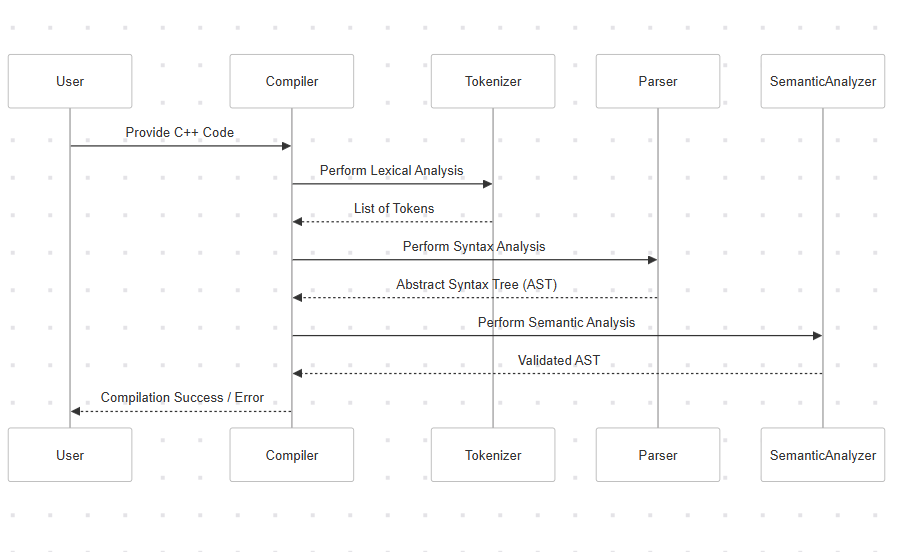


**Lab Terminal**

**Name : Muhammad Wasiq**

**Reg. no . : fa21-bcs-039**

**Course : Compiler Construction**

**Q.1:  
  
**

**Q.2:**

### **1. Lexical Analysis**

Function: std::vector<Token> lexicalAnalysis(const std::string &code)

Purpose:

* Converts the raw source code (text) into a structured sequence of tokens.
* Each token represents a meaningful unit like keywords (int, return), identifiers (main), numbers (42), operators (+, -), and symbols ({, }).

Key Steps:

* Uses regular expressions to define patterns for different types of tokens.
* Matches the input code against these patterns to extract tokens.
* Skips irrelevant parts like whitespace while retaining the meaningful elements.

Example: Input:

int main() { return 42; }

Output:

(KEYWORD, int), (IDENTIFIER, main), (SYMBOL, (), (SYMBOL, )), (SYMBOL, {), (KEYWORD, return), (NUMBER, 42), (SYMBOL, ;), (SYMBOL, })

### **2. Syntax Analysis**

Function: std::vector<ASTNode> syntaxAnalysis(const std::vector<Token> &tokens)

Purpose:

* Constructs an Abstract Syntax Tree (AST) from the tokens.
* The AST represents the logical structure of the code, focusing on its meaning rather than syntax.

Key Steps:

* Processes tokens sequentially.
* Recognizes patterns, such as return statements, and creates corresponding AST nodes.
* Throws an error if the tokens do not form a valid structure.

Example: Input Tokens:

(KEYWORD, return), (NUMBER, 42), (SYMBOL, ;)

Output AST:

[{type: "return", value: 42}]

### **3. Semantic Analysis**

Function: void semanticAnalysis(const std::vector<ASTNode> &ast)

Purpose:

* Validates the AST to ensure logical correctness.
* Checks for semantic rules, such as type constraints and value validity.

Key Steps:

* Iterates through the AST.
* For each return node, ensures the value being returned is an integer.
* Throws an error if any semantic rule is violated.

Example: Input AST:

[{type: "return", value: 42}]

Output:

Semantic Analysis Passed!