COMSATS Institute of Information Technology
Department of Computer Science, Attock Campus

"CC Mid Lab"

Name: **ASMA AKHTAR**
M.Saif Ullah

Reg No**: Fa20-BCS-003**
Fa20-BCS-039

Course**: CC (lab)**

PROGRAM**: BS(CS)-7A**

Course Instructor**: Mr. Bilal Haider**

Date**:28-12-2023**

Q4:

STEP WISE PROCESS:

Input:
Receive Java source code as input.

<u>Scanner (Lexical Analysis):</u>

Tokenization:
The process of breaking down the source code into meaningful units known as tokens, encompassing keywords, identifiers, operators, constants, and symbols.

Regular Expressions and Patterns:
The Lexical Analyzer utilizes regular expressions or patterns to define rules for identifying various types of tokens.

Scanning:
The systematic examination of the source code character by character.

Recognizing Tokens:
The identification of tokens during the scanning process.

Building Tokens:
Once a token is recognized, the Lexical Analyzer constructs the token by assembling the characters that constitute it.

Ignoring Whitespace and Comments:
The Lexical Analyzer typically disregards whitespace characters and comments during the tokenization process.

Output Token Stream:
The result of the Lexical Analyzer's work—a stream of tokens produced as its output.

Passing Tokens to the Syntax Analyzer:
The generated token stream is then forwarded to the subsequent phase of the compiler, the Syntax Analyzer.

<u>Semantic Analysis:</u>

Input:
Semantic analysis takes the results of the lexical and syntax analysis phases as its input.

Symbol Table Construction:
The compiler generates a symbol table, a data structure that holds information about variables, functions, and other program entities. This table tracks names, types, and scopes of these entities.

Type Checking:
Ensures that operations, such as attempting to add a string to an integer, adhere to type compatibility rules.

Scope Analysis:
The compiler examines the code for violations related to variable scope, ensuring adherence to scope rules.

Declaration Checking:

Semantic analysis verifies that variables and functions are properly declared before their usage in the code.

Function Overloading and Signature Matching:

In the case of function overloading support, the compiler checks that functions sharing the same name possess distinct parameter lists.

Constant Folding:

Involves the evaluation of constant expressions during compile-time to optimize the generated code.

Error Reporting:

In the event of semantic errors, such as type mismatches, undeclared variables, or scope violations, the compiler communicates these errors to the user.

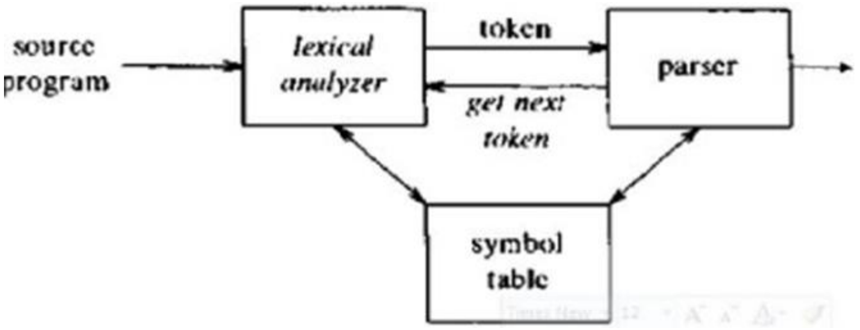
Intermediate Code Generation:

In certain compiler designs, semantic analysis may include the additional task of producing intermediate code.

Output:

The result of the semantic analysis phase is either an error report, revealing the existence and nature of semantic errors, or, in the absence of errors, an improved representation of the program that accurately captures its semantic meaning.

Class Diagram:


source program
lexical analyzer
token
get next token
parser
symbol table