

**Lab Task # 12**

**Submitted By:**

Alveena

FA21-BCS-014

**Course Instructor:**

Mr. Syed Bilal Bukhari

**Course:**

Compiler Construction

**Date:**

November 26, 2024

**DEPARTMENT OF COMPUTER SCIENCE**

**COMSATS UNIVERSITY ISLAMABAD, ATTOCK CAMPUS**

**Lab Task:**

Understand the integrated code, execute it to get the desired output.

using System;

using System.Collections.Generic;

using System.Text.RegularExpressions;

using System.Windows.Forms;

namespace Lab12\_Task

{

public partial class Form1 : Form

{

// Symbol Table and related data

List<List<String>> SymbolTable;

List<String> VariableList;

List<String> KeywordList;

// Regular Expressions

Regex VariableRegex;

Regex ConstantRegex;

Regex OperatorRegex;

Regex SpecialCharacterRegex;

public Form1()

{

InitializeComponent();

// Initialize Symbol Table and Lists

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

VariableList = new List<String>();

KeywordList = new List<String> { "int", "float", "begin", "end", "print", "if", "else", "while", "main", "new" };

// Initialize Regular Expressions

VariableRegex = new Regex(@"^[A-Za-z\_][A-Za-z0-9]\*$");

ConstantRegex = new Regex(@"^[0-9]+(**\.**[0-9]+)?(e[+-]?[0-9]+)?$");

OperatorRegex = new Regex(@"[+**\-**\*/=<>(){};]");

SpecialCharacterRegex = new Regex(@"^[.,'**\[\]**{}();:?]$");

// Clear Output and Symbol Table displays

tfTokens.Text = "";

tfSymbolTable.Text = "";

}

private void btn\_Input\_Click(object sender, EventArgs e)

{

// Get input from the text box

string userInput = tfInput.Text;

// Buffers and Lists

List<string> FinalTokens = new List<string>();

List<char> TempBuffer = new List<char>();

// Process Input Character by Character

char[] inputCharacters = userInput.ToCharArray();

for (int i = 0; i < inputCharacters.Length; i++)

{

char currentChar = inputCharacters[i];

// Check if the current character matches any token type

if (VariableRegex.IsMatch(currentChar.ToString()) || ConstantRegex.IsMatch(currentChar.ToString()) ||

OperatorRegex.IsMatch(currentChar.ToString()) || SpecialCharacterRegex.IsMatch(currentChar.ToString()) || currentChar == ' ')

{

TempBuffer.Add(currentChar);

}

// End of line or space triggers token formation

if (currentChar == '\n' || currentChar == ' ')

{

if (TempBuffer.Count > 0)

{

string token = new string(TempBuffer.ToArray()).Trim();

FinalTokens.Add(token);

TempBuffer.Clear();

}

}

// Operators and special characters form immediate tokens

if (OperatorRegex.IsMatch(currentChar.ToString()) || SpecialCharacterRegex.IsMatch(currentChar.ToString()))

{

if (TempBuffer.Count > 0)

{

string token = new string(TempBuffer.ToArray()).Trim();

FinalTokens.Add(token);

TempBuffer.Clear();

}

FinalTokens.Add(currentChar.ToString());

}

}

// Process remaining characters in buffer

if (TempBuffer.Count > 0)

{

string remainingToken = new string(TempBuffer.ToArray()).Trim();

FinalTokens.Add(remainingToken);

TempBuffer.Clear();

}

// Display Tokens and Populate Symbol Table

tfTokens.Clear();

SymbolTable.Clear();

foreach (string token in FinalTokens)

{

if (KeywordList.Contains(token))

{

tfTokens.AppendText($"< keyword, {token} >\n");

}

else if (VariableRegex.IsMatch(token) && !KeywordList.Contains(token))

{

tfTokens.AppendText($"< variable, {token} >\n");

if (!VariableList.Contains(token))

{

VariableList.Add(token);

SymbolTable.Add(new List<string> { "Variable", token });

}

}

else if (ConstantRegex.IsMatch(token))

{

tfTokens.AppendText($"< constant, {token} >\n");

SymbolTable.Add(new List<string> { "Constant", token });

}

else if (OperatorRegex.IsMatch(token))

{

tfTokens.AppendText($"< operator, {token} >\n");

}

else if (SpecialCharacterRegex.IsMatch(token))

{

tfTokens.AppendText($"< special, {token} >\n");

}

else

{

tfTokens.AppendText($"< unknown, {token} >\n");

}

}

// Display Symbol Table

DisplaySymbolTable();

}

private void DisplaySymbolTable()

{

tfSymbolTable.Clear();

tfSymbolTable.AppendText("Type\t\tToken\n");

tfSymbolTable.AppendText("\n\n----------------------------\n\n");

foreach (var entry in SymbolTable)

{

tfSymbolTable.AppendText($"\n{entry[0]}\t\n\t\n{entry[1]}\n");

}

}

}

}

**Ouput:**

