**LAB (12) TASK (1)**

**Code:**

using System;

using System.Collections.Generic;

using System.Text.RegularExpressions;

class LexicalAnalyzer

{

static void Main()

{

string inputCode = @"

int x = 10;

float y = 20.5;

if (x > 5) {

y = y + 1;

}

";

AnalyzeCode(inputCode);

}

static void AnalyzeCode(string input)

{

// Initialize symbol table

List<SymbolEntry> symbolTable = new List<SymbolEntry>();

// Define keywords

HashSet<string> keywords = new HashSet<string> { "int", "float", "if", "else", "while", "main" };

// Define regular expressions

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

Regex constantReg = new Regex(@"^[0-9]+(\.[0-9]+)?([eE][+-]?[0-9]+)?$");

Regex operatorReg = new Regex(@"^[+\-\*/=<>!&|]+$");

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

// Split input into lines

string[] lines = input.Split(new[] { '\n', '\r' }, StringSplitOptions.RemoveEmptyEntries);

int lineNumber = 0;

foreach (string line in lines)

{

lineNumber++;

string trimmedLine = line.Trim();

if (string.IsNullOrEmpty(trimmedLine)) continue;

Console.WriteLine($"\nProcessing line {lineNumber}: {trimmedLine}");

// Tokenize the line

string[] tokens = TokenizeLine(trimmedLine);

// Process tokens

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

{

string token = tokens[i];

if (keywords.Contains(token))

{

Console.WriteLine($"<keyword, {token}>");

// Check for variable declaration

if (token == "int" || token == "float")

{

if (i + 1 < tokens.Length && variableReg.IsMatch(tokens[i + 1]))

{

string varName = tokens[i + 1];

string type = token;

string value = "null";

// Check for initialization

if (i + 3 < tokens.Length && tokens[i + 2] == "=")

{

value = tokens[i + 3].TrimEnd(';');

}

// Add to symbol table

symbolTable.Add(new SymbolEntry

{

Index = symbolTable.Count + 1,

Name = varName,

Type = type,

Value = value,

LineNumber = lineNumber

});

}

}

}

else if (variableReg.IsMatch(token))

{

// Check if variable exists in symbol table

SymbolEntry entry = symbolTable.Find(e => e.Name == token);

if (entry != null)

{

Console.WriteLine($"<var{entry.Index}, {entry.Index}>");

}

else

{

Console.WriteLine($"<identifier, {token}>");

}

}

else if (constantReg.IsMatch(token))

{

Console.WriteLine($"<constant, {token}>");

}

else if (operatorReg.IsMatch(token))

{

Console.WriteLine($"<operator, {token}>");

}

else if (specialReg.IsMatch(token))

{

Console.WriteLine($"<special, {token}>");

}

}

}

// Display symbol table

Console.WriteLine("\nSymbol Table:");

Console.WriteLine("Index\tName\tType\tValue\tLine");

foreach (var entry in symbolTable)

{

Console.WriteLine($"{entry.Index}\t{entry.Name}\t{entry.Type}\t{entry.Value}\t{entry.LineNumber}");

}

}

static string[] TokenizeLine(string line)

{

// This is a simplified tokenizer - a real implementation would be more sophisticated

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

string currentToken = "";

foreach (char c in line)

{

if (char.IsWhiteSpace(c))

{

if (!string.IsNullOrEmpty(currentToken))

{

tokens.Add(currentToken);

currentToken = "";

}

}

else if (IsSpecialCharacter(c))

{

if (!string.IsNullOrEmpty(currentToken))

{

tokens.Add(currentToken);

currentToken = "";

}

tokens.Add(c.ToString());

}

else

{

currentToken += c;

}

}

if (!string.IsNullOrEmpty(currentToken))

{

tokens.Add(currentToken);

}

return tokens.ToArray();

}

static bool IsSpecialCharacter(char c)

{

return c == '=' || c == '+' || c == '-' || c == '\*' || c == '/' ||

c == '(' || c == ')' || c == '{' || c == '}' ||

c == ';' || c == '<' || c == '>' || c == '!';

}

}

class SymbolEntry

{

public int Index { get; set; }

public string Name { get; set; }

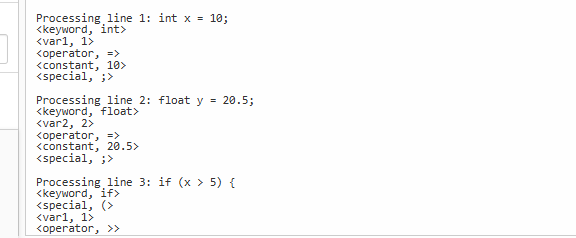
public string Type { get; set; }

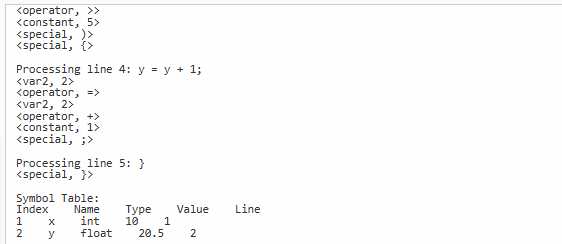
public string Value { get; set; }

public int LineNumber { get; set; }

}

**Output:**

****

****