Project Guidelines

CSCI415 – Compiler Design

FALL 2025

Project Title

Mini Compiler for SQL-like Languages

Project Overview

Students will design and implement a compiler for a SQL-like query language. The compiler will be built in three phases, each submitted and graded individually. The final submission MUST perform lexical, syntax and semantic analysis on queries.

- Phase 01: Lexical Analyzer
- Phase 02: Syntax Analyzer
- Phase 03: Semantic Analyzer

Project Timeline

Month	Day	Phase
Oct	04 – 10	Teams Registration
	11	Phase 01 Starts
Nov	01	Phase 01 Deadline
	22	Phase 02 Starts
Dec	19	Phase 02 Deadline
	20	Phase 03 Starts
	26	Final Project Submission
Dec – Jan	27 – 01	Project Discussion

Notes:

- Take into consideration that all phases are able to build on each other to avoid issues.
- You may use any coding language you prefer to complete the project.
- A discussion will be held with ALL team members; all members must participate in implementation.
- DO NOT use ready functions or libraries; build your own methods.
- **ANY PLAGARISM DETECTED WILL LEAD TO A ZERO AND NO DISCUSSION WILL BE CONDUCTED FOR THE TEAM.**

Phase 01 - Lexical Analysis

Objective

In this phase, students will implement a lexical analyzer (scanner) for a SQL-like language. The lexical analyzer reads source code character by character, groups them into tokens, classifies them, and constructs a symbol table.

Responsibilities

- Ignore **whitespace** and **comments**.
- Detect **illegal characters** and report their positions.
- Handle **case-sensitivity** (do not treat select and SELECT the same).
- Generate a **stream of tokens** to be consumed by the parser in Phase 2

Specifications

Case-sensitive

Keywords

Reserved SQL words like (SELECT, FROM, WHERE, INSERT, INTO, VALUES, UPDATE, SET, DELETE, CREATE, TABLE, INT, FLOAT, TEXT, AND, OR, NOT)

Identifiers

User-defined names (table/column names). Must start with a letter and may contain digits or underscores.

Literals

String and numeric constants. Strings are enclosed in single quotes.

Operators

Arithmetic and comparison operators.

Delimiters/Punctuation

Symbols used to separate SQL constructs like parenthesis, commas, semicolons and similar symbols.

Comments

(Must be skipped by the lexer.)

Single line comments start with -- as for multi-line comments they start and end with double hashes (#).

Error Handling

Your lexical analyzer must:

- Detect **unknown or invalid symbols**.
  Example: `@id` → "Error: invalid character '@' at line 3, position 5."
- Handle **unclosed string literals**.
  Example: `'Ali` → "Error: unclosed string starting at line 2."
- Handle **unterminated comments**.
  Example: `/* comment` → "Error: unclosed comment starting at line 5."

All errors must include:

- Line number
- Column number
- Descriptive message

Sample

Input (text file)

```
CREATE TABLE students (id INT, name TEXT);
INSERT INTO students VALUES (1, 'Ali');
SELECT name FROM students WHERE id = 1;
```

Output (similar not exact)

```
Token: CREATE, Lexeme: CREATE
Token: TABLE, Lexeme: TABLE
Token: IDENTIFIER, Lexeme: students
Token: LEFT_PAREN, Lexeme: (
Token: IDENTIFIER, Lexeme: id
Token: TYPE, Lexeme: INT
Token: COMMA, Lexeme: ,
Token: IDENTIFIER, Lexeme: name
Token: TYPE, Lexeme: TEXT
Token: RIGHT_PAREN, Lexeme: )
Token: SEMICOLON, Lexeme: ;
Token: INSERT, Lexeme: INSERT
Token: INTO, Lexeme: INTO
```

```
Token: IDENTIFIER, Lexeme: students
Token: VALUES, Lexeme: VALUES
Token: LEFT_PAREN, Lexeme: (
Token: NUMBER, Lexeme: 1
Token: COMMA, Lexeme: ,
Token: STRING, Lexeme: 'Ali'
Token: RIGHT_PAREN, Lexeme: )
Token: SEMICOLON, Lexeme: ;
Token: SELECT, Lexeme: SELECT
Token: IDENTIFIER, Lexeme: name
Token: FROM, Lexeme: FROM
Token: IDENTIFIER, Lexeme: students
Token: WHERE, Lexeme: WHERE
Token: IDENTIFIER, Lexeme: id
Token: EQUAL, Lexeme: =
Token: NUMBER, Lexeme: 1
Token: SEMICOLON, Lexeme: ;
```

Deliverable

Each team must submit:

1. **Source code**
2. **Input text file**
3. **Brief Report (1 – 2 pages)** mentioning:
    a. Tokens implemented
    b. How did you handle errors?
    c. Faced challenges and how it was fixed.