**LAB (14) TASK(1)**

**Code:**

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;145

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();

}146

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", "" },

{ "$", "" },147

{ "}", "" },

{ "+", "" },

{ "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", "" },148

{ "10", "" },

{ "0", "" },

{ ";", "" },

{ "if", "" },

{ ">", "" },

{ "print", "" },

{ "else", "" },

{ "$", "" },

{ "}", "" },

{ "+", "" },

{ "end", "" },

{ "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", "" },173

{ "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;174

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)175

{

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))176

{ 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; }

}177

}

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("}"))178

{

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);

}179

}

#endregion

///////////////////////////////////////END\_Check\_And\_Make\_Entries////////

/////////////

#region Print Stack

void Print\_Stack()

{

foreach (String i in Stack)

{

Output.AppendText(i);

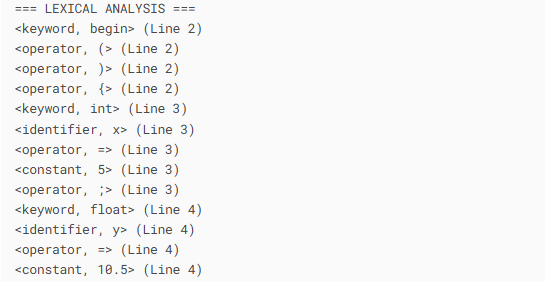
}

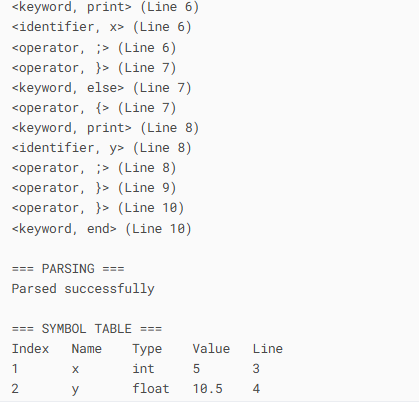
Output.AppendText("\n");

}

#endregion }}

**Output:**

****

****