

**CC LAB Terminal PROJECT**

**MISBAH NAZIR (FA20-BCS-015)**

**SOFIA AHMED (FA20-BCS-053)**

**BCS 7A**

**Submitted to: Mr Bilal Haider**

**Q1:Brief Description of the project:**

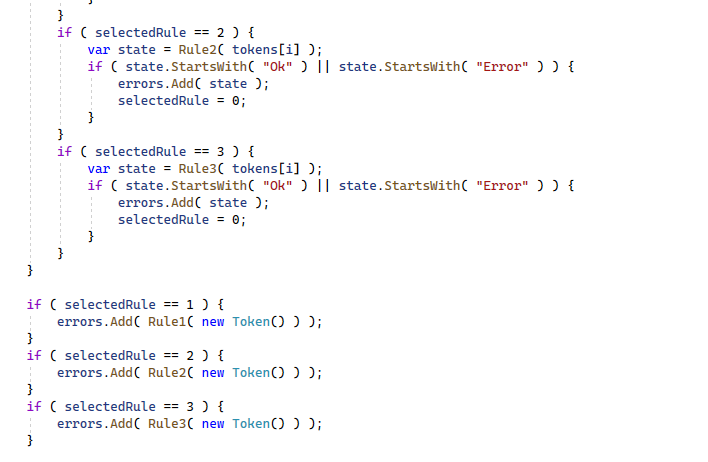
In the context of this laboratory terminal project, we've crafted a semantic analyzer to pinpoint errors within the code block. The SemanticAnalyzer() function is designed to process a list of tokens obtained by the Scanner. It evaluates whether the sequence of tokens adheres to any of the three rules outlined in the project description using Top-Down Parsing. Subsequently, the function generates a list enumerating any errors detected. To provide a concise view, here's a summarized version of the SemanticAnalyzer() function with collapsed code.

Within the SemanticAnalyzer() function, a primary "for" loop systematically traverses the provided list of tokens. It assesses each token to determine which of the three rules is applicable to initiate acceptance. Once a rule commences acceptance, it becomes the chosen rule for evaluation. This selected rule persists until it either reports an error or signifies successful validation with an "Ok" status, indicating that the preceding sequence of tokens adhered to the rule accurately.



For each rule, a token is extracted during every loop iteration, and the previously observed token is stored in a variable called "previousInput." Within each rule, the upper portion delineates the conditions triggering the initiation, termination, and continuation of that rule's application. Meanwhile, the lower part of the rule handles the identification of errors and provides feedback on what the rule anticipated instead.

Roles 1, 2, and 3:



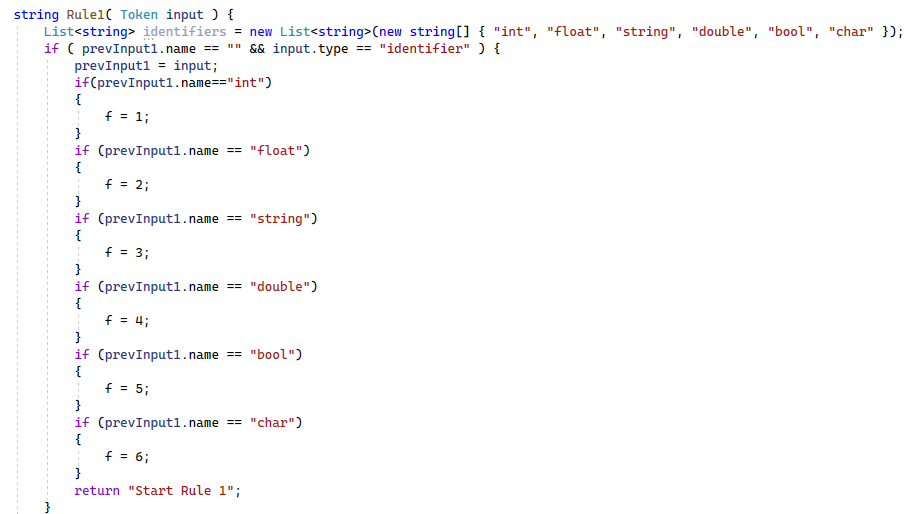
**Q2:**

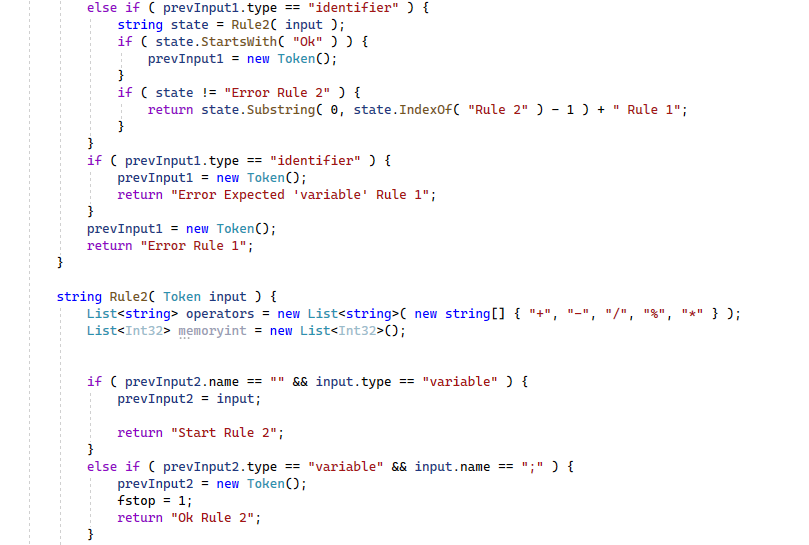
**Functional Explanation**

This function serves the purpose of transforming a user-input sequence of characters, representing a code block, into a sequence of tokens. Instead of employing DFA or splitting methods, we've opted for the regular expression match approach. This choice offers the advantage of specifying the desired tokens in a manner analogous to the provided image and is notably straightforward.

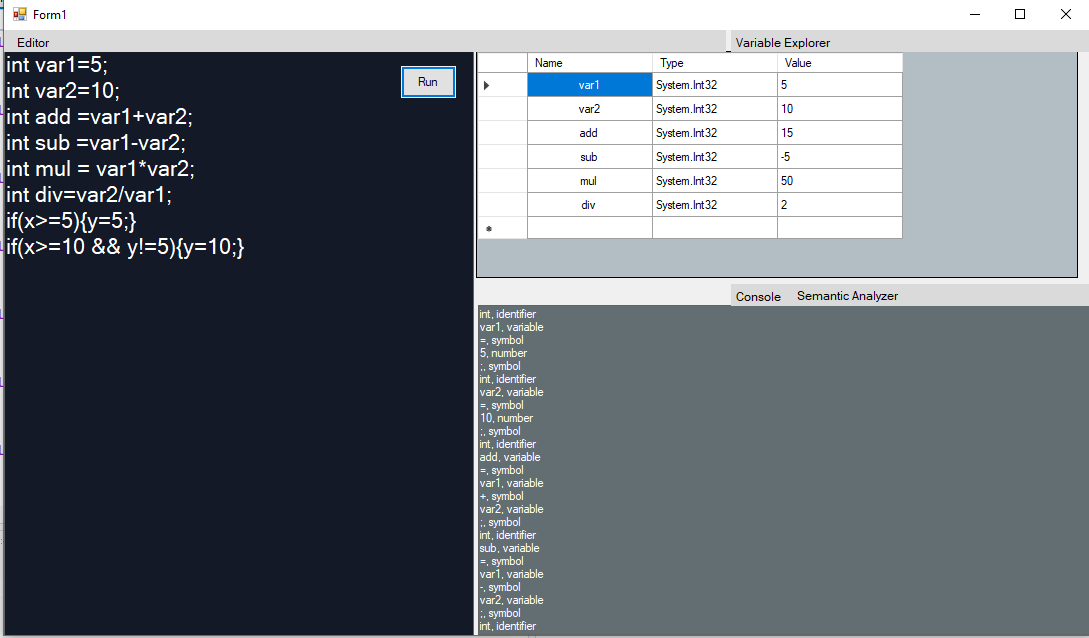
The "Splitter" component returns a list of matched strings.

To facilitate subsequent use, a Token class is established to store both the name and type of tokens.

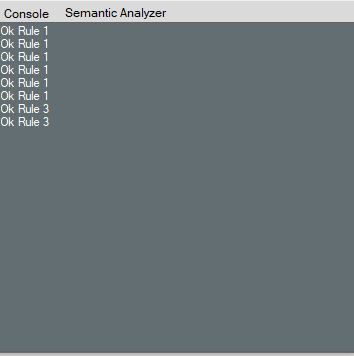




**Q3:User Input:**

****

**Output:**

****

**Semantic Analyzer**

Checking the lines with roles

Ok Rule 1 Ok Rule 1 Ok Rule 1 Ok Rule 1 Ok Rule 1 Ok Rule 1 Ok Rule 1 Ok Rule 3 Ok Rule 3

If we replace by this input then,

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

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

with

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

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

**Q4:**

**How Function Works:**

During this stage, we store the value of a variable in memory, and each variable is displayed on the data grid view. Our class encompasses a variable and a list of class variables.

To begin, in the rule1 function, a flag (denoted as "f") is set to 1. This flag serves to indicate that the identifier is an integer.

Within the rule 2 function, we initialize the "firstVar" variable to capture the variable preceding the "=" symbol, and the "prevVar" variable is employed to capture the initial variable following the "=" symbol.

In this condition the variable becomes any integer value

For example, in the condition "int x=3;," the integer variable "x" assumes the value of another integer variable. Similarly, in the condition "int x= y;," the variable "x" takes on the value of the variable "y."

In this scenario, we capture the numerical value preceding the operator. The identified number then takes on the role of this numerical value.

In this situation, we capture the variable preceding the operator. The variable designated as "prevVar" then assumes the role of this identified variable.

In this scenario, we extract the values associated with the "+" operator in the following two cases:

* When the expression involves a numerical value added to another numerical value (Num + Num).
* When an integer variable is added to a numerical value (Integer variable + Num).

In this situation, we extract the values related to the "-" operator in the following two cases:

* When the expression involves subtracting a numerical value from another numerical value (Num - Num).
* When subtracting a numerical value from an integer variable (Integer variable - Num).

In this case, we extract the values associated with the "\*" operator in two specific instances:

* When the expression involves multiplying a numerical value by another numerical value (Num \* Num).
* When multiplying an integer variable by a numerical value (Integer variable \* Num).

In this circumstance, we capture the values corresponding to the "/" operator in two distinct scenarios:

* When the expression involves dividing a numerical value by another numerical value (Num / Num).
* When dividing an integer variable by a numerical value (Integer variable / Num).

In this scenario, we extract the values associated with the "%" operator in two specific cases:

* When the expression involves calculating the modulus of a numerical value by another numerical value (Num % Num).
* When calculating the modulus of an integer variable by a numerical value (Integer variable % Num).

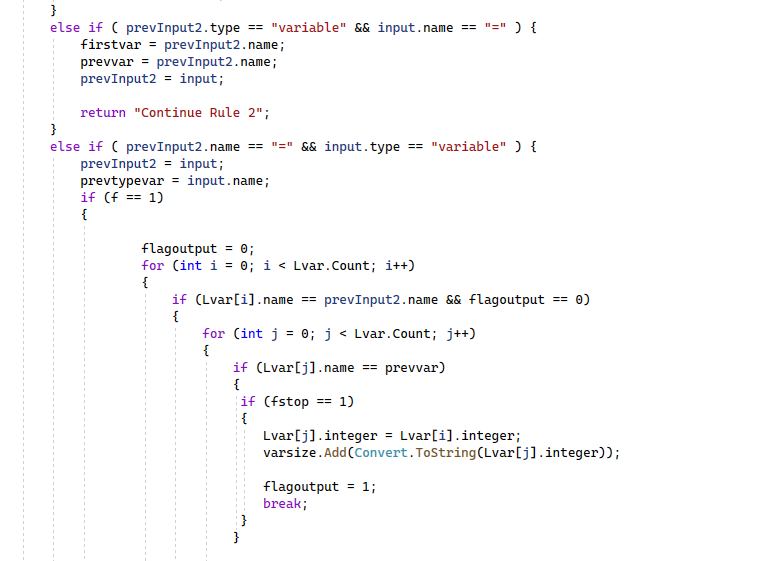
In this condition, we consider the values associated with the "+" operator when it involves the addition of two integer variables (Integer variable + Integer variable).

In this case, we consider the values associated with the "-" operator when it involves subtracting one integer variable from another (Integer variable - Integer variable).

In this scenario, we consider the values associated with the "\*" operator when it involves the multiplication of two integer variables (Integer variable \* Integer variable).

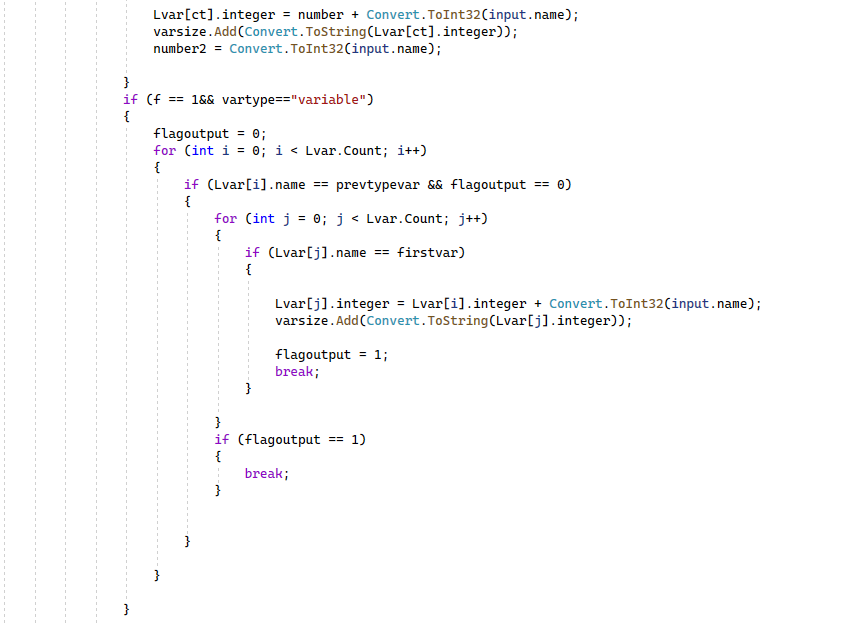
n this situation, we consider the values associated with the "/" operator when it involves dividing one integer variable by another (Integer variable / Integer variable).

In this circumstance, we consider the values associated with the "%" operator when it involves calculating the modulus of one integer variable by another (Integer variable % Integer variable).







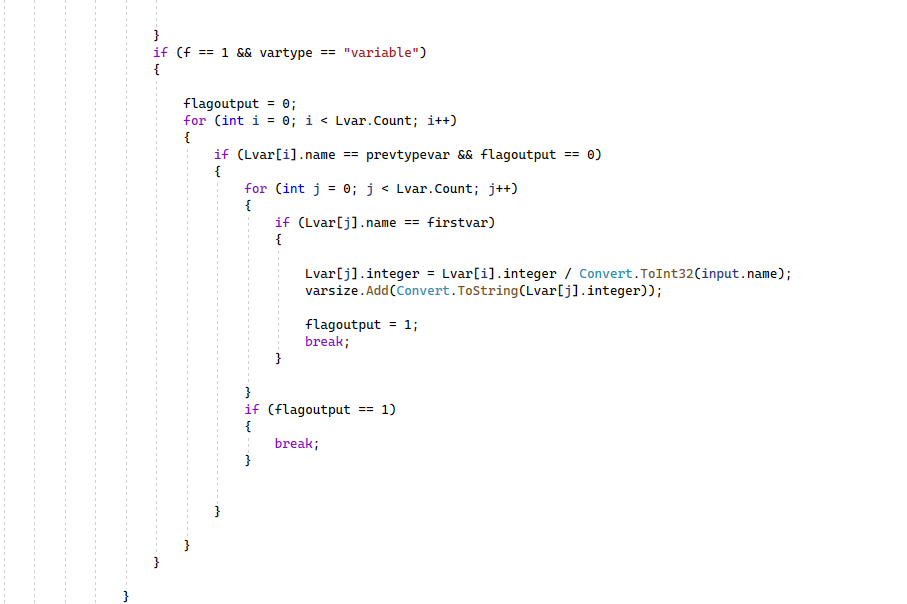








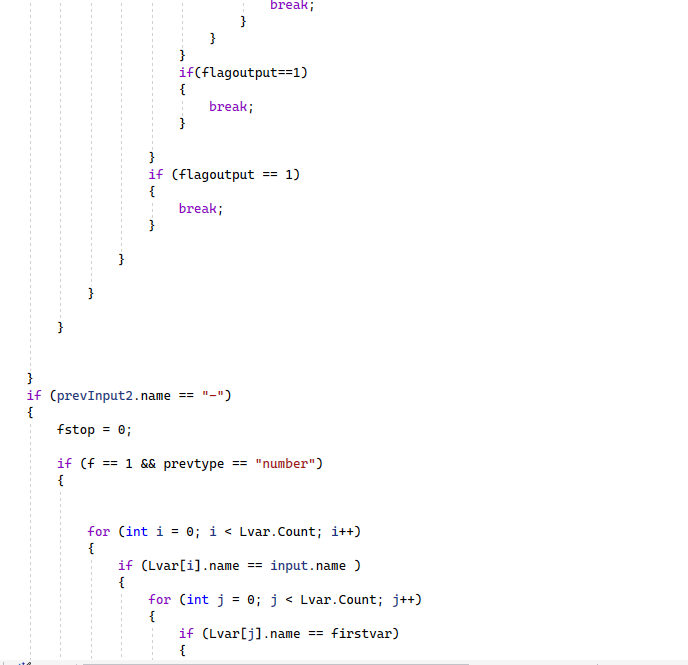


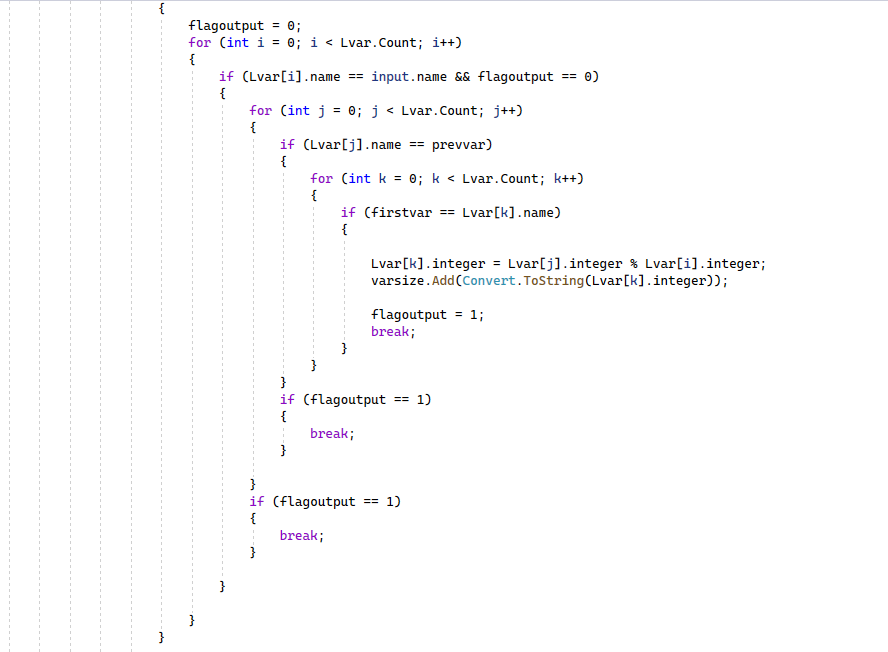
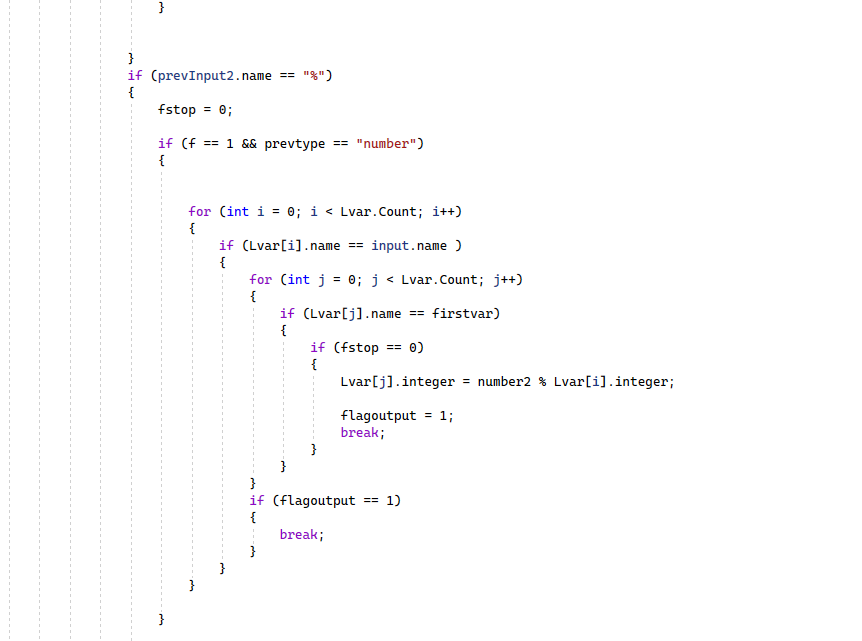
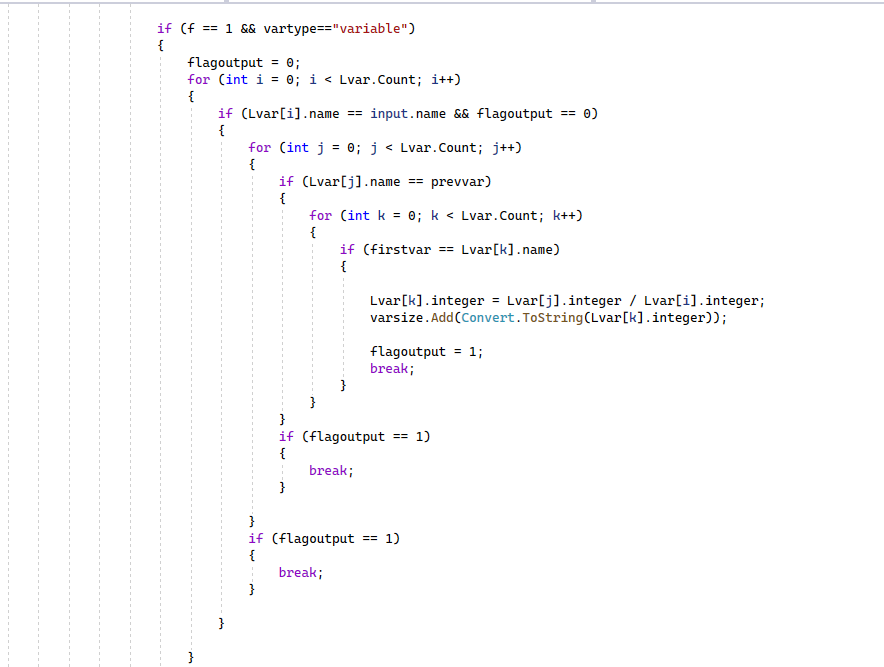
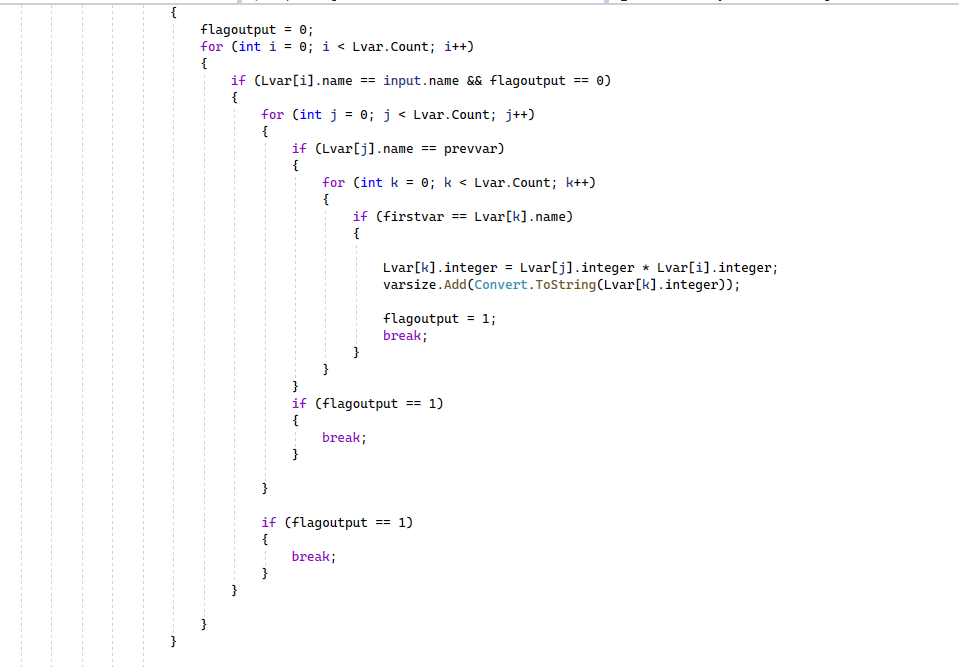
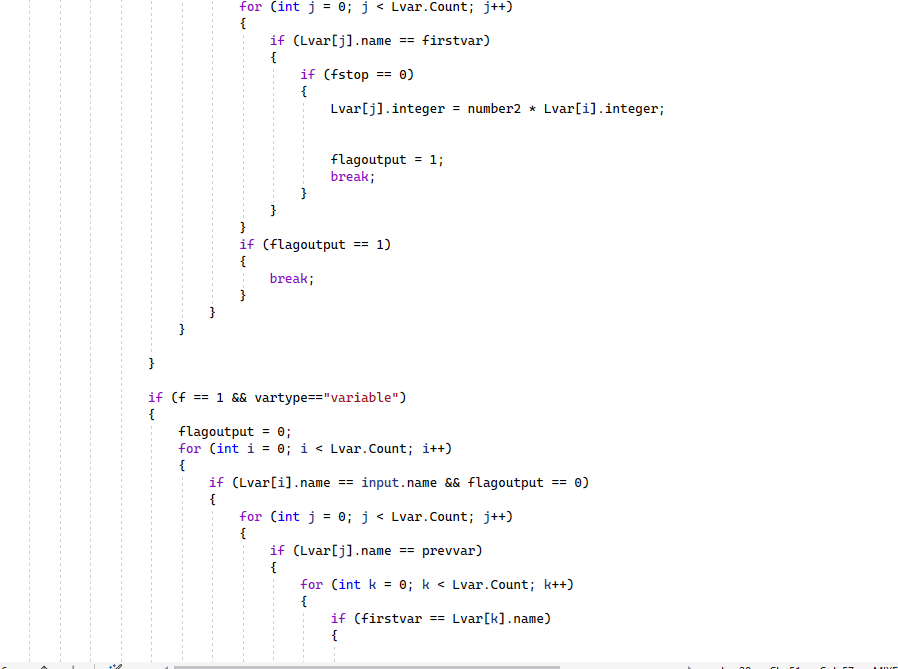
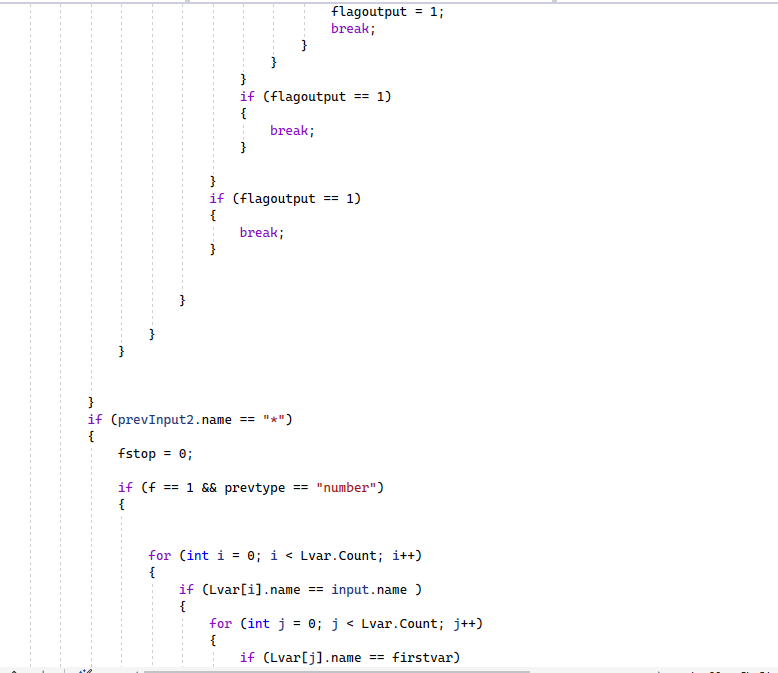


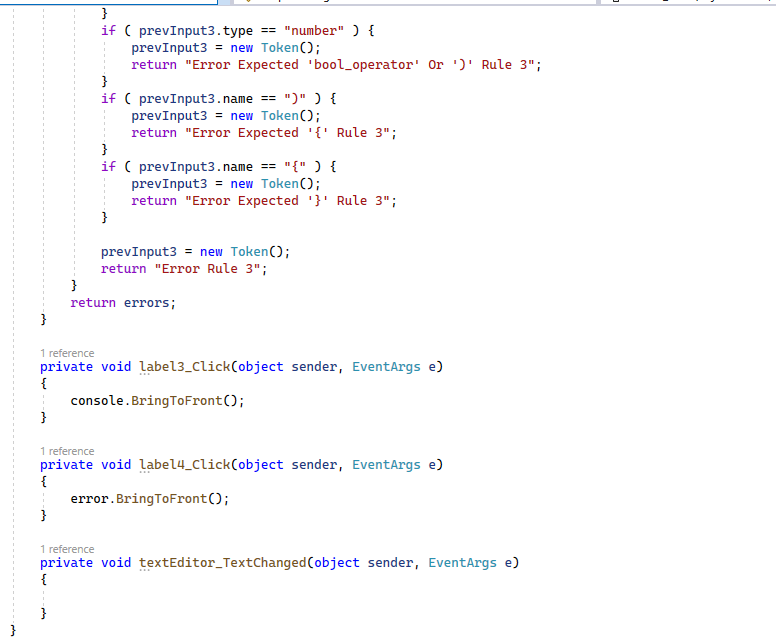




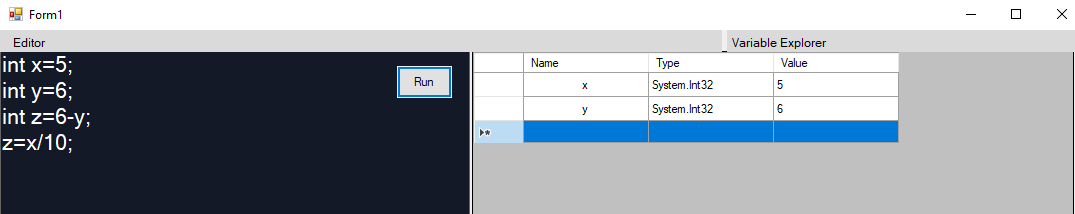








**Test Case:**

****

**Challenges Encountered During the Project:**

In the process of developing this project, which incorporates a semantic analyzer, we faced several challenges. The primary objective of this project is to receive user input, scan the entire code, and provide feedback on whether the input is correct. Additionally, it offers step-by-step explanations of the given input statements.

The main challenges we encountered during the project were associated with handling syntax and logical errors. Understanding the intricacies of language grammar and performing lexical analysis proved to be particularly challenging. Initially, we had to gain a comprehensive understanding of these aspects before proceeding to construct the semantic analyzer. Eventually, after overcoming these hurdles, the project ran successfully.