CSCI415 – Compiler Design

Phase 01: Lexical Analyzer Report

Project Title: Mini Compiler for SQL-like Languages

Team Members:

- **Sama Eldessouky – 221000579**
- **Ziad Wael – 221001480**
- **AbdelRahman AlSammany – 221000915**
- **Shaden Abdelsalam – 221001937**

---

1. Objective

The goal of this phase was to implement a Lexical Analyzer for a SQL-like language. The lexical analyzer reads the input source code character by character, groups them into tokens, classifies each token type, builds a symbol table, and reports lexical errors such as invalid characters, unclosed strings, and unterminated comments.

---

2. Tokens Implemented

The lexer identifies and classifies the following categories of tokens:

- **Keywords:** SELECT, FROM, WHERE, INSERT, INTO, VALUES, UPDATE, SET, DELETE, CREATE, TABLE, INT, FLOAT, TEXT, AND, OR, NOT.
- **Identifiers:** User-defined names such as table and column names that start with a letter and may include digits or underscores.
- **Literals:** String literals enclosed in single quotes, and numeric literals including integers and floating-point numbers.
- **Operators:** Arithmetic and comparison operators such as equal, not equal, less than, greater than, and others.
- **Delimiters and punctuation:** Parentheses, commas, semicolons, and similar symbols.
- **Comments:** Single-line comments starting with two dashes and multi-line comments enclosed within double hashes.

- **End of File:** A special token added to mark the end of the input.

---

3. Error Handling

The lexical analyzer detects and reports various types of lexical errors with line and column numbers and a descriptive message. The main error types include invalid characters, unclosed string literals, and unterminated multi-line comments. Each error is printed immediately when detected through the report_error function.

Invalid characters are also added as tokens of type ILLEGAL to maintain consistency in the token stream and allow scanning to continue without interruption.

---

4. How Errors Were Handled

Each error message includes the exact line and column numbers where the error occurred to make debugging easier.
Unclosed strings and unterminated comments are detected and reported while allowing the lexer to continue scanning the rest of the file.
Invalid symbols are not ignored; they are recorded as ILLEGAL tokens so the parsing phase can still proceed without breaking.

---

5. Challenges Faced and Solutions

One main challenge was handling multi-line comments using the double hash syntax. This was solved by scanning until two consecutive hash symbols are found and raising an error if the end of file is reached before closing the comment.
Another challenge was managing unclosed string literals and detecting the end of file correctly, which was handled by checking for missing closing quotes inside the string handler.
Tracking line and column positions accurately was also difficult, and it was solved by updating counters in the advance function each time a character was processed.
Differentiating between keywords and identifiers required using a lookup dictionary that matches only exact uppercase keywords, ensuring that the language remains case-sensitive.

---

6. Conclusion

The lexical analyzer successfully converts SQL-like queries into a structured stream of tokens. It ignores whitespace and comments, correctly identifies valid tokens, and reports errors with detailed information. The produced token stream is ready to be used by the Syntax Analyzer in Phase 02 of the compiler project.