**PRACTICAL # 08**

**OBJECT:**

Completing lexical analyzer with lexical error handling.

**THEORY:**

We have covered the basic token types and their recognition from the source code. This labs aims to complete the lexical analyzer by integrating all the individual tokens recognition functions and handling lexical errors.

**Program:**

This program completes the basic lexical analyzer.

*using System;*

*using System.Collections.Generic;*

*using System.Linq;*

*using System.Text;*

*using System.Threading.Tasks;*

*namespace LexicalAnalyser*

*{*

*class Token*

*{*

*private string tokenClass;*

*private string lexeme;*

*private int lineNum;*

*public Token( string cls, string lexm ) {*

*this.tokenClass = cls;*

*this.lexeme = lexm;*

*}*

*public Token( string cls, string lexm, int lineNum ) {*

*this.tokenClass = cls;*

*this.lexeme = lexm;*

*this.lineNum = lineNum;*

*}*

*public string TokenClass {*

*get {*

*return this.tokenClass;*

*}*

*}*

*public string Lexeme {*

*get {*

*return this.lexeme;*

*}*

*}*

*public int LineNum*

*{*

*get*

*{*

*return this.lineNum;*

*}*

*}*

*public string toString() {*

*return "("+ tokenClass + ", " + lexeme + ", " + lineNum + ")" ;*

*}*

*}*

*}*

*using System;*

*using System.Collections.Generic;*

*using System.Linq;*

*using System.Text;*

*using System.Threading.Tasks;*

*namespace LexicalAnalyser*

*{*

*class Scanner*

*{*

*private static string[] keywords =*

*{ //"char", "int", "float", "double",*

*"for", "do", "while", "if", "else" };*

*private static string[] dataTypes =*

*{ "char", "int", "float", "double" };*

*public static Token recognizeID(string src, ref int i, int lineNum) {*

*char peek = src[i];*

*string identifier = "";*

*if (peek >= 'a' && peek <= 'z'*

*||*

*peek >= 'A' && peek <= 'Z') {*

*do{*

*identifier += peek;*

*i++;*

*peek = src[i];*

*} while (peek >= 'a' && peek <= 'z'*

*||*

*peek >= 'A' && peek <= 'Z'*

*||*

*peek >= '0' && peek <= '9');*

*i--;*

*}*

*Token t;*

*if (keywords.Contains(identifier) == true)*

*{*

*t = new Token(identifier, identifier, lineNum);*

*}else if(dataTypes.Contains(identifier)){*

*t = new Token("DT", identifier);*

*}*

*else {*

*t = new Token("ID", identifier, lineNum);*

*}*

*Console.WriteLine(t.toString());*

*System.IO.File.AppendAllText("tokens.txt", t.toString() + "\n");*

*return t;*

*}*

*public static Token intDFA(string src, ref int i, int lineNum) {*

*char peek = src[i];*

*int val = 0;*

*Token t = null;*

*if ( peek >= '0' && peek <='9' )*

*{*

*val = 0;*

*do*

*{ //12a*

*val = val \* 10 + Convert.ToInt32 (peek.ToString());*

*i++;*

*peek = src[i];*

*} while (peek >= '0' && peek <= '9');*

*i--;*

*if (peek >= 'a' && peek <= 'z') {*

*t = new Token("invalid int",*

*val.ToString()+peek,*

*lineNum);*

*Console.WriteLine(t.toString());*

*}*

*if (val != 0) {*

*t = new Token("int\_const",*

*val.ToString(),*

*lineNum);*

*Console.WriteLine(t.toString());*

*System.IO.File.AppendAllText("tokens.txt", t.toString() + "\n" );*

*}*

*}*

*return t;*

*}*

*public static Token recognizePunctuations(string src, ref int i, int lineNum) {*

*char peek ;*

*Token t = null;*

*peek = src[i];*

*if( peek == ';' ||*

*peek == ',' ||*

*peek == '{' ||*

*peek == '}' ||*

*peek == '(' ||*

*peek == ')'*

*){*

*t = new Token(peek.ToString(),*

*peek.ToString(),*

*lineNum);*

*Console.WriteLine(t.toString());*

*System.IO.File.AppendAllText("tokens.txt", t.toString() + "\n");*

*}*

*if (peek == '>' || peek == '<' )*

*{*

*t = new Token("relop",*

*peek.ToString(),*

*lineNum);*

*Console.WriteLine(t.toString());*

*System.IO.File.AppendAllText("tokens.txt", t.toString() + "\n");*

*}*

*return t;*

*}*

*public static Token recognizeOperators(string src, ref int i, int lineNum)*

*{*

*char peek ;*

*Token t = null;*

*peek = src[i];*

*if( peek == '+' ||*

*peek == '-' ||*

*peek == '\*' ||*

*peek == '/' ||*

*peek == '='*

*){*

*t = new Token(peek.ToString(), peek.ToString(), lineNum);*

*Console.WriteLine(t.toString());*

*System.IO.File.AppendAllText("tokens.txt", t.toString() + "\n");*

*}*

*return t;*

*}*

*}*

*}*

*class Program*

*{*

*private static int tokenIndex = 0;*

*private static List<Token> tokens = new List<Token>();*

*public static void dummy(){*

*//curr\_scope = dummy\_1*

*int a;*

*}*

*static void Main(string[] args)*

*{*

*lexicalAnalysis();*

*}*

*public static void lexicalAnalysis() {*

*int lineNum = 0;*

*Token token;*

*string srcCode = System.IO.File.ReadAllText("srcCode.x");*

*Console.WriteLine( "source code = " + srcCode);*

*Console.WriteLine( "Lexical Analyser Started " );*

*char peek;*

*int i = 0;*

*while (i < srcCode.Length)*

*{*

*token = null;*

*peek = srcCode[i];*

*//Check ID*

*if (peek >= 'a' && peek <= 'z'*

*||*

*peek >= 'A' && peek <= 'Z')*

*{*

*token = Scanner.recognizeID(srcCode, ref i, lineNum);*

*tokens.Add(token);*

*}*

*//Check Number*

*else if (peek >= '0' && peek <= '9')*

*{*

*token = Scanner.intDFA(srcCode, ref i, lineNum);*

*if (token == null)*

*{*

*Console.WriteLine("invlid token");*

*break;*

*}*

*tokens.Add(token);*

*}*

*//Check Punctuation*

*else if (peek == ';' ||*

*peek == ',' ||*

*peek == '{' ||*

*peek == '}' ||*

*peek == '>' ||*

*peek == '<' ||*

*peek == '(' ||*

*peek == ')'*

*)*

*{*

*token = Scanner.recognizePunctuations(srcCode, ref i, lineNum);*

*tokens.Add(token);*

*}*

*//Check Operator*

*else if (peek == '+' || peek == '-' || peek == '\*' || peek == '/' || peek == '=')*

*{*

*token = Scanner.recognizeOperators(srcCode, ref i, lineNum);*

*tokens.Add(token);*

*}*

*else if (peek == '\n')*

*{*

*lineNum++;*

*}*

*i++;*

*}*

*Console.WriteLine("Lexical Analyser Ended ");*

*}*

*}*

**ACTIVITIES**

**Activity 1**

Write a complete lexical analyzer for your language and execute it on a sample code. Check if it correctly recognized all the tokens and outputs error on invalid tokens.

**REVIEW QUESTIONS**

1. What are lexical errors?
2. What is the output of lexical analyzer if there is no lexical error in the source code?