NASEEM HASSAN (FA20-BCS-068)
MUHAMMAD TALHA TAJ (FA20-BCS-058)

CC TERMINAL

LEXER ANALYZER

Q1 Project Brief: Lexical Analyzer

Answer:

Objective:

Creating a program that performs lexical analysis on input text to identify and classify tokens based on defined rules.

Code Overview:

Language/Framework: C# using System and System.Collections.Generic namespaces.

Class Structure: Analyze class within the Analyzer namespace.

Functionalities:

loadTransitionTable: Reads transition rules from a file to set up the rules for token identification.

getNextState: Determines the next state based on the current state and input character.

isKeyword: Checks if a given token is a keyword or not.

Result:

Performs lexical analysis on the provided text.

Processes characters one by one, filtering out comments and identifying tokens based on predefined rules.

Recognizes identifiers, integers, floats, strings, and operators.

Builds a result string by replacing identified tokens with corresponding placeholders (<ID>, <INT>, <FLOAT>, <STR>, <OPR>).

Usage:

Input: Text input to be analyzed.

Transition Table: Rules for the analyzer loaded from a file (defaults to "matrix.txt").

Output: Resultant string after lexical analysis, replacing identified tokens with placeholders.

Functionality Notes:

The analyzer distinguishes between keywords, identifiers, integers, floats, strings, and operators based on the defined rules.

Comments in the input text are filtered out and not included in the output.

Identified tokens are replaced with their corresponding placeholders in the final output.

Q2: Give two functionalities of the project?

Answer:

Control Flow Explanation:

1. Input and Initialization:

The Result method is the entry point.

Input text is provided for lexical analysis (txt), along with an optional file path for transition rules (tt).

Initialization of variables (txtIndex, iState, cTemp, cChar, sToken, flag).

2. Transition Rules Loading:

loadTransitionTable is called within Result to load transition rules.

If successful, the rules are stored in the rules list.

3. Lexical Analysis Loop:

Loop Initialization:

Sets up a loop to iterate through each character in the input text.

Character Processing:

Character Analysis:

Checks for comments, handles one character at a time.

Uses getNextState to determine the character type and transition between states.

Tokenization and Output:

Constructs tokens based on transitions.

Identifies and categorizes tokens (identifiers, keywords, numbers, symbols).

Builds the output string by appending identified tokens or placeholders.

4. Functions Involved in Control Flow:

loadTransitionTable Function:

Reads transition rules and initializes the rules list.

getNextState Function:

Determines the next state based on the current state and input character.

Guides the transition between different states in the lexical analysis process.

isKeyword Function:

Determines if a given token is a keyword by comparing it to a predefined list of keywords.

5. Error Handling:

Exception Handling:

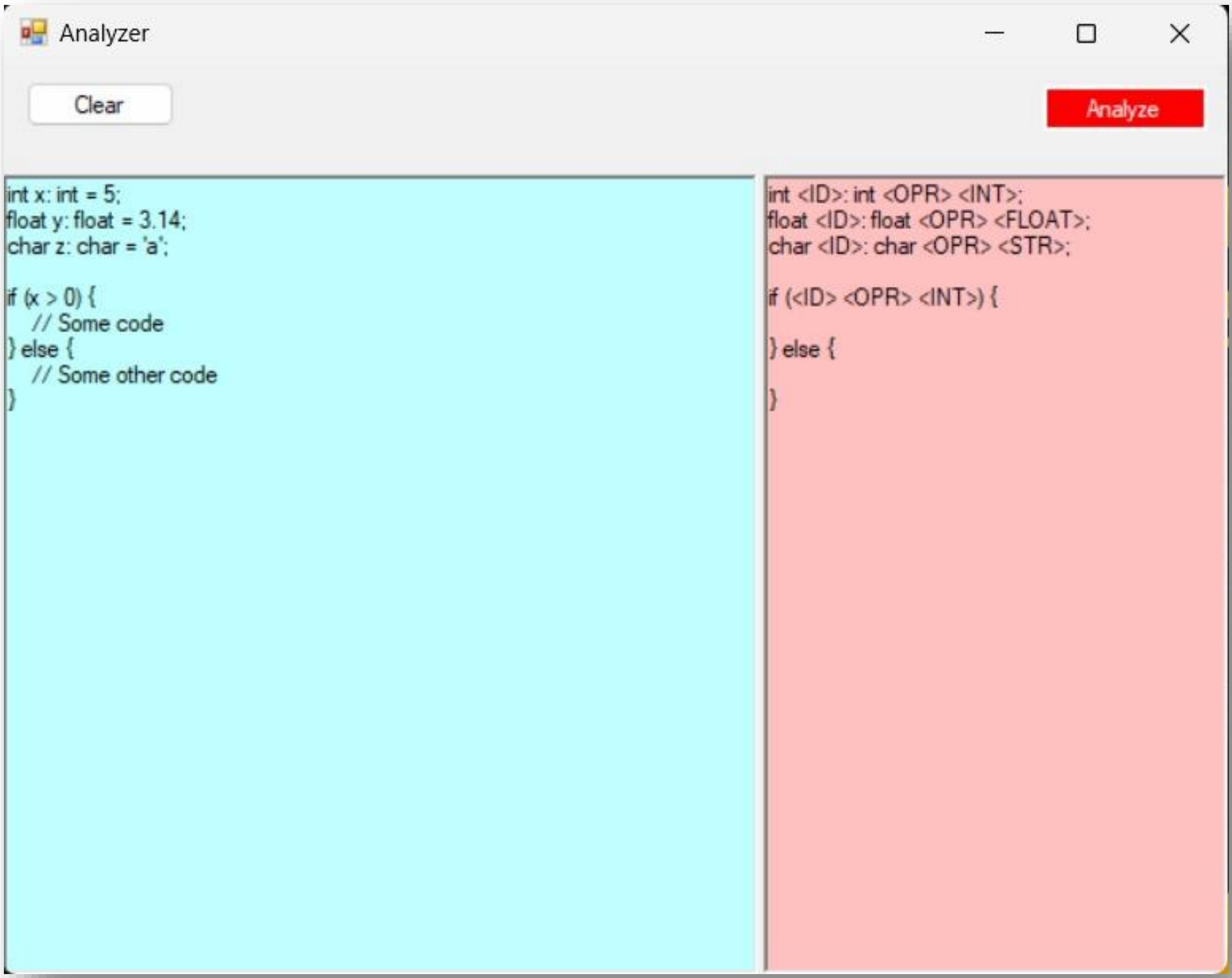
If there's an issue loading transition rules, an exception is caught.

Returns an error message indicating the failure to load transition rules.

6. Output:

Result String:

The output of the Result method is the processed string based on identified tokens and placeholders.



```
int x: int = 5;
float y: float = 3.14;
char z: char = 'a';

if (x > 0) {
   // Some code
} else {
   // Some other code
}
```

```
int <ID>: int <OPR> <INT>;
float <ID>: float <OPR> <FLOAT>;
char <ID>: char <OPR> <STR>;

if (<ID> <OPR> <INT>) {

} else {

}
```

Question # 03: Working of project with output.

Data Flow Explanation:

1. Input Data:

Text to Analyze: This is the string passed to the Result method for lexical analysis. For instance, consider "if (x == 5) { return x; }" as the input text.

2. loadTransitionTable Function:

Input: Path to the transition rules file (e.g., "matrix.txt").

Output: Populates the rules list, which represents the transition rules for the lexical analyzer.

3. Result Function:

Initialization:

Loads transition rules using loadTransitionTable.

Initializes variables (txtIndex, iState, cTemp, cChar, sToken, flag).

Lexical Analysis Loop:

Character Processing:

Iteration through Characters: Analyzes each character in the input text sequentially.

Character Classification:

Determines the type of character (letter, digit, symbol) using getNextState.

Constructs tokens and identifies their types.

Tokenization and Output:

Appending Results: Builds the output string based on identified tokens.

Filtering Out Comments: Ignores comments in the input text.

4. getNextState Function:

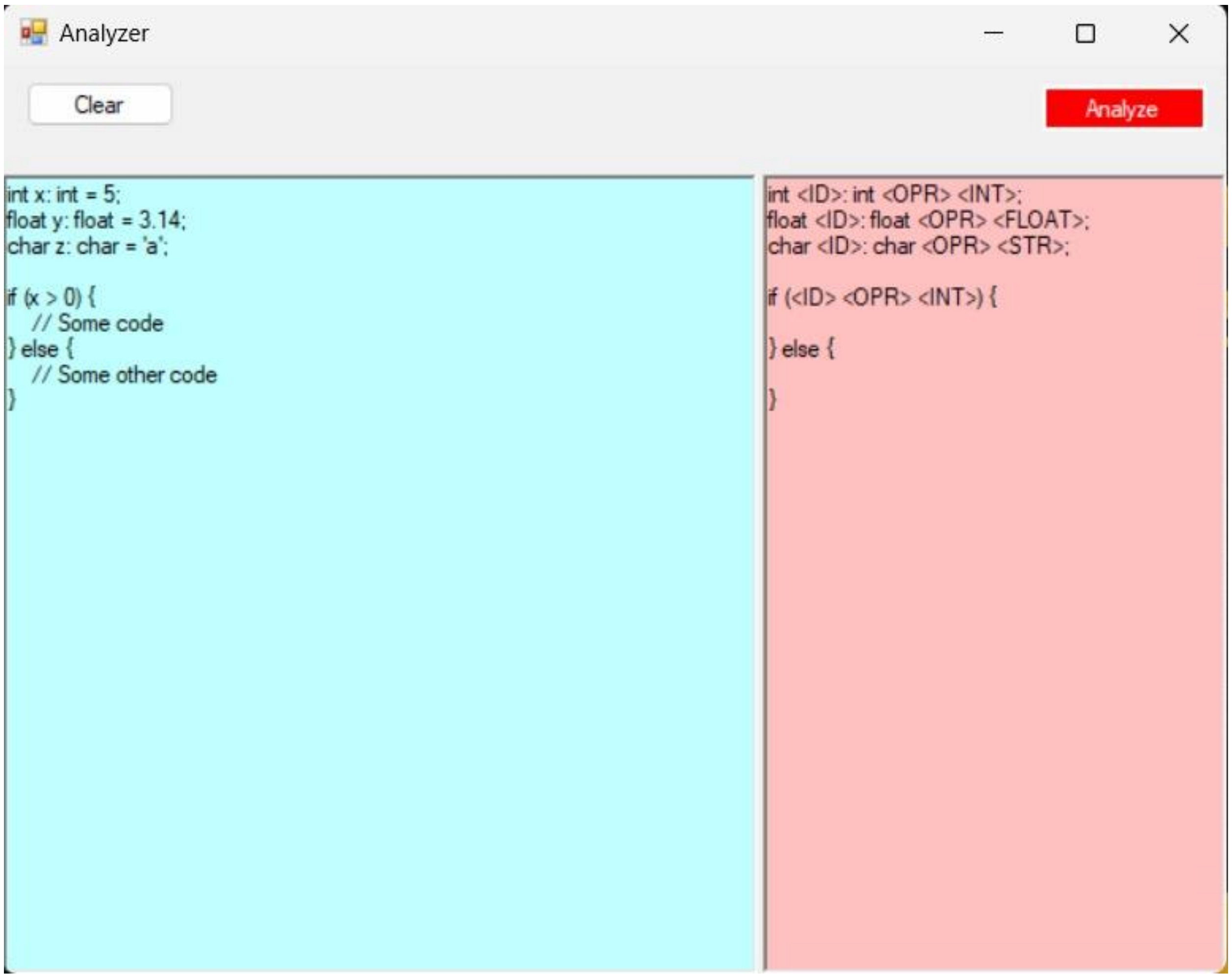
Input: Current state (iState) and the current character (cChar).

Output: Determines the next state based on the input character and the rules defined in the rules list.

5. isKeyword Function:

Input: A token (e.g., an identifier).

Output: Checks if the given token is a keyword from the predefined list.

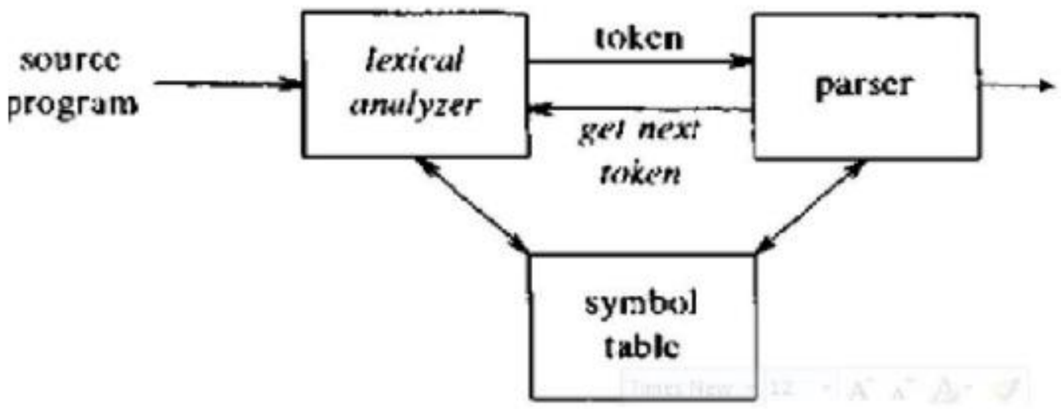


```
int x: int = 5;
float y: float = 3.14;
char z: char = 'a';

if (x > 0) {
   // Some code
} else {
   // Some other code
}
```

```
int <ID>: int <OPR> <INT>;
float <ID>: float <OPR> <FLOAT>;
char <ID>: char <OPR> <STR>;

if (<ID> <OPR> <INT>) {

} else {

}
```

Q4: Class diagram of the project?

Answer:



Q5: Challenges faced during this project?

Answer:

First, we have to clear the concept of Semantic and lexical analyses. It gives many errors due to our own concepts was not clear at the beginning but later we keep on hard working take help from google then resolved error.

Building an efficient lexer to tokenize the input source code is a challenge for us. Handling complex language constructs, identifying keywords, and dealing with special characters was very tricky.

Ensuring that the grammar is unambiguous and can be parsed accurately was very crucial.

Choosing the right parsing technique, whether it's top-down or bottom-up parsing, was challenging. Implementing a parser that can handle the grammar rules effectively was a key part of the project.

Integrating the lexical and semantic analysis components with a Windows Forms application may require careful design to ensure responsiveness and a smooth user experience.

Debugging issues in the parsing and analysis phases was challenging.

user-friendly interface in a Windows Forms application are important but can be overlooked in the midst of technical challenges.

SEMANTIC ANALYZER

Q1: 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 provided in the project description using Top-Down Parsing, then it return a list of the errors found.

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.

Q2:Functionalities of Project

ANSWER:

The two main functionalities of the project are:

Lexical Analysis and Tokenization:

This involves breaking down the source code into individual tokens (words) and classifying them into different categories such as identifiers, numbers, variables, and operators.

Semantic Analysis and Error Checking:

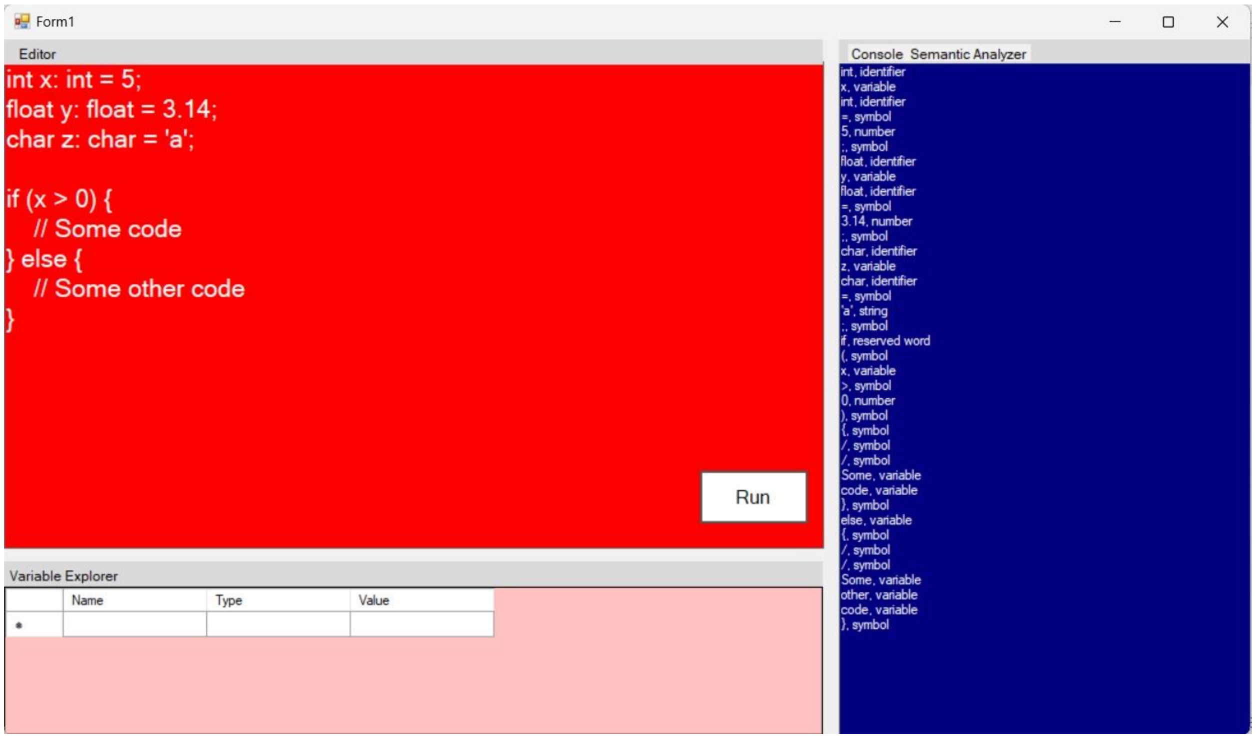
This involves checking the code for errors that make it nonsensical, such as using undeclared variables or performing operations on incompatible types.

Q3:Input with output

ANSWER:

INPUT WITH OUTPUT:

```
int var1 = 10;

int var2 = 20;

int var3 = var1 + 20;

int add = var1 + var2;

int sub = var1 - var2;

int mul = var1 * var2;

int div = var2 / var1;

if(x >= 10) { y = 10; }

if(x >= 20 && y != 10 ) { y = 20;}
```



```
int x: int = 5;
float y: float = 3.14;
char z: char = 'a';

if (x > 0) {
    // Some code
} else {
    // Some other code
}
```

```
int, identifier
x, variable
int, identifier
=, symbol
5, number
;, symbol
float, identifier
y, variable
float, identifier
=, symbol
3.14, number
;, symbol
char, identifier
z, variable
char, identifier
=, symbol
'a', string
;, symbol
if, reserved word
(, symbol
x, variable
>, symbol
0, number
), symbol
{, symbol
/, symbol
/, symbol
Some, variable
code, variable
}, symbol
else, variable
{, symbol
/, symbol
/, symbol
Some, variable
other, variable
code, variable
}, symbol
```

The error shows in right side window is missing semicolon that

violates the rule that are defined earlier.

QUESTION NO 4: How functions works step by step?

ANSWER:

SemanticAnalyzer: Following are the steps involve in SemanticAnalyzer function:

1. Loop through Tokens: The code uses a for loop to iterate over the list of tokens.

2. Rule Selection: Inside the loop, it checks the current token against three rules (Rule1, Rule2, and Rule3) to determine which rule to apply. If any of the rules indicate a match with the current token (by returning a string starting with "Start"), it sets the selectedRule variable accordingly.

3. Rule Application: If a rule is selected, it applies the corresponding rule to the current token. If the rule application returns a state that starts with "Ok" or "Error," it adds the state to the errors list and resets selectedRule to 0, indicating that the next token should be processed using rule selection.

4. Backtracking for Unmatched Token: If selectedRule is still set after processing all tokens, it means that the last rule is still active. In that case, it applies the corresponding rule to a new (empty) token and adds the result to the errors list.

5. Errors: The final result is a list of error messages in the errors list, indicating issues found during semantic analysis.

QUESTION NO 05: What challenges your faces during the project?

ANSWER:

Implementing complex semantic rules:

Handling semantic rules beyond basic operations can be challenging. This includes function calls, loops, conditional statements, and type checking. Ensuring correct code execution involves careful rule implementation.