Lab terminal CC

TO:

Lecturer: Mr. Bilal Bukhari

BY:

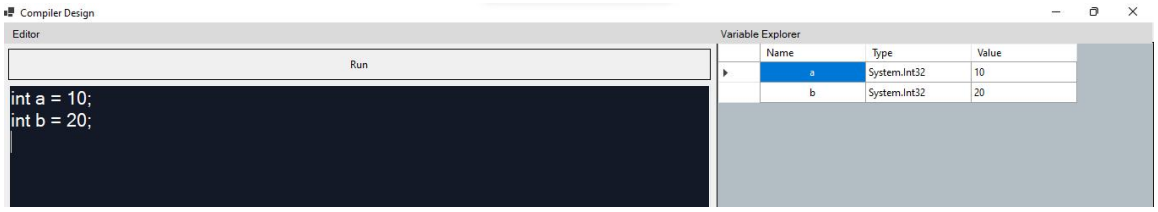
Muhammad Naqeeb
(Fa20-bcs-012)

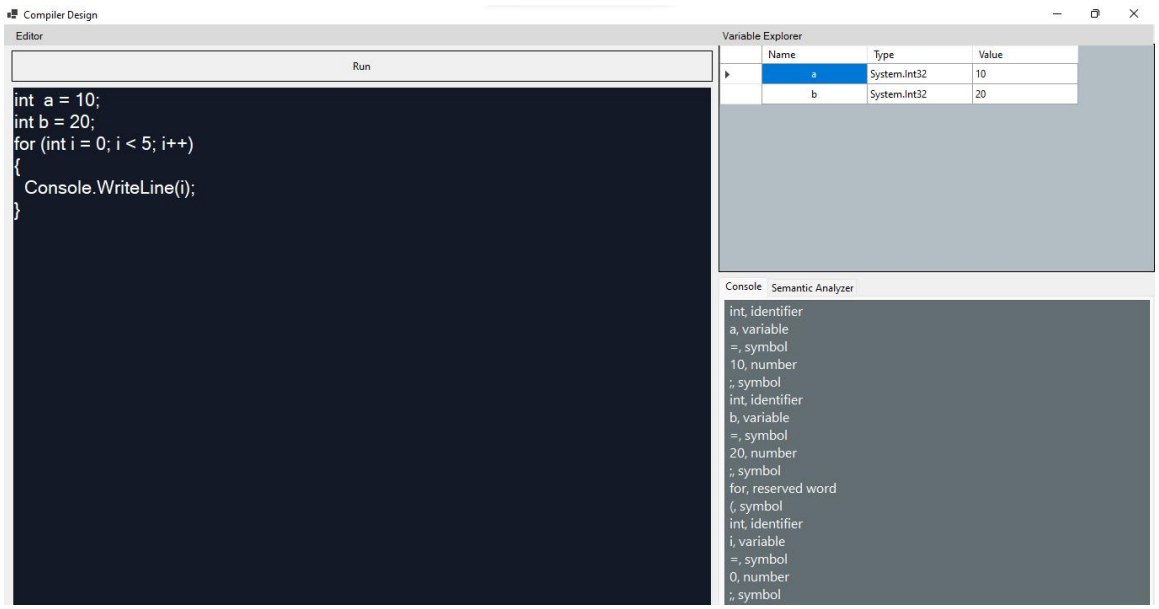
Question # 2

Lexical Analyzer

Lexical Analysis is the first phase of compiler also known as scanner. It converts the **input program into a sequence of Tokens**.It can be implemented with the regular expressions.

The Lexical Analyzer's key role is the identification and classification of language constructs through regular expressions, resulting in a stream of tokens that forms the basis for subsequent phases.





Token Types Enumeration:

Defines an enumeration TokenType to represent different types of tokens (e.g., Keyword, Identifier, Operator, Constant).

```
enum TokenType
{
    Keyword,
    Identifier,
    Operator,
    Constant,
    // Add more as needed
}
```

Token Structure:

Defines a Token structure to hold information about each token, including its type and value.

```
struct Token
{
    public TokenType Type;
    public string Value;
}
```

LexicalAnalyzer Class:

Creates a class named LexicalAnalyzer to encapsulate the functionality.

```
class LexicalAnalyzer
{
    // ...
}
```

Analyze Method:

- The Analyze method takes the source code as input and returns a list of tokens.
- Initializes a list to store the tokens.
- Defines regular expressions for various token patterns (e.g., keywords, identifiers, operators, constants).

```
static List<Token> Analyze(string sourceCode)
{
    List<Token> tokens = new List<Token>();

    // Regular expressions for token patterns
    Regex keywordRegex = new Regex(@"¥b(if|else|while|int|float)¥b");
    Regex identifierRegex = new Regex(@"[a-zA-Z_][a-zA-Z0-9_]*");
    Regex operatorRegex = new Regex(@"[+¥-*/=]");
    Regex constantRegex = new Regex(@"¥b¥d+(¥.¥d+)?¥b");

   }
```

Tokenization Loop:

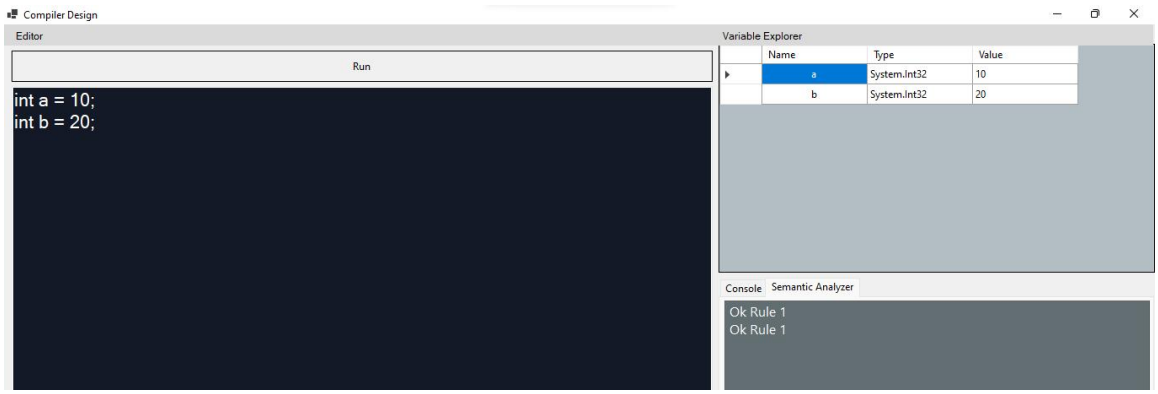
- Uses a StringReader to iterate through each character in the source code.
- Checks each character against the defined regular expressions to identify token patterns.
- Adds recognized tokens to the list.

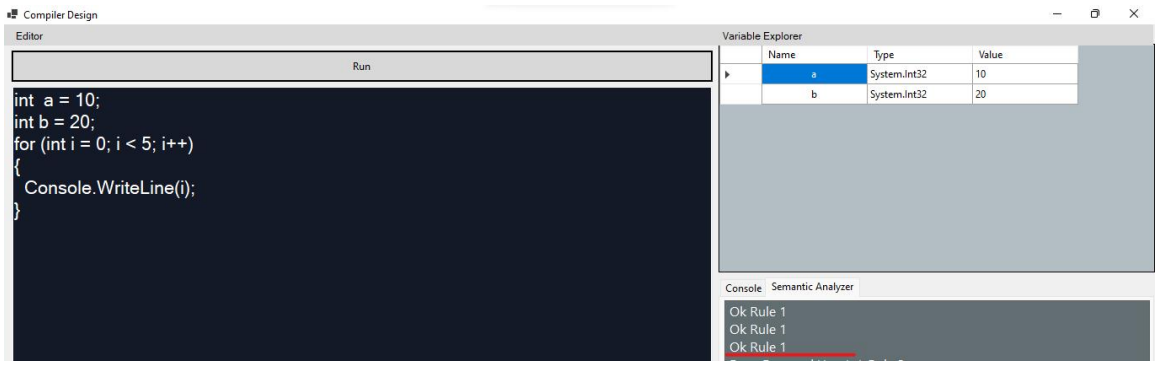
```
using (StringReader reader = new StringReader(sourceCode))
{
    int currentChar;
    while ((currentChar = reader.Read()) != -1)
    {
        char currentCharValue = (char)currentChar.ToString();

        // Token matching
        if (char.IsWhiteSpace(currentCharValue))
        {
            // Skip whitespace
        }
        else if (keywordRegex.IsMatch(currentCharValue.ToString()))
        {
            tokens.Add(new Token { Type = TokenType.Keyword, Value =
currentCharValue.ToString() });
        }
        else if (identifierRegex.IsMatch(currentCharValue.ToString()))
        {
            tokens.Add(new Token { Type = TokenType.Identifier, Value =
currentCharValue.ToString() });
        }
        else if (operatorRegex.IsMatch(currentCharValue.ToString()))
        {
            tokens.Add(new Token { Type = TokenType.Operator, Value =
currentCharValue.ToString() });
        }
        else if (constantRegex.IsMatch(currentCharValue.ToString()))
        {
```

```
            tokens.Add(new Token { Type = TokenType.Constant, Value =
currentCharValue.ToString() });
        }
        // Add more token types and patterns as needed
        else
        {
            // Handle unrecognized characters or tokens
            Console.WriteLine($"Error: Unrecognized character '{currentCharValue}'");
        }
    }
}
```

Semantic analysis

Semantic analysis is the task of ensuring that the declarations and statements of a program are semantically correct, i.e,that their meaning is clear and consistent with the way in which control structures and data types are supposed to be used.





Namespace and Class Definition:

The code defines a class named SemanticAnalyzer within the System namespace.

```
using System;
using System.Collections.Generic;

class SemanticAnalyzer
```

```
{
    // ...
}
```

Symbol Table:

A dictionary named symbolTable is declared to store variable symbols along with their types.

```
Dictionary<string, string> symbolTable = new Dictionary<string, string>();
Token Structure:
A Token structure is defined to represent tokens, where each token has a type and a value.
csharp
Copy code
struct Token
{
    public string Type;
    public string Value;
}
```

AnalyzeSemantics Method:

This method performs semantic analysis on a list of tokens.

It iterates through each token, checking for specific semantic rules.

For example, if a token is a keyword "int," it calls DeclareVariable to handle the declaration of an integer variable.

If an identifier is encountered and it is not found in the symbol table, it reports an error for an undeclared variable.

```
static void AnalyzeSemantics(List<Token> tokens)
{
    foreach (Token token in tokens)
    {
        if (token.Type == "Keyword" && token.Value == "int")
        {
            DeclareVariable(tokens);
        }
        else if (token.Type == "Identifier" && !symbolTable.ContainsKey(token.Value))
        {
            Console.WriteLine($"Error: Undeclared variable '{token.Value}'");
        }
            }
}
```

DeclareVariable Method:

This method handles the declaration of integer variables.

It assumes that the next token is an identifier and adds the variable to the symbol table.

```
static void DeclareVariable(List<Token> tokens)
{
    Token nextToken = GetNextToken(tokens);

    if (nextToken.Type == "Identifier")
    {
        symbolTable[nextToken.Value] = "int";
```

```
    }
    else
    {
        Console.WriteLine($"Error: Expected identifier after 'int', but found
'{nextToken.Value}'");
    }
}
```

GetNextToken Method:

This method simulates obtaining the next token from the list of tokens for processing.

```
static Token GetNextToken(List<Token> tokens)
{
    Token nextToken = tokens[0];
    tokens.RemoveAt(0);
    return nextToken;
}
```