Graphic Era deemed to be University
DEHRADUN

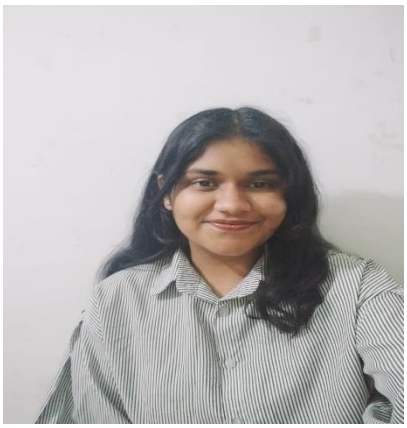
PROJECT AND TEAM INFORMATION

Project Title

A Mini Lexical Analyzer for C++ and General Code Analysis

Student/Team Information

Team Name:	CD-VI-T204
Team member 1 (Team Lead) Jha Ashutosh Munindra – 220211124 ajha11oct@gmail.com	

Team member 2 Negi Rahul – 220211268 negirahul4167@gmail.com	
Team member 3 Akula Sai Santhosh Kumar – 220212055 akulasanthosh56.ya@gmail.com	
Team member 4 Rastogi Parkhi – 220221920 parkhirastogi269@gmail.com	

PROJECT PROGRESS DESCRIPTION

Project Abstract

Developed in C++ and compiled to Web Assembly for browser execution, this project provides a web-based mini lexical analyzer for processing C++ and general code. Two modes are supported by the analyzer: general mode tokenizes all components, including comments, and C++ mode tokenizes conventional C++ code (e.g., int x=5; → int (keyword), x (identifier), = (operator), 5 (number), ; (punctuator)) while disregarding comments. using the use of a web interface made using HTML, Tailwind CSS, and JavaScript, users can enter code by typing it in or uploading.cpp files. Tokens and symbol tables are among the outputs that can be exported as a.txt file and are shown in the browser. The project functions as a teaching aid for compiler design, showcasing lexical analysis concepts through an intuitive user interface and strong functionality.

Updated Project Approach and Architecture

The application is a web-based utility that features a C++ backend compiled to Web Assembly using Emscripten and a frontend using HTML, CSS and JavaScript. The application can process C++ and general code in two modes:

C++ Mode: Tokenizes standard C++ code while ignoring comments, and generates tokens (keywords, identifiers, operators, numbers, punctuators).

General Mode: Tokenizes all code elements, including comments and give names. This mode offers users a broader tool for analysis.

Workflow: User interacts with a web UI to input code in one of two ways (typing in textarea or uploading a .cpp file). The WebAssembly backend reads in the input, tokenizes it according to the user's selected mode, and generates a list of tokens as well as symbol table. Everything is displayed in the browser and the users can store the tokenized input by download as a .txt file.

Architecture: The application uses a modular design, which concentrates on the client-server architecture. The C++ lexical analyzer is separate from the web UI. The lexical analyzer generates a lexer.js and lexer.wasm, which tokenizes the user-generated input. The frontend is styled to be interactive for input, visualizing the analysis, and exporting code for review. JavaScript acts as a bridge between the frontend and backend with the proper API calls to send the user input to the backend for processing.

Tasks Completed

Task Completed	Team Member
• Lexical Analyzer and Developing	Ashutosh Munindra Jha
• Lexical Analyzing Designing	Akula Sai Santhosh Kumar
• C++ Integration and API	Rahul Negi
• Frontend Developing	Parkhi Rastogi
• Testing, Final Optimization and Documentation	All Team Members

Challenges/Roadblocks

Managing Edge Cases: There were cases of malformed inputs, and various special characters that required enough logic for tokenization. We had built a state machine in the C++ backend to encapsulate token types that are multi-character long and handled special cases for those tokens to manage proper flow and accurately scan tokens through the lexer.
Input Buffering on Multi-Character Tokens: Parsing tokens that incorporated pre and post string values gave us trouble because we needed look-ahead to parse those. We altered the lexer to a sliding window approach and made the necessary checks to ensure we were recognizing the tokens correctly without blowing performance.
Error Detection and Reporting: Detecting and reporting invalid tokens and reporting on illegal characters took precision and planning regarding how we desired to handle errors. We added the right levels of error messages in the backend, so they were printable through the web UI and designed them to recover gracefully in case of malformed inputs.
Responsive Web UI Layout: Displaying the variable-length token outputs and symbol tables as a responsive web-based interactive UI went better than expected but was still not trivial. We used Tailwind CSS to help with adaptive layout management, and we implemented dynamic table rendering with .txt export capability for large number of token inputs.
WebAssembly: Compiling the C++ backend as WebAssembly by Emscripten gave us some hard time initially with file I/O and managing the integrity of the JavaScript backend frontend connections. After configuring the virtual filesystem in Emscripten, we then optimized the JavaScript glue code to ensure the framework would have a seamless response to interactions and communication globally across both ends.
Ultimately, we demonstrated that these best resolutions could yield a strong, user-friendly and robust.

Tasks Pending

Task Pending	Team Member (to complete the task)
• None (All tasks are complete, and the project is fully operational.)	None

Project Outcome/Deliverables

The project delivers a fully functional web-based mini lexical analyzer with the following:

- **C++ Source Code**: A lexical analyzer implemented in C++ (lexer.cpp), compiled to WebAssembly (lexer.js, lexer.wasm).
- **Web UI**: A responsive interface built with HTML, CSS and JavaScript, supporting typed input and .cpp file uploads.
- **Outputs**: Tokens and symbol tables displayed in the browser and exportable as .txt files.
- **Sample Inputs**: Example .cpp files (e.g., int x=5;) for testing.
- **Documentation**: Comprehensive guide covering installation, usage, and architecture, included in the repository.

The analyzer is an effective educational tool for compiler design, offering clear visualization and robust functionality for learning lexical analysis.

Progress Overview

The project is now 100% complete. The C++ backend, which was compiled to WebAssembly, is fully functional for both C++ and general mode tokenization; it identifies the code's inputs and outputs tokens and symbol tables. The web UI was styled using CSS, is fully responsive and welcoming user input, .cpp file uploads, .txt exports, etc. We thoroughly tested the program using edge cases, including malformed inputs and large files. Usage of WebAssembly allowed for performance optimization and the potential for increased scalability once the tool is proven iteratively. We provided documentation, as well as several sample inputs for users, and we hope it will be invaluable as a learning tool in compiler design courses. Our team worked well throughout the project and finished on time by working effectively as a team and approaching challenges in perfect manner.

Codebase Information

- **Repository:** https://github.com/assk2005/MiniLexcialAnalyzer_for_CPP
- **Branch:** main, code and testing
- **Key Commits:**
  - Implemented C++ lexical analysis module
  - Built general mode analysis
  - Developed web UI with CSS
  - Compiled to WebAssembly and integrated modules
  - Added error handling and edge case tests
  - Polished UI and implemented .txt export
  - Final optimization and documentation

Testing and Validation Status

Test Type	Status (Pass/Fail)	Notes
• C++ Mode Tokenization	Pass	Correctly tokenizes as a C++ compiler, ignoring comments
• General Mode Tokenization	Pass	Tokenize all elements, including comments
• Web UI Input (Typed/.cpp)	Pass	Handles both input types perfectly
• Output (Browser)	Pass	Displays tokens and symbol tables correctly
• Output (.txt Export)	Pass	Exports results accurately to .txt files
• Edge Case Handling	Pass	Handles malformed inputs and special characters robustly.

Deliverables Progress

All deliverables are complete:

- **C++ Lexical Analyzer**: Fully implemented in lexer.cpp, compiled to WebAssembly, supporting both modes.
- **Web UI**: Responsive interface with Tailwind CSS, supporting input, visualization, and .txt export.
- **Sample Inputs**: Example .cpp files provided for testing.
- **Documentation**: Comprehensive guide on installation, usage, and architecture included in the repository.

The project is fully operational, delivering a robust educational tool for lexical analysis.