CONSTRUCTION COMPILER LAB TERMINAL (Q4)

SUBMITTED BY : FAJAR AAMIR SHEIKH

REGISTRATION NO: SP22-BCS-031

SUBMITTED TO : SIR BILAL BUKHARI

SUBMISSION DATE: 18TH JUNE2025

<u>QUESTION 4:</u> What Challenges Did You Face During the Project?

Developing a mini compiler, even for a limited scope, involved several challenges across different phases. These challenges were both **technical** and **logical**, especially since the compiler mimics the behavior of real-world compilers in a simplified form.

1. Lexical Analysis Complexity

- **Challenge:** Designing a clean and minimal lexer that correctly identifies keywords, identifiers, numbers, operators, and handles invalid characters.
- **Issue Faced:** At first, unexpected characters caused the program to crash due to lack of error handling in Lexer.NextToken().
- **How Resolved:** Introduced throw new Exception() to catch and display meaningful error messages for invalid characters.

2. Syntax Validation Logic

- **Challenge:** Ensuring only valid assignment statements like int x = 5; are accepted.
- **Issue Faced:** Incorrect ordering of tokens (e.g., missing ; or misplacing =) was hard to debug.
- **How Resolved:** Implemented a strict Eat(TokenType) method and added detailed position-based error messages to help identify syntax issues.

3. Symbol Table & Semantic Checks

- **Challenge:** Handling redeclaration and type mismatches using a Dictionary.
- **Issue Faced:** Accidentally allowed redeclaration of variables multiple times.
- **How Resolved:** Added logic to SymbolTable.Declare() to check for existing entries and throw a semantic error if redeclared.

4. No User Input During Syntax Phase

- **Challenge:** Syntax analyzer directly used tokens without re-prompting the user.
- **Issue Faced:** Users expected the compiler to ask for corrections, but the syntax analyzer didn't allow input correction.
- **How Resolved:** This was accepted as a limitation in a mini compiler. Compilation stops with a helpful error message instead.

5. Manual Intermediate & Target Code Generation

- **Challenge:** Designing logic to convert parsed variables into intermediate and target code without a real backend.
- **Issue Faced:** Mapping every variable to temporary values (t1, etc.) while keeping it readable.
- **How Resolved:** Simplified the IR and target code formats to readable strings:
    - IR → t1 = value, x = t1
    - Target → LOAD value, STORE x

6. Debugging and Testing Multiple Lines

- **Challenge:** Allowing users to enter 5 lines of code and parsing each independently.
- **Issue Faced:** If one line failed, the whole compilation process stopped.
- **How Resolved:** Used a loop to compile each line separately, displaying errors line-by-line, while allowing others to proceed.

7. **Optimization Limitations**

- **Challenge:** Implementing general-purpose constant folding.
- **Issue Faced:** Only hardcoded case (2 + 3) worked.
- **How Resolved:** Left as a placeholder for future extension with expression evaluation logic.