# COMPILER CONSTRUCTION

# LAB TERMINAL

**NAME:**

**USAMA MANSOOR**

**REG. NO:**

**FA20-BCS-026**

**CLASS:**

**BCS-7B**

**SUBMITTED TO:**

**SIR BILAL HAIDER**

**DATE:**

**27-12-2023**

# SEMANTIC ANALYZER:

- **QUESTIO NO 01:**

**Brief Introduction of Project**

- **ANSWER:**

The project involves a basic compiler or interpreter that processes user-entered source code. The main components include a Scanner, Semantic Analyzer.

- **Semantic Analyzer:**

This component applies semantic rules to the tokens generated by the scanner. It checks for semantic errors, ensuring that the code makes logical sense. This includes checking for proper variable declaration and assignment, as well as verifying that operations are performed between compatible types.

Following are the rules that my Semantic Analyzer follows:

**RULE 1:** First rule checks the datatypes that used in source code (input).

**RULE 2:** Second rule checks operators (+, -, *, /) in source code (input).

**RULE 3**: Third rule checks logical operators (>=, <=, >, <) in source code (input).

The SemanticAnalyzer() function takes the list of tokens identified by the Scanner, and checks if the tokens sequence follow any of the three rules that are mentioned above using Top-Down Parsing, then it return a list of the errors found if any.

- **Scanner:**

The scanner is responsible for breaking down the source code into individual tokens. It classifies these tokens into categories such as identifiers, numbers, variables, operators, etc.

## LEXICAL ANALYZER:

### QUESTION NO 01:

**Give the brief explanation of the CC final project?**

**ANSWER:**

This project involves the creation of a basic interpreter for arithmetic expressions using C#. The interpreter is designed to process and evaluate mathematical expressions entered as strings, breaking them down into individual components through a process called lexical analysis or tokenization. The main components of this interpreter include the Lexer class, the Parser class, and associated supporting structures.

In the initial stage, the Lexer class is responsible for tokenizing the input string, identifying and categorizing individual elements such as numbers, operators (addition, subtraction, multiplication, and division), and parentheses. The Lexer utilizes the Token class to represent each token, which includes properties for the token's type (TokenType) and its actual value.

Following the tokenization phase, the Parser class takes the sequence of tokens produced by the Lexer and interprets the syntax of the arithmetic expressions. The parsing process, implemented using a recursive descent parsing approach, involves breaking down expressions into terms, factors, and numbers. The Parser class includes methods such as ParseExpression, ParseTerm, and ParseFactor to navigate through the token stream and construct a syntactic representation of the input expression.