

**Lab Tasks – lab 4**

**Submitted By:**

Ramsha Kokab

FA21-BCS-010

**Course Instructor:**

Mr. Syed Bilal Haider

**Course:**

Compiler Construction

**Date:**

27th September, 2024

**DEPARTMENT OF COMPUTER SCIENCE**

**COMSATS UNIVERSITY ISLAMABAD, ATTOCK CAMPUS**

**Task#1: Implement lexical analyzer using two buffers**

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.Text.RegularExpressions;

using System.Windows.Forms;

namespace lab3Task

{

public partial class Form1 : Form

{

static int BUFFER\_SIZE = 20; // Size of each buffer

static char[] buffer1 = new char[BUFFER\_SIZE + 1]; // Adding sentinel at the end

static char[] buffer2 = new char[BUFFER\_SIZE + 1];

static int currentBuffer = 1; // To track which buffer is currently being read

static int bufferIndex = 0; // Index for current character in the buffer

public Form1()

{

InitializeComponent();

}

private void button1\_Click(object sender, EventArgs e)

{

// Clear previous tokens output

tfTokens.Clear();

// Get user input from the rich text box

string userInput = tfInput.Text;

// Load the buffers with the user input

LoadBuffers(userInput);

// Perform lexical analysis using two-buffer scheme

LexicalAnalysis();

}

// Load the buffers with the user input

private void LoadBuffers(string input)

{

// Split the input into two buffers

for (int i = 0; i < BUFFER\_SIZE && i < input.Length; i++)

{

buffer1[i] = input[i];

}

buffer1[BUFFER\_SIZE] = '\0'; // Sentinel character

for (int i = BUFFER\_SIZE; i < 2 \* BUFFER\_SIZE && i < input.Length; i++)

{

buffer2[i - BUFFER\_SIZE] = input[i];

}

buffer2[BUFFER\_SIZE] = '\0'; // Sentinel character

}

// Get the next character from the buffer

private char NextChar()

{

char c;

if (currentBuffer == 1)

{

c = buffer1[bufferIndex];

if (c == '\0') // If we hit the sentinel, switch to buffer2

{

currentBuffer = 2;

bufferIndex = 0;

return NextChar(); // Recursively fetch from buffer2

}

}

else

{

c = buffer2[bufferIndex];

if (c == '\0') // If we hit the sentinel, there's no more input

{

return '\0'; // End of input

}

}

bufferIndex++;

return c;

}

// Lexical Analysis using two-buffer scheme

private void LexicalAnalysis()

{

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

// Regular Expressions for matching tokens

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

Regex digitRegex = new Regex(@"^\d+(**\.**\d+)?$");

Regex operatorRegex = new Regex(@"^[+**\-**\*/><=]+$");

Regex punctuationRegex = new Regex(@"^[;{}()]$");

string currentToken = "";

char currentChar;

// Process each character until the end of input

while ((currentChar = NextChar()) != '\0')

{

if (char.IsWhiteSpace(currentChar))

{

// Process the current token before resetting

if (currentToken != "")

{

tokens.Add(ClassifyToken(currentToken, variableRegex, digitRegex, operatorRegex, punctuationRegex));

currentToken = ""; // Reset token

}

}

else

{

currentToken += currentChar;

}

}

// Process any remaining token

if (currentToken != "")

{

tokens.Add(ClassifyToken(currentToken, variableRegex, digitRegex, operatorRegex, punctuationRegex));

}

// Display the tokens in the rich text box

foreach (var token in tokens)

{

tfTokens.AppendText(token + "\n");

}

}

// Classify tokens based on regular expressions

private string ClassifyToken(string token, Regex variableRegex, Regex digitRegex, Regex operatorRegex, Regex punctuationRegex)

{

if (variableRegex.IsMatch(token))

{

return $"<Variable, {token}>";

}

else if (digitRegex.IsMatch(token))

{

return $"<Digit, {token}>";

}

else if (operatorRegex.IsMatch(token))

{

return $"<Operator, {token}>";

}

else if (punctuationRegex.IsMatch(token))

{

return $"<Punctuation, {token}>";

}

else

{

return $"<Unknown, {token}>";

}

}

}

}

**Output:**

