - The generated Truth Table.
 - The minimized Sum-of-Products (SOP) expression (from Quine-McCluskey).
 - A node-and-edge graph representation of the logic circuit.
- **2. Node.js REST API (`server.js`):** A lightweight web server that acts as the "wrapper" and mediator. It exposes simple HTTP endpoints, receives requests from the frontend, calls the C executable, and securely returns its JSON output.
- **3. React Frontend Application:** A dynamic, single-page application (SPA) that provides the complete user experience, including:
 - An input form for the Boolean expression.
 - Clear, formatted displays for the resulting truth table and minimized expression.
 - An interactive, graphical visualization of the logic circuit diagram (using a library like React Flow).
- **4. Complete Source Code Repository:** A Git repository containing all C source code (`.c`, `.h`), the Node.js server code (`server.js`), all React frontend code, and a Makefile for compiling the C program.

Progress Overview

The project is approximately **60% complete** and is currently **On Track**.

The foundational architecture, combining a C core with a Node.js API, has been successfully implemented and tested. The core C modules for stack operations, parsing, and initial truth table generation are built (`stack.c`, `boolean_parser.c`, `truth_table.c`). The Node.js server (`server.js`) is successfully communicating with the compiled C executable (`bool_eval.exe`) by passing input and capturing its output.

The current focus is on implementing the complex algorithms (Quine-McCluskey, circuit graph generation) in C and building the React frontend components to consume the API.

Codebase Information

Repository Link - <https://github.com/lakshya-singhal/Logic-Expression-Evaluator-and-Truth-Table-Generator.git>

Branch – Main branch

Deliverables Progress

• 1. C Logic Engine: In Progress

- `stack.c / stack.h`: **Completed**
- `boolean_parser.c`: **In Progress** (Shunting-yard conversion is complete; error handling is pending)
- `truth_table.c`: **In Progress** (Basic table generation is complete; integration with parser is pending)
- Minimization (Quine-McCluskey): **Pending**
- Circuit Graph Generation: **Pending**

• 2. Node.js REST API: In Progress

- `server.js` setup: **Completed**
- `child_process` communication with `bool_eval.exe`: **Completed**
- API endpoint for expression evaluation: **In Progress** (Basic functionality works; robust error handling is pending)

• 3. React Frontend Application: Pending

- Basic project setup (Create React App): **In Progress**
- UI components (Input form, tables): **In Progress**
- Circuit visualization (React Flow): **Pending**

• 4. Source Code Repository: In Progress

- Repository is set up and active.
- Makefile for C compilation is **Completed**.