LAB NO 4:

Activity 1:

LexicalAnalyzerV1.Form1 btn_Input_Click(object sender, EventArgs e)

```
List<string> keywordList = new List<string> { "int", "float", "while", "main", "if

int row = 1;
int count = 1;
int line_num = 0;

string[,] SymbolTable = new string[20, 6];
List<string> varListinSymbolTable = new List<string>();
ArrayList finalArray = new ArrayList();
ArrayList finalArrayc = new ArrayList();
ArrayList tempArray = new ArrayList();

char[] charinput = userInput.ToCharArray();

Regex variable_Reg = new Regex(@"^[A-Za-z_][A-Za-z0-9]*$");
Regex constants_Reg = new Regex(@"^[0-9]+(\.[0-9]+)?([eE][+-]?[0-9]+)?$");
Regex operators_Reg = new Regex(@"[-+*/><=&|]");
Regex Special_Reg = new Regex(@"[.,'\[\]{}();:?]");

for (int itr = 0; itr < charinput.Length; itr++)
{
    string currentChar = charinput[itr].ToString();

    if (variable_Reg.IsMatch(currentChar) || constants_Reg.IsMatch(currentChar) ||
```

```
    if (variable_Reg.IsMatch(currentChar) || constants_Reg.IsMatch(currentChar) ||
        operators_Reg.IsMatch(currentChar) || Special_Reg.IsMatch(currentChar) ||
        charinput[itr] == ' ')
    {
        tempArray.Add(currentChar);
    }

    if (charinput[itr] == '\n')
    {
        if (tempArray.Count != 0)
        {
            finalArray.Add(string.Join("", tempArray.ToArray()));
            tempArray.Clear();
        }
    }
}

if (tempArray.Count != 0)
{
    finalArray.Add(string.Join("", tempArray.ToArray()));
    tempArray.Clear();
}
```

```
foreach (string line in finalArray)
{
    line_num++;
    char[] lineChar = line.ToCharArray();

    for (int itr = 0; itr < lineChar.Length; itr++)
    {
        string currentChar = lineChar[itr].ToString();

        if (variable_Reg.IsMatch(currentChar) || constants_Reg.IsMatch(currentChar
        {
            tempArray.Add(currentChar);
        }

        if (lineChar[itr] == ' ' || operators_Reg.IsMatch(currentChar) || Special_
        {
            if (tempArray.Count != 0)
            {
                finalArrayc.Add(string.Join("", tempArray.ToArray()));
                tempArray.Clear();
            }

            if (operators_Reg.IsMatch(currentChar) || Special_Reg.IsMatch(currentC
```

Output:

```
int a = 212
int b = 30
int c = a*b
```

```
=
*
```

```
<keyword, int> < op, => < digit, 212>
<keyword, int> < op, => < digit, 30>
<keyword, int> < op, => < op, *>
```

Graded Task 1:

```
using System;
using System.Collections.Generic;
using System.IO;

3 references
class LexicalAnalyzer
{
    const int BUFFER_SIZE = 10; // Define buffer size
    char[] bufferA = new char[BUFFER_SIZE];
    char[] bufferB = new char[BUFFER_SIZE];
    int bufferPointer = 0;
    bool isBufferAActive = true;
    StreamReader fileReader;

    // List of keywords
    HashSet<string> keywords = new HashSet<string> { "if", "else", "while", "return", "int", "floa

    1 reference
    public LexicalAnalyzer(string filePath)
    {
        fileReader = new StreamReader(filePath);
        LoadBuffer(); // Load initial buffer
    }

```

```
        if (isBufferAActive)
        {
            Array.Clear(bufferB, 0, BUFFER_SIZE);
            fileReader.Read(bufferB, 0, BUFFER_SIZE);
        }
        else
        {
            Array.Clear(bufferA, 0, BUFFER_SIZE);
            fileReader.Read(bufferA, 0, BUFFER_SIZE);
        }

        isBufferAActive = !isBufferAActive; // Swap buffers
        bufferPointer = 0;
    }

    3 references
    private char GetNextChar()
    {
        if (bufferPointer >= BUFFER_SIZE)
        {
            LoadBuffer();
            bufferPointer = 0;
        }
```

Program.cs | What's New? | lexicalAnalyzer

lexicalAnalyzer | Program | Main()

```
56          char currentChar;
57          string lexeme = "";
58
59          while (!fileReader.EndOfStream)
60          {
61              currentChar = GetNextChar();
62
63              if (char.IsLetter(currentChar)) // Identifier or Keyword
64              {
65                  lexeme += currentChar;
66                  while (char.IsLetterOrDigit(currentChar = GetNextChar()))
67                      lexeme += currentChar;
68
69                  if (keywords.Contains(lexeme))
70                      Console.WriteLine($"Keyword: {lexeme}");
71                  else
72                      Console.WriteLine($"Identifier: {lexeme}");
73
74                  lexeme = "";
75                  bufferPointer--; // Rollback to unread character
76              }
77              else if (char.IsDigit(currentChar)) // Number
78              {
79                  lexeme += currentChar;
```

100 % | 0 | 1 | Ln: 101 | Ch: 7 | SPC | CRLF

Output

Program.cs | What's New? | lexicalAnalyzer

lexicalAnalyzer | Program | Main()

```
72                  Console.WriteLine($"Identifier: {lexeme}");
73
74                  lexeme = "";
75                  bufferPointer--; // Rollback to unread character
76              }
77              else if (char.IsDigit(currentChar)) // Number
78              {
79                  lexeme += currentChar;
80                  while (char.IsDigit(currentChar = GetNextChar()))
81                      lexeme += currentChar;
82
83                  Console.WriteLine($"Number: {lexeme}");
84                  lexeme = "";
85                  bufferPointer--;
86              }
87              else if ("+-*/=;(){}".Contains(currentChar)) // Operators & Symbols
88              {
89                  Console.WriteLine($"Symbol: {currentChar}");
90              }
91              else if (char.IsWhiteSpace(currentChar)) // Ignore spaces
92              {
93                  continue;
94              }
95          }
```

100 % | 0 | 1 | Ln: 101 | Ch: 7 | SPC | CR

```
Program.cs   What's New?                                   lexicalAnaly
lexicalAnalyzer                Program                     Main()
    }
}

0 references
class Program
{
    0 references
    static void Main()
    {
        Console.Write("Enter file path to analyze: ");
        string filePath = Console.ReadLine();

        if (!File.Exists(filePath))
        {
            Console.WriteLine("Error: File not found.");
            return;
        }

        LexicalAnalyzer lexer = new LexicalAnalyzer(filePath);
        lexer.Analyze();

        Console.WriteLine("\nLexical Analysis Completed.");
    }
}
00 %   0   1                                   Ln: 101   Ch: 7
Output
```

Output:

```
Microsoft Visual Studio Debu

Enter file path to analyze: C:\Users\HP\Documents\test.txt
Keyword: int
Identifier: x
Symbol: =
Number: 10
Symbol: ;
Keyword: if
Symbol: (
Identifier: x
Number: 5
Symbol: )
Symbol: {
Keyword: return
Identifier: x
Symbol: ;

Lexical Analysis Completed.

C:\Users\HP\source\repos\lexicalAnalyzer\lexicalAnalyzer\bin\Debug\net8.0\lexicalAnalyzer.exe (process 6584) exited with
 code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```