Lab Manual

CSC441-Compiler Construction

CUI

Department of Computer Science
Islamabad Campus

Lab Contents:

Fundamentals of C#; Implemntation of Lexical Analyzer: Recognition of operators/variables/keywords; Recognition of Constants/Special Symbols/Integers; Input Buffering scheme; Construction of Symbol Table;Top-down Parser: Finding the first set of a given grammar;Finding the Follow set of a given grammar;Bottom-up Parser: Implementation of DFA from the given Grammar; Parsing Stack using SLR Parsing Table;Syntax-Directed Translation; Semantic Analyzer;Integration of Lexical Analyzer and symbol table (Ph-1);Integration of Semantic Analyzer with Ph-1. Mini-Compiler.

Student Outcomes (SO)

S.#	Description
2	Identify, formulate, research literature, and solve complex computing problems reaching substantiated conclusions using fundamental principles of mathematics, computing sciences, and relevant domain disciplines
3	Design and evaluate solutions for complex computing problems, and design and evaluate systems, components, or processes that meet specified needs with appropriate consideration for public health and safety, cultural, societal, and environmental considerations
4	Create, select, adapt and apply appropriate techniques, resources, and modern computing tools to complex computing activities, with an understanding of the limitations.
5	Function effectively as an individual and as a member or leader in diverse teams and in multi-disciplinary settings.

Intended Learning Outcomes

Sr.#	Description	Blooms Taxonomy Learning Level	SO
CLO-4	Implement a lexical, syntax and semantic analyzer.	*Applying*	2,4
CLO-5	Develop a mini compiler for a language.	*Creating*	3,5

Lab Assessment Policy

The lab work done by the student is evaluated using rubrics defined by the course instructor, viva-voce, project work/performance. Marks distribution is as follows:

Assignments	Lab Mid Term Exam	Lab Terminal Exam	Total
25	25	50	100

Note: Midterm and Final term exams must be computer based.

List of Labs

Lab 01

Fundamentals Of C#

Objective:

This Lab will provide you an introduction to C# syntax so that you can easily design compiler in C#.

Activity Outcomes:

On completion of this lab students will be able to

- Doing arithmetic operations in C#
- Displaying and retrieving values from DatagridView in C#
- Implementing Stack data structure in C#

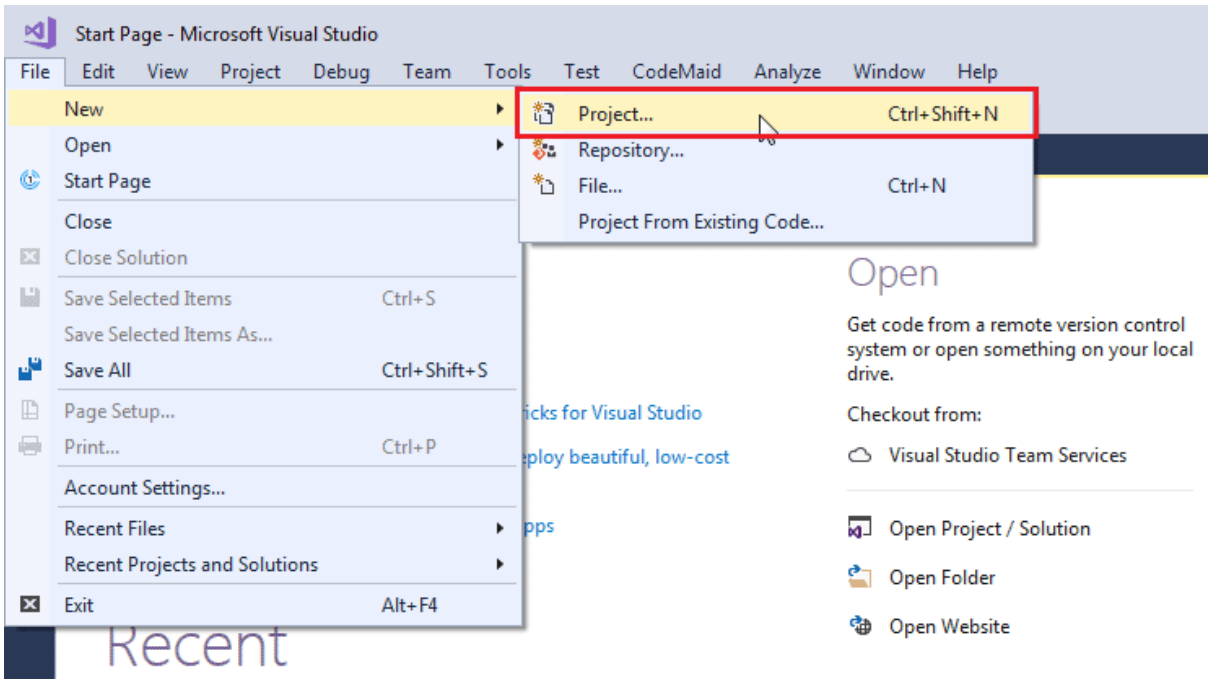
Instructor Note:

As a pre-lab activity, read chapter 01 from the book "C# 8.0 and .NET Core 3.0 – Modern Cross-Platform Development" 4th Edition by Mark J.Price

1) Useful Concepts

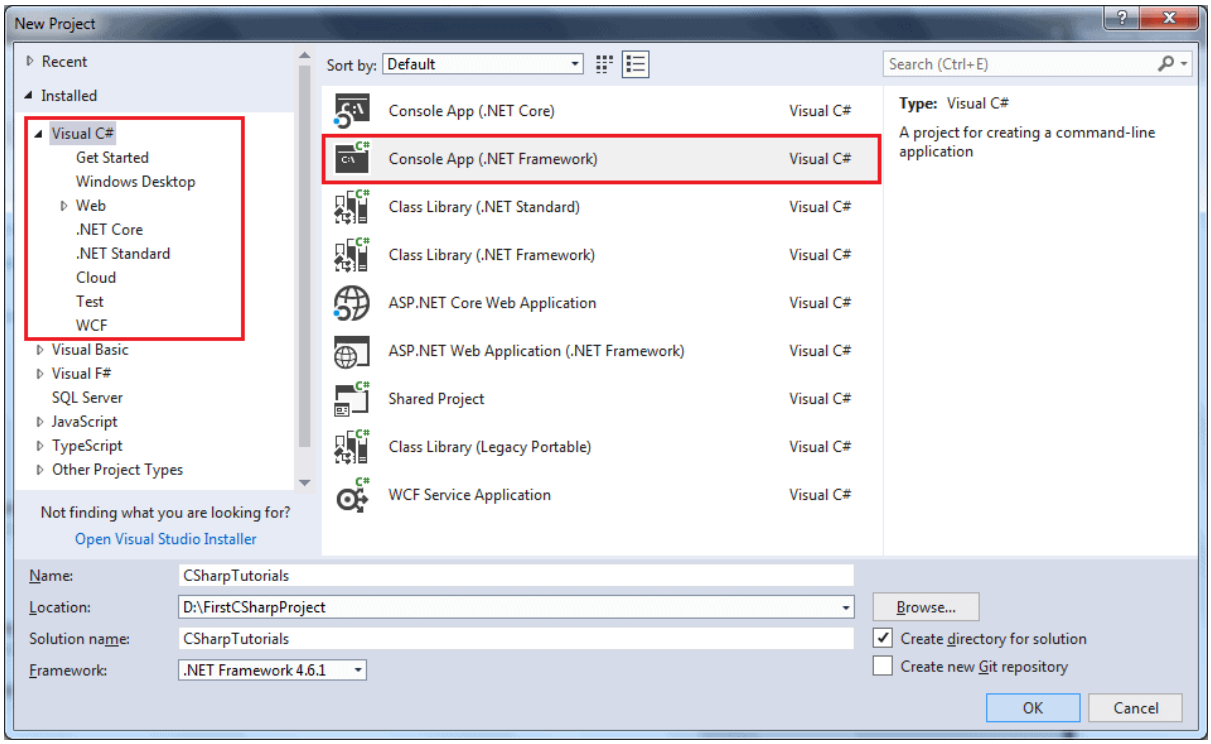
C# can be used in a window-based, web-based, or console application. To start with, we will create a console application to work with C#.

Open Visual Studio (2017 or later) installed on your local machine. Click on File -> New Project... from the top menu, as shown below.



Create a New Project in Visual Studio 2019

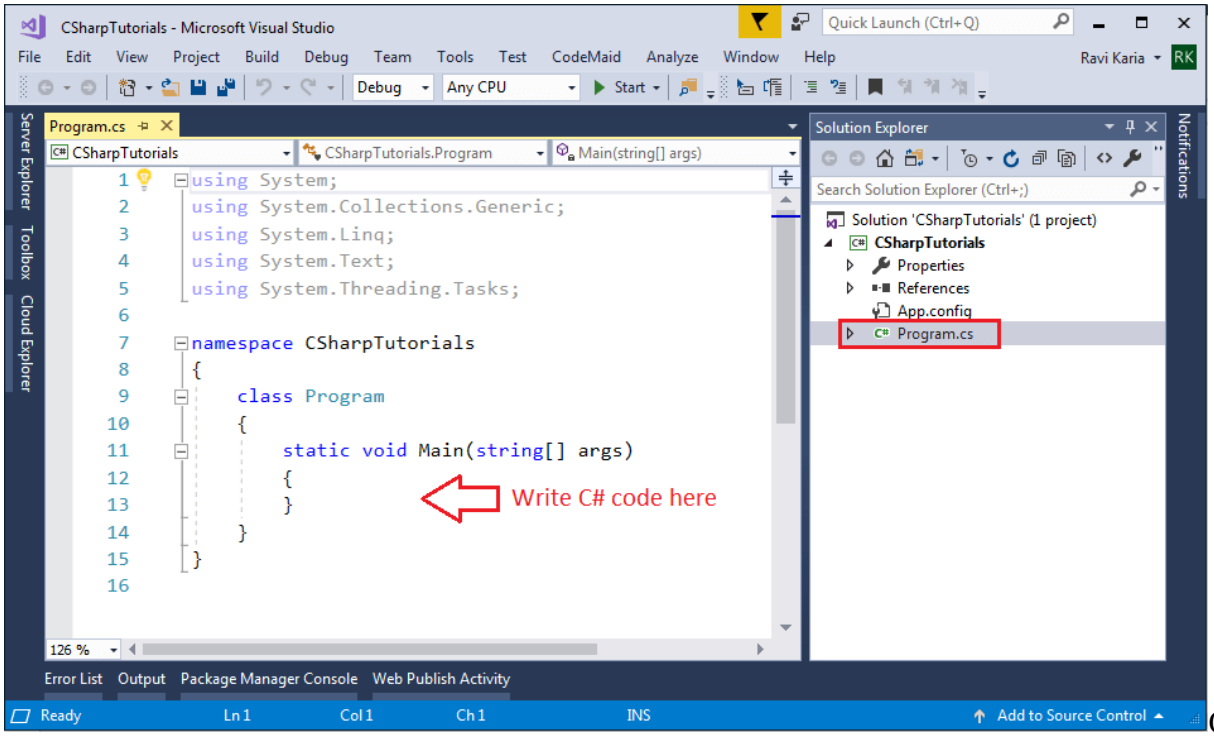
From the **New Project** popup, shown below, select Visual C# in the left side panel and select the Console App in the right-side panel.



Select Visual C# Console App Template

In the name section, give any appropriate project name, a location where you want to create all the project files, and the name of the project solution.

Click OK to create the console project. **Program.cs** will be created as default a C# file in Visual Studio where you can write your C# code in Program class, as shown below. (The .cs is a file extension for C# file.)



C#

Console Program

Every console application starts from the Main() method of the Program class. The following example displays "Hello World!!" on the console.

```
Example: C# Console Application
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CSharpTutorials
{
    class Program
    {
        static void Main(string[] args)
        {
            string message = "Hello World!!";
            Console.WriteLine(message);
        }
    }
}
```

1. Every .NET application takes the reference of the necessary .NET framework namespaces that it is planning to use with the using keyword, e.g., using System.Text.
2. Declare the namespace for the current class using the namespace keyword, e.g., namespace CSharpTutorials.FirstProgram
3. We then declared a class using the class keyword: class Program
4. The Main() is a method of Program class is the entry point of the console application.
5. String is a data type.
6. A message is a variable that holds the value of a specified data type.
7. "Hello World!!" is the value of the message variable.
8. The Console.WriteLine() is a static method, which is used to display a text on the console.

Note:

Every line or statement in C# must end with a semicolon (;).

Compile and Run C# Program

To see the output of the above C# program, we have to compile it and run it by pressing Ctrl + F5 or clicking the Run button or by clicking the "Debug" menu and clicking "Start Without Debugging". You will see the following output in the console:

Output:

```
Hello World!!
```

Declaration of varaiables in C#

data_type variable_name = value;

Example: int age = 20;

Declaration of Arrays in C#

```
int[] arr = new int[6]; // one dimensional array of size 6
```

Functions

```
int sum(int x, int y)
{
 return x+y;
 }
```

Loops

```
for(int i=1; i<=10; i++)
{ Console.WriteLine(i);}
```

while(Condition)

```
{  Statements; }
```

if(condition)

```
{ Statements;  }
```

Above is the basic code items that you will use almost in every C# code.

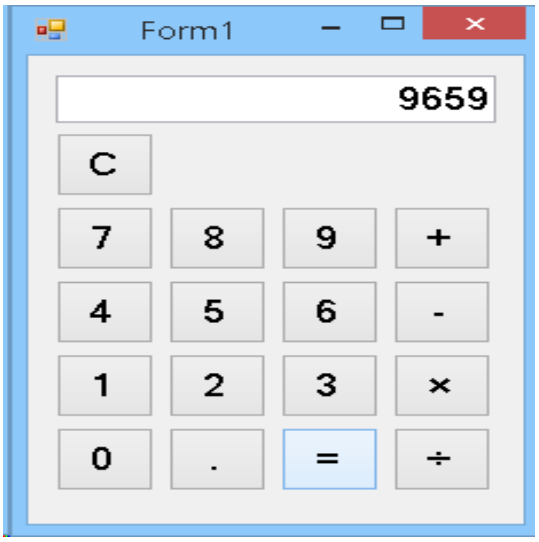
2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	***25 mins***	***Low***	***CLO-5***
Activity 2	***25 mins***	***Low***	***CLO-5***
Activity 3	***15 mins***	***Low***	***CLO-5***

Activity 1:
Design a calculator in C# Windows Form Application

Solution:

- Open a Windows Form Application
- Drag some buttons and a textbox from Toolbox onto Form. Example is provided below:



- Copy and paste the code provided below into your class.

```
using System;
using System.Windows.Forms;

namespace RedCell.App.Calculator.Example
{
    public partial class Form1 : Form
    {
        private double accumulator = 0;
        private char lastOperation;

        public Form1()
        {
            InitializeComponent();
        }

        private void Operator_Pressed(object sender, EventArgs e)
        {
            // An operator was pressed; perform the last operation
and store the new operator.
            char operation = (sender as Button).Text[0];
            if (operation == 'C')
            {
                accumulator = 0;
            }
            else
            {
                double currentValue = double.Parse(Display.Text);
                switch (lastOperation)
                {
                    case '+': accumulator += currentValue; break;
                    case '-': accumulator -= currentValue; break;
                    case '×': accumulator *= currentValue; break;
                    case '÷': accumulator /= currentValue; break;
                    default: accumulator = currentValue; break;
                }
```

```
            }

            lastOperation = operation;
            Display.Text = operation == '=' ?
accumulator.ToString() : "0";
        }

        private void Number_Pressed(object sender, EventArgs e)
        {
            // Add it to the display.
            string number = (sender as Button).Text;
            Display.Text = Display.Text == "0" ? number :
Display.Text + number;
        }
    }
}
```

1. There are two kinds of buttons, **numbers** and **operators**.
2. There is a **display** that shows entries and results.
3. There is an accumulator variable to store the accumulated value.
4. There is a lastOperation variable to store the last operator, because we won't evaluate until another operator is pressed.

When a number is pressed, it is added to the end of the number currently on the display. If a 0 was on the display we replace it, just to look nicer.

If the C operator is pressed, we reset the accumulator to 0.

Otherwise we perform the last operation against the accumulator and the currently entered number. If there wasn't a lastOperation, then we must be starting a new calculation, so we set the accumulator to the currentValue as the first operation.

Activity 2:
Display and retrieve data from data grid view

Solution:

- Displaying Data in Data Grid View

    1) Create a windows Form application
    2) Drag data grid view tool and a button from toolbox on form.
    3) Copy and paste the code provided below behind the button.

```
using System;
using System.Data;
using System.Windows.Forms;
using System.Data.SqlClient;

namespace WindowsApplication1
{
   public partial class Form1 : Form
```

```
        {
            public Form1()
            {
                InitializeComponent();
            }
```

```
        private void button1_Click(object sender, EventArgs e)
            {
                dataGridView1.ColumnCount = 3;
                dataGridView1.Columns[0].Name = "Product ID";
                dataGridView1.Columns[1].Name = "Product Name";
                dataGridView1.Columns[2].Name = "Product Price";

                string[] row = new string[]
        { "1", "Product 1", "1000" };
                dataGridView1.Rows.Add(row);
                row = new string[] { "2", "Product 2", "2000" };
                dataGridView1.Rows.Add(row);
                row = new string[] { "3", "Product 3", "3000" };
                dataGridView1.Rows.Add(row);
                row = new string[] { "4", "Product 4", "4000" };
                dataGridView1.Rows.Add(row);
            }
        }
    }
```

- Data Retrieval from Data Grid View
    1) First populate the data grid view with some data
    2) You can retrieve data from data grid view via loops.

```
for (int rows = 0; rows < dataGrid.Rows.Count; rows++)
{
     for (int col= 0; col < dataGrid.Rows[rows].Cells.Count;
col++)
    {
        string value =
dataGrid.Rows[rows].Cells[col].Value.ToString();

    }
}
example without using index

foreach (DataGridViewRow row in dataGrid.Rows)
{
    foreach (DataGridViewCell cell in row.Cells)
    {
        string value = cell.Value.ToString();

    }
}
```

Activity 3:
Implement stack data structure

Solution:

```
using System;
using System.Collections;

namespace CollectionsApplication
{
   class Program
   {
            static void Main(string[] args)
      {
         Stack st = new Stack();

         st.Push('A');
         st.Push('M');
         st.Push('G');
         st.Push('W');

         Console.WriteLine("Current stack: ");
         foreach (char c in st)
         {
            Console.Write(c + " ");
         }

         Console.WriteLine();

         st.Push('V');
         st.Push('H');
         Console.WriteLine("The next poppable value in stack: {0}",
st.Peek());
         Console.WriteLine("Current stack: ");
         foreach (char c in st)
         {
            Console.Write(c + " ");
         }

         Console.WriteLine();

         Console.WriteLine("Removing values ");
         st.Pop();
         st.Pop();
         st.Pop();

         Console.WriteLine("Current stack: ");
         foreach (char c in st)
         {
            Console.Write(c + " ");
         }
      }
   }
}
```

When the above code is compiled and executed, it produces the following result:

```
Current stack:
W G M A
The next poppable value in stack: H
Current stack:
H V W G M A
Removing values
Current stack:
G M A
```

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1:

Implement scientific calculator with Sine, Cosine, Tangent, Log Functions.

Lab Task 2:

Insert values into Data grid View at run time

Lab 02

Lexical Analyzer:Recognition of Operators/Variables/keywords.

Objective:

This lab is designed to demonstrate the implementation of tokenization using regular expression

Activity Outcomes:

This lab teaches you

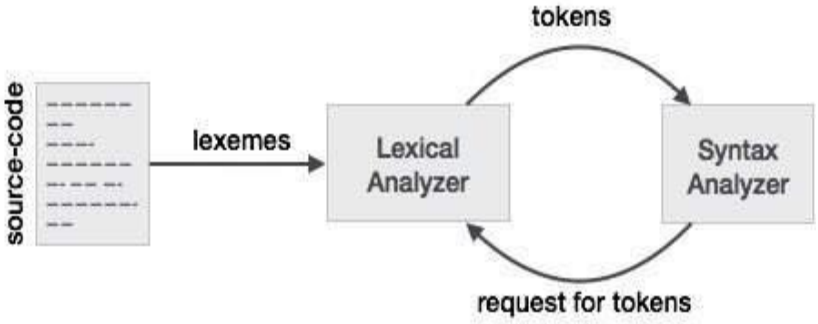
- How to use regular expressions for pattern matching.
- How to recognize operators from a source program written in a high-level language
- How to recognize variables from a source program written in a high-level language

Instructor Note:

As for this lab activity, read chapter 02 from the book "Compiler Construction-Principles and Practices" by Kenneth C. Louden.

1) Useful Concepts

Lexical analysis is the first phase of a compiler. It takes the modified source code from language preprocessors that are written in the form of sentences. The lexical analyzer breaks these syntaxes into a series of tokens, by removing any whitespace or comments in the source code.If the lexical analyzer finds a token invalid, it generates an error. The lexical analyzer works closely with the syntax analyzer. It reads character streams from the source code, checks for legal tokens, and passes the data to the syntax analyzer when it demands.



In programming language, keywords, constants, identifiers, strings, numbers, operators and punctuations symbols can be considered as tokens.

A *regular expression* is a pattern that describes a set of strings. Regular expressions are constructed analogously to arithmetic expressions by using various operators to combine smaller expressions.

The fundamental building blocks are the regular expressions that match a single character. Most characters, including all letters and digits, are regular expressions that match themselves. Any meta character with special meaning may be quoted by preceding it with a backslash. In basic regular expressions the metacharacters "?", "+", "{", "|", "(", and ")" lose their special meaning; instead use the backslashed versions "\?", "\+", "\{", "\|", "\(", and "\)".

Implement Regular Expressions using RegEx class.
Regex file. A file can be parsed with Regex.The Regex can process each line to find all matching parts. This is useful for log files or output from other programs.

Regex g = new Regex(@"\s/Content/([a-zA-Z0-9\-]+?)\.aspx");

The Regex.Replace(String, String, MatchEvaluator) method is useful for replacing a regular expression match if any of the following conditions is true: The replacement string cannot readily be specified by a regular expression replacement pattern. The replacement string results from some processing done on the matched string.

```
string result = rgx.Replace(input, replacement);
```

2) Solved Lab Activities

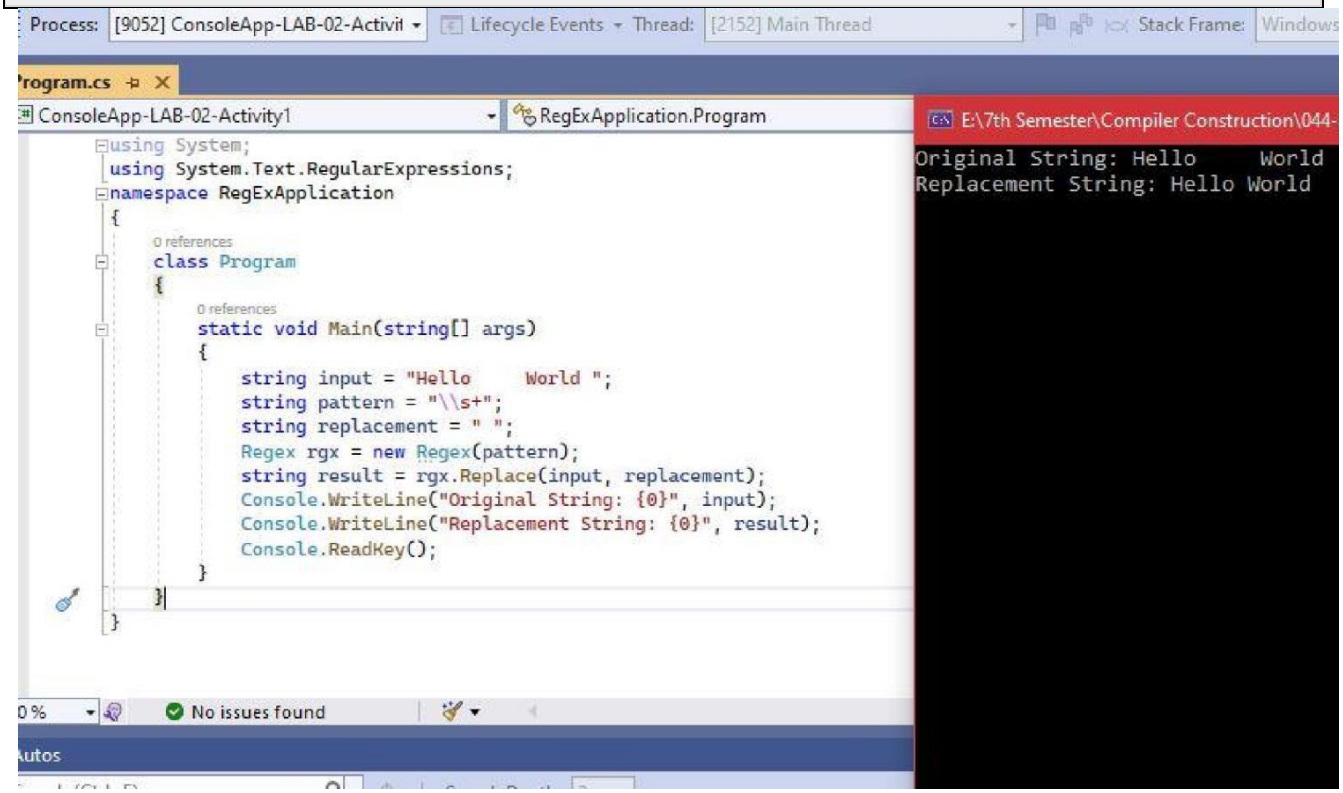
Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	***15 mins***	***Low***	***CLO-5***
Activity 2	***20 mins***	***Low***	***CLO-5***
Activity 3	***20 mins***	***Low***	***CLO-5***

Activity 1:
This activity demonstrates the removal of extra white spaces:

Solution:

```
using System;
using System.Text.RegularExpressions;

namespace RegExApplication
{
   class Program
   {
      static void Main(string[] args)
      {
         string input = "Hello   World   ";
         string pattern = "\\s+";
         string replacement = " ";
         Regex rgx = new Regex(pattern);
         string result = rgx.Replace(input, replacement);

         Console.WriteLine("Original String: {0}", input);
         Console.WriteLine("Replacement String: {0}", result);
```

```
            Console.ReadKey();
        }
    }
}
```



Activity 2:

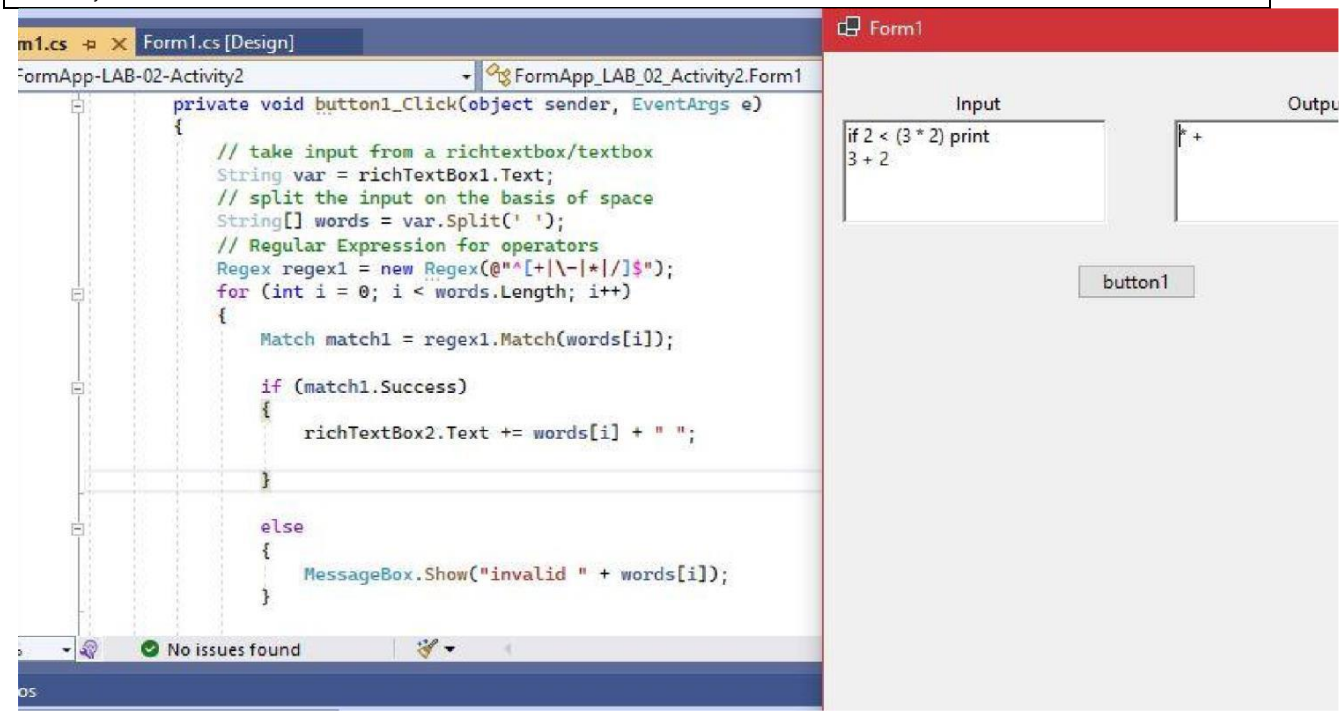
Design regular expression for arithmetic operators:

Regular Expression for operators: [+|*|/|-]

Solution:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Text.RegularExpressions;

namespace Sessional1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
```

```
        private void button1_Click(object sender, EventArgs e)
        {


        // take input from a richtextbox/textbox

            String var = richTextBox1.Text;

            // split the input on the basis of space

            String[] words = var.Split(' ');

            // Regular Expression for operators

            Regex regex1 = new Regex(@"^[+|\-|*|/]$");
for (int i = 0; i < words.Length; i++)
            {
                Match match1 = regex1.Match(words[i]);

                if (match1.Success)
                {
                        richTextBox2.Text += words[i] + " ";

                }

                else {
                    MessageBox.Show("invalid "+words[i]);
                }

            }
        }
    }
```



Activity 3:

Any meta character with special meaning may be quoted by preceding it with a backslash. In basic regular expressions the metacharacters "?", "+", "{", "|", "(", and ")" lose their special meaning; instead use the backslashed versions "\?", "\+", "\{", "\|", "\(", and "\)".

Regular Expression for variables is: [A-Za-z]([A-Za-z|0-9])*

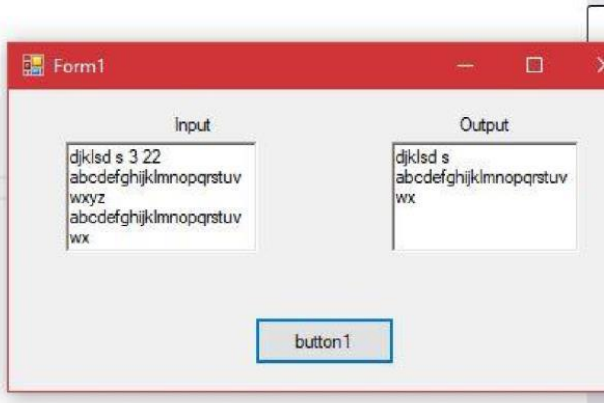
Design a regular expression for variables that should start with a letter, have a length not greater than 10 and can contain combination of digits and letters afterwards.

Solution:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Text.RegularExpressions;

namespace Sessional1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
        // take input from a richtextbox/textbox

            String var = richTextBox1.Text;

            // split the input on the basis of space

            String[] words = var.Split(' ');

            // Regular Expression for variables
            Regex regex1 = new Regex(@"^[A-Za-z][A-Za-z|0-9]{0,24} $";

            for (int i = 0; i < words.Length; i++)
            {
                Match match1 = regex1.Match(words[i]);

                if (match1.Success)
                {
                        richTextBox2.Text += words[i] + " ";

                }

                else {
```

```
                    MessageBox.Show("invalid "+words[i]);
                }
            }
        }
    }
```

```
{
    InitializeComponent();
}

1 reference
private void button1_Click(object sender, EventArgs e)
{
    // take input from a richtextbox/textbox
    String var = richTextBox1.Text;
    // split the input on the basis of space
    String[] words = var.Split(' ');
    // Regular Expression for variables
    Regex regex1 = new Regex(@"^[A-Za-z][A-Za-z|0-9]{0,24}$");

    for (int i = 0; i < words.Length; i++)
    {
        Match match1 = regex1.Match(words[i]);

        if (match1.Success)
        {
            richTextBox2.Text += words[i] + " ";

        }

        else
        {
```



3)Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1:

Design regular expression for logical operators.

Lab Task 2:

Design regular expression for relational operators:

Lab 03

Lexical Analyzer: Recognition of constants, special symbols and integers

Objective:

The objective of this lab is to design a lexical analyzer that can generate tokens by recognizing constants, special symbols and integers.

Activity Outcomes:

On completion of this lab, students will be able to:

- Implement tokernizer that can recognize constants, special symbols and integers from a source program written in a high level language.

Instructor Note:

As for this lab activity, read chapter 02 & 03 from the book “Compiler Construction-Principles and Practices” by Kenneth C. Louden.

1) Useful Concepts:

The fundamental building blocks are the regular expressions that match a single character. Most characters, including all letters and digits, are regular expressions that match themselves. Any meta character with special meaning may be quoted by preceding it with a backslash. In basic regular expressions the metacharacters "?", "+", "{", "|", "(", and ")" lose their special meaning; instead use the backslashed versions "\?", "\+", "\{", "\|", "\(", and "\)".Regular expressions are constructed analogously to arithmetic expressions by using various operators to combine smaller expressions.

2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	***25 mins***	***Medium***	***CLO-5***
Activity 2	***20 mins***	***Low***	***CLO-5***

Activity 1:

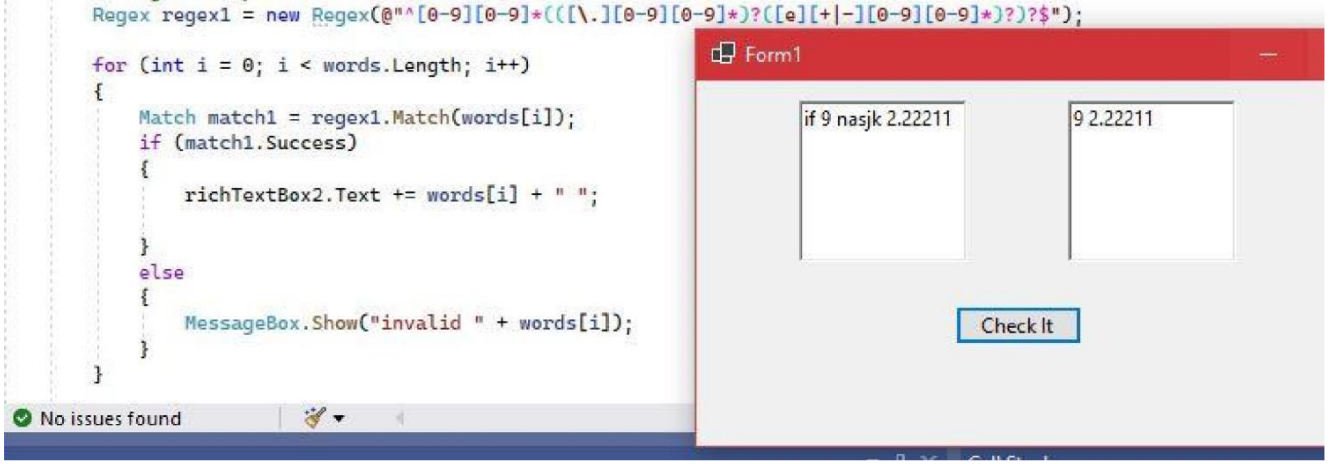
Design a regular expression for constants (digits plus floating point numbers): Regular Expression for Constants: [0-9]+((.[0-9]+)?([e][+|-][0-9]+)?)? Using Datagrid view.

Solution:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Text.RegularExpressions;

namespace Lab3
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {


        // take input from a richtextbox/textbox

            String var = richTextBox1.Text;

            // split the input on the basis of space

            String[] words = var.Split(' ');

            // Regular Expression for variables

 Regex regex1 = new Regex(@"^[0-9][0-9]*(([\.][0-9][0-
9]*)?([e][+|-][0-9][0-9]*)?)?$");
          for (int i = 0; i < words.Length; i++)
            {
                Match match1 = regex1.Match(words[i]);

                if (match1.Success)
                {
                        richTextBox2.Text += words[i] + " ";

                }

                else {
                    MessageBox.Show("invalid "+words[i]);
                }
```

```
            }
        }
    }
```

```
1 reference
private void button1_Click(object sender, EventArgs e)
{
    // take input from a richtextbox/textbox
    String var = richTextBox1.Text;
    // split the input on the basis of space
    String[] words = var.Split(' ');

    // Regular Expression for variables
    Regex regex1 = new Regex(@"^[0-9][0-9]*(([\.][0-9][0-9]*)?([e][+|-][0-9][0-9]*)?)?$");

    for (int i = 0; i < words.Length; i++)
    {
        Match match1 = regex1.Match(words[i]);
        if (match1.Success)
        {
            richTextBox2.Text += words[i] + " ";

        }
        else
        {
            MessageBox.Show("invalid " + words[i]);
        }
    }
```

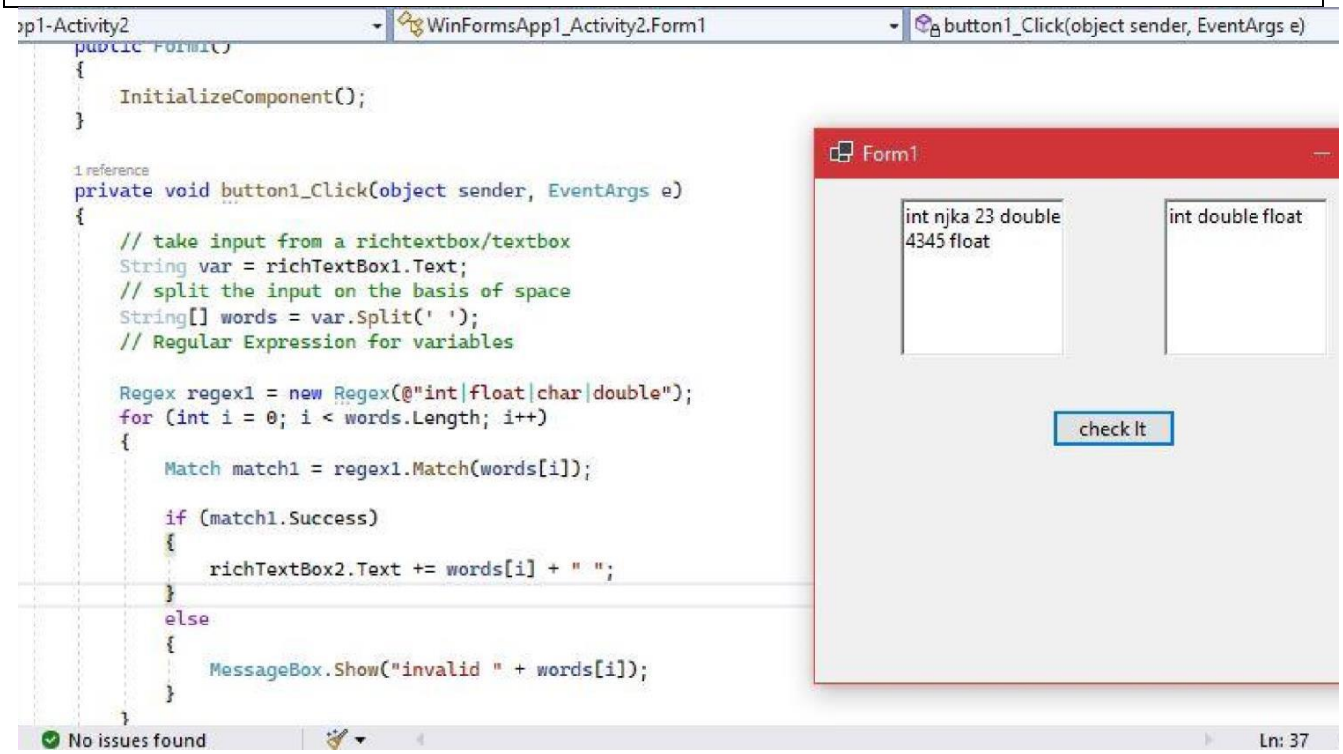


Activity 2:
Design a regular expression for keywords.(Using Datagrid view).
Regular Expression for keywords: [int | float | double | char]

Solution:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Text.RegularExpressions;

namespace Sessional1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {

        // take input from a richtextbox/textbox
```

```
            String var = richTextBox1.Text;

            // split the input on the basis of space

            String[] words = var.Split(' ');

            // Regular Expression for variables

             Regex regex1 = new Regex(@"^[int | float | char]*$");


             for (int i = 0; i < words.Length; i++)
            {
                Match match1 = regex1.Match(words[i]);

                if (match1.Success)
                {
                        richTextBox2.Text += words[i] + " ";
                }

                else {
                    MessageBox.Show("invalid "+words[i]);
                }
            }
        }
    }
```



3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1:

Design a regular expression for floating point numbers having length not greater than 6.

Lab Task 2:

Design a single regular expression for following numbers: 8e4, 5e-2 , 6e9
(Using Datagrid view).

Lab Task 3:

Design a regular expression for finding all the words starting with 't' and 'm' in the following document(Using Datagrid view).

Lab 04

Lexical Analyzer: Input Buffering scheme

Objective:

This lab is designed to demonstrate the implementation of lexical analyzer using buffering scheme.

Activity Outcomes:

On completion of this lab, students will be able to:

- implement lexical analyzer using input buffering scheme

Instructor Note:

As for this lab activity, read chapter 02 from the book "Compilers:Principles, Techniques and Tools" by Ullman Sethi.

1) Useful Concepts

In this lab, we implement the lexical analyzer using buffering scheme. Lexical analyzer reads source code character by character and produces tokens for each valid word. Specialized buffering techniques thus have been developed to reduce the amount of overhead required to process a single input character.

Two pointers to the input are maintained:

Pointer *Lexeme Begin*, marks the beginning of the current lexeme, whose extent we are attempting to determine

Pointer *Forward*, scans ahead until a pattern match is found.

Once the next lexeme is determined, *forward* is set to character at its right end.Then, after the lexeme is recorded as an attribute value of a token returned to the parser, *Lexeme Begin* is set to the character immediately after the lexeme just found.

If we use the scheme of Buffer pairs we must check, each time we advance forward, that we have not moved off one of the buffers; if we do, then we must also reload the other buffer. Thus, for each character read, we make two tests: one for the end of the buffer, and one to determine what character is read (the latter may be a multiway branch). We can combine the buffer-end test with the test for the current character if we extend each buffer to hold a sentinel character at the end. The sentinel is a special character that cannot be part of the source program, and a natural choice is the character **EOF.**

Note that **EOF** retains its use as a marker for the end of the entire input. Any **EOF** that appears other than at the end of a buffer means that the input is at an end.

2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	*45 mins*	*Medium*	*CLO-5*

Activity 1:
Implement lexical analyzer using input buffering scheme

Solution:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Collections;
namespace LexicalAnalyzerV1
{
    public partial class Form1 : Form
    {

        public Form1()
        {
            InitializeComponent();

        }

        private void btn_Input_Click(object sender, EventArgs e)
        {
            //taking user input from rich textbox
            String userInput = tfInput.Text;
            //List of keywords which will be used to seperate keywords
from variables
            List<String> keywordList = new List<String>();
            keywordList.Add("int");
            keywordList.Add("float");
            keywordList.Add("while");
            keywordList.Add("main");
            keywordList.Add("if");
            keywordList.Add("else");
            keywordList.Add("new");
            //row is an index counter for symbol table
            int row = 1;

            //count is a variable to incremenet variable id in tokens
            int count = 1;
```

```
            //line_num is a counter for lines in user input
            int line_num = 0;

            //SymbolTable is a 2D array that has the following
structure
            //[Index][Variable Name][type][value][line#]
            //rows are incremented with each variable information entry

            String[,] SymbolTable = new String[20, 6];
            List<String> varListinSymbolTable = new List<String>();

            //Input Buffering

            ArrayList finalArray = new ArrayList();
            ArrayList finalArrayc = new ArrayList();
            ArrayList tempArray = new ArrayList();

            char[] charinput = userInput.ToCharArray();

            //Regular Expression for Variables
            Regex variable_Reg = new Regex(@"^[A-Za-z|_][A-Za-z|0-
9]*$");
            //Regular Expression for Constants
            Regex constants_Reg = new Regex(@"^[0-9]+([.][0-
9]+)?([e]([+|-])?[0-9]+)?$");
            //Regular Expression for Operators
            Regex operators_Reg = new Regex(@"^[-*+/><&&||=]$");
            //Regular Expression for Special_Characters
            Regex Special_Reg = new Regex(@"^[.,'\[\]{}();:?]$");

            for (int itr = 0; itr < charinput.Length; itr++)
            {
                Match Match_Variable =
variable_Reg.Match(charinput[itr] + "");
                Match Match_Constant =
constants_Reg.Match(charinput[itr] + "");
                Match Match_Operator =
operators_Reg.Match(charinput[itr] + "");
                Match Match_Special = Special_Reg.Match(charinput[itr]
+ "");
                if (Match_Variable.Success || Match_Constant.Success ||
Match_Operator.Success || Match_Special.Success ||
charinput[itr].Equals(' '))
                {
                    tempArray.Add(charinput[itr]);
                }
                if (charinput[itr].Equals('\n'))
                {
                    if (tempArray.Count != 0)
                    {
                        int j = 0;
                        String fin = "";
                        for (; j < tempArray.Count; j++)
                        {
```

```
                            fin += tempArray[j];
                        }

                        finalArray.Add(fin);
                        tempArray.Clear();
                    }
                }
            }
            if (tempArray.Count != 0)
            {
                int j = 0;
                String fin = "";
                for (; j < tempArray.Count; j++)
                {
                    fin += tempArray[j];
                }
                finalArray.Add(fin);
                tempArray.Clear();
            }

// Final Array SO far correct
            tfTokens.Clear();

            symbolTable.Clear();

            //looping on all lines in user input
            for (int i = 0; i < finalArray.Count; i++)
            {
               String line = finalArray[i].ToString();
               //tfTokens.AppendText(line + "\n");
               char[] lineChar = line.ToCharArray();
               line_num++;
               //taking current line and splitting it into lexemes by
space

                for (int itr = 0; itr < lineChar.Length; itr++)
                {

                    Match Match_Variable =
variable_Reg.Match(lineChar[itr] + "");
                    Match Match_Constant =
constants_Reg.Match(lineChar[itr] + "");
                    Match Match_Operator =
operators_Reg.Match(lineChar[itr] + "");
                    Match Match_Special =
Special_Reg.Match(lineChar[itr] + "");
                    if (Match_Variable.Success ||
Match_Constant.Success)
                    {
                        tempArray.Add(lineChar[itr]);
                    }
                    if (lineChar[itr].Equals(' '))
                    {
                        if (tempArray.Count != 0)
```

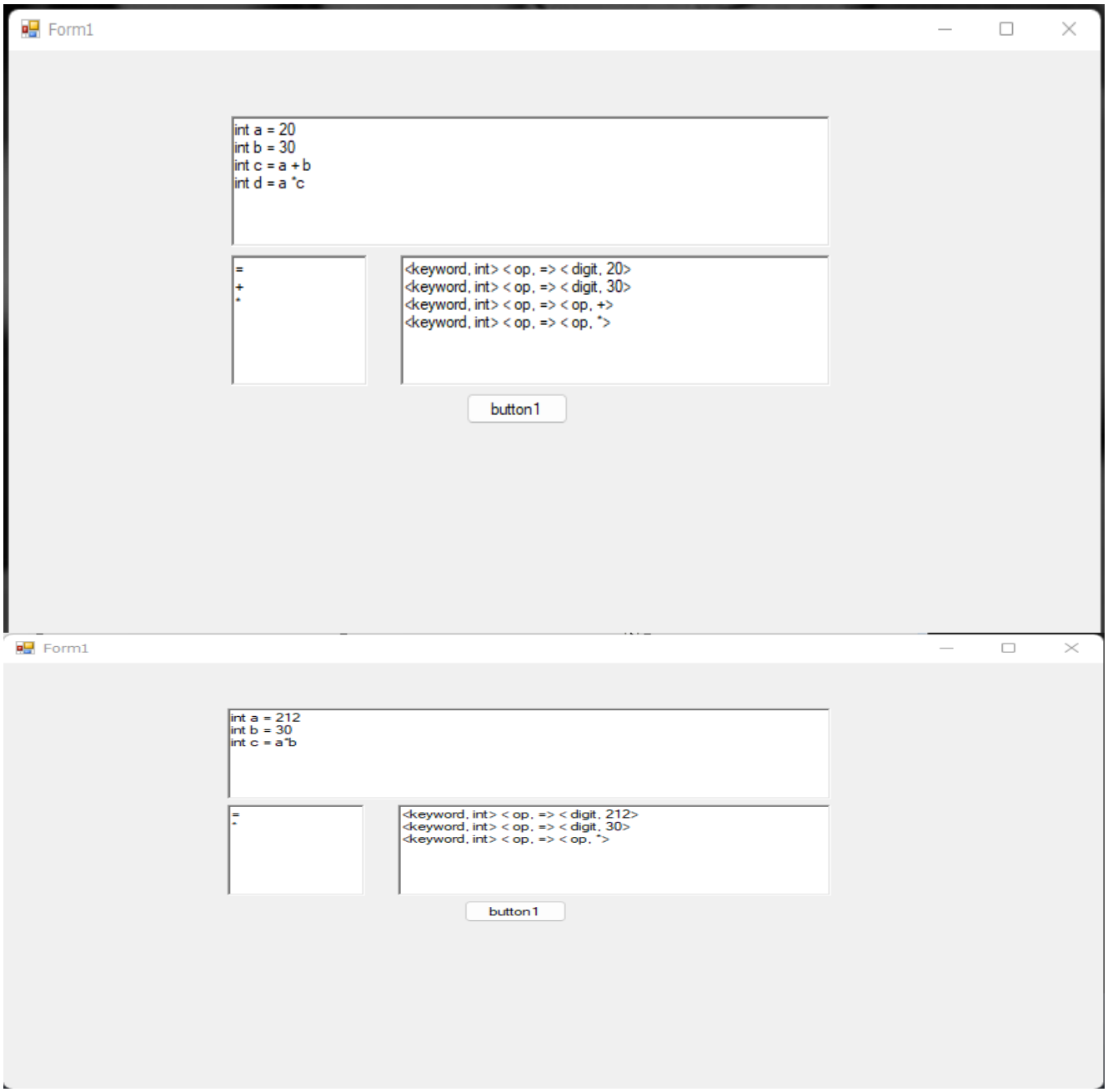
```
                            {
                                int j = 0;
                                String fin = "";
                                for (; j < tempArray.Count; j++)
                                {
                                    fin += tempArray[j];
                                }
                                finalArrayc.Add(fin);
                                tempArray.Clear();
                            }

                        }
                        if (Match_Operator.Success ||
Match_Special.Success)
                        {
                            if (tempArray.Count != 0)
                            {
                                int j = 0;
                                String fin = "";
                                for (; j < tempArray.Count; j++)
                                {
                                    fin += tempArray[j];
                                }
                                finalArrayc.Add(fin);
                                tempArray.Clear();
                            }
                            finalArrayc.Add(lineChar[itr]);
                        }
                    }
                    if (tempArray.Count != 0)
                    {
                            String fina = "";
                            for (int k = 0; k < tempArray.Count; k++)
                            {
                                fina += tempArray[k];
                            }

                            finalArrayc.Add(fina);
                            tempArray.Clear();
                    }

                    // we have asplitted line here

                       for (int x = 0; x < finalArrayc.Count; x++)
                        {

                            Match operators =
operators_Reg.Match(finalArrayc[x].ToString());
                            Match variables =
variable_Reg.Match(finalArrayc[x].ToString());
                            Match digits =
constants_Reg.Match(finalArrayc[x].ToString());
                            Match punctuations =
Special_Reg.Match(finalArrayc[x].ToString());
```

```
                        if (operators.Success)
                        {
                            // if a current lexeme is an operator then
make a token e.g. < op, = >
                            tfTokens.AppendText("< op, " +
finalArrayc[x].ToString() + "> ");
                        }
                        else if (digits.Success)
                        {
                            // if a current lexeme is a digit then make
a token e.g. < digit, 12.33 >
                            tfTokens.AppendText("< digit, " +
finalArrayc[x].ToString() + "> ");
                        }
                        else if (punctuations.Success)
                        {
                            // if a current lexeme is a punctuation
then make a token e.g. < punc, ; >
                            tfTokens.AppendText("< punc, " +
finalArrayc[x].ToString() + "> ");
                        }

                        else if (variables.Success)
                        {
                            // if a current lexeme is a variable and
not a keyword
                            if
(!keywordList.Contains(finalArrayc[x].ToString())) // if it is not a
keyword
                            {
                                // check what is the category of
varaible, handling only two cases here
                                //Category1- Variable initialization of
type digit e.g. int count = 10 ;
                                //Category2- Variable initialization of
type String e.g. String var = ' Hello ' ;

                                Regex reg1 = new
Regex(@"^(int|String|float|double)\s([A-Za-z|_][A-Za-z|0-
9]{0,10})\s(=)\s([0-9]+([.][0-9]+)?([e][+|-]?[0-9]+)?)\s(;)$"); // line
of type int alpha = 2 ;
                                Match category1 = reg1.Match(line);

                                Regex reg2 = new
Regex(@"^(String|char)\s([A-Za-z|_][A-Za-z|0-9]{0,10})\s(=)\s[']\s([A-
Za-z|_][A-Za-z|0-9]{0,30})\s[']\s(;)$"); // line of type String alpha =
' Hello ' ;
                                Match category2 = reg2.Match(line);

                                //if it is a category 1 then add a row
in symbol table containing the information related to that variable

                                if (category1.Success)
```

```
                                    {
                                        SymbolTable[row, 1] =
row.ToString(); //index

                                        SymbolTable[row, 2] =
finalArrayc[x].ToString(); //variable name

                                        SymbolTable[row, 3] = finalArrayc[x
- 1].ToString(); //type

                                        SymbolTable[row, 4] =
finalArrayc[x+2].ToString(); //value

                                        SymbolTable[row, 5] =
line_num.ToString(); // line number

                                        tfTokens.AppendText("<var" + count
+ ", " + row + "> ");

symbolTable.AppendText(SymbolTable[row, 1].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 2].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 3].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 4].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 5].ToString() + " \n ");
                                        row++;
                                        count++;
                                    }
                                    //if it is a category 2 then add a row
in symbol table containing the information related to that variable
                                    else if (category2.Success)
                                    {
                                        // if  a line such as String var =
' Hello ' ; comes and the loop moves to index of array containing Hello
'
                                        //then this if condition prevents
addition of Hello in symbol Table because it is not a variable it is
just a string

                                        if (!(finalArrayc[x-
1].ToString().Equals("'") && finalArrayc[x+1].ToString().Equals("'")))

                                        {
                                            SymbolTable[row, 1] =
row.ToString(); // index

                                            SymbolTable[row, 2] =
finalArrayc[x].ToString(); //varname

                                            SymbolTable[row, 3] =
finalArrayc[x-1].ToString(); //type
```

```
                                        SymbolTable[row, 4] =
finalArrayc[x+3].ToString(); //value

                                        SymbolTable[row, 5] =
line_num.ToString(); // line number

                                        tfTokens.AppendText("<var" +
count + ", " + row + "> ");

symbolTable.AppendText(SymbolTable[row, 1].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 2].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 3].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 4].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 5].ToString() + " \n ");
                                        row++;
                                        count++;
                                    }
                                    else
                                    {
                                        tfTokens.AppendText("<String" +
count + ", " + finalArrayc[x].ToString() + "> ");
                                    }

                                }

                                else
                                {
                                    // if any other category line comes
in we check if we have initializes that varaible before,
                                    // if we have initiazed it before
then we put the index of that variable in symbol table, in its token
                                    String ind = "Default";
                                    String ty = "Default";
                                    String val = "Default";
                                    String lin = "Default";
                                    for (int r = 1; r <=
SymbolTable.GetLength(0); r++)
                                    {
                                        //search in the symbol table if
variable entry already exists
                                        if (SymbolTable[r,
2].Equals(finalArrayc[x].ToString()))
                                        {
                                            ind = SymbolTable[r, 1];
                                            ty = SymbolTable[r, 3];
                                            val = SymbolTable[r, 4];
                                            lin = SymbolTable[r, 5];
                                            tfTokens.AppendText("<var"
+ ind + ", " + ind + "> ");
```

```
                                                    break;
                                                }
                                            }
                                        }

                                    }
                                    // if a current lexeme is not a variable
but a keyword then make a token such as: <keyword, int>
                                    else
                                    {
                                        tfTokens.AppendText("<keyword, " +
finalArrayc[x].ToString() + "> ");
                                    }
                                }
                            }
                        tfTokens.AppendText("\n");
                        finalArrayc.Clear();
                    }
                }
            }
        }
```



3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1:
Implement lexical analyzer using two buffers

Lab 05

Construction of Symbol Table

Objective:

In this lab, previous lab work will be used to generate Symbol Table.

Activity Outcomes:

This lab teaches you

- Implementation of symbol table with arrays

Instructor Note:

As for this lab activity, read chapter 02 & 03 from the book "Compiler Construction-Principles and Practices" by Kenneth C. Louden

1) Useful concepts:

A **symbol table** is a data structure used by a language translator such as a compiler or interpreter, where each identifier in a program's source code is associated with information relating to its declaration or appearance in the source. A common implementation technique is to use a hash table. There are also trees, linear lists and self-organizing lists which can be used to implement a symbol table. It also simplifies the classification of literals in tabular format. The symbol table is accessed by most phases of a compiler, beginning with the lexical analysis to optimization.

Introduction

Consider the following program written in C:

```
// Declare an external function
extern double bar(double x);

// Define a public function
double foo(int count)
{
   double  sum = 0.0;

   // Sum all the values bar(1) to bar(count)
   for (int i = 1;  i <= count;  i++)
      sum += bar((double) i);
   return sum;
}
```

A C compiler that parses this code will contain at least the following symbol table entries:

Symbol name	Type	Scope
bar	function, double	extern
x	double	function parameter
foo	function, double	global
count	int	function parameter
sum	double	block local
i	int	for-loop statement

2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	***60 mins***	***High***	***CLO-5***

Activity 1:
Implement symbol table using array data structure.

Solution:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Collections;

namespace LexicalAnalyzerV1
{
    public partial class Form1 : Form
    {

        public Form1()
        {
            InitializeComponent();

        }

        private void btn_Input_Click(object sender, EventArgs e)
        {
            //taking user input from rich textbox
            String userInput = tfInput.Text;
```

```
            //List of keywords which will be used to seperate keywords
from variables
            List<String> keywordList = new List<String>();
            keywordList.Add("int");
            keywordList.Add("float");
            keywordList.Add("while");
            keywordList.Add("main");
            keywordList.Add("if");
            keywordList.Add("else");
            keywordList.Add("new");
            //row is an index counter for symbol table
            int row = 1;

            //count is a variable to incremenet variable id in tokens
            int count = 1;

            //line_num is a counter for lines in user input
            int line_num = 0;

            //SymbolTable is a 2D array that has the following structure
            //[Index][Variable Name][type][value][line#]
            //rows are incremented with each variable information entry

            String[,] SymbolTable = new String[20, 6];
            List<String> varListinSymbolTable = new List<String>();

            //Input Buffering

            ArrayList finalArray = new ArrayList();
            ArrayList finalArrayc = new ArrayList();
            ArrayList tempArray = new ArrayList();

            char[] charinput = userInput.ToCharArray();

            //Regular Expression for Variables
            Regex variable_Reg = new Regex(@"^[A-Za-z|_][A-Za-z|0-9]*$");
            //Regular Expression for Constants
            Regex constants_Reg = new Regex(@"^[0-9]+([.][0-
9]+)?([e]([+|-])?[0-9]+)?$");
            //Regular Expression for Operators
            Regex operators_Reg = new Regex(@"^[-*+/><&&||=]$");
            //Regular Expression for Special_Characters
            Regex Special_Reg = new Regex(@"^[.,'\[\]{}();:?]$");

            for (int itr = 0; itr < charinput.Length; itr++)
            {
                Match Match_Variable = variable_Reg.Match(charinput[itr]
+ "");
                Match Match_Constant = constants_Reg.Match(charinput[itr]
+ "");
                Match Match_Operator = operators_Reg.Match(charinput[itr]
+ "");
                Match Match_Special = Special_Reg.Match(charinput[itr] +
"");
```

```
                if (Match_Variable.Success || Match_Constant.Success ||
Match_Operator.Success || Match_Special.Success ||
charinput[itr].Equals(' '))
                {
                    tempArray.Add(charinput[itr]);
                }
                if (charinput[itr].Equals('\n'))
                {
                    if (tempArray.Count != 0)
                    {
                        int j = 0;
                        String fin = "";
                        for (; j < tempArray.Count; j++)
                        {
                            fin += tempArray[j];
                        }

                        finalArray.Add(fin);
                        tempArray.Clear();
                    }

                }
            }
            if (tempArray.Count != 0)
            {
                int j = 0;
                String fin = "";
                for (; j < tempArray.Count; j++)
                {
                    fin += tempArray[j];
                }
                finalArray.Add(fin);
                tempArray.Clear();
            }

// Final Array SO far correct
            tfTokens.Clear();

            symbolTable.Clear();

            //looping on all lines in user input
            for (int i = 0; i < finalArray.Count; i++)
            {
               String line = finalArray[i].ToString();
               //tfTokens.AppendText(line + "\n");
               char[] lineChar = line.ToCharArray();
               line_num++;
               //taking current line and splitting it into lexemes by
space

                for (int itr = 0; itr < lineChar.Length; itr++)
                {

                    Match Match_Variable =
```

```
variable_Reg.Match(lineChar[itr] + "");
                    Match Match_Constant =
constants_Reg.Match(lineChar[itr] + "");
                    Match Match_Operator =
operators_Reg.Match(lineChar[itr] + "");
                    Match Match_Special = Special_Reg.Match(lineChar[itr]
+ "");
                    if (Match_Variable.Success || Match_Constant.Success)
                    {
                        tempArray.Add(lineChar[itr]);
                    }
                    if (lineChar[itr].Equals(' '))
                    {
                        if (tempArray.Count != 0)
                        {
                            int j = 0;
                            String fin = "";
                            for (; j < tempArray.Count; j++)
                            {
                                fin += tempArray[j];
                            }
                            finalArrayc.Add(fin);
                            tempArray.Clear();
                        }

                    }
                    if (Match_Operator.Success || Match_Special.Success)
                    {
                        if (tempArray.Count != 0)
                        {
                            int j = 0;
                            String fin = "";
                            for (; j < tempArray.Count; j++)
                            {
                                fin += tempArray[j];
                            }
                            finalArrayc.Add(fin);
                            tempArray.Clear();
                        }
                        finalArrayc.Add(lineChar[itr]);
                    }
                }
                if (tempArray.Count != 0)
                {
                        String fina = "";
                        for (int k = 0; k < tempArray.Count; k++)
                        {
                            fina += tempArray[k];
                        }

                        finalArrayc.Add(fina);
                        tempArray.Clear();
                }
```

```
                // we have asplitted line here

                    for (int x = 0; x < finalArrayc.Count; x++)
                    {

                        Match operators =
operators_Reg.Match(finalArrayc[x].ToString());
                        Match variables =
variable_Reg.Match(finalArrayc[x].ToString());
                        Match digits =
constants_Reg.Match(finalArrayc[x].ToString());
                        Match punctuations =
Special_Reg.Match(finalArrayc[x].ToString());

                        if (operators.Success)
                        {
                            // if a current lexeme is an operator then
make a token e.g. < op, = >
                            tfTokens.AppendText("< op, " +
finalArrayc[x].ToString() + "> ");
                        }
                        else if (digits.Success)
                        {
                            // if a current lexeme is a digit then make a
token e.g. < digit, 12.33 >
                            tfTokens.AppendText("< digit, " +
finalArrayc[x].ToString() + "> ");
                        }
                        else if (punctuations.Success)
                        {
                            // if a current lexeme is a punctuation then
make a token e.g. < punc, ; >
                            tfTokens.AppendText("< punc, " +
finalArrayc[x].ToString() + "> ");
                        }

                        else if (variables.Success)
                        {
                            // if a current lexeme is a variable and not
a keyword
                            if
(!keywordList.Contains(finalArrayc[x].ToString())) // if it is not a
keyword
                            {
                                // check what is the category of
varaible, handling only two cases here
                                //Category1- Variable initialization of
type digit e.g. int count = 10 ;
                                //Category2- Variable initialization of
type String e.g. String var = ' Hello ' ;

                                Regex reg1 = new
Regex(@"^(int|float|double)\s([A-Za-z|_][A-Za-z|0-9]{0,10})\s(=)\s([0-
9]+([.][0-9]+)?([e][+|-]?[0-9]+)?)\s(;)$"); // line of type int alpha = 2
```

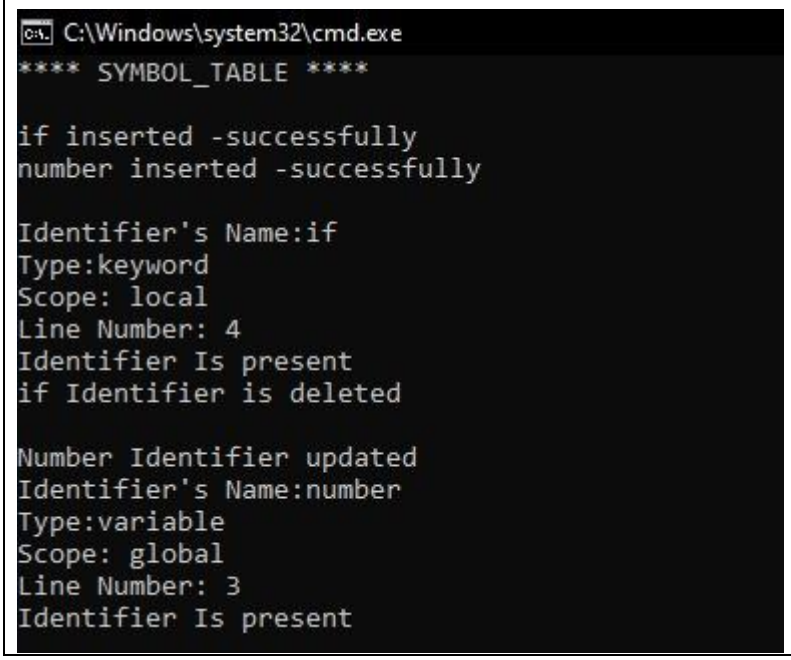
```
;
                                    Match category1 = reg1.Match(line);

                                    Regex reg2 = new
Regex(@"^(String|char)\s([A-Za-z|_][A-Za-z|0-9]{0,10})\s(=)\s[']\s([A-Za-
z|_][A-Za-z|0-9]{0,30})\s[']\s(;)$"); // line of type String alpha = '
Hello ' ;
                                    Match category2 = reg2.Match(line);

                                    //if it is a category 1 then add a row in
symbol table containing the information related to that variable

                                    if (category1.Success)
                                    {
                                        SymbolTable[row, 1] = row.ToString();
//index

                                        SymbolTable[row, 2] =
finalArrayc[x].ToString(); //variable name

                                        SymbolTable[row, 3] = finalArrayc[x -
1].ToString(); //type

                                        SymbolTable[row, 4] =
finalArrayc[x+2].ToString(); //value

                                        SymbolTable[row, 5] =
line_num.ToString(); // line number

                                        tfTokens.AppendText("<var" + count +
", " + row + "> ");

symbolTable.AppendText(SymbolTable[row, 1].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 2].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 3].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 4].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 5].ToString() + " \n ");
                                        row++;
                                        count++;
                                    }
                                    //if it is a category 2 then add a row in
symbol table containing the information related to that variable
                                    else if (category2.Success)
                                    {
                                        // if  a line such as String var = '
Hello ' ; comes and the loop moves to index of array containing Hello ,
                                        //then this if condition prevents
addition of Hello in symbol Table because it is not a variable it is just
a string
```

```
                                    if (!(finalArrayc[x-
1].ToString().Equals("'") && finalArrayc[x+1].ToString().Equals("'")))

                                    {
                                        SymbolTable[row, 1] =
row.ToString(); // index

                                        SymbolTable[row, 2] =
finalArrayc[x].ToString(); //varname


                                        SymbolTable[row, 3] =
finalArrayc[x-1].ToString(); //type

                                        SymbolTable[row, 4] =
finalArrayc[x+3].ToString(); //value

                                        SymbolTable[row, 5] =
line_num.ToString(); // line number

                                        tfTokens.AppendText("<var" +
count + ", " + row + "> ");

symbolTable.AppendText(SymbolTable[row, 1].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 2].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 3].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 4].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 5].ToString() + " \n ");
                                        row++;
                                        count++;
                                    }

                                    else
                                    {
                                        tfTokens.AppendText("<String" +
count + ", " + finalArrayc[x].ToString() + "> ");
                                    }

                                }

                                else
                                {
                                    // if any other category line comes
in we check if we have initializes that varaible before,
                                    // if we have initiazed it before
then we put the index of that variable in symbol table, in its token
                                    String ind = "Default";
                                    String ty = "Default";
                                    String val = "Default";
                                    String lin = "Default";
```

```
                                    for (int r = 1; r <=
SymbolTable.GetLength(0); r++)
                                    {
                                        //search in the symbol table if
variable entry already exists
                                        if (SymbolTable[r,
2].Equals(finalArrayc[x].ToString()))
                                        {
                                            ind = SymbolTable[r, 1];
                                            ty = SymbolTable[r, 3];
                                            val = SymbolTable[r, 4];
                                            lin = SymbolTable[r, 5];
                                            tfTokens.AppendText("<var" +
ind + ", " + ind + "> ");

                                            break;
                                        }
                                    }
                                }

                            }
                            // if a current lexeme is not a variable but
a keyword then make a token such as: <keyword, int>
                            else
                            {
                                tfTokens.AppendText("<keyword, " +
finalArrayc[x].ToString() + "> ");
                            }
                        }
                    }
                tfTokens.AppendText("\n");
                finalArrayc.Clear();

            }


        }
    }

}
#region Display Symbol Table
            for (int j = 0; j < Symboltable.Count; j++)
            {
                for (int z = 0; z < Symboltable[j].Count; z++)
                { ST.AppendText(Symboltable[j][z] + "\t"); }
                ST.AppendText("\n");
            }
            #endregion
        }

#region Make Entry Symbol Table
        void Check_And_Make_Entries()
        {
            KeyWords.Remove("begin"); KeyWords.Remove("end");
```

```
KeyWords.Remove("print");
            KeyWords.Remove("if"); KeyWords.Remove("else");
            if (lexemes_per_line - 4 == 0 || lexemes_per_line - 7 == 0)
            {
                if (lexemes_per_line == 7)
                {
                    Variables.RemoveAt(Variables.Count - 1);
Variables.RemoveAt(Variables.Count - 1);
                }
                for (; ST_index < KeyWords.Count; ST_index++)
                {
                    Symboltable.Add(new List<string>());
                    Symboltable[ST_index].Add(ST_index + 1 + "");
                    Symboltable[ST_index].Add(Variables[ST_index] + "");
                    Symboltable[ST_index].Add(KeyWords[ST_index] + "");
                    Symboltable[ST_index].Add(Constants[ST_index] + "");
                    Symboltable[ST_index].Add(LineNumber[ST_index] + "");
                }
            }
            if (lexemes_per_line - 6 == 0)
            {
                Variables.RemoveAt(Variables.Count - 1);
Variables.RemoveAt(Variables.Count - 1);
Variables.RemoveAt(Variables.Count - 1);
            }
        }
        #endregion
```

OUTPUT :



3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1:

Implement symbol table using hash function

Lab 06

Top-down Parser: Finding the First set of a given grammar.

Objective:

In this lab we will find the first set of a given grammar using Array for a Top-down Parser.

Activity Outcomes:

This lab teaches you

- How to find the tokens/variables that are the starting symbols of a grammar rule.

Instructor Note:

As for this lab activity, read chapter 04 from the book “Compiler Construction-Principles and Practices” by Kenneth C. Louden.

1) Useful Concepts.

Syntax analysis is the second phase of a compiler. The lexical analyzer works closely with the syntax analyzer. It reads character streams from the source code, checks for legal tokens, and passes the data to the syntax analyzer for further processing.
Each time a predictive parser makes a decision, it needs to determine which production rule to apply to the leftmost non-terminal in an intermediate form, based on the next terminal (*i.e.* the lookahead symbol).
Take the minimalistic grammar

1. S -> aAb
2. A -> a | <epsilon>

and let us first parse the statement 'aab', so that the parser starts from looking at the (*intermediate form*, *input*) pair ('S','aab').
There is no real choice here (since 'S' expands in only one way), but we can still see that this is the production to choose because FIRST(S) = {a}, and arrive at the pair ('aAb', 'aab'). If we started from ('S', 'z'), we'd already know that there's a syntax error, because no expansion of S begins with 'z' - that's how come FIRST(S) doesn't have a 'z' in it.
Moving along, ('aAb', 'aab') doesn't begin with a non-terminal to decide a production for, so we just verify that 'a' matches 'a', which leaves us with ('Ab','ab'). The nonterminal 'A' *does* have multiple ways to expand - it can either become an 'a', or vanish. Since FIRST(A) = {a} as well, the former choice is the right one, so we choose that, and get ('ab', 'ab'). Having run out of nonterminals, the rest is just to verify that 'a' is in the right place to leave ('b','b'), and 'b' matches as well, so in the end, the statement is accepted by the grammar.
This is the significance of the FIRST sets: they tell you when a nonterminal can produce the lookahead symbol as the beginning of a statement, so that it can be matched away and reduce the input. These derivations were direct, but if the grammar were

1. S -> aDb
2. D -> E
3. E -> 'a' | <epsilon>

you would find 'a' in FIRST(S), FIRST(D), and FIRST(E) to drive essentially the same choices, just using one additional step of derivation.

First sets are the set of all what can begin a production rule. For example, a number must begin with a digit, a identifier must begin with a letter, ...

2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	*60 mins*	*High*	*CLO-5*

Activity 1:
Write a program that takes at least six grammar rules. Based on these rules, find the first sets of these non-terminals.

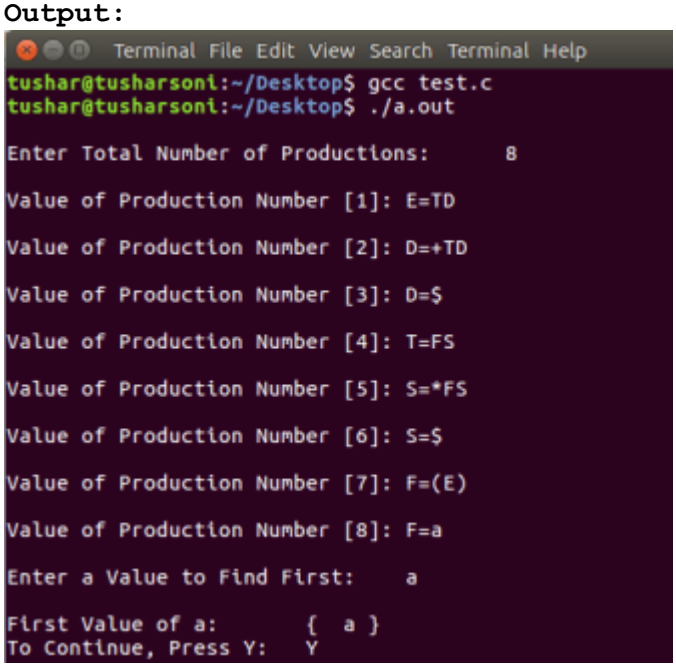
Solution:

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using System.Text.RegularExpressions;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace FirstSets
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        Hashtable productionRulez = new Hashtable();
        Hashtable firstSets = new Hashtable();
        private void button1_Click(object sender, EventArgs e)
        {
            productionRulez.Clear();
            firstSets.Clear();

            String temp2 = "";

            bool flag = true;

            var productionRules = richTextBox1.Text.Split('\n');

            foreach (var productionRule in productionRules)
            {
                var temp = productionRule.Split('>');
                if (!productionRulez.Contains(temp[0]))
                {
```

```
                    productionRulez.Add(temp[0], temp[1]);
                    var te = temp[0].ToCharArray()[0];
                    if(!(new Regex(@"^[A-Z]$")).Match(te+"").Success)
                    {
                        flag = false;
                        MessageBox.Show("Non terminals cant be small
letters");
                    }
                }
                else
                {
                    productionRulez[temp[0]] += "|" + temp[1];
                }
            }
            if (flag)
            {
                foreach (DictionaryEntry rule in productionRulez)
                {
                    List<String[]> rules = new List<String[]>();

                    var alpha = rule.Value.ToString().Split('|');
                    foreach (var rul in alpha)
                    {
                        rules.Add(rul.Split(' '));
                    }
                    foreach (var rul in rules)
                    {
                        if (!firstSets.Contains(rule.Key))
                        {
                            firstSets.Add(rule.Key,
calculateFirst(rul, 0));
                        }
                        else
                        {
                            firstSets[rule.Key] += "," +
calculateFirst(rul, 0);
                        }
                    }
                }
                foreach (DictionaryEntry x in firstSets)
                {
                    richTextBox2.AppendText("First(" +
x.Key.ToString() + ") = " + "{" + x.Value.ToString() + "}\n");
                }
            }
        }
        private string calculateFirst(String[] alpha, int index)
        {

            if (!productionRulez.Contains(alpha[0]) && alpha[0] !=
"~")
            {
                return alpha[0];
            }
```

```
            else if (alpha[0] != "~" && alpha.Length >= 1)
            {
                String[] beta = null;
                if (productionRulez.Contains(alpha[index]))
                {
                    beta =
productionRulez[alpha[index]].ToString().Split(' ');
                }
                else
                {
                    return alpha[index];
                }
                var x = calculateFirst(beta, index);

                if (x != "~")
                {
                    return x;
                }
                else
                {
                    return calculateFirst(alpha, index + 1);
                }
            }
            return "~";
        }
    }
}
```

Output:



3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1:

Write a code for any given grammar that satisfy the criterion of JAVA language constructs.

Lab 07

Top-down Parser: Finding the Follow set of a given grammar.

Objective:

In this lab we will find the follow set of a given grammar using Array for a Top-down Parser.

Activity Outcomes:

On completion of this lab, student will be able to:

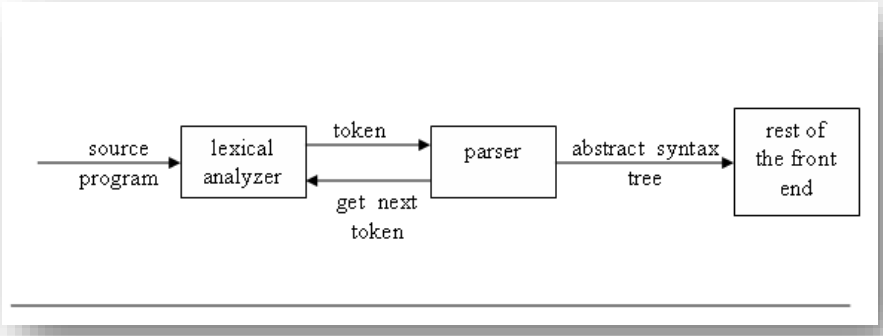
- write a top-down partser than can find the tokens/variables that are the ending symbols of a grammar rule.

Instructor Note:

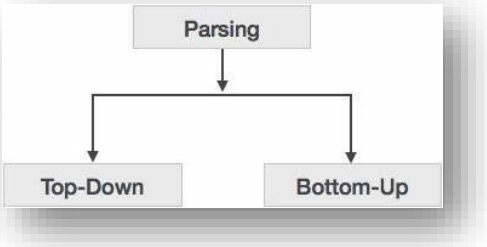
As for this lab activity, read chapter 04 from the book "Compiler Construction-Principles and Practices" by Kenneth C. Louden

1) Useful Concepts

A parser or syntax analyzer is a compiler component that breaks data into smaller elements for easy translation into another language. A parser takes input in the form of a sequence of tokens or program instructions, validates sentence sequence and usually builds a data structure in the form of a parse tree or abstract syntax tree.



Syntax analyzers follow production rules defined by means of context-free grammar. The way the production rules are implemented (derivation) divides parsing into two types: top-down parsing and bottom-up parsing.



Syntax analysis is the second phase of a compiler.The lexical analyzer works closely with the syntax analyzer. It reads character streams from the source code, checks for legal tokens, and passes the data to the syntax analyzer for further processing. Follow set in parsing is the continuation of first set.

FOLLOW covers the possibility that the leftmost non-terminal can disappear, so that the lookahead symbol is not actually a part of what we're presently expanding, but rather the beginning of the next construct.

FOLLOW(S) for nonterminal S, to be the set of terminals a that can appear immediately to the right of A in some sentential form; that is, the set of terminals a such that there exists a derivation of the form S =>* aAa.

Consider parsing the string 'ab', which starts us off at ('S','ab'). The first decision comes from FIRST(S) again, and goes through ('aAb','ab'), to ('Ab','b'). In this situation, we need the A to vanish; although A can not directly match 'b', 'b' can *follow* A: FOLLOW(A) = {b} because b is found immediately to the right of A in the result of the first production, and A can produce the empty string.
A -> <epsilon> can't be chosen whenever strings begin with <epsilon>, because all strings do. It *can*, however, be chosen as a consequence of noticing that we need A to go away before we can make further progress. Hence, seeing ('Ab','b'), the A -> <epsilon> production yields ('b','b'), and success in the next step.

This is the significance of the FOLLOW sets: they tell you when a non-terminal can hand you the lookahead symbol at the beginning of a statement by disappearing. Choosing productions that give <epsilon> doesn't reduce the input string, but you still have to make a rule for when the parser needs to take them, and the appropriate conditions are found from the FOLLOW set of the troublesome non-terminal.

Both Top-Down and Bottom-Up Parsers make use of FIRST and FOLLOW for the production of Parse Tree from a grammar. In top down parsing, FIRST and FOLLOW is used to choose which among the grammar is to apply, based on the next input symbol (lookhead terminals) in the given string. During panic-mode error recovery, sets of tokens produced by FOLLOW can be used as synchronizing tokens.

2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	***60 mins***	***High***	***CLO-6***

Activity 1:
Write a program that takes at least six grammar rules. Based on these rules and after calculating the first of all the non-terminals, find the follow sets of these variables.

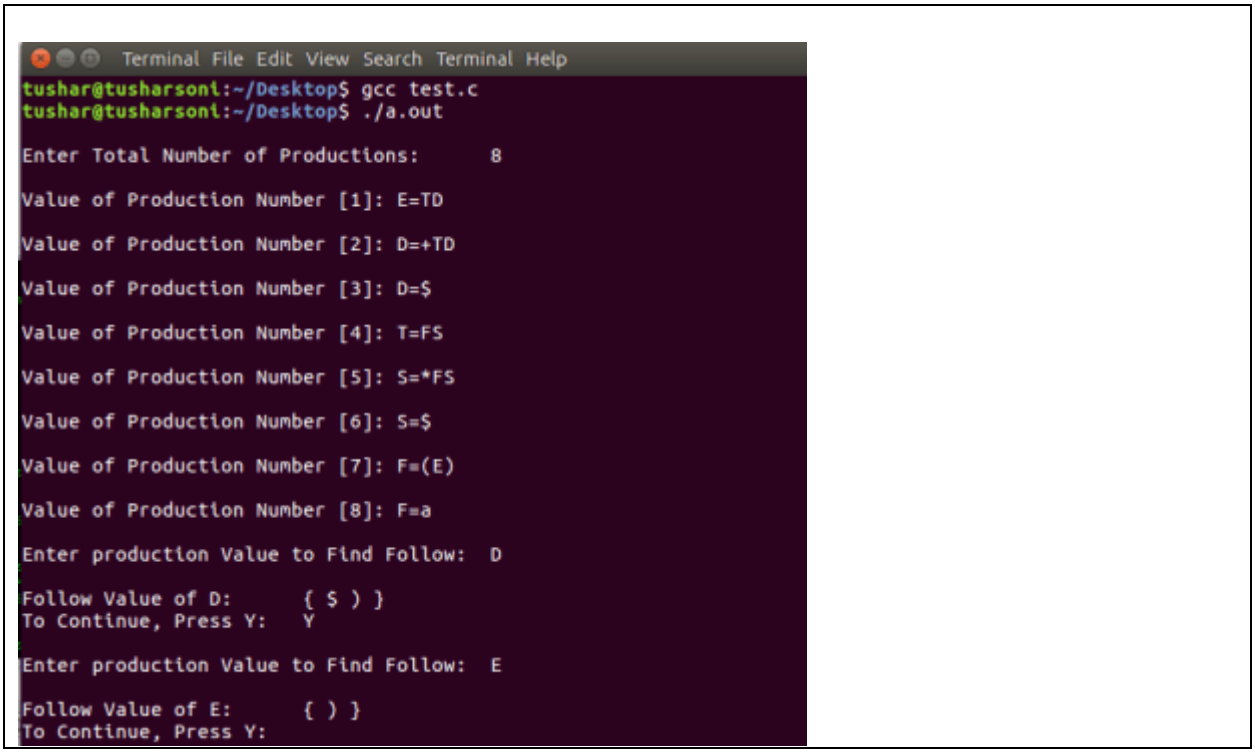
Solution:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace FirstFollowSet
{
```

```
    class Program
    {
        static int limit, x = 0;
        static char[,] production = new char[10, 10];
        static char[] array = new char[10];

        static void Main(string[] args)
        {
            for(int i = 0; i < 10; i++){
                for (int j = 0; j < 10; j++) {
                    //To signify empty space.
                    production[i,j] = '-';
                }
            }

            int count;
            char option, ch;
            Console.WriteLine("\nEnter Total Number of
Productions:\t");
            limit = Convert.ToInt32(Console.ReadLine());
            for (count = 0; count < limit; count++)
            {
                Console.WriteLine("\nValue of Production Number
{0}:\t", count + 1);
                String temp = Console.ReadLine();

                for (int i = 0; i < temp.Length; i++ )
                {
                    production[count, i] = temp[i];
                }
            }
            // Keep asking the user for non-terminal for which
follow_set is needed.
            do
            {
                x = 0;
                Console.WriteLine("\nEnter production Value to Find
Follow:\t");
                ch = Console.ReadKey().KeyChar;
                find_follow(ch);
                Console.WriteLine("\nFollow Value of {0}:\t{", ch);
                for (count = 0; count < x; count++)
                {
                    Console.Write(array[count]);
                }
                Console.Write("}\n");
                Console.Write("To Continue, Press Y:\t");
                option = ch = Console.ReadKey().KeyChar;
            } while (option == 'y' || option == 'Y');

            for (int i = 0; i < 10; i++)
            {
                for (int j = 0; j < 10; j++)
                {
```

```
                    Console.Write(production[i, j]);
                }
                Console.Write("\n");

            }
            Console.ReadKey();
        }
        static void find_follow(char ch)
        {
            int i = 0, j;

            for (int k = 0; k < 10; k++)
            {
                if(){
                }
            }

            int length = production[i,0].Length;
            if (Convert.ToChar(production[0, 0]).Equals(ch))
            {
                array_manipulation('$');
            }
            for (i = 0; i < limit; i++)
            {
                for (j = 2; j < length; j++)
                {
                    if (Convert.ToChar(production[i, j]).Equals(ch))
                    {
                        if (Convert.ToChar(production[i, j +
1]).Equals('\0'))
                        {
                            find_first(Convert.ToChar(production[i, j
+ 1]));
                        }
                        if (Convert.ToChar(production[i, j +
1]).Equals('\0') && ch.Equals(Convert.ToChar(production[i, 0])))
                        {
                            find_follow(Convert.ToChar(production[i,
0]));
                        }
                    }
                }
            }
        }

        static void find_first(char ch)
        {
            int i = 0, k;
            //Check for uppercase letter.
            int val = System.Convert.ToInt32(ch);

            if (!(val >= 97 && val <= 122))
            {
                array_manipulation(ch);
```

```
            }
            for (k = 0; k < limit; k++)
            {
                if (production[k, 0].Equals(ch))
                {
                    if (production[k, 2].Equals('$'))
                    {
                        find_follow(Convert.ToChar(production[i,
0]));
                    }
                    //Check for lowercase.
                    else if (Convert.ToInt32((production[k, 2])) >=
97 && Convert.ToInt32((production[k, 2])) <= 122)
                    {

array_manipulation(Convert.ToChar(production[k, 2]));
                    }
                    else
                    {
                        find_first(Convert.ToChar(production[k, 2]));
                    }
                }
            }
        }

        static void array_manipulation(char ch)
        {
            int count;
            for (count = 0; count <= x; count++)
            {
                if (array[count].Equals(ch))
                {
                    return;
                }
            }
            array[x++] = ch;
        }
    }
}
```



3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1:

Write a code for the grammar with at least 4 Non-Terminals and 4 Terminals.

Lab 08

BottomUp Parser: Implementation of Deterministic Finite Automata

Objective:

In this lab students will be able to implement DFA from the given Grammar that will be used for further processing in checking the syntax of the given grammar.

Activity Outcomes:

On completion of this lab, students will be able to:

- Implement deterministic finite automata which will be used in bottom up parser

Instructor Note:

As for this lab activity, read chapter 05 from the book “Compiler Construction-Principles and Practices” by Kenneth C. Louden

1) Useful Concepts

Bottom-up parsing starts with the input symbols and tries to construct the parse tree up to the start symbol. For example
Input string : a + b * c
Production rules:

```
S → E
E → E + T
E → E * T
E → T
T → id
```

Let us start bottom-up parsing

```
a + b * c
```

Read the input and check if any production matches with the input:

```
a + b * c
T + b * c
E + b * c
E + T * c
E * c
E * T
E
```

S

For designing bottom up parser you need to know how to implement deterministic finite automata (DFA) and simple LR. In this lab you will learn how to implement a DFA.
Deterministic Finite Automata is a finite-state machine that accepts and rejects finite strings of symbols and only produces a unique computation (or run) of the automaton for each input string.

2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	***45 mins***	***Medium***	***CLO-5***

Activity 1:
Design a Deterministic finite automata which accepts the input 'abcc'.

Solution:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {

        public Form1()
        {
            InitializeComponent();

        }

        private void Compile_Click(object sender, EventArgs e)
        {
            String Initial_State = "S0";
            String Final_State = "S3";

            var dict = new Dictionary<string, Dictionary<char,
object>>();
            dict.Add("S0", new Dictionary<char, object>()
            {
                { 'a', "S1" },
                { 'b', "Se" },
                { 'c', "Se" }
            });
            dict.Add("S1", new Dictionary<char, object>()
            {
```

```
                    { 'a', "Se" },
                    { 'b', "S2" },
                    { 'c', "Se" }
                });
                dict.Add("S2", new Dictionary<char, object>()
                {
                    { 'a', "Se" },
                    { 'b', "Se" },
                    { 'c', "S3" }
                });
                dict.Add("S3", new Dictionary<char, object>()
                {
                    { 'a', "Se" },
                    { 'b', "Se" },
                    { 'c', "S3" }
                });

                char check;
                String state;
                string strinput = Input.Text;
                char[] charinput = strinput.ToCharArray();

                    check = charinput[0];
                    state = Initial_State;
                    int j = 0;
                    while(check!='\\' && state!="Se")
                    {
                        state = dict[state][check]+"";
                        j++;
                        check = charinput[j];
                    }
                    if (state.Equals(Final_State))
                    { Output.Text = "RESULT OKAY"; }
                    else
                    { Output.Text = "ERROR"; }

            }
        }
}
```

OUTPUT:

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1:

Design a deterministic finite automaton which will accept variables of C.

Lab 10

Bottom up parser: Implementation of SLR Parser.

Objective:

In this lab student will implement SLR Parser from the given grammar with the help of DFA.

Activity Outcomes:

On completion of this lab, students will be able to:

- Implement SLR for a bottom up parser

Instructor Note:

As for this lab activity, read chapter 05 from the book "Compiler Construction-Principles and Practices" by Kenneth C. Louden

1) Useful Concepts

A Simple LR or SLR parser is a type of LR parser with small parse tables and a relatively simple parser generator algorithm. It is the smallest class of grammar having few number of states.

SLR parser is quite efficient at finding the single correct bottom-up parse in a single left-to-right scan over the input stream, without guesswork or backtracking. The parser is mechanically generated from a formal grammar for the language.

Steps for constructing the SLR parsing table :

1. Writing augmented grammar
2. LR(0) collection of items to be found
3. Find FOLLOW of LHS of production
4. Defining 2 functions:goto[list of terminals] and action[list of non-terminals] in the parsing table

EXAMPLE – Construct LR parsing table for the given context-free grammar

S–>AA

A–>aA|b

Solution:

STEP1 – Find augmented grammar

The augmented grammar of the given grammar is:-

S'–>.S [0th production]

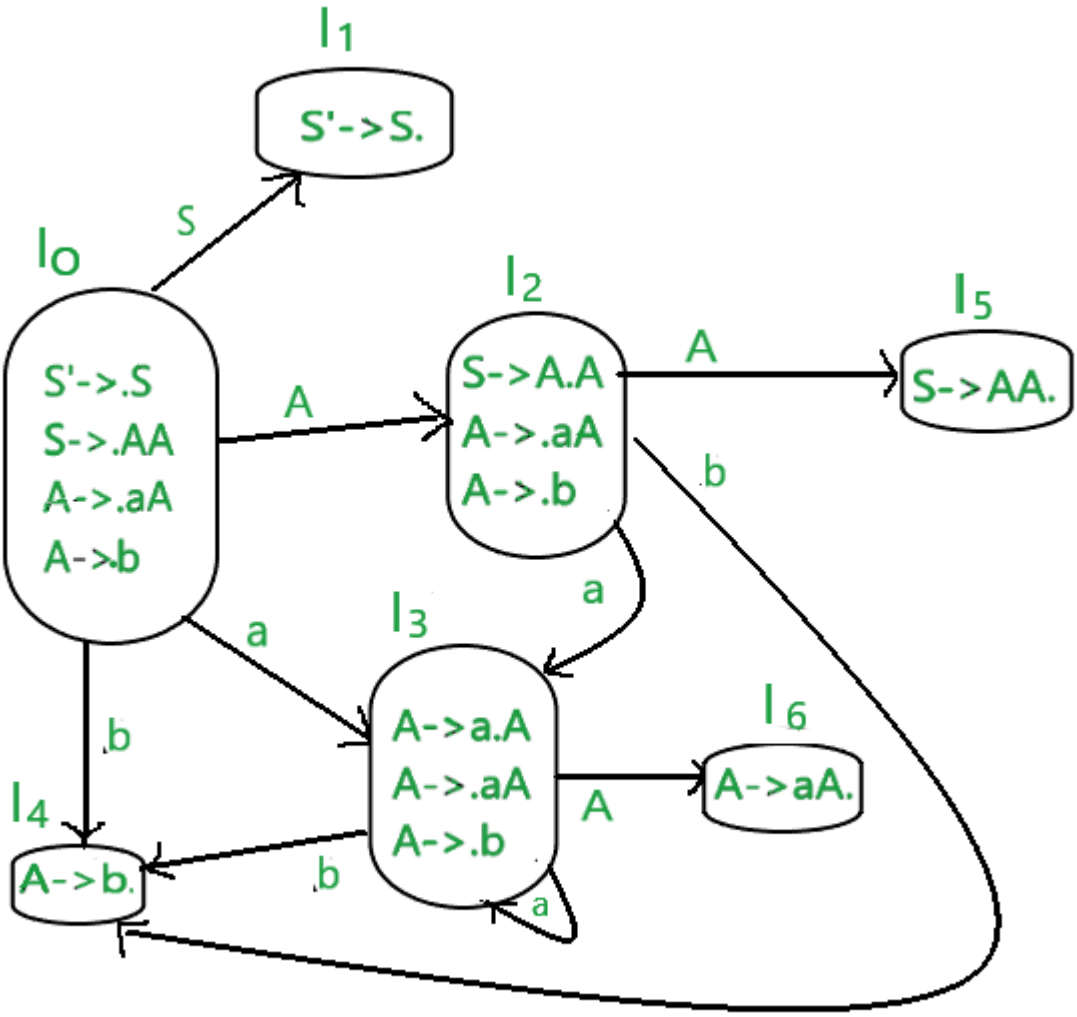
S–>.AA [1st production]

A–>.aA [2nd production]

A–>.b [3rd production]

STEP2 – Find LR(0) collection of items

Below is the figure showing the LR(0) collection of items. We will understand everything one by one.



The terminals of this grammar are {a,b}.
The non-terminals of this grammar are {S,A}

RULE –
If any non-terminal has ‘ . ‘ preceding it, we have to write all its production and add ‘ . ‘ preceding each of its production.
RULE –
from each state to the next state, the ‘ . ‘ shifts to one place to the right.

- In the figure, I0 consists of augmented grammar.
- Io goes to I1 when ‘ . ‘ of 0th production is shifted towards the right of S(S’->S.). this state is the accepted state. S is seen by the compiler.
- Io goes to I2 when ‘ . ‘ of 1st production is shifted towards right (S->A.A) . A is seen by the compiler
- I0 goes to I3 when ‘ . ‘ of the 2nd production is shifted towards right (A->a.A) . a is seen by the compiler.
- I0 goes to I4 when ‘ . ‘ of the 3rd production is shifted towards right (A->b.) . b is seen by the compiler.

- I2 goes to I5 when ‘ . ‘ of 1st production is shifted towards right (S->AA.) . A is seen by the compiler
- I2 goes to I4 when ‘ . ‘ of 3rd production is shifted towards right (A->b.) . b is seen by the compiler.
- I2 goes to I3 when ‘ . ‘ of the 2nd production is shifted towards right (A->a.A) . a is seen by the compiler.
- I3 goes to I4 when ‘ . ‘ of the 3rd production is shifted towards right (A->b.) . b is seen by the compiler.
- I3 goes to I6 when ‘ . ‘ of 2nd production is shifted towards the right (A->aA.) . A is seen by the compiler
- I3 goes to I3 when ‘ . ‘ of the 2nd production is shifted towards right (A->a.A) . a is seen by the compiler.

STEP3 –
Find FOLLOW of LHS of production
FOLLOW(S)=$
FOLLOW(A)=a,b,$

To find FOLLOW of non-terminals, please read follow set in syntax analysis.
STEP 4-
Defining 2 functions:goto[list of non-terminals] and action[list of terminals] in the parsing table.
Below is the SLR parsing table.

	ACTION			
	a	b	$	A
0	S3	S4		2
1			accept	
2	S3	S4		5
3	S3	S4		6
4	R3	R3	R3	
5			R1	
6	R2	R2	R2	

- $ is by default a nonterminal that takes accepting state.
- 0,1,2,3,4,5,6 denotes I0,I1,I2,I3,I4,I5,I6
- I0 gives A in I2, so 2 is added to the A column and 0 rows.
- I0 gives S in I1,so 1 is added to the S column and 1 row.
- similarly 5 is written in A column and 2 row, 6 is written in A column and 3 row.
- I0 gives a in I3 .so S3(shift 3) is added to a column and 0 row.
- I0 gives b in I4 .so S4(shift 4) is added to the b column and 0 row.
- Similarly, S3(shift 3) is added on a column and 2,3 row ,S4(shift 4) is added on b column and 2,3 rows.
- I4 is reduced state as ‘ . ‘ is at the end. I4 is the 3rd production of grammar(A–>.b).LHS of this production is A. FOLLOW(A)=a,b,$. write r3(reduced 3) in the columns of a,b,$ and 4th row
- I5 is reduced state as ‘ . ‘ is at the end. I5 is the 1st production of grammar(S–>.AA). LHS of this production is S.
  FOLLOW(S)=$. write r1(reduced 1) in the column of $ and 5th row

- I6 is a reduced state as ‘ . ‘ is at the end. I6 is the 2nd production of grammar(A–>.aA). The LHS of this production is A.
  FOLLOW(A)=a,b,$. write r2(reduced 2) in the columns of a,b,$ and 6th row

2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	*50 mins*	*Medium*	*CLO-4*

Activity 1:
Design SLR for the CFG of TINY C.

TINY C

Keywords: begin(){, }end, int, float, if, for, else, then, print

Operators: +, =, <

Variables: same criterion as that of C language

Constants: digits and floating point numbers

Punctuation Symbols: {, }, (,), ;

Input string for making SLR

Begin(){

int a=5;

int b=10;

int c=0;

c=a+b;

if(c>a)

print a;

else print c;

}end

Solution:

```
Store the input in an array named finalArray having an index named pointer.
   //Initializations

        ArrayList States = new ArrayList();
                Stack<String> Stack = new Stack<String>();
                String Parser;
                String[] Col = { "begin" ,"(",")","{","int","a","b",
        "c","=","5","10","0",";","if",">","print",

        "else","$","}","+","end","Program","DecS","AssS","IffS","PriS","V
        ar","Const" };
```

```
#region Bottom Up Parser
            States.Add("Program_begin ( ) { DecS Decs Decs AssS IffS }
end");
            States.Add("DecS_int Var = Const ;");
            States.Add("AssS_Var = Var + Var ;");
            States.Add("IffS_if ( Var > Var ) { PriS } else { PriS }");
            States.Add("PriS_print Var ;");
            States.Add("Var_a");
            States.Add("Var_b");
            States.Add("Var_c");
            States.Add("Const_5");
            States.Add("Const_10");
            States.Add("Const_0");
            Stack.Push("0");
            finalArray.Add("$");
            int pointer = 0;
            #region ParseTable
            var dict = new Dictionary<string, Dictionary<String,
object>>();
            dict.Add("0", new Dictionary<String, object>()
            {
                { "begin", "S2" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "1" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
dict.Add("1", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
```

```
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "Accept" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("2", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "S3" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
```

```
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("3", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "S4" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("4", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "S5" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
```

```
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("5", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "S13" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "6" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("6", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "S13" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
```

```
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "7" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("7", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "S13" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "8" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
dict.Add("8", new Dictionary<String, object>()
            {
                { "begin", "" },
```

```
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "S40" },
            { "b", "S42" },
            { "c", "S44" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "9" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "18" },
            { "Const", "" }
        });
        dict.Add("9", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "S24" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
```

```
                { "IffS", "10" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
dict.Add("10", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "S11" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("11", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
```

```
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "S12" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
dict.Add("12", new Dictionary<String, object>()
            {
                { "begin", "R1" },
                { "(", "R1" },
                { ")", "R1" },
                { "{", "R1" },
                { "int", "R1" },
                { "a", "R1" },
                { "b", "R1" },
                { "c", "R1" },
                { "=", "R1" },
                { "5", "R1" },
                { "10", "R1" },
                { "0", "R1" },
                { ";", "R1" },
                { "if", "R1" },
                { ">", "R1" },
                { "print", "R1" },
                { "else", "R1" },
                { "$", "R1" },
                { "}", "R1" },
                { "+", "R1" },
                { "end", "R1" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("13", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "S40" },
                { "b", "S42" },
                { "c", "S44" },
```

```
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "14" },
                { "Const", "" }
            });


dict.Add("14", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "S15" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "2" },
                { "IffS", "1" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
```

```
            dict.Add("15", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "S41" },
                { "10", "S43" },
                { "0", "S45" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "16" }
            });
            dict.Add("16", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "S17" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
```

```
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("17", new Dictionary<String, object>()
        {
            { "begin", "R2" },
            { "(", "R2" },
            { ")", "R2" },
            { "{", "R2" },
            { "int", "R2" },
            { "a", "R2" },
            { "b", "R2" },
            { "c", "R2" },
            { "=", "R2" },
            { "5", "R2" },
            { "10", "R2" },
            { "0", "R2" },
            { ";", "R2" },
            { "if", "R2" },
            { ">", "R2" },
            { "print", "R2" },
            { "else", "R2" },
            { "$", "R2" },
            { "}", "R2" },
            { "+", "R2" },
            { "end", "R2" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("18", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "S19" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
```

```
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("19", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "S40" },
            { "b", "S42" },
            { "c", "S44" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "20" },
            { "Const", "" }
        });
        dict.Add("20", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
```

```
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "S21" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
dict.Add("21", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "S40" },
                { "b", "S42" },
                { "c", "S44" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "22" },
                { "Const", "" }
```

```
            });
            dict.Add("22", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "S23" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("23", new Dictionary<String, object>()
            {
                { "begin", "R3" },
                { "(", "R3" },
                { ")", "R3" },
                { "{", "R3" },
                { "int", "R3" },
                { "a", "R3" },
                { "b", "R3" },
                { "c", "R3" },
                { "=", "R3" },
                { "5", "R3" },
                { "10", "R3" },
                { "0", "R3" },
                { ";", "R3" },
                { "if", "R3" },
                { ">", "R3" },
                { "print", "R3" },
                { "else", "R3" },
                { "$", "R3" },
                { "}", "R3" },
                { "+", "R3" },
```

```
                { "end", "R3" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("24", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "S25" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("25", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "S40" },
                { "b", "S42" },
                { "c", "S44" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
```

```
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "26" },
            { "Const", "" }
        });

        dict.Add("26", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "S27" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("27", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
```

```
            { "{", "" },
            { "int", "" },
            { "a", "S40" },
            { "b", "S42" },
            { "c", "S44" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "28" },
            { "Const", "" }
        });
        dict.Add("28", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "S29" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "2" },
            { "IffS", "1" },
            { "PriS", "" },
```

```
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("29", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "S30" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "2" },
            { "IffS", "1" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("30", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "S37" },
            { "else", "" },
            { "$", "" },
```

```
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "31" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("31", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "S32" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("32", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
```

```
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "S33" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("33", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "S34" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("34", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
```

```
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "S37" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "35" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("35", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "S36" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "2" },
            { "IffS", "1" },
```

```
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("36", new Dictionary<String, object>()
            {
                { "begin", "R4" },
                { "(", "R4" },
                { ")", "R4" },
                { "{", "R4" },
                { "int", "R4" },
                { "a", "R4" },
                { "b", "R4" },
                { "c", "R4" },
                { "=", "R4" },
                { "5", "R4" },
                { "10", "R4" },
                { "0", "R4" },
                { ";", "R4" },
                { "if", "R4" },
                { ">", "R4" },
                { "print", "R4" },
                { "else", "R4" },
                { "$", "R4" },
                { "}", "R4" },
                { "+", "R4" },
                { "end", "R4" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("37", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "S40" },
                { "b", "S42" },
                { "c", "S44" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
```

```
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "38" },
            { "Const", "" }
        });
        dict.Add("38", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "S39" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("39", new Dictionary<String, object>()
        {
            { "begin", "R5" },
            { "(", "R5" },
            { ")", "R5" },
            { "{", "R5" },
            { "int", "R5" },
            { "a", "R5" },
            { "b", "R5" },
            { "c", "R5" },
            { "=", "R5" },
```

```
            { "5", "R5" },
            { "10", "R5" },
            { "0", "R5" },
            { ";", "R5" },
            { "if", "R5" },
            { ">", "R5" },
            { "print", "R5" },
            { "else", "R5" },
            { "$", "R5" },
            { "}", "R5" },
            { "+", "R5" },
            { "end", "R5" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("40", new Dictionary<String, object>()
        {
            { "begin", "R6" },
            { "(", "R6" },
            { ")", "R6" },
            { "{", "R6" },
            { "int", "R6" },
            { "a", "R6" },
            { "b", "R6" },
            { "c", "R6" },
            { "=", "R6" },
            { "5", "R6" },
            { "10", "R6" },
            { "0", "R6" },
            { ";", "R6" },
            { "if", "R6" },
            { ">", "R6" },
            { "print", "R6" },
            { "else", "R6" },
            { "$", "R6" },
            { "}", "R6" },
            { "+", "R6" },
            { "end", "R6" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });

        dict.Add("41", new Dictionary<String, object>()
        {
```

```
            { "begin", "R9" },
            { "(", "R9" },
            { ")", "R9" },
            { "{", "R9" },
            { "int", "R9" },
            { "a", "R9" },
            { "b", "R9" },
            { "c", "R9" },
            { "=", "R9" },
            { "5", "R9" },
            { "10", "R9" },
            { "0", "R9" },
            { ";", "R9" },
            { "if", "R9" },
            { ">", "R9" },
            { "print", "R9" },
            { "else", "R9" },
            { "$", "R9" },
            { "}", "R9" },
            { "+", "R9" },
            { "end", "R9" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("42", new Dictionary<String, object>()
        {
            { "begin", "R7" },
            { "(", "R7" },
            { ")", "R7" },
            { "{", "R7" },
            { "int", "R7" },
            { "a", "R7" },
            { "b", "R7" },
            { "c", "R7" },
            { "=", "R7" },
            { "5", "R7" },
            { "10", "R7" },
            { "0", "R7" },
            { ";", "R7" },
            { "if", "R7" },
            { ">", "R7" },
            { "print", "R7" },
            { "else", "R7" },
            { "$", "R7" },
            { "}", "R7" },
            { "+", "R7" },
            { "end", "R7" },
            { "Program", "" },
            { "DecS", "" },
```

```
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("43", new Dictionary<String, object>()
            {
                { "begin", "R10" },
                { "(", "R10" },
                { ")", "R10" },
                { "{", "R10" },
                { "int", "R10" },
                { "a", "R10" },
                { "b", "R10" },
                { "c", "R10" },
                { "=", "R10" },
                { "5", "R10" },
                { "10", "R10" },
                { "0", "R10" },
                { ";", "R10" },
                { "if", "R10" },
                { ">", "R10" },
                { "print", "R10" },
                { "else", "R10" },
                { "$", "R10" },
                { "}", "R10" },
                { "+", "R10" },
                { "end", "R10" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("44", new Dictionary<String, object>()
            {
                { "begin", "R8" },
                { "(", "R8" },
                { ")", "R8" },
                { "{", "R8" },
                { "int", "R8" },
                { "a", "R8" },
                { "b", "R8" },
                { "c", "R8" },
                { "=", "R8" },
                { "5", "R8" },
                { "10", "R8" },
                { "0", "R8" },
                { ";", "R8" },
                { "if", "R8" },
                { ">", "R8" },
```

```
                { "print", "R8" },
                { "else", "R8" },
                { "$", "R8" },
                { "}", "R8" },
                { "+", "R8" },
                { "end", "R8" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("45", new Dictionary<String, object>()
            {
                { "begin", "R11" },
                { "(", "R11" },
                { ")", "R11" },
                { "{", "R11" },
                { "int", "R11" },
                { "a", "R11" },
                { "b", "R11" },
                { "c", "R11" },
                { "=", "R11" },
                { "5", "R11" },
                { "10", "R11" },
                { "0", "R11" },
                { ";", "R11" },
                { "if", "R11" },
                { ">", "R11" },
                { "print", "R11" },
                { "else", "R11" },
                { "$", "R11" },
                { "}", "R11" },
                { "+", "R11" },
                { "end", "R11" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
#endregion

            while (true)
            {
                if (!Col.Contains(finalArray[pointer]))
                {
                    Output.AppendText("Unable to Parse Unknown Input");
                    break;
                }
```

```
                Parser = dict[Stack.Peek() + ""][finalArray[pointer] +
""] + "";
                if (Parser.Contains("S"))
                {
                    Stack.Push(finalArray[pointer] + "");
                    Parser = Parser.TrimStart('S');
                    Stack.Push(Parser);
                    pointer++;
                    Print_Stack();
                }
                if (Parser.Contains("R"))
                {
                    Parser = Parser.TrimStart('R');
                    String get = States[Convert.ToInt32(Parser) - 1] +
"";
                    String[] Splitted = get.Split('_');
                    String[] Final_ = Splitted[1].Split(' ');
                    int test = Final_.Length;
                    for (int i = 0; i < test * 2; i++)
                    { Stack.Pop(); }
                    String row = Stack.Peek() + "";
                    Stack.Push(Splitted[0]);
                    Stack.Push(dict[row][Stack.Peek()] + "");
                    Print_Stack();
                }
                if (Parser.Contains("Accept"))
                {
                    Output.AppendText("Parsed");
                    break;
                }
                if (Parser.Equals(""))
                {
                    Output.AppendText("Unable to Parse");
                    break;
                }
            }
            finalArray.Remove("$");
            finalArray.Remove("begin");
            #endregion
```

OUTPUT:

	Action						Goto		
State	id	+	*	(	)	$	E	T	F
0	S5			S4			1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				
9		R1	S7		R1	R1			
10		R3	R3		R3	R3			
11		R5	R5		R5	R5			

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1:

Understand the SLR code, mentioned in lab activity 1, written with the help of dictionary and stack classes of C#, execute the same with the output.

Lab 11

Syntax-Directed Translation for Semantic Analyzer

Objective:

In this lab students will learn to convert Parse tree into Annotated Parse tree by syntax-directed Translation.

Activity Outcomes:

On completion of this lab, student will be able to:

- Implement semantic analyzer

Instructor Note:

As for this lab activity, read chapter 06 from the book "Compiler Construction-Principles and Practices" by Kenneth C. Louden.

1) Useful Concepts

Semantic analysis, also context sensitive analysis, is a process in compiler construction, usually after parsing, to gather necessary semantic information from the source code. It usually includes type checking, or makes sure a variable is declared before use which is impossible to describe in Extended Backus–Naur Form and thus not easily detected during parsing.

Semantics of a language provide meaning to its constructs, like tokens and syntax structure. Semantics help interpret symbols, their types, and their relations with each other. Semantic analysis judges whether the syntax structure constructed in the source program derives any meaning or not.

```
CFG + semantic rules = Syntax Directed Definitions
```

For example:

```
int a = "value";
```

should not issue an error in lexical and syntax analysis phase, as it is lexically and structurally correct, but it should generate a semantic error as the type of the assignment differs. These rules are set by the grammar of the language and evaluated in semantic analysis.

2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	***75 mins***	***High***	***CLO-4***

Activity 1:
Implement the semantic analyzer for checking type incompatibilities in the source program

Solution:

i. Initialize finalArray with input

ii. Semantic analyzer uses information of symbol table so you must have the symbol table implemented before doing this lab.

iii. variable_Reg = new Regex(@"^[A-Za-z|_][A-Za-z|0-9]*$");

```
#region Semantic Analyzer
        void Semantic_Analysis(int k)
        {
            if (finalArray[k].Equals("+"))
            {
                if (variable_Reg.Match(finalArray[k - 1] +
"").Success && variable_Reg.Match(finalArray[k + 1] + "").Success)
                {
                    String type = finalArray[k - 4] + "";
                    String left_side = finalArray[k - 3] + "";
                    int left_side_i = 0;
                    int left_side_j = 0;
                    String before = finalArray[k - 1] + "";
                    int before_i = 0;
                    int before_j = 0;
                    String after = finalArray[k + 1] + "";
                    int after_i = 0;
                    int after_j = 0;
                    for (int i = 0; i < Symboltable.Count; i++)
                    {
                        for (int j = 0; j < Symboltable[i].Count;
j++)
                        {
                            if
(Symboltable[i][j].Equals(left_side))
                            { left_side_i = i; left_side_j = j; }
                            if (Symboltable[i][j].Equals(before))
                            { before_i = i; before_j = j; }
                            if (Symboltable[i][j].Equals(after))
                            { after_i = i; after_j = j; }
                        }
                    }
                    if (type.Equals(Symboltable[before_i][2]) &&
type.Equals(Symboltable[after_i][2]) &&
Symboltable[before_i][2].Equals(Symboltable[after_i][2]))
                    {
                        int Ans =
Convert.ToInt32(Symboltable[before_i][3]) +
Convert.ToInt32(Symboltable[after_i][3]);
                        Constants.Add(Ans);
                    }
                    if
(Symboltable[left_side_i][2].Equals(Symboltable[before_i][2]) &&
```

```
Symboltable[left_side_i][2].Equals(Symboltable[after_i][2]) &&
Symboltable[before_i][2].Equals(Symboltable[after_i][2]))
                    {
                        int Ans =
Convert.ToInt32(Symboltable[before_i][3]) +
Convert.ToInt32(Symboltable[after_i][3]);
                        Constants.RemoveAt(Constants.Count - 1);
                        Constants.Add(Ans);
                        Symboltable[left_side_i][3] = Ans + "";
                    }
                }
            }
            if (finalArray[k].Equals("-"))
            {
                if (variable_Reg.Match(finalArray[k - 1] +
"").Success && variable_Reg.Match(finalArray[k + 1] + "").Success)
                {
                    String type = finalArray[k - 4] + "";
                    String left_side = finalArray[k - 3] + "";
                    int left_side_i = 0;
                    int left_side_j = 0;
                    String before = finalArray[k - 1] + "";
                    int before_i = 0;
                    int before_j = 0;
                    String after = finalArray[k + 1] + "";
                    int after_i = 0;
                    int after_j = 0;
                    for (int i = 0; i < Symboltable.Count; i++)
                    {
                        for (int j = 0; j < Symboltable[i].Count;
j++)
                        {
                            if
(Symboltable[i][j].Equals(left_side))
                            { left_side_i = i; left_side_j = j; }
                            if (Symboltable[i][j].Equals(before))
                            { before_i = i; before_j = j; }
                            if (Symboltable[i][j].Equals(after))
                            { after_i = i; after_j = j; }
                        }
                    }
                    if (type.Equals(Symboltable[before_i][2]) &&
type.Equals(Symboltable[after_i][2]) &&
Symboltable[before_i][2].Equals(Symboltable[after_i][2]))
                    {
                        int Ans =
Convert.ToInt32(Symboltable[before_i][3]) -
Convert.ToInt32(Symboltable[after_i][3]);
                        Constants.Add(Ans);
                    }
                    if
(Symboltable[left_side_i][2].Equals(Symboltable[before_i][2]) &&
Symboltable[left_side_i][2].Equals(Symboltable[after_i][2]) &&
Symboltable[before_i][2].Equals(Symboltable[after_i][2]))
```

```
                    {
                        int Ans =
Convert.ToInt32(Symboltable[before_i][3]) +
Convert.ToInt32(Symboltable[after_i][3]);
                        Constants.RemoveAt(Constants.Count - 1);
                        Constants.Add(Ans);
                        Symboltable[left_side_i][3] = Ans + "";
                    }
                }
            }
            if (finalArray[k].Equals(">"))
            {
                if (variable_Reg.Match(finalArray[k - 1] +
"").Success && variable_Reg.Match(finalArray[k + 1] + "").Success)
                {
                    String before = finalArray[k - 1] + "";
                    int before_i = 0;
                    int before_j = 0;
                    String after = finalArray[k + 1] + "";
                    int after_i = 0;
                    int after_j = 0;
                    for (int i = 0; i < Symboltable.Count; i++)
                    {
                        for (int j = 0; j < Symboltable[i].Count;
j++)
                        {
                            if (Symboltable[i][j].Equals(before))
                            { before_i = i; before_j = j; }
                            if (Symboltable[i][j].Equals(after))
                            { after_i = i; after_j = j; }
                        }
                    }
                    if (Convert.ToInt32(Symboltable[before_i][3])
> Convert.ToInt32(Symboltable[after_i][3]))
                    {
                        int start_of_else =
finalArray.IndexOf("else");
                        int end_of_else = finalArray.Count - 1;
                        for (int i = end_of_else; i >=
start_of_else; i--)
                        {
                            if (finalArray[i].Equals("}"))
                            {
                                if (i < finalArray.Count - 2)
                                { end_of_else = i; }
                            }
                        }

                        for (int i = start_of_else; i <=
end_of_else; i++)
                        { finalArray.RemoveAt(start_of_else); }
                    }
                    else
                    {
```

```
                            int start_of_if =
finalArray.IndexOf("if");
                            int end_of_if = finalArray.IndexOf("}");

                            for (int i = start_of_if; i <= end_of_if;
i++)
                            { finalArray.RemoveAt(start_of_if); }
                            if_deleted = true;
                        }
                    }
                }
            }
            #endregion
```

Output:

```
Tokenizing src/main/resources/tests/lexer02.txt...
INT (1,1)
MAIN (1,5)
LPAREN (1,9)
RPAREN (1,10)
LBRACE (1,11)
INT (2,3)
ID (2,7): x
SEMI (2,8)
ID (3,3): x
SEMI (3,4)
ID (4,3): x
ASSIGN (4,5)
INT_CONST (4,7): 2
PLUS (4,9)
INT_CONST (4,11): 5
PLUS (4,13)
LPAREN (4,15)
INT_CONST (4,16): 4
TIMES (4,18)
INT_CONST (4,20): 8
RPAREN (4,21)
PLUS (4,23)
CHAR_CONST (4,27): l
DIV (4,29)
FLOAT_CONST (4,31): 9.0
SEMI (4,34)
IF (5,3)
LPAREN (5,5)
ID (5,6): x
PLUS (5,8)
ID (5,10): y
RPAREN (5,11)
LBRACE (5,12)
IF (6,5)
LPAREN (6,8)
ID (6,10): x
NEQ (6,11)
INT_CONST (6,14): 4
RPAREN (6,15)
LBRACE (6,16)
```

```
ID (7,7): x
ASSIGN (7,9)
INT_CONST (7,11): 6
SEMI (7,12)
ID (8,7): y
ASSIGN (8,9)
INT_CONST (8,11): 10
SEMI (8,13)
ID (9,7): i
ASSIGN (9,9)
INT_CONST (9,11): 11
SEMI (9,13)
RBRACE (10,5)
RBRACE (11,3)
RBRACE (12,1)
EOF (12,2)
---
```

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1:

Implement Syntax Directed Translation for each node of the above grammar

Lab 12

Integration: Lexical Analyzer and symbol table (Ph-1)

Objective:

The objective of this lab is to implement the integration of lexical analyzer with the symbol table.

Activity Outcomes:

On completion of this lab, students will be able to:

- integrate lexical analyzer with symbol table

Instructor Note:

As for this lab activity, read article 6.4(Chapter 6) from the book "Compiler Construction-Principles and Practices" by Kenneth C. Louden

1) Useful Concepts.

Task of lexical analyzer is token generation. Generated tokens are then passed to the parser for syntax checking but lexical analyzer is also responsible for storing the information of variables i.e. their name, data type, line number and value in the symbol table.

2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	***60 mins***	***Medium***	***CLO-5***

Activity 1:
Integrate lexical analyzer with symbol table

Solution:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Text.RegularExpressions;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Collections;

namespace LexicalAnalyzerV1
{
    public partial class Form1 : Form
    {
List<List<String>> Symboltable = new List<List<String>>();
```

```
        ArrayList LineNumber;
        ArrayList Variables;
        ArrayList KeyWords;
        ArrayList Constants;
        ArrayList finalArray;
        ArrayList tempArray;

        Regex variable_Reg;
        Regex constants_Reg;
        Regex operators_Reg;

        int lexemes_per_line;
        int ST_index;

        public Form1()
        {
            InitializeComponent();
String[] k_ = { "int", "float", "begin", "end", "print", "if",
"else" };
            ArrayList key = new ArrayList(k_);

            LineNumber = new ArrayList();
            Variables = new ArrayList();
            KeyWords = new ArrayList();
            Constants = new ArrayList();

            finalArray = new ArrayList();
            tempArray = new ArrayList();

            variable_Reg = new Regex(@"^[A-Za-z|_][A-Za-z|0-9]*$");
            constants_Reg = new Regex(@"^[0-9]+([.][0-
9]+)?([e]([+|-])?[0-9]+)?$");
            operators_Reg = new Regex(@"[+-/*=;>(){}]");

            int L = 1;

            Output.Text = "";
            ST.Text = "";

            Symboltable.Clear();

            if_deleted = false;

            string strinput = Input.Text;
            char[] charinput = strinput.ToCharArray();
        }

        private void btn_Input_Click(object sender, EventArgs e)
        {
            //taking user input from rich textbox
            String userInput = tfInput.Text;
            //List of keywords which will be used to seperate
keywords from variables
```

```
            List<String> keywordList = new List<String>();
            keywordList.Add("int");
            keywordList.Add("float");
            keywordList.Add("while");
            keywordList.Add("main");
            keywordList.Add("if");
            keywordList.Add("else");
            keywordList.Add("new");
            //row is an index counter for symbol table
            int row = 1;

            //count is a variable to incremenet variable id in
tokens
            int count = 1;

            //line_num is a counter for lines in user input
            int line_num = 0;

            //SymbolTable is a 2D array that has the following
structure
            //[Index][Variable Name][type][value][line#]
            //rows are incremented with each variable information
entry

            String[,] SymbolTable = new String[20, 6];
            List<String> varListinSymbolTable = new List<String>();

            //Input Buffering

            ArrayList finalArray = new ArrayList();
            ArrayList finalArrayc = new ArrayList();
            ArrayList tempArray = new ArrayList();

            char[] charinput = userInput.ToCharArray();

            //Regular Expression for Variables
            Regex variable_Reg = new Regex(@"^[A-Za-z|_][A-Za-z|0-
9]*$");
            //Regular Expression for Constants
            Regex constants_Reg = new Regex(@"^[0-9]+([.][0-
9]+)?([e]([+|-])?[0-9]+)?$");
            //Regular Expression for Operators
            Regex operators_Reg = new Regex(@"^[-*+/><&&||=]$");
            //Regular Expression for Special_Characters
            Regex Special_Reg = new Regex(@"^[.,'\[\]{}();:?]$");

            for (int itr = 0; itr < charinput.Length; itr++)
            {
                Match Match_Variable =
variable_Reg.Match(charinput[itr] + "");
                Match Match_Constant =
constants_Reg.Match(charinput[itr] + "");
                Match Match_Operator =
operators_Reg.Match(charinput[itr] + "");
```

```
                Match Match_Special =
Special_Reg.Match(charinput[itr] + "");
                if (Match_Variable.Success ||
Match_Constant.Success || Match_Operator.Success ||
Match_Special.Success || charinput[itr].Equals(' '))
                {
                    tempArray.Add(charinput[itr]);
                }
                if (charinput[itr].Equals('\n'))
                {
                    if (tempArray.Count != 0)
                    {
                        int j = 0;
                        String fin = "";
                        for (; j < tempArray.Count; j++)
                        {
                            fin += tempArray[j];
                        }

                        finalArray.Add(fin);
                        tempArray.Clear();
                    }

                }
            }
            if (tempArray.Count != 0)
            {
                int j = 0;
                String fin = "";
                for (; j < tempArray.Count; j++)
                {
                    fin += tempArray[j];
                }
                finalArray.Add(fin);
                tempArray.Clear();
            }

// Final Array SO far correct
            tfTokens.Clear();

            symbolTable.Clear();

            //looping on all lines in user input
            for (int i = 0; i < finalArray.Count; i++)
            {
               String line = finalArray[i].ToString();
               //tfTokens.AppendText(line + "\n");
               char[] lineChar = line.ToCharArray();
               line_num++;
               //taking current line and splitting it into lexemes
by space

                for (int itr = 0; itr < lineChar.Length; itr++)
                {
```

```
                    Match Match_Variable =
variable_Reg.Match(lineChar[itr] + "");
                    Match Match_Constant =
constants_Reg.Match(lineChar[itr] + "");
                    Match Match_Operator =
operators_Reg.Match(lineChar[itr] + "");
                    Match Match_Special =
Special_Reg.Match(lineChar[itr] + "");
                    if (Match_Variable.Success ||
Match_Constant.Success)
                    {
                        tempArray.Add(lineChar[itr]);
                    }
                    if (lineChar[itr].Equals(' '))
                    {
                        if (tempArray.Count != 0)
                        {
                            int j = 0;
                            String fin = "";
                            for (; j < tempArray.Count; j++)
                            {
                                fin += tempArray[j];
                            }
                            finalArrayc.Add(fin);
                            tempArray.Clear();
                        }

                    }
                    if (Match_Operator.Success ||
Match_Special.Success)
                    {
                        if (tempArray.Count != 0)
                        {
                            int j = 0;
                            String fin = "";
                            for (; j < tempArray.Count; j++)
                            {
                                fin += tempArray[j];
                            }
                            finalArrayc.Add(fin);
                            tempArray.Clear();
                        }
                        finalArrayc.Add(lineChar[itr]);
                    }
                }
                if (tempArray.Count != 0)
                {
                        String fina = "";
                        for (int k = 0; k < tempArray.Count; k++)
                        {
                            fina += tempArray[k];
                        }
                        finalArrayc.Add(fina);
```

```
                        tempArray.Clear();
                    }
                // we have asplitted line here
                    for (int x = 0; x < finalArrayc.Count; x++)
                    {
                        Match operators =
operators_Reg.Match(finalArrayc[x].ToString());
                        Match variables =
variable_Reg.Match(finalArrayc[x].ToString());
                        Match digits =
constants_Reg.Match(finalArrayc[x].ToString());
                        Match punctuations =
Special_Reg.Match(finalArrayc[x].ToString());

                        if (operators.Success)
                        {
                            // if a current lexeme is an operator
then make a token e.g. < op, = >
                            tfTokens.AppendText("< op, " +
finalArrayc[x].ToString() + "> ");
                        }
                        else if (digits.Success)
                        {
                            // if a current lexeme is a digit then
make a token e.g. < digit, 12.33 >
                            tfTokens.AppendText("< digit, " +
finalArrayc[x].ToString() + "> ");
                        }
                        else if (punctuations.Success)
                        {
                            // if a current lexeme is a punctuation
then make a token e.g. < punc, ; >
                            tfTokens.AppendText("< punc, " +
finalArrayc[x].ToString() + "> ");
                        }

                        else if (variables.Success)
                        {
                            // if a current lexeme is a variable
and not a keyword
                            if
(!keywordList.Contains(finalArrayc[x].ToString())) // if it is not
a keyword
                            {
                                // check what is the category of
varaible, handling only two cases here
                                //Category1- Variable
initialization of type digit e.g. int count = 10 ;
                                //Category2- Variable
initialization of type String e.g. String var = ' Hello ' ;

                                Regex reg1 = new
Regex(@"^(int|float|double)\s([A-Za-z|_][A-Za-z|0-
9]{0,10})\s(=)\s([0-9]+([.][0-9]+)?([e][+|-]?[0-9]+)?)\s(;)$"); //
```

```
line of type int alpha = 2 ;
                                    Match category1 = reg1.Match(line);

                                    Regex reg2 = new
Regex(@"^(String|char)\s([A-Za-z|_][A-Za-z|0-
9]{0,10})\s(=)\s[']\s([A-Za-z|_][A-Za-z|0-9]{0,30})\s[']\s(;)$");
// line of type String alpha = ' Hello ' ;
                                    Match category2 = reg2.Match(line);

                                    //if it is a category 1 then add a
row in symbol table containing the information related to that
variable

                                    if (category1.Success)
                                    {
                                        SymbolTable[row, 1] =
row.ToString(); //index

                                        SymbolTable[row, 2] =
finalArrayc[x].ToString(); //variable name

                                        SymbolTable[row, 3] =
finalArrayc[x - 1].ToString(); //type

                                        SymbolTable[row, 4] =
finalArrayc[x+2].ToString(); //value

                                        SymbolTable[row, 5] =
line_num.ToString(); // line number

                                        tfTokens.AppendText("<var" +
count + ", " + row + "> ");

symbolTable.AppendText(SymbolTable[row, 1].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 2].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 3].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 4].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 5].ToString() + " \n ");
                                        row++;
                                        count++;
                                    }
                                    //if it is a category 2 then add a
row in symbol table containing the information related to that
variable
                                    else if (category2.Success)
                                    {
                                        // if  a line such as String
var = ' Hello ' ; comes and the loop moves to index of array
containing Hello ,
                                        //then this if condition
```

```
prevents addition of Hello in symbol Table because it is not a
variable it is just a string

                                    if (!(finalArrayc[x-
1].ToString().Equals("'") &&
finalArrayc[x+1].ToString().Equals("'")))

                                    {
                                        SymbolTable[row, 1] =
row.ToString(); // index

                                        SymbolTable[row, 2] =
finalArrayc[x].ToString(); //varname

                                        SymbolTable[row, 3] =
finalArrayc[x-1].ToString(); //type

                                        SymbolTable[row, 4] =
finalArrayc[x+3].ToString(); //value

                                        SymbolTable[row, 5] =
line_num.ToString(); // line number

                                        tfTokens.AppendText("<var"
+ count + ", " + row + "> ");

symbolTable.AppendText(SymbolTable[row, 1].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 2].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 3].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 4].ToString() + " \t ");

symbolTable.AppendText(SymbolTable[row, 5].ToString() + " \n ");
                                        row++;
                                        count++;
                                    }
                                    else
                                    {
                                tfTokens.AppendText("<String" + count +
", " + finalArrayc[x].ToString() + "> ");
                                    }
                                }
                                else
                                {
// if any other category line comes in we check if we have
initializes that varaible before,
// if we have initiazed it before then we put the index of that
variable in symbol table, in its token
                                    String ind = "Default";
                                    String ty = "Default";
                                    String val = "Default";
                                    String lin = "Default";
```

```
for (int r = 1; r <= SymbolTable.GetLength(0); r++)
  {
    //search in the symbol table if variable entry already exists
                                            if (SymbolTable[r,
2].Equals(finalArrayc[x].ToString()))
                                            {
                                                ind = SymbolTable[r,
1];
                                                ty = SymbolTable[r, 3];
                                                val = SymbolTable[r,
4];
                                                lin = SymbolTable[r,
5];

tfTokens.AppendText("<var" + ind + ", " + ind + "> ");

                                                break;
                                            }
                                        }
                                    }
                                }
 // if a current lexeme is not a variable but a keyword then make a
token such as: <keyword, int>
                                else
                                {
                                    tfTokens.AppendText("<keyword, " +
finalArrayc[x].ToString() + "> ");
                                }
                            }
                        }
                    tfTokens.AppendText("\n");
                    finalArrayc.Clear();
                }
            }
        }
}
#region Display Symbol Table
            for (int j = 0; j < Symboltable.Count; j++)
            {
                for (int z = 0; z < Symboltable[j].Count; z++)
                { ST.AppendText(Symboltable[j][z] + "\t"); }
                ST.AppendText("\n");
            }
            #endregion
        }
#region Make Entry Symbol Table
        void Check_And_Make_Entries()
        {
            KeyWords.Remove("begin"); KeyWords.Remove("end");
KeyWords.Remove("print");
            KeyWords.Remove("if"); KeyWords.Remove("else");
            if (lexemes_per_line - 4 == 0 || lexemes_per_line - 7
== 0)
            {
```

```
                if (lexemes_per_line == 7)
                {
                    Variables.RemoveAt(Variables.Count - 1);
Variables.RemoveAt(Variables.Count - 1);
                }
                for (; ST_index < KeyWords.Count; ST_index++)
                {
                    Symboltable.Add(new List<string>());
                    Symboltable[ST_index].Add(ST_index + 1 + "");
                    Symboltable[ST_index].Add(Variables[ST_index] +
"");
                    Symboltable[ST_index].Add(KeyWords[ST_index] +
"");
                    Symboltable[ST_index].Add(Constants[ST_index] +
"");
                    Symboltable[ST_index].Add(LineNumber[ST_index]
+ "");
                }
            }
            if (lexemes_per_line - 6 == 0)
            {
                Variables.RemoveAt(Variables.Count - 1);
Variables.RemoveAt(Variables.Count - 1);
Variables.RemoveAt(Variables.Count - 1);
            }
        }
      #endregion
```

OUTPUT:

```
ERROR: RPAREN at line 4, column 6; Expected EXPRESSION

ERROR: LBRACE at line 4, column 7; Expected RPAREN
```

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1:
Understand the integrated code, execute it to get the desired output.

Lab 13

Integration: Ph-1 and Semantic Analyzer(Ph-2)

Objective:

Lexical analyzer generates tokens and passes them to the parser for syntax analysis.

Activity Outcomes:

This lab teaches you

- How to integrate lexical analyzer with parser

Instructor Note:

As for this lab activity, read chapter 6 & 8 from the book "Compiler Construction-Principles and Practices" by Kenneth C. Louden

1) Useful Concepts

Task of lexical analyzer is token generation. Generated tokens are then passed to the parser for syntax checking. Parser verifies their syntax with the help of context free grammar of that language. This parser uses bottom up strategy to parse the tokens.

2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	***60 mins***	***HIGH***	***CLO-5***

Activity 1:
Integrate lexical analyzer and parser

Solution:

```
public partial class Form1 : Form
    {
        ArrayList States = new ArrayList();
        Stack<String> Stack = new Stack<String>();
        String Parser;
        String[] Col = { "begin" ,"(",")","{","int","a","b",
"c","=","5","10","0",";","if",">","print",

"else","$","}","+","end","Program","DecS","AssS","IffS","PriS","Var","Con
st" };

        public Form1()
        {
            InitializeComponent();
        }

        List<List<String>> Symboltable = new List<List<String>>();

        ArrayList LineNumber;
        ArrayList Variables;
        ArrayList KeyWords;
        ArrayList Constants;
        ArrayList finalArray;
        ArrayList tempArray;

        Regex variable_Reg;
        Regex constants_Reg;
        Regex operators_Reg;

        int lexemes_per_line;
        int ST_index;

        Boolean if_deleted;

        private void Compile_Click(object sender, EventArgs e)
        {
            String[] k_ = { "int", "float", "begin", "end", "print",
"if", "else" };
            ArrayList key = new ArrayList(k_);

            LineNumber = new ArrayList();
            Variables = new ArrayList();
            KeyWords = new ArrayList();
            Constants = new ArrayList();

            finalArray = new ArrayList();
            tempArray = new ArrayList();

            variable_Reg = new Regex(@"^[A-Za-z|_][A-Za-z|0-9]*$");
            constants_Reg = new Regex(@"^[0-9]+([.][0-9]+)?([e]([+|-
])?[0-9]+)?$");
            operators_Reg = new Regex(@"[+-/*=;>(){}]");

            int L = 1;
```

```
            Output.Text = "";
            ST.Text = "";

            Symboltable.Clear();

            if_deleted = false;

            string strinput = Input.Text;
            char[] charinput = strinput.ToCharArray();

////////////////////////////////Start_Split
Function/////////////////////////////////////////////////////////
            #region Input Buffering
            for (int itr = 0; itr < charinput.Length; itr++)
            {
                Match Match_Variable = variable_Reg.Match(charinput[itr]
+ "");
                Match Match_Constant = constants_Reg.Match(charinput[itr]
+ "");
                Match Match_Operator = operators_Reg.Match(charinput[itr]
+ "");

                if (Match_Variable.Success || Match_Constant.Success)
                {
                    tempArray.Add(charinput[itr]);
                }
                if (charinput[itr].Equals(' '))
                {
                    if (tempArray.Count != 0)
                    {
                        int j = 0;
                        String fin = "";
                        for (; j < tempArray.Count; j++)
                        {
                            fin += tempArray[j];
                        }
                        finalArray.Add(fin);
                        tempArray.Clear();
                    }
                }
                if (Match_Operator.Success)
                {
                    if (tempArray.Count != 0)
                    {
                        int j = 0;
                        String fin = "";
                        for (; j < tempArray.Count; j++)
                        {
                            fin += tempArray[j];
                        }
                        finalArray.Add(fin);
                        tempArray.Clear();
                    }
```

```
                    finalArray.Add(charinput[itr]+"");
                }
            }
            if (tempArray.Count != 0)
            {
                String final = "";
                for (int k = 0; k < tempArray.Count; k++)
                {
                    final += tempArray[k];
                }
                finalArray.Add(final);
            }
            #endregion
////////////////////////////////End_Split
Function////////////////////////////////////////////////////////////
            #region Bottom Up Parser
            States.Add("Program_begin ( ) { DecS Decs Decs AssS IffS }
end");
            States.Add("DecS_int Var = Const ;");
            States.Add("AssS_Var = Var + Var ;");
            States.Add("IffS_if ( Var > Var ) { PriS } else { PriS }");
            States.Add("PriS_print Var ;");
            States.Add("Var_a");
            States.Add("Var_b");
            States.Add("Var_c");
            States.Add("Const_5");
            States.Add("Const_10");
            States.Add("Const_0");
            Stack.Push("0");
            finalArray.Add("$");
            int pointer = 0;
            #region ParseTable
            var dict = new Dictionary<string, Dictionary<String,
object>>();
            dict.Add("0", new Dictionary<String, object>()
            {
                { "begin", "S2" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
```

```
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "1" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("1", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "Accept" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("2", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "S3" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
```

```
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("3", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "S4" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("4", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
```

```
            { ")", "" },
            { "{", "S5" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("5", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "S13" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "6" },
            { "AssS", "" },
            { "IffS", "" },
```

```
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("6", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "S13" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "7" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("7", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "S13" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
```

```
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "8" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("8", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "S40" },
                { "b", "S42" },
                { "c", "S44" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "9" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "18" },
                { "Const", "" }
            });
            dict.Add("9", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
```

```
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "S24" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "10" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("10", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "S11" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("11", new Dictionary<String, object>()
        {
            { "begin", "" },
```

```
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "S12" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("12", new Dictionary<String, object>()
        {
            { "begin", "R1" },
            { "(", "R1" },
            { ")", "R1" },
            { "{", "R1" },
            { "int", "R1" },
            { "a", "R1" },
            { "b", "R1" },
            { "c", "R1" },
            { "=", "R1" },
            { "5", "R1" },
            { "10", "R1" },
            { "0", "R1" },
            { ";", "R1" },
            { "if", "R1" },
            { ">", "R1" },
            { "print", "R1" },
            { "else", "R1" },
            { "$", "R1" },
            { "}", "R1" },
            { "+", "R1" },
            { "end", "R1" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
```

```
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("13", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "S40" },
            { "b", "S42" },
            { "c", "S44" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "14" },
            { "Const", "" }
        });
        dict.Add("14", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "S15" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
```

```
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "2" },
                { "IffS", "1" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("15", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "S41" },
                { "10", "S43" },
                { "0", "S45" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "16" }
            });
            dict.Add("16", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
```

```
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "S17" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("17", new Dictionary<String, object>()
        {
            { "begin", "R2" },
            { "(", "R2" },
            { ")", "R2" },
            { "{", "R2" },
            { "int", "R2" },
            { "a", "R2" },
            { "b", "R2" },
            { "c", "R2" },
            { "=", "R2" },
            { "5", "R2" },
            { "10", "R2" },
            { "0", "R2" },
            { ";", "R2" },
            { "if", "R2" },
            { ">", "R2" },
            { "print", "R2" },
            { "else", "R2" },
            { "$", "R2" },
            { "}", "R2" },
            { "+", "R2" },
            { "end", "R2" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("18", new Dictionary<String, object>()
        {
```

```
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "S19" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("19", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "S40" },
            { "b", "S42" },
            { "c", "S44" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
```

```
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "20" },
                { "Const", "" }
            });
            dict.Add("20", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "S21" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("21", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "S40" },
                { "b", "S42" },
                { "c", "S44" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
```

```
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "22" },
                { "Const", "" }
            });
            dict.Add("22", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "S23" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("23", new Dictionary<String, object>()
            {
                { "begin", "R3" },
                { "(", "R3" },
                { ")", "R3" },
                { "{", "R3" },
                { "int", "R3" },
                { "a", "R3" },
                { "b", "R3" },
```

```
            { "c", "R3" },
            { "=", "R3" },
            { "5", "R3" },
            { "10", "R3" },
            { "0", "R3" },
            { ";", "R3" },
            { "if", "R3" },
            { ">", "R3" },
            { "print", "R3" },
            { "else", "R3" },
            { "$", "R3" },
            { "}", "R3" },
            { "+", "R3" },
            { "end", "R3" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("24", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "S25" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("25", new Dictionary<String, object>()
```

```
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "S40" },
                { "b", "S42" },
                { "c", "S44" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "26" },
                { "Const", "" }
            });
            dict.Add("26", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "S27" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
```

```
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("27", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "S40" },
                { "b", "S42" },
                { "c", "S44" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "28" },
                { "Const", "" }
            });
            dict.Add("28", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "S29" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
```

```
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "2" },
                { "IffS", "1" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("29", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "S30" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "2" },
                { "IffS", "1" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("30", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
```

```
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "S37" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "31" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("31", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "S32" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
```

```
            dict.Add("32", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "S33" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("33", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "S34" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
```

```
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("34", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "S37" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "35" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("35", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
```

```
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "S36" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "2" },
            { "IffS", "1" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("36", new Dictionary<String, object>()
        {
            { "begin", "R4" },
            { "(", "R4" },
            { ")", "R4" },
            { "{", "R4" },
            { "int", "R4" },
            { "a", "R4" },
            { "b", "R4" },
            { "c", "R4" },
            { "=", "R4" },
            { "5", "R4" },
            { "10", "R4" },
            { "0", "R4" },
            { ";", "R4" },
            { "if", "R4" },
            { ">", "R4" },
            { "print", "R4" },
            { "else", "R4" },
            { "$", "R4" },
            { "}", "R4" },
            { "+", "R4" },
            { "end", "R4" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("37", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
```

```
            { "a", "S40" },
            { "b", "S42" },
            { "c", "S44" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "38" },
            { "Const", "" }
        });
        dict.Add("38", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "S39" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
```

```
            });
            dict.Add("39", new Dictionary<String, object>()
            {
                { "begin", "R5" },
                { "(", "R5" },
                { ")", "R5" },
                { "{", "R5" },
                { "int", "R5" },
                { "a", "R5" },
                { "b", "R5" },
                { "c", "R5" },
                { "=", "R5" },
                { "5", "R5" },
                { "10", "R5" },
                { "0", "R5" },
                { ";", "R5" },
                { "if", "R5" },
                { ">", "R5" },
                { "print", "R5" },
                { "else", "R5" },
                { "$", "R5" },
                { "}", "R5" },
                { "+", "R5" },
                { "end", "R5" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("40", new Dictionary<String, object>()
            {
                { "begin", "R6" },
                { "(", "R6" },
                { ")", "R6" },
                { "{", "R6" },
                { "int", "R6" },
                { "a", "R6" },
                { "b", "R6" },
                { "c", "R6" },
                { "=", "R6" },
                { "5", "R6" },
                { "10", "R6" },
                { "0", "R6" },
                { ";", "R6" },
                { "if", "R6" },
                { ">", "R6" },
                { "print", "R6" },
                { "else", "R6" },
                { "$", "R6" },
                { "}", "R6" },
                { "+", "R6" },
```

```
                { "end", "R6" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("41", new Dictionary<String, object>()
            {
                { "begin", "R9" },
                { "(", "R9" },
                { ")", "R9" },
                { "{", "R9" },
                { "int", "R9" },
                { "a", "R9" },
                { "b", "R9" },
                { "c", "R9" },
                { "=", "R9" },
                { "5", "R9" },
                { "10", "R9" },
                { "0", "R9" },
                { ";", "R9" },
                { "if", "R9" },
                { ">", "R9" },
                { "print", "R9" },
                { "else", "R9" },
                { "$", "R9" },
                { "}", "R9" },
                { "+", "R9" },
                { "end", "R9" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("42", new Dictionary<String, object>()
            {
                { "begin", "R7" },
                { "(", "R7" },
                { ")", "R7" },
                { "{", "R7" },
                { "int", "R7" },
                { "a", "R7" },
                { "b", "R7" },
                { "c", "R7" },
                { "=", "R7" },
                { "5", "R7" },
                { "10", "R7" },
                { "0", "R7" },
```

```
                { ";", "R7" },
                { "if", "R7" },
                { ">", "R7" },
                { "print", "R7" },
                { "else", "R7" },
                { "$", "R7" },
                { "}", "R7" },
                { "+", "R7" },
                { "end", "R7" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("43", new Dictionary<String, object>()
            {
                { "begin", "R10" },
                { "(", "R10" },
                { ")", "R10" },
                { "{", "R10" },
                { "int", "R10" },
                { "a", "R10" },
                { "b", "R10" },
                { "c", "R10" },
                { "=", "R10" },
                { "5", "R10" },
                { "10", "R10" },
                { "0", "R10" },
                { ";", "R10" },
                { "if", "R10" },
                { ">", "R10" },
                { "print", "R10" },
                { "else", "R10" },
                { "$", "R10" },
                { "}", "R10" },
                { "+", "R10" },
                { "end", "R10" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("44", new Dictionary<String, object>()
            {
                { "begin", "R8" },
                { "(", "R8" },
                { ")", "R8" },
                { "{", "R8" },
```

```
                { "int", "R8" },
                { "a", "R8" },
                { "b", "R8" },
                { "c", "R8" },
                { "=", "R8" },
                { "5", "R8" },
                { "10", "R8" },
                { "0", "R8" },
                { ";", "R8" },
                { "if", "R8" },
                { ">", "R8" },
                { "print", "R8" },
                { "else", "R8" },
                { "$", "R8" },
                { "}", "R8" },
                { "+", "R8" },
                { "end", "R8" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("45", new Dictionary<String, object>()
            {
                { "begin", "R11" },
                { "(", "R11" },
                { ")", "R11" },
                { "{", "R11" },
                { "int", "R11" },
                { "a", "R11" },
                { "b", "R11" },
                { "c", "R11" },
                { "=", "R11" },
                { "5", "R11" },
                { "10", "R11" },
                { "0", "R11" },
                { ";", "R11" },
                { "if", "R11" },
                { ">", "R11" },
                { "print", "R11" },
                { "else", "R11" },
                { "$", "R11" },
                { "}", "R11" },
                { "+", "R11" },
                { "end", "R11" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
```

```
                { "Const", "" }
            });
#endregion

            while (true)
            {
                if (!Col.Contains(finalArray[pointer]))
                {
                    Output.AppendText("Unable to Parse Unknown Input");
                    break;
                }
                Parser = dict[Stack.Peek() + ""][finalArray[pointer] +
""] + "";
                if (Parser.Contains("S"))
                {
                    Stack.Push(finalArray[pointer] + "");
                    Parser = Parser.TrimStart('S');
                    Stack.Push(Parser);
                    pointer++;
                    Print_Stack();
                }
                if (Parser.Contains("R"))
                {
                    Parser = Parser.TrimStart('R');
                    String get = States[Convert.ToInt32(Parser) - 1] +
"";
                    String[] Splitted = get.Split('_');
                    String[] Final_ = Splitted[1].Split(' ');
                    int test = Final_.Length;
                    for (int i = 0; i < test * 2; i++)
                    { Stack.Pop(); }
                    String row = Stack.Peek() + "";
                    Stack.Push(Splitted[0]);
                    Stack.Push(dict[row][Stack.Peek()] + "");
                    Print_Stack();
                }
                if (Parser.Contains("Accept"))
                {
                    Output.AppendText("Parsed");
                    break;
                }
                if (Parser.Equals(""))
                {
                    Output.AppendText("Unable to Parse");
                    break;
                }
            }
            finalArray.Remove("$");
            finalArray.Remove("begin");
            #endregion
////////////////////////////////////Pasing_Finished//////////////////////
////////////////////////////////////
            #region Syntax Analyzer
            lexemes_per_line = 0;
```

```
            ST_index = 0;

            for (int k = 0; k < finalArray.Count; k++)
            {
                if (if_deleted == true)
                {
                    k = k - 4;
                    if_deleted = false;
                }
                Match Match_Variable = variable_Reg.Match(finalArray[k] +
"");
                Match Match_Constant = constants_Reg.Match(finalArray[k]
+ "");
                Match Match_Operator = operators_Reg.Match(finalArray[k]
+ "");

                if (Match_Variable.Success)
                {
                    if (key.Contains(finalArray[k]))
                    {
                        if (finalArray[k].Equals("print"))
                        {
                            String print_on_Screen = finalArray[k + 1] +
"";
                            int index = 0;
                            for (int i = 0; i < Symboltable.Count; i++)
                            {
                                for (int j = 0; j < Symboltable[i].Count;
j++)
                                {
                                    if
(Symboltable[i][j].Equals(print_on_Screen))
                                        { index = i; }
                                }
                            }
                            CodeOutput.Text = Symboltable[index][3];
                        }
                        KeyWords.Add(finalArray[k]); lexemes_per_line++;
                    }
                    else
                    {
                        Variables.Add(finalArray[k]);
                        if (!LineNumber.Contains(L))
                        {
                            LineNumber.Add(L);
                        }
                        lexemes_per_line = lexemes_per_line + 2;
                    }
                }
                if (Match_Constant.Success)
                {
                    Constants.Add(finalArray[k]); lexemes_per_line++;
                }
                if (Match_Operator.Success)
```

```
                {
                    if (finalArray[k].Equals(";") ||
finalArray[k].Equals("}") || finalArray[k].Equals("{") ||
finalArray[k].Equals(")"))
                    {
                        L++; lexemes_per_line = 0;
                    }
                    if (operators_Reg.Match(finalArray[k] + "").Success)
                    {
                        Semantic_Analysis(k);
                    }
                }
                Check_And_Make_Entries();
            }
#endregion
//////////////////////////////////Symbol Table
Generated////////////////////////////////////////////////////////////
            #region Display Symbol Table
            for (int j = 0; j < Symboltable.Count; j++)
            {
                for (int z = 0; z < Symboltable[j].Count; z++)
                { ST.AppendText(Symboltable[j][z] + "\t"); }
                ST.AppendText("\n");
            }
            #endregion
        }

#region Make Entry Symbol Table
        void Check_And_Make_Entries()
        {
            KeyWords.Remove("begin"); KeyWords.Remove("end");
KeyWords.Remove("print");
            KeyWords.Remove("if"); KeyWords.Remove("else");
            if (lexemes_per_line - 4 == 0 || lexemes_per_line - 7 == 0)
            {
                if (lexemes_per_line == 7)
                {
                    Variables.RemoveAt(Variables.Count - 1);
Variables.RemoveAt(Variables.Count - 1);
                }
                for (; ST_index < KeyWords.Count; ST_index++)
                {
                    Symboltable.Add(new List<string>());
                    Symboltable[ST_index].Add(ST_index + 1 + "");
                    Symboltable[ST_index].Add(Variables[ST_index] + "");
                    Symboltable[ST_index].Add(KeyWords[ST_index] + "");
                    Symboltable[ST_index].Add(Constants[ST_index] + "");
                    Symboltable[ST_index].Add(LineNumber[ST_index] + "");
                }
            }
            if (lexemes_per_line - 6 == 0)
            {
                Variables.RemoveAt(Variables.Count - 1);
Variables.RemoveAt(Variables.Count - 1);
```

```
Variables.RemoveAt(Variables.Count - 1);
            }
        }
        #endregion
////////////////////////////////////END_Check_And_Make_Entries////////
////////////////////////////////////
        #region Print Stack
        void Print_Stack()
        {
            foreach (String i in Stack)
            {
                Output.AppendText(i);
            }
            Output.AppendText("\n");
        }
        #endregion
```

Output:

Declaration Error: MULTIPLE_DECLARATION, variable (x)
Declaration Error: MULTIPLE_DECLARATION, variable (y)
Declaration Error: NO_DECLARATION, variable (jjj)
Declaration Error: NO_DECLARATION, variable (b)
Casting Error: FLOAT_INT_CASTING, variable (x)

3)Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1:

Execute the integrated code with desired output.

Lab 14

Integration: Ph-2 and Code Generator.

Objective:

In this lab students will understand semantic analyzer by attaching meaning to every node of the syntax tree that was generated by the Parser.

Activity Outcomes:

This lab teaches you

- How to integrate lexical analyzer, parser and semantic analyzer

Instructor Note:

As for this lab activity, read chapter 8 from the book "Compiler Construction-Principles and Practices" by Kenneth C. Louden

1) Useful Concepts

Task of lexical analyzer is token generation. Generated tokens are then passed to the parser for syntax checking. Parser verifies their syntax with the help of context free grammar of that language. This parser uses bottom up strategy to parse the tokens. And semantic analyzer checks for type incompatibilities.

2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	*60 mins*	*HIGH*	*CLO-5*

Activity 1:
Integrate lexical analyzer, parser and semantic analyzer

Solution:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Text.RegularExpressions;
using System.Collections;

namespace WindowsFormsApplication1
{
```

```
    public partial class Form1 : Form
    {
        ArrayList States = new ArrayList();
        Stack<String> Stack = new Stack<String>();
        String Parser;
        String[]    Col    =    {    "begin"    ,"(",")","{","int","a","b",
"c","=","5","10","0",";","if",">","print",

"else","$","}","+","end","Program","DecS","AssS","IffS","PriS","Var","Con
st" };

        public Form1()
        {
            InitializeComponent();
        }

        List<List<String>> Symboltable = new List<List<String>>();

        ArrayList LineNumber;
        ArrayList Variables;
        ArrayList KeyWords;
        ArrayList Constants;
        ArrayList finalArray;
        ArrayList tempArray;

        Regex variable_Reg;
        Regex constants_Reg;
        Regex operators_Reg;

        int lexemes_per_line;
        int ST_index;

        Boolean if_deleted;

        private void Compile_Click(object sender, EventArgs e)
        {
            String[]  k_  =  {  "int",  "float",  "begin",  "end",  "print",
"if",  "else" };
            ArrayList key = new ArrayList(k_);

            LineNumber = new ArrayList();
            Variables = new ArrayList();
            KeyWords = new ArrayList();
            Constants = new ArrayList();

            finalArray = new ArrayList();
            tempArray = new ArrayList();

            variable_Reg = new Regex(@"^[A-Za-z|_][A-Za-z|0-9]*$");
            constants_Reg    =    new    Regex(@"^[0-9]+([.][0-9]+)?([e]([+|-
])?[0-9]+)?$");
            operators_Reg = new Regex(@"[+-/*=;>(){}]");

            int L = 1;
```

```
            Output.Text = "";
            ST.Text = "";

            Symboltable.Clear();

            if_deleted = false;

            string strinput = Input.Text;
            char[] charinput = strinput.ToCharArray();

////////////////////////////////Start_Split
Function/////////////////////////////////////////////////////////
            #region Input Buffering
            for (int itr = 0; itr < charinput.Length; itr++)
            {
                Match Match_Variable = variable_Reg.Match(charinput[itr]
+ "");
                Match Match_Constant = constants_Reg.Match(charinput[itr]
+ "");
                Match Match_Operator = operators_Reg.Match(charinput[itr]
+ "");

                if (Match_Variable.Success || Match_Constant.Success)
                {
                    tempArray.Add(charinput[itr]);
                }
                if (charinput[itr].Equals(' '))
                {
                    if (tempArray.Count != 0)
                    {
                        int j = 0;
                        String fin = "";
                        for (; j < tempArray.Count; j++)
                        {
                            fin += tempArray[j];
                        }
                        finalArray.Add(fin);
                        tempArray.Clear();
                    }
                }
                if (Match_Operator.Success)
                {
                    if (tempArray.Count != 0)
                    {
                        int j = 0;
                        String fin = "";
                        for (; j < tempArray.Count; j++)
                        {
                            fin += tempArray[j];
                        }
                        finalArray.Add(fin);
                        tempArray.Clear();
                    }
```

```
                    finalArray.Add(charinput[itr]+"");
                }
            }
            if (tempArray.Count != 0)
            {
                String final = "";
                for (int k = 0; k < tempArray.Count; k++)
                {
                    final += tempArray[k];
                }
                finalArray.Add(final);
            }
            #endregion
////////////////////////////////End_Split
Function////////////////////////////////////////////////////////////
            #region Bottom Up Parser
            States.Add("Program_begin ( ) { DecS Decs Decs AssS IffS }
end");
            States.Add("DecS_int Var = Const ;");
            States.Add("AssS_Var = Var + Var ;");
            States.Add("IffS_if ( Var > Var ) { PriS } else { PriS }");
            States.Add("PriS_print Var ;");
            States.Add("Var_a");
            States.Add("Var_b");
            States.Add("Var_c");
            States.Add("Const_5");
            States.Add("Const_10");
            States.Add("Const_0");
            Stack.Push("0");
            finalArray.Add("$");
            int pointer = 0;
            #region ParseTable
            var   dict   =   new   Dictionary<string,   Dictionary<String,
object>>();
            dict.Add("0", new Dictionary<String, object>()
            {
                { "begin", "S2" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
```

```
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "1" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("1", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "Accept" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("2", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "S3" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
```

```
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("3", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "S4" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("4", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
```

```
            { ")", "" },
            { "{", "S5" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("5", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "S13" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "6" },
            { "AssS", "" },
            { "IffS", "" },
```

```
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("6", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "S13" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "7" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("7", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "S13" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
```

```
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "8" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("8", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "S40" },
            { "b", "S42" },
            { "c", "S44" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "9" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "18" },
            { "Const", "" }
        });
        dict.Add("9", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
```

```
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "S24" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "10" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("10", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "S11" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("11", new Dictionary<String, object>()
        {
            { "begin", "" },
```

```
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "S12" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("12", new Dictionary<String, object>()
        {
            { "begin", "R1" },
            { "(", "R1" },
            { ")", "R1" },
            { "{", "R1" },
            { "int", "R1" },
            { "a", "R1" },
            { "b", "R1" },
            { "c", "R1" },
            { "=", "R1" },
            { "5", "R1" },
            { "10", "R1" },
            { "0", "R1" },
            { ";", "R1" },
            { "if", "R1" },
            { ">", "R1" },
            { "print", "R1" },
            { "else", "R1" },
            { "$", "R1" },
            { "}", "R1" },
            { "+", "R1" },
            { "end", "R1" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
```

```
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("13", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "S40" },
            { "b", "S42" },
            { "c", "S44" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "14" },
            { "Const", "" }
        });
        dict.Add("14", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "S15" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
```

```
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "2" },
            { "IffS", "1" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("15", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "S41" },
            { "10", "S43" },
            { "0", "S45" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "16" }
        });
        dict.Add("16", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
```

```
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "S17" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("17", new Dictionary<String, object>()
        {
            { "begin", "R2" },
            { "(", "R2" },
            { ")", "R2" },
            { "{", "R2" },
            { "int", "R2" },
            { "a", "R2" },
            { "b", "R2" },
            { "c", "R2" },
            { "=", "R2" },
            { "5", "R2" },
            { "10", "R2" },
            { "0", "R2" },
            { ";", "R2" },
            { "if", "R2" },
            { ">", "R2" },
            { "print", "R2" },
            { "else", "R2" },
            { "$", "R2" },
            { "}", "R2" },
            { "+", "R2" },
            { "end", "R2" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("18", new Dictionary<String, object>()
        {
```

```
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "S19" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("19", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "S40" },
            { "b", "S42" },
            { "c", "S44" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
```

```
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "20" },
                { "Const", "" }
            });
            dict.Add("20", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "S21" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("21", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "S40" },
                { "b", "S42" },
                { "c", "S44" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
```

```
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "22" },
                { "Const", "" }
            });
            dict.Add("22", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "S23" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("23", new Dictionary<String, object>()
            {
                { "begin", "R3" },
                { "(", "R3" },
                { ")", "R3" },
                { "{", "R3" },
                { "int", "R3" },
                { "a", "R3" },
                { "b", "R3" },
```

```
            { "c", "R3" },
            { "=", "R3" },
            { "5", "R3" },
            { "10", "R3" },
            { "0", "R3" },
            { ";", "R3" },
            { "if", "R3" },
            { ">", "R3" },
            { "print", "R3" },
            { "else", "R3" },
            { "$", "R3" },
            { "}", "R3" },
            { "+", "R3" },
            { "end", "R3" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("24", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "S25" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("25", new Dictionary<String, object>()
```

```
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "S40" },
                { "b", "S42" },
                { "c", "S44" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "26" },
                { "Const", "" }
            });
            dict.Add("26", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "S27" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
```

```
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("27", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "S40" },
                { "b", "S42" },
                { "c", "S44" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "28" },
                { "Const", "" }
            });
            dict.Add("28", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "S29" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
```

```
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "2" },
                { "IffS", "1" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("29", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "S30" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "2" },
                { "IffS", "1" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("30", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
```

```
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "S37" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "31" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("31", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "S32" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
```

```
            dict.Add("32", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "S33" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("33", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "S34" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
```

```
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("34", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
                { "if", "" },
                { ">", "" },
                { "print", "S37" },
                { "else", "" },
                { "$", "" },
                { "}", "" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "35" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("35", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
                { "a", "" },
                { "b", "" },
                { "c", "" },
                { "=", "" },
                { "5", "" },
                { "10", "" },
                { "0", "" },
                { ";", "" },
```

```
                { "if", "" },
                { ">", "" },
                { "print", "" },
                { "else", "" },
                { "$", "" },
                { "}", "S36" },
                { "+", "" },
                { "end", "" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "2" },
                { "IffS", "1" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("36", new Dictionary<String, object>()
            {
                { "begin", "R4" },
                { "(", "R4" },
                { ")", "R4" },
                { "{", "R4" },
                { "int", "R4" },
                { "a", "R4" },
                { "b", "R4" },
                { "c", "R4" },
                { "=", "R4" },
                { "5", "R4" },
                { "10", "R4" },
                { "0", "R4" },
                { ";", "R4" },
                { "if", "R4" },
                { ">", "R4" },
                { "print", "R4" },
                { "else", "R4" },
                { "$", "R4" },
                { "}", "R4" },
                { "+", "R4" },
                { "end", "R4" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("37", new Dictionary<String, object>()
            {
                { "begin", "" },
                { "(", "" },
                { ")", "" },
                { "{", "" },
                { "int", "" },
```

```
            { "a", "S40" },
            { "b", "S42" },
            { "c", "S44" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "38" },
            { "Const", "" }
        });
        dict.Add("38", new Dictionary<String, object>()
        {
            { "begin", "" },
            { "(", "" },
            { ")", "" },
            { "{", "" },
            { "int", "" },
            { "a", "" },
            { "b", "" },
            { "c", "" },
            { "=", "" },
            { "5", "" },
            { "10", "" },
            { "0", "" },
            { ";", "S39" },
            { "if", "" },
            { ">", "" },
            { "print", "" },
            { "else", "" },
            { "$", "" },
            { "}", "" },
            { "+", "" },
            { "end", "" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
```

```
            });
            dict.Add("39", new Dictionary<String, object>()
            {
                { "begin", "R5" },
                { "(", "R5" },
                { ")", "R5" },
                { "{", "R5" },
                { "int", "R5" },
                { "a", "R5" },
                { "b", "R5" },
                { "c", "R5" },
                { "=", "R5" },
                { "5", "R5" },
                { "10", "R5" },
                { "0", "R5" },
                { ";", "R5" },
                { "if", "R5" },
                { ">", "R5" },
                { "print", "R5" },
                { "else", "R5" },
                { "$", "R5" },
                { "}", "R5" },
                { "+", "R5" },
                { "end", "R5" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("40", new Dictionary<String, object>()
            {
                { "begin", "R6" },
                { "(", "R6" },
                { ")", "R6" },
                { "{", "R6" },
                { "int", "R6" },
                { "a", "R6" },
                { "b", "R6" },
                { "c", "R6" },
                { "=", "R6" },
                { "5", "R6" },
                { "10", "R6" },
                { "0", "R6" },
                { ";", "R6" },
                { "if", "R6" },
                { ">", "R6" },
                { "print", "R6" },
                { "else", "R6" },
                { "$", "R6" },
                { "}", "R6" },
                { "+", "R6" },
```

```
            { "end", "R6" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("41", new Dictionary<String, object>()
        {
            { "begin", "R9" },
            { "(", "R9" },
            { ")", "R9" },
            { "{", "R9" },
            { "int", "R9" },
            { "a", "R9" },
            { "b", "R9" },
            { "c", "R9" },
            { "=", "R9" },
            { "5", "R9" },
            { "10", "R9" },
            { "0", "R9" },
            { ";", "R9" },
            { "if", "R9" },
            { ">", "R9" },
            { "print", "R9" },
            { "else", "R9" },
            { "$", "R9" },
            { "}", "R9" },
            { "+", "R9" },
            { "end", "R9" },
            { "Program", "" },
            { "DecS", "" },
            { "AssS", "" },
            { "IffS", "" },
            { "PriS", "" },
            { "Var", "" },
            { "Const", "" }
        });
        dict.Add("42", new Dictionary<String, object>()
        {
            { "begin", "R7" },
            { "(", "R7" },
            { ")", "R7" },
            { "{", "R7" },
            { "int", "R7" },
            { "a", "R7" },
            { "b", "R7" },
            { "c", "R7" },
            { "=", "R7" },
            { "5", "R7" },
            { "10", "R7" },
            { "0", "R7" },
```

```
                { ";", "R7" },
                { "if", "R7" },
                { ">", "R7" },
                { "print", "R7" },
                { "else", "R7" },
                { "$", "R7" },
                { "}", "R7" },
                { "+", "R7" },
                { "end", "R7" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("43", new Dictionary<String, object>()
            {
                { "begin", "R10" },
                { "(", "R10" },
                { ")", "R10" },
                { "{", "R10" },
                { "int", "R10" },
                { "a", "R10" },
                { "b", "R10" },
                { "c", "R10" },
                { "=", "R10" },
                { "5", "R10" },
                { "10", "R10" },
                { "0", "R10" },
                { ";", "R10" },
                { "if", "R10" },
                { ">", "R10" },
                { "print", "R10" },
                { "else", "R10" },
                { "$", "R10" },
                { "}", "R10" },
                { "+", "R10" },
                { "end", "R10" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("44", new Dictionary<String, object>()
            {
                { "begin", "R8" },
                { "(", "R8" },
                { ")", "R8" },
                { "{", "R8" },
```

```
                { "int", "R8" },
                { "a", "R8" },
                { "b", "R8" },
                { "c", "R8" },
                { "=", "R8" },
                { "5", "R8" },
                { "10", "R8" },
                { "0", "R8" },
                { ";", "R8" },
                { "if", "R8" },
                { ">", "R8" },
                { "print", "R8" },
                { "else", "R8" },
                { "$", "R8" },
                { "}", "R8" },
                { "+", "R8" },
                { "end", "R8" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
                { "Const", "" }
            });
            dict.Add("45", new Dictionary<String, object>()
            {
                { "begin", "R11" },
                { "(", "R11" },
                { ")", "R11" },
                { "{", "R11" },
                { "int", "R11" },
                { "a", "R11" },
                { "b", "R11" },
                { "c", "R11" },
                { "=", "R11" },
                { "5", "R11" },
                { "10", "R11" },
                { "0", "R11" },
                { ";", "R11" },
                { "if", "R11" },
                { ">", "R11" },
                { "print", "R11" },
                { "else", "R11" },
                { "$", "R11" },
                { "}", "R11" },
                { "+", "R11" },
                { "end", "R11" },
                { "Program", "" },
                { "DecS", "" },
                { "AssS", "" },
                { "IffS", "" },
                { "PriS", "" },
                { "Var", "" },
```

```
                { "Const", "" }
            });
#endregion

            while (true)
            {
                if (!Col.Contains(finalArray[pointer]))
                {
                    Output.AppendText("Unable to Parse Unknown Input");
                    break;
                }
                Parser = dict[Stack.Peek() + ""][finalArray[pointer] +
""] + "";
                if (Parser.Contains("S"))
                {
                    Stack.Push(finalArray[pointer] + "");
                    Parser = Parser.TrimStart('S');
                    Stack.Push(Parser);
                    pointer++;
                    Print_Stack();
                }
                if (Parser.Contains("R"))
                {
                    Parser = Parser.TrimStart('R');
                    String get = States[Convert.ToInt32(Parser) - 1] +
"";
                    String[] Splitted = get.Split('_');
                    String[] Final_ = Splitted[1].Split(' ');
                    int test = Final_.Length;
                    for (int i = 0; i < test * 2; i++)
                    { Stack.Pop(); }
                    String row = Stack.Peek() + "";
                    Stack.Push(Splitted[0]);
                    Stack.Push(dict[row][Stack.Peek()] + "");
                    Print_Stack();
                }
                if (Parser.Contains("Accept"))
                {
                    Output.AppendText("Parsed");
                    break;
                }
                if (Parser.Equals(""))
                {
                    Output.AppendText("Unable to Parse");
                    break;
                }
            }
            finalArray.Remove("$");
            finalArray.Remove("begin");
            #endregion
////////////////////////////////////Pasing_Finished/////////////////////////
////
            #region Syntax Analyzer
            lexemes_per_line = 0;
```

```
            ST_index = 0;

            for (int k = 0; k < finalArray.Count; k++)
            {
                if (if_deleted == true)
                {
                    k = k - 4;
                    if_deleted = false;
                }
                Match Match_Variable = variable_Reg.Match(finalArray[k] +
"");
                Match  Match_Constant  =  constants_Reg.Match(finalArray[k]
+ "");
                Match  Match_Operator  =  operators_Reg.Match(finalArray[k]
+ "");

                if (Match_Variable.Success)
                {
                    if (key.Contains(finalArray[k]))
                    {
                        if (finalArray[k].Equals("print"))
                        {
                            String print_on_Screen = finalArray[k + 1] +
"";
                            int index = 0;
                            for (int i = 0; i < Symboltable.Count; i++)
                            {
                                for (int j = 0; j < Symboltable[i].Count;
j++)
                                {
                                    if
(Symboltable[i][j].Equals(print_on_Screen))
                                        { index = i; }
                                }
                            }
                            CodeOutput.Text = Symboltable[index][3];
                        }
                        KeyWords.Add(finalArray[k]); lexemes_per_line++;
                    }
                    else
                    {
                        Variables.Add(finalArray[k]);
                        if (!LineNumber.Contains(L))
                        {
                            LineNumber.Add(L);
                        }
                        lexemes_per_line = lexemes_per_line + 2;
                    }
                }
                if (Match_Constant.Success)
                {
                    Constants.Add(finalArray[k]); lexemes_per_line++;
                }
                if (Match_Operator.Success)
```

```
                {
                    if            (finalArray[k].Equals(";")            ||
finalArray[k].Equals("}")       ||       finalArray[k].Equals("{")       ||
finalArray[k].Equals(")"))
                    {
                        L++; lexemes_per_line = 0;
                    }
                    if (operators_Reg.Match(finalArray[k] + "").Success)
                    {
                        Semantic_Analysis(k);
                    }
                }
                Check_And_Make_Entries();
            }
#endregion
//////////////////////////Symbol              Table               Generated
//////////////////////////////
            #region Display Symbol Table
            for (int j = 0; j < Symboltable.Count; j++)
            {
                for (int z = 0; z < Symboltable[j].Count; z++)
                { ST.AppendText(Symboltable[j][z] + "\t"); }
                ST.AppendText("\n");
            }
            #endregion
        }

////////////////////////////////////////////////END//////////////////////
//////////////
        #region Semantic Analyzer
        void Semantic_Analysis(int k)
        {
            if (finalArray[k].Equals("+"))
            {
                if (variable_Reg.Match(finalArray[k - 1] + "").Success &&
variable_Reg.Match(finalArray[k + 1] + "").Success)
                {
                    String type = finalArray[k - 4] + "";
                    String left_side = finalArray[k - 3] + "";
                    int left_side_i = 0;
                    int left_side_j = 0;
                    String before = finalArray[k - 1] + "";
                    int before_i = 0;
                    int before_j = 0;
                    String after = finalArray[k + 1] + "";
                    int after_i = 0;
                    int after_j = 0;
                    for (int i = 0; i < Symboltable.Count; i++)
                    {
                        for (int j = 0; j < Symboltable[i].Count; j++)
                        {
                            if (Symboltable[i][j].Equals(left_side))
                            { left_side_i = i; left_side_j = j; }
                            if (Symboltable[i][j].Equals(before))
```

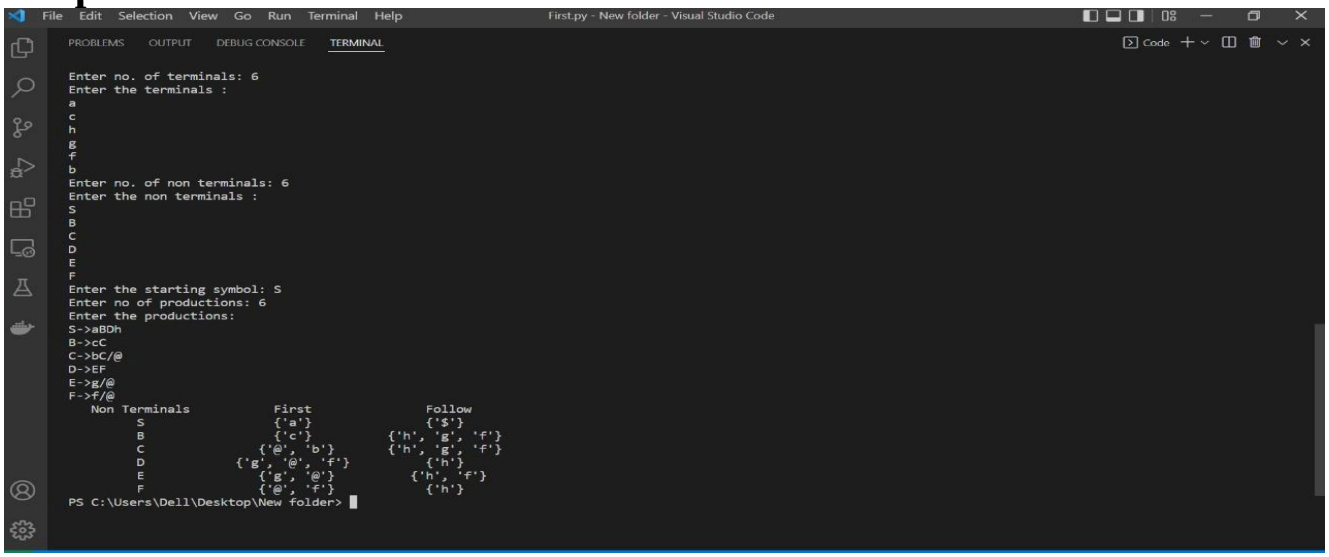
```
                            { before_i = i; before_j = j; }
                            if (Symboltable[i][j].Equals(after))
                            { after_i = i; after_j = j; }
                        }
                    }
                    if      (type.Equals(Symboltable[before_i][2])      &&
type.Equals(Symboltable[after_i][2])                                    &&
Symboltable[before_i][2].Equals(Symboltable[after_i][2]))
                    {
                        int                         Ans                 =
Convert.ToInt32(Symboltable[before_i][3])                               +
Convert.ToInt32(Symboltable[after_i][3]);
                        Constants.Add(Ans);
                    }
                    if
(Symboltable[left_side_i][2].Equals(Symboltable[before_i][2])           &&
Symboltable[left_side_i][2].Equals(Symboltable[after_i][2])             &&
Symboltable[before_i][2].Equals(Symboltable[after_i][2]))
                    {
                        int                         Ans                 =
Convert.ToInt32(Symboltable[before_i][3])                               +
Convert.ToInt32(Symboltable[after_i][3]);
                        Constants.RemoveAt(Constants.Count - 1);
                        Constants.Add(Ans);
                        Symboltable[left_side_i][3] = Ans + "";
                    }
                }
            }
            if (finalArray[k].Equals("-"))
            {
                if (variable_Reg.Match(finalArray[k - 1] + "").Success &&
variable_Reg.Match(finalArray[k + 1] + "").Success)
                {
                    String type = finalArray[k - 4] + "";
                    String left_side = finalArray[k - 3] + "";
                    int left_side_i = 0;
                    int left_side_j = 0;
                    String before = finalArray[k - 1] + "";
                    int before_i = 0;
                    int before_j = 0;
                    String after = finalArray[k + 1] + "";
                    int after_i = 0;
                    int after_j = 0;
                    for (int i = 0; i < Symboltable.Count; i++)
                    {
                        for (int j = 0; j < Symboltable[i].Count; j++)
                        {
                            if (Symboltable[i][j].Equals(left_side))
                            { left_side_i = i; left_side_j = j; }
                            if (Symboltable[i][j].Equals(before))
                            { before_i = i; before_j = j; }
                            if (Symboltable[i][j].Equals(after))
                            { after_i = i; after_j = j; }
                        }
```

```
                    }
                    if      (type.Equals(Symboltable[before_i][2])      &&
type.Equals(Symboltable[after_i][2])                                    &&
Symboltable[before_i][2].Equals(Symboltable[after_i][2]))
                    {
                        int                         Ans                 =
Convert.ToInt32(Symboltable[before_i][3])                               -
Convert.ToInt32(Symboltable[after_i][3]);
                        Constants.Add(Ans);
                    }
                    if
(Symboltable[left_side_i][2].Equals(Symboltable[before_i][2])           &&
Symboltable[left_side_i][2].Equals(Symboltable[after_i][2])             &&
Symboltable[before_i][2].Equals(Symboltable[after_i][2]))
                    {
                        int                         Ans                 =
Convert.ToInt32(Symboltable[before_i][3])                               +
Convert.ToInt32(Symboltable[after_i][3]);
                        Constants.RemoveAt(Constants.Count - 1);
                        Constants.Add(Ans);
                        Symboltable[left_side_i][3] = Ans + "";
                    }
                }
            }
            if (finalArray[k].Equals(">"))
            {
                if (variable_Reg.Match(finalArray[k - 1] + "").Success &&
variable_Reg.Match(finalArray[k + 1] + "").Success)
                {
                    String before = finalArray[k - 1] + "";
                    int before_i = 0;
                    int before_j = 0;
                    String after = finalArray[k + 1] + "";
                    int after_i = 0;
                    int after_j = 0;
                    for (int i = 0; i < Symboltable.Count; i++)
                    {
                        for (int j = 0; j < Symboltable[i].Count; j++)
                        {
                            if (Symboltable[i][j].Equals(before))
                            { before_i = i; before_j = j; }
                            if (Symboltable[i][j].Equals(after))
                            { after_i = i; after_j = j; }
                        }
                    }
                    if     (Convert.ToInt32(Symboltable[before_i][3])     >
Convert.ToInt32(Symboltable[after_i][3]))
                    {
                        int start_of_else = finalArray.IndexOf("else");
                        int end_of_else = finalArray.Count - 1;
                        for (int i = end_of_else; i >= start_of_else; i--
)
                        {
                            if (finalArray[i].Equals("}"))
```

```
                                {
                                    if (i < finalArray.Count - 2)
                                    { end_of_else = i; }
                                }
                            }

                            for (int i = start_of_else; i <= end_of_else;
i++)
                            { finalArray.RemoveAt(start_of_else); }
                        }
                        else
                        {
                            int start_of_if = finalArray.IndexOf("if");
                            int end_of_if = finalArray.IndexOf("}");

                            for (int i = start_of_if; i <= end_of_if; i++)
                            { finalArray.RemoveAt(start_of_if); }
                            if_deleted = true;
                        }
                    }
                }
            }
            #endregion
//////////////////////////////////////////END_Semantic_Analysis////////
/////////////
            #region Make Entry Symbol Table
            void Check_And_Make_Entries()
            {
                KeyWords.Remove("begin");                KeyWords.Remove("end");
KeyWords.Remove("print");
                KeyWords.Remove("if"); KeyWords.Remove("else");
                if (lexemes_per_line - 4 == 0 || lexemes_per_line - 7 == 0)
                {
                    if (lexemes_per_line == 7)
                    {
                        Variables.RemoveAt(Variables.Count         -         1);
Variables.RemoveAt(Variables.Count - 1);
                    }
                    for (; ST_index < KeyWords.Count; ST_index++)
                    {
                        Symboltable.Add(new List<string>());
                        Symboltable[ST_index].Add(ST_index + 1 + "");
                        Symboltable[ST_index].Add(Variables[ST_index] + "");
                        Symboltable[ST_index].Add(KeyWords[ST_index] + "");
                        Symboltable[ST_index].Add(Constants[ST_index] + "");
                        Symboltable[ST_index].Add(LineNumber[ST_index] + "");
                    }
                }
                if (lexemes_per_line - 6 == 0)
                {
                    Variables.RemoveAt(Variables.Count          -          1);
Variables.RemoveAt(Variables.Count                    -                    1);
Variables.RemoveAt(Variables.Count - 1);
                }
```

```
        }
        #endregion
////////////////////////////////////END_Check_And_Make_Entries////////
//////////////
        #region Print Stack
        void Print_Stack()
        {
            foreach (String i in Stack)
            {
                Output.AppendText(i);
            }
            Output.AppendText("\n");
        }
        #endregion
    }
}
```

Output:

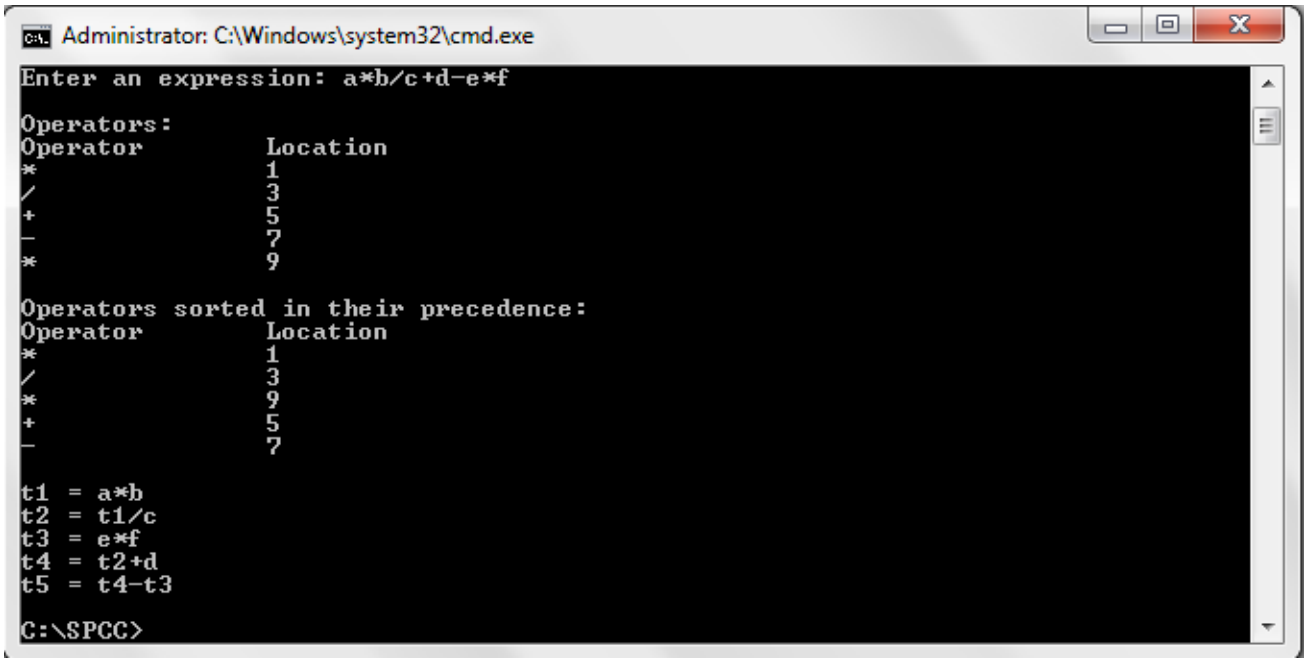



THE GRAMMAR IS AS FOLLOWS
S -> S+T
S -> T
T -> T*F
T -> F
F -> (S)
F -> t
I0 :
Z -> .S
S -> .S+T
S -> .T
T -> .T*F
T -> .F
F -> .(S)
F -> .t
I1 :
Z -> S.
S -> S.+T
I2 :
S -> T.
T -> T.*F
I3 :
T -> F.
I4 :
F -> (.S)
S -> .S+T
S -> .T
T -> .T*F
T -> .F
F -> .(S)
F -> .t
I5 :
F -> t.

```
THE GRAMMAR IS AS FOLLOWS
                                S -> S+T
                                S -> T
                                T -> T*F
                                T -> F
                                F -> (S)
                                F -> t

I0 :
        Z -> .S
        S -> .S+T
        S -> .T
        T -> .T*F
        T -> .F
        F -> .(S)
        F -> .t

I1 :
        Z -> S.
        S -> S.+T

I2 :
        S -> T.
        T -> T.*F

I3 :
        T -> F.

I4 :
        F -> (.S)
        S -> .S+T
        S -> .T
        T -> .T*F
        T -> .F
        F -> .(S)
        F -> .t

I5 :
        F -> t.
```

```
I5 :
        F -> t.

I6 :
        S -> S+.T
        T -> .T*F
        T -> .F
        F -> .(S)
        F -> .t

I7 :
        T -> T*.F
        F -> .(S)
        F -> .t

I8 :
        F -> (S.)
        S -> S.+T

I9 :
        S -> S+T.
        T -> T.*F

I10 :
        T -> T*F.

I11 :
        F -> (S).                       DFA TABLE IS AS FOLLOWS


I0 : 'S'->I1 | 'T'->I2 | 'F'->I3 | '('->I4 | 't'->I5 |


I1 : '+'->I6 |


I2 : '*'->I7 |
```

```
 Enter any String
 a+a*a$
 0       a+a*a$
 0a5             +a*a$
 0F3             +a*a$
 0T2             +a*a$
 0E1             +a*a$
 0E1+6           a*a$
 0E1+6a5         *a$
 0E1+6F3         *a$
 0E1+6T9         *a$
 0E1+6T9*7               a$
 0E1+6T9*7a5             $
 0E1+6T9*7F10            $
 0E1+6T9         $
 0E1             $
 Given String is accepted
 PS C:\Users\Dell\Downloads>
```



3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficult and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1:

Apply the optimization techniques on the above mini-compiler to increase its efficiency and decreasing memory space.