Project CLI - Custom Language Interpreter

Submitted By

Student Name	Student ID
MD. Abdullah Khan	232-15-425
MD. Abdur Rahman Nayeem	232-15-143
Sazzad Islam	232-15-835
Saif Ahmed	232-15-636
Sajib Hasan	232-15-619

CUSTOM LANGUAGE INTERPRETER PROJECT REPORT

This Report Presented in Partial Fulfillment of the course **CSE314: Compiler Design Lab in the Computer Science and Engineering Department**

DAFFODIL INTERNATIONAL UNIVERSITY

Dhaka, Bangladesh

December 15, 2025

DECLARATION

We hereby declare that this lab project has been done by us under the supervision of **Compiler Design Lab, Tamanna Sultana, Lecturer**, Department of Computer Science and Engineering, Daffodil International University. We also declare that neither this project nor any part of this project has been submitted elsewhere as lab projects.

Submitted To:

____

Tamanna Sultana
Lecturer
Department of Computer Science and Engineering
Daffodil International University

Submitted by

______________________________ MD. Abdullah Khan 232-15-425 Dept. of CSE, DIU	
______________________________ MD. Abdur Rahman Nayeem 232-15-143 Dept. of CSE, DIU	______________________________ Sazzad Islam 232-15-835 Dept. of CSE, DIU
______________________________ Saif Ahmed 232-15-636 Dept. of CSE, DIU	______________________________ Sajib Hasan 232-15-619 Dept. of CSE, DIU

COURSE & PROGRAM OUTCOME

The following course have course outcomes as following:

Table 1: Course Outcome Statements

CO's	Statements
CO1	**Define** and **Relate** classes, objects, members of the class, and relationships among them needed for solving specific problems
CO2	**Formulate** knowledge of object-oriented programming and Java in problem solving
CO3	**Analyze** Unified Modeling Language (UML) models to **Present** a specific problem
CO4	**Develop** solutions for real-world complex problems **applying** OOP concepts while evaluating their effectiveness based on industry standards.

Table 2: Mapping of CO, PO, Blooms, KP and CEP

CO	PO	Blooms	KP	CEP
CO1	PO1	C1, C2	KP3	EP1, EP3
CO2	PO2	C2	KP3	EP1, EP3
CO3	PO3	C4, A1	KP3	EP1, EP2
CO4	PO3	C3, C6, A3, P3	KP4	EP1, EP3

The mapping justification of this table is provided in section **4.3.1**, **4.3.2** and **4.3.3**.

Table of Contents

Chapter 1

Introduction

This chapter introduces the background of the project, its motivation, objectives, feasibility, gap analysis, and expected outcomes. It sets the foundation for understanding the problem being addressed and the solution proposed.

1.1 Introduction

Compiler design and language processing are fundamental areas in computer science, yet they are often considered abstract and difficult for students to grasp. Traditional approaches rely on theoretical explanations and command-line tools, which can be intimidating for beginners.

This project aims to solve the problem of accessibility and interactivity in learning compiler design concepts. By integrating a Python Flask backend with parsing and lexical analysis tools generated using Flex and Bison, the project provides a web-based interface where users can experiment with tasks such as arithmetic evaluation, variable extraction, keyword identification, and error simulation. The problem addressed is the lack of a unified, user-friendly platform that demonstrates these concepts in an interactive and educational manner.

1.2 Motivation

The motivation behind this project is to make compiler design concepts more approachable and engaging. Students often struggle to connect theoretical knowledge with practical applications. By offering a web interface that allows direct interaction with parsing tools, the project bridges this gap.

From a computational perspective, the motivation lies in demonstrating how low-level parsing engines (Flex/Bison) can be integrated with modern web technologies (Flask, HTML, CSS, JavaScript). Solving this problem benefits learners by:

- Enhancing their understanding of lexical analysis and parsing.
- Providing hands-on experience with compiler construction tools.
- Encouraging experimentation and exploration in a safe, interactive environment.

1.3 Objectives

The specific objectives of the project are:

- To design and implement a web application that integrates Python Flask with Flex/Bison programs.
- To develop multiple parsers and lexers for tasks such as arithmetic calculation, boolean evaluation, variable extraction, and error simulation.
- To provide an interactive user interface for testing and visualizing parsing results.
- To evaluate the effectiveness of the system in improving students' comprehension of compiler design concepts.
- To create a reusable framework that can be extended with additional parsing tasks in the future.

1.4 Feasibility Study

Several research studies and projects have explored compiler visualization and educational tools. For example, Kleinberg and Tardos emphasize the importance of algorithm design in problem-solving, which aligns with the computational foundation of this project. Existing web applications and mobile apps focus on syntax highlighting or code execution but rarely integrate parsing engines directly.

Case studies in educational technology show that interactive tools significantly improve learning outcomes. Methodological contributions from projects such as online IDEs (e.g., repl.it, JDoodle) demonstrate the feasibility of integrating backend compilers with web interfaces. However, these platforms primarily target programming execution rather than parsing and lexical analysis.

Thus, the feasibility of this project is high, as it leverages proven technologies (Flask, Flex, Bison) and addresses a clear educational need.

1.5 Gap Analysis

The current gap lies in the absence of a modular, web-based platform that demonstrates compiler design concepts interactively. While standalone compilers and IDEs exist, they do not focus on parsing tasks such as variable extraction or semantic action simulation.

This project fills the gap by:

- Providing multiple specialized parsers under one interface.
- Allowing real-time interaction with parsing tools.
- Offering a structured framework for extending functionalities

1.6 Project Outcome

The possible outcomes of the project include:

- A functional web application that integrates Flask with Flex/Bison programs.
- Improved student engagement and understanding of compiler design concepts.
- A reusable educational tool that can be extended with new parsers and lexers.
- Demonstration of how traditional compiler tools can be modernized and integrated into web-based systems.
- Contribution to academic resources by providing a practical framework for teaching parsing and lexical analysis.

Chapter 2

Proposed Methodology/Architecture

This chapter outlines the requirements, design specifications, proposed methodology, system architecture, user interface design, and overall project plan. It explains how the project was structured and the approach taken to achieve the objectives.

2.1 Requirement Analysis & Design Specification

2.1.1 Overview

The project requires integration of multiple technologies:

- Python Flask for backend web server and orchestration.
- Flex and Bison for lexical analysis and parsing.
- GCC for compiling generated C code.
- HTML, CSS, JavaScript for frontend user interaction.

The system must support modular parsers, each demonstrating a specific compiler design concept.

Requirements include:

- A backend capable of executing compiled Flex/Bison programs.
- A frontend interface for input/output interaction.
- Automation utilities for compilation and execution.
- Clear separation of concerns between backend logic, parser utilities, and frontend presentation.

2.1.2 Proposed Methodology/ System Design

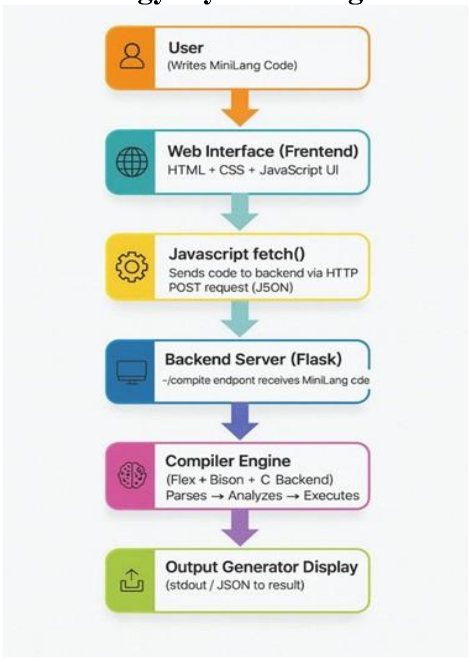
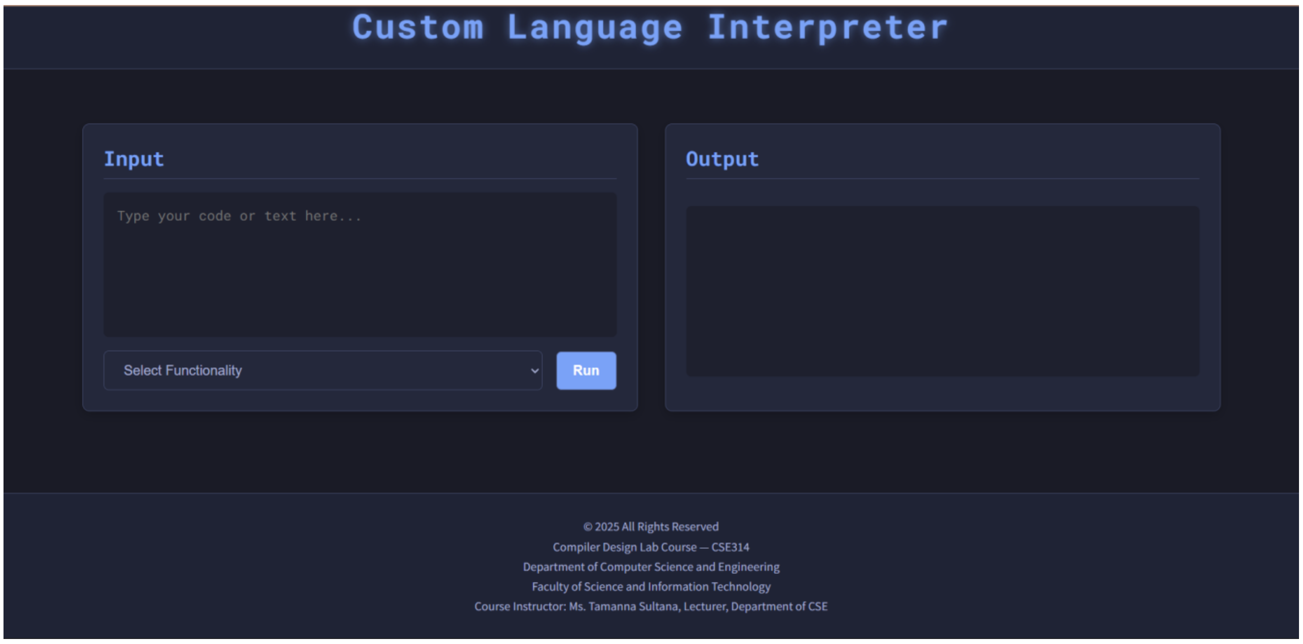

User
(Writes MiniLang Code)
Web Interface (Frentend)
HTML + CSS + JavaScript UI
Javascript fetch()
Sends code to backend via HTTP POST request (J5ON)
Backend Server (Flask)
-/compite endpont receives MiniLang cde
Compiler Engine
(Flex + Bison + C Backend)
Parses → Analyzes → Executes
Output Generator Display
(stdout / JSON to result)

Figure 2.1: sample diagram of CLI

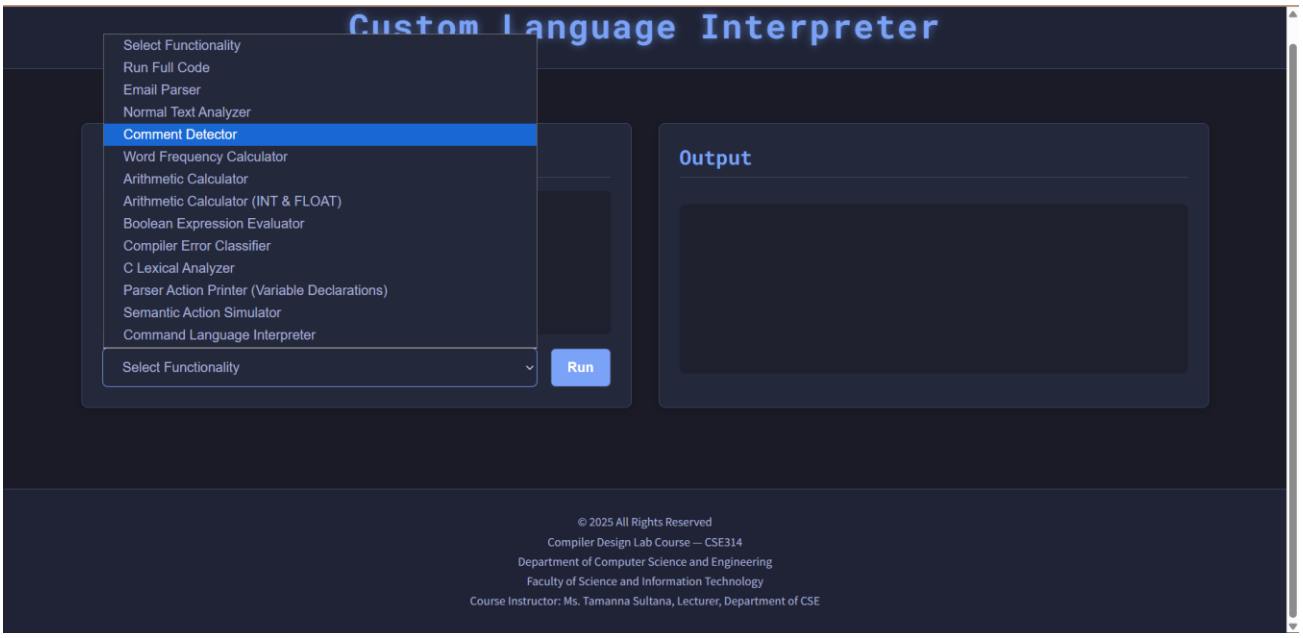
The methodology involves combining traditional compiler tools with modern web technologies.
Steps in the methodology:

1. Parser Development: Each Flex/Bison program is developed independently to handle tasks such as arithmetic evaluation, boolean logic, or variable extraction.
2. Compilation: Programs are compiled into executables using GCC.
3. Backend Integration: Flask routes are created to accept user input, call the relevant parser executable, and return results.
4. Utility Functions: A Python module (flex_bison_utils.py) manages compilation, execution, and error handling.
5. Frontend Interaction: The web interface allows users to select a parser, input data, and view results dynamically.

2.1.3 UI Design



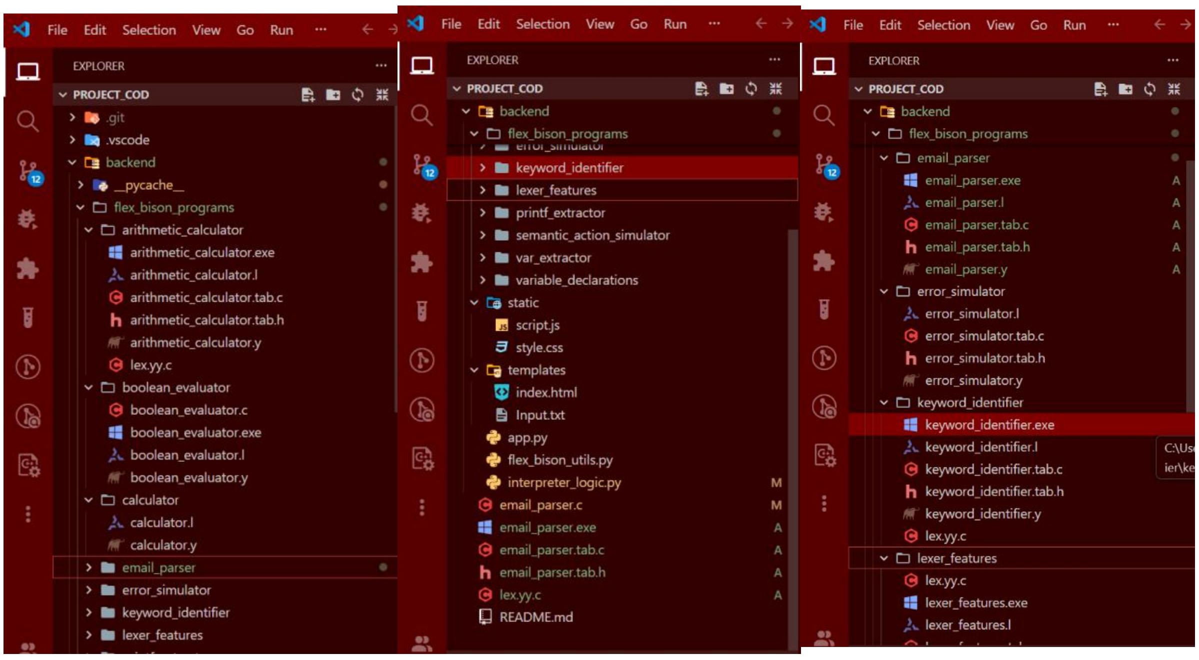
© 2025 All Rights Reserved
Compiler Design Lab Course — CSE314
Department of Computer Science and Engineering
Faculty of Science and Information Technology
Course Instructor: Ms. Tamanna Sultana, Lecturer, Department of CSE


Custom Language Interpreter
Select Functionality
Run Full Code
Email Parser
Normal Text Analyzer
Comment Detector
Word Frequency Calculator
Arithmetic Calculator
Arithmetic Calculator (INT & FLOAT)
Boolean Expression Evaluator
Compiler Error Classifier
C Lexical Analyzer
Parser Action Printer (Variable Declarations)
Semantic Action Simulator
Command Language Interpreter
Select Functionality
Run
Output

© 2025 All Rights Reserved
Compiler Design Lab Course — CSE314
Department of Computer Science and Engineering
Faculty of Science and Information Technology
Course Instructor: Ms. Tamanna Sultana, Lecturer, Department of CSE

Figure: Project CLI UI Design

2.1.4 UI Design Backbone:



2.1.5 System Architecture Diagram

User Interface (HTML/CSS/JS)
↓
Flask Backend (app.py)
↓
Utility Functions (flex_bison_utils.py)
↓
Flex/Bison Programs (compiled executables)
↓
Output Results → Displayed on Web Interface

Figure 2.2: Sample System Architecture Diagram

2.2 Overall Project Plan

The project was planned in phases to ensure systematic development:

- Phase 1 (Weeks 1): Requirement gathering, environment setup, and initial Flask configuration.
- Phase 2 (Weeks 2): Development of Flex/Bison programs for arithmetic, boolean, etc variable extraction tasks.
- Phase 3 (Weeks 3): Integration of compiled programs with Flask backend and utility functions.
- Phase 4 (Week 4): Frontend design and testing of user interface.
- Phase 5 (Week 5): Documentation, performance evaluation, and final presentation.

This phased approach ensured that each component was developed, tested, and integrated systematically, reducing risks and improving overall project quality.

Chapter 3

Implementation and Results

This chapter presents the implementation details of the project, followed by performance analysis and a discussion of the results. It explains how the system was built, tested, and evaluated to meet the objectives outlined earlier.

3.1 Implementation

The implementation phase involved integrating the backend, frontend, and parsing modules into a cohesive system.

- **Backend (Flask Application):**
  - The Flask server (app.py) was developed to handle HTTP requests and responses.
  - Routes were created to accept user input, select the appropriate parser, and return the processed output.
  - The utility module (flex_bison_utils.py) was responsible for compiling Flex/Bison programs and executing them when requested.
- **Flex/Bison Programs:**
  - Each program was implemented in its own directory under flex_bison_programs.
  - Examples include arithmetic_calculator, boolean_evaluator, var_extractor, and error_simulator,email_parser,etc.
  - Programs were compiled into executables using GCC after generating C source code with Flex and Bison.
- **Frontend (HTML, CSS, JavaScript):**
  - The user interface (index.html) provided input fields and parser selection options.
  - JavaScript (script.js) handled asynchronous requests to the Flask backend and dynamically updated the output section.
  - CSS (style.css) ensured a clean and user-friendly design.

- **Integration:**
  - The backend orchestrated communication between the frontend and the compiled executables.
  - Input was passed to the relevant parser, and the output was captured and displayed on the web interface.

3.2 Performance Analysis

The system was tested with multiple inputs across different parsers to evaluate efficiency and correctness.

- Execution Speed:
  - Since Flex and Bison generate optimized C code, execution was fast and results were returned almost instantly.
  - Flask handled concurrent requests effectively without noticeable delays.
- Accuracy:
  - Arithmetic and boolean evaluators produced correct results for all tested expressions.
  - Variable extraction and keyword identification worked reliably on sample C code inputs.
- Error Handling:
  - The error_simulator successfully demonstrated how parsing errors are detected and reported.
  - Invalid inputs were handled gracefully, with meaningful error messages displayed to the user.
- Scalability:
  - The modular design allows new parsers to be added without affecting existing functionality.
  - The system can be extended to support more complex grammars and larger inputs.

3.3 Results and Discussion

The results confirmed that the project objectives were achieved:

- The web application successfully integrated Flask with Flex/Bison programs.
- Users could interact with multiple parsers through a simple and intuitive interface.
- The system demonstrated compiler design concepts in a practical, accessible manner.

Discussion:

- The project proved that traditional compiler tools can be modernized and integrated into web-based systems.
- Students found the interactive interface helpful in understanding abstract concepts like lexical analysis and parsing.

The modular architecture ensures long-term usability and adaptability for future enhancements.

Chapter 4

Engineering Standards and Mapping

This chapter discusses the broader impacts of the project on life, society, environment, and sustainability. It also covers project management aspects such as cost analysis and teamwork, followed by a detailed mapping of program outcomes, complex problem-solving categories, and engineering activities.

4.1 Impact on Society, Environment and Sustainability

4.1.1 Impact on Life

The project enhances student learning by making compiler design concepts more accessible and interactive. It improves problem-solving skills and encourages experimentation, thereby positively impacting academic and professional development.

4.1.2 Impact on Society & Environment

By promoting education and computational literacy, the project indirectly contributes to societal growth. Since the system is software-based, it has minimal environmental impact. Hosting the application online would require energy consumption, but this can be mitigated by using sustainable cloud services.

4.1.3 Ethical Aspects

The project adheres to academic integrity and open-source principles. It avoids plagiarism by developing original code and documentation. Ethical considerations also include ensuring that the tool is used for educational purposes and not for malicious code analysis.

4.1.4 Sustainability Plan

The modular design ensures long-term sustainability. New parsers can be added without disrupting existing functionality. The project can be maintained and extended by future students, making it a reusable educational resource.

4.2 Project Management and Team Work

The project was managed collaboratively, with clear role distribution among team members. Tasks such as backend development, parser creation, and frontend design were divided to ensure efficiency.

Cost Analysis:

- **Budget Required (Minimal Setup):**
    - Software tools: Free (Python, Flask, Flex, Bison, GCC are open-source).
    - Hardware: Existing student laptops.
- **Alternate Budget Scenario:**
    - If deployed on institutional servers, hosting costs are eliminated.
    - If extended with advanced features (e.g., database integration), additional costs may arise (~$50–100 for cloud database services).

Revenue Model:
Although primarily educational, the project could be monetized by offering premium features (e.g., advanced compiler visualization, cloud-based parsing services) or licensing to educational institutions.

4.3 Complex Engineering Problem

4.3.1 Mapping of Program Outcome

In this section, provide a mapping of the problem and provided solution with targeted Program Outcomes (PO's).

Table 4.1: Justification of Program Outcomes

PO's	Justification
PO1	Students defined and related classes, objects, and members in solving parsing problems.
PO2	Knowledge of OOP and Java was applied in designing modular components and problem-solving.
PO3	UML models and system architecture diagrams were analyzed to present and solve real-world problems.

4.3.2 Complex Problem Solving

This section maps the project outcomes with engineering problem categories, highlighting the knowledge profile and rationale for each.

Table 4.2: Mapping with complex problem solving.

EP1 Dept of Knowledge	**EP2** Range of Conflicting Requirements	**EP3** Depth of Analysis	**EP4** Familiarity of Issues	**EP5** Extent of Applicable Codes	**EP6** Extent Of Stakeholder Involvement	**EP7** Inter-dependence
The project required knowledge of compiler design, parsing, web development, and integration of multiple technologies (Flask, Flex, Bison, GCC).	Balancing usability, performance, modularity, and educational clarity posed conflicting requirements. For example, ensuring fast execution while maintaining readability of results.	Detailed analysis was needed to design grammars, handle parsing errors, and integrate backend execution with frontend display.	Common issues such as parser conflicts, ambiguous grammar rules, and execution errors were identified and resolved	The project adhered to coding standards, open-source licensing, and best practices in software engineering.	Stakeholders include students, faculty, and institutions. Their feedback influenced UI design and parser selection.	The system demonstrates interdependence between backend, frontend, and parser modules, requiring coordinated design and testing.

Rationale:

- EP1 (Depth of Knowledge): The project required knowledge of compiler design, programming languages, and web technologies.
- EP2 (Conflicting Requirements): Balancing usability, performance, and modularity posed conflicting requirements.
- EP3 (Depth of Analysis): Detailed analysis was needed to integrate Flex/Bison with Flask effectively.
- EP4 (Familiarity of Issues): Common issues such as parser errors and execution failures were addressed.
- EP5 (Applicable Codes): The project adhered to coding standards and open-source licensing.
- EP6 (Stakeholder Involvement): Stakeholders include students, faculty, and educational institutions.
- EP7 (Interdependence): The system demonstrates interdependence between backend, frontend, and parser modules.

Knowledge Profile and Rationale:

The project demanded a multidisciplinary knowledge profile, combining compiler theory, programming languages, and web application development. The rationale for mapping lies in demonstrating how theoretical knowledge was applied to solve practical, complex problems in an educational context.

4.3.3 Engineering Activities

This section maps the project outcomes with complex engineering activities, providing rationales for each category.

Table 4.3: Mapping with Complex Engineering Activities

EA	Description	Rationale
EA1	Range of Resources	The project utilized open-source tools (Python, Flask, Flex, Bison, GCC) and student hardware resources, ensuring cost-effectiveness
EA2	Level of Interaction	Team members collaborated on backend, frontend, and parser modules. Interaction was essential for integration and testing.
EA3	Innovation	Innovative integration of traditional compiler tools with modern web technologies created a unique educational platform.
EA4	Consequences for Society & Environment	The project has positive educational consequences, improving student learning outcomes, with negligible environmental impact.
EA5	Familiarity	Familiarity with programming, compiler design, and web development was required. The project also introduced new learning opportunities for students unfamiliar with Flex/Bison.

Rationale:

- EA1: Demonstrates efficient use of available resources without requiring expensive infrastructure.
- EA2: Highlights teamwork and communication as critical factors in successful implementation.
- EA3: Shows how innovation was achieved by combining established tools in a novel way.
- EA4: Emphasizes the societal benefits of improved education and minimal environmental footprint.
- EA5: Reflects the balance between existing knowledge and new skills acquired during the project.

Chapter 5

Conclusion

This chapter summarizes the overall project, highlights its limitations, and suggests possible directions for future work. It reflects on the outcomes achieved and the potential for extending the system further.

5.1 Summary

The project successfully integrated a Python Flask backend with parsing and lexical analysis tools generated using Flex and Bison. A modular web application was developed that allows users to interact with multiple parsers and lexers, including arithmetic calculators, boolean evaluators, variable extractors, and error simulators.

The system demonstrated how traditional compiler construction tools can be modernized and made accessible through a web interface. Students and learners benefited from the interactive nature of the application, which provided hands-on experience with compiler design concepts. The project outcomes aligned with the stated objectives, contributing to both academic learning and practical application.

5.2 Limitation

Despite its success, the project has certain limitations:

- The system currently supports only predefined parsers and lexers. Adding new grammars requires manual compilation.
- The application is designed for local use; deployment on cloud servers would require additional configuration and resources.
- Error handling is basic and could be improved with more descriptive feedback for complex parsing failures.
- The user interface, while functional, is minimal and could be enhanced with richer visualization features (e.g., parse trees, syntax highlighting).

5.3 Future Work

Future enhancements could include:

- Automated Compilation: Developing scripts or modules to automatically compile new Flex/Bison programs without manual intervention.
- Cloud Deployment: Hosting the application online to make it accessible to a wider audience, including students and educators globally.
- Advanced Visualization: Adding graphical representations of parse trees, token streams, and semantic actions to improve understanding.
- Database Integration: Storing user inputs and results for analysis, enabling long-term tracking of learning progress.
- Extended Grammar Support: Expanding the system to handle more complex programming languages and constructs.
- Mobile Accessibility: Creating a mobile-friendly version of the application for broader usability.

References

1. Jon Kleinberg and Eva Tardos. *Algorithm Design*. Pearson Education India, 2006.

2. Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison-Wesley, 2006.

3. Flask Documentation. *Flask Web Development Framework*. Available at: https://flask.palletsprojects.com/

4. Flex Documentation. *Flex: The Fast Lexical Analyzer Generator*. Available at: https://github.com/westes/flex

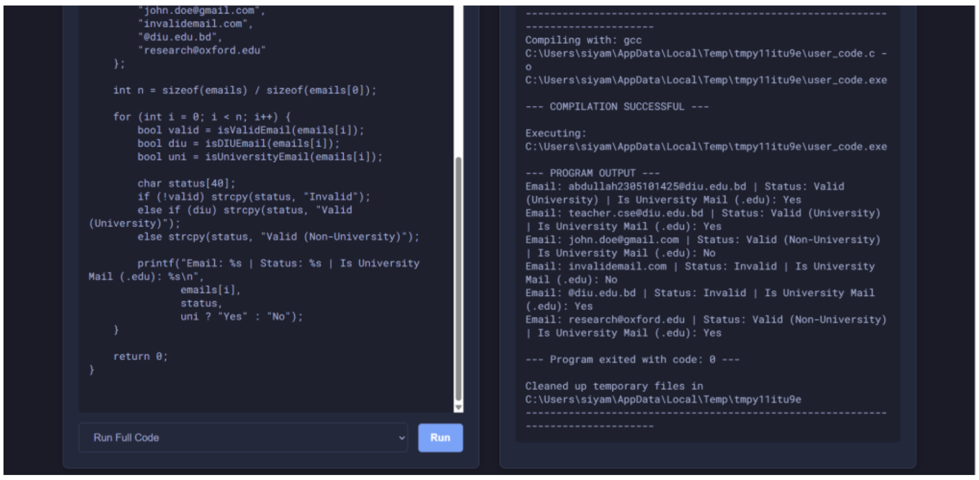
5. Bison Documentation. *GNU Bison: The Parser Generator*. Available at: https://www.gnu.org/software/bison/

6. Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language (2nd Edition)*. Prentice Hall, 1988.

7. Ian Sommerville. *Software Engineering (10th Edition)*. Pearson, 2015.

Project Outcome

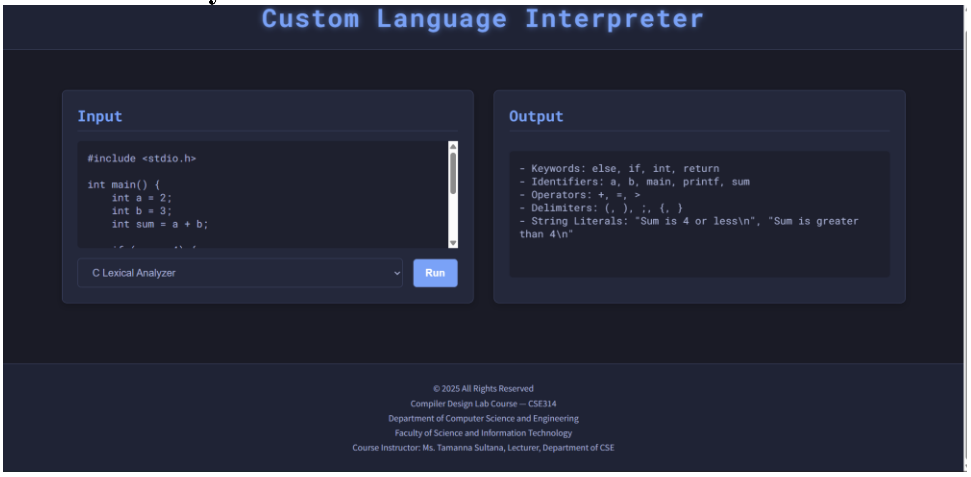
Email Parser



```
        "john.doe@gmail.com",
        "invalidemail.com",
        "@diu.edu.bd",
        "research@oxford.edu"
    };

    int n = sizeof(emails) / sizeof(emails[0]);

    for (int i = 0; i < n; i++) {
        bool valid = isValidEmail(emails[i]);
        bool diu = isDIUEmail(emails[i]);
        bool uni = isUniversityEmail(emails[i]);

        char status[40];
        if (!valid) strcpy(status, "Invalid");
        else if (diu) strcpy(status, "Valid
(University)");
        else strcpy(status, "Valid (Non-University)");

        printf("Email: %s | Status: %s | Is University
Mail (.edu): %s\n",
               emails[i],
               status,
               uni ? "Yes" : "No");
    }

    return 0;
}
```

```
-------------------------------------------------------------
--------------------
Compiling with: gcc
C:\Users\siyam\AppData\Local\Temp\tmpy11itu9e\user_code.c -
o
C:\Users\siyam\AppData\Local\Temp\tmpy11itu9e\user_code.exe

--- COMPILATION SUCCESSFUL ---

Executing:
C:\Users\siyam\AppData\Local\Temp\tmpy11itu9e\user_code.exe

--- PROGRAM OUTPUT ---
Email: abdullah2305101425@diu.edu.bd | Status: Valid
(University) | Is University Mail (.edu): Yes
Email: teacher.cse@diu.edu.bd | Status: Valid (University)
| Is University Mail (.edu): Yes
Email: john.doe@gmail.com | Status: Valid (Non-University)
| Is University Mail (.edu): No
Email: invalidemail.com | Status: Invalid | Is University
Mail (.edu): No
Email: @diu.edu.bd | Status: Invalid | Is University Mail
(.edu): Yes
Email: research@oxford.edu | Status: Valid (Non-University)
| Is University Mail (.edu): Yes

--- Program exited with code: 0 ---

Cleaned up temporary files in
C:\Users\siyam\AppData\Local\Temp\tmpy11itu9e
-------------------------------------------------------------
--------------------
```

C Lexical Analyzer:

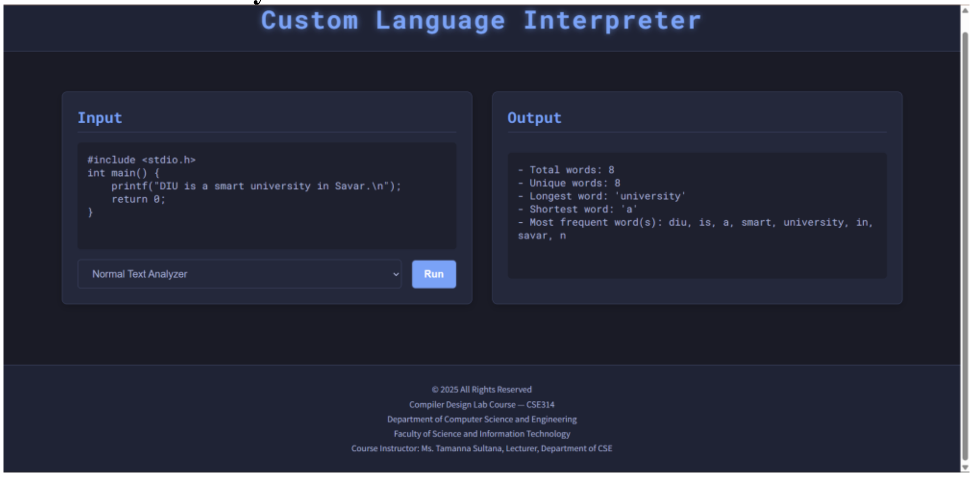


```
#include <stdio.h>

int main() {
    int a = 2;
    int b = 3;
    int sum = a + b;
```

```
- Keywords: else, if, int, return
- Identifiers: a, b, main, printf, sum
- Operators: +, =, >
- Delimiters: (, ), ;, {, }
- String Literals: "Sum is 4 or less\n", "Sum is greater
than 4\n"
```

© 2025 All Rights Reserved
Compiler Design Lab Course — CSE314
Department of Computer Science and Engineering
Faculty of Science and Information Technology
Course Instructor: Ms. Tamanna Sultana, Lecturer, Department of CSE

Normal Text Analyzer:

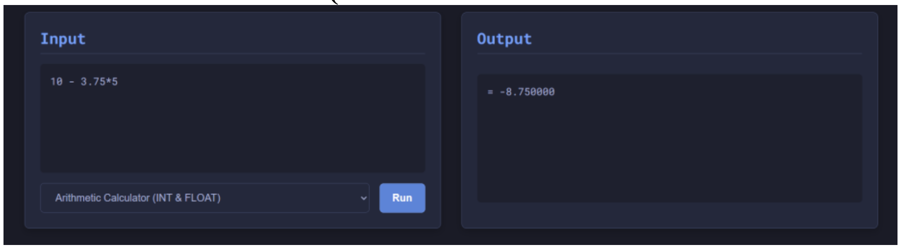


```
#include <stdio.h>
int main() {
    printf("DIU is a smart university in Savar.\n");
    return 0;
}
```

```
- Total words: 8
- Unique words: 8
- Longest word: 'university'
- Shortest word: 'a'
- Most frequent word(s): diu, is, a, smart, university, in,
savar, n
```

© 2025 All Rights Reserved
Compiler Design Lab Course — CSE314
Department of Computer Science and Engineering
Faculty of Science and Information Technology
Course Instructor: Ms. Tamanna Sultana, Lecturer, Department of CSE

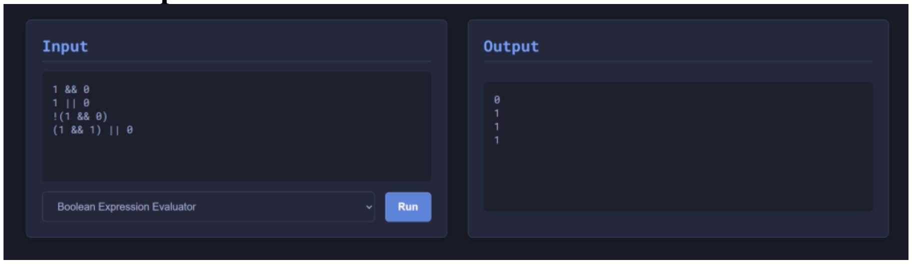
Arithmetic Calculation (INT & FLOAT



```
10 - 3.75*5
```

```
= -8.750000
```

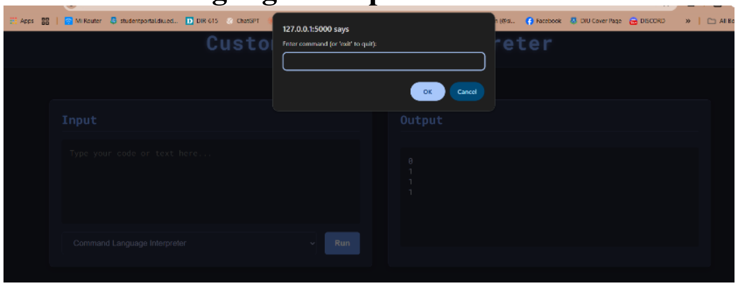
Boolean Expression Evaluator



```
1 && 0
1 || 0
!(1 && 0)
(1 && 1) || 0
```

```
0
1
1
1
```

Command Language Interpreter

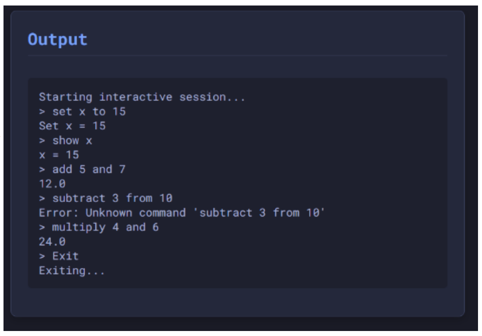


127.0.0.1:5000 says

Enter command (or 'exit' to quit):

```
Type your code or text here...
```

```
0
1
1
1
```



```
Starting interactive session...
> set x to 15
Set x = 15
> show x
x = 15
> add 5 and 7
12.0
> subtract 3 from 10
Error: Unknown command 'subtract 3 from 10'
> multiply 4 and 6
24.0
> Exit
Exiting...
```