

LAB NO 5:

Activity 1:

```
Program.cs
lab5 activity1
LexicalAnalyzer
Tokenize(string input)

1  using System;
2  using System.Collections.Generic;
3  using System.Text;
4  using System.Text.RegularExpressions;
5
6  class LexicalAnalyzer
7  {
8      // Define token types
9      private static readonly Dictionary<string, string> tokenDefinitions = new Dictionary<string, string>
10     {
11         { @"\b(int|float|char|double|string|if|else|while|for|return|void)\b", "KEYWORD" },
12         { @"\b[A-Za-z_][A-Za-z0-9_]*\b", "IDENTIFIER" },
13         { @"\b\d+\b", "NUMBER" },
14         { @"\+", "PLUS_OPERATOR" },
15         { @"\-", "MINUS_OPERATOR" },
16         { @"\*", "MULTIPLICATION_OPERATOR" },
17         { @"/", "DIVISION_OPERATOR" },
18         { @"\=", "ASSIGNMENT_OPERATOR" },
19         { @"\=", "EQUALITY_OPERATOR" },
20         { @"\!=", "NOT_EQUAL_OPERATOR" },
21         { @"\<", "LESS_THAN_OPERATOR" },
22         { @"\>", "GREATER_THAN_OPERATOR" },
23         { @"\<=", "LESS_EQUAL_OPERATOR" },
24         { @"\>=", "GREATER_EQUAL_OPERATOR" },
25         { @"\(", "LEFT_PARENTHESIS" },
26         { @"\)", "RIGHT_PARENTHESIS" },
27         { @"\{", "LEFT_CURLY_BRACE" },
28         { @"\}", "RIGHT_CURLY_BRACE" },
29         { @";", "SEMICOLON" },
30         { @",", "COMMA" },
31         { @"\s+", "WHITESPACE" } // Handle whitespace (ignored)
32     };
33
34     public static List<string, string> Tokenize(string input)
35     {
36         List<string, string> tokens = new List<string, string>();
37
38         while (!string.IsNullOrEmpty(input))
39         {
40             bool matched = false;
41
42             foreach (var tokenDefinition in tokenDefinitions)
43             {
44                 Match match = Regex.Match(input, tokenDefinition.Key);
45                 if (match.Success)
46                 {
47                     tokens.Add(match.Value, tokenDefinition.Value);
48                     input = input.Substring(match.Length);
49                     matched = true;
50                 }
51             }
52
53             if (!matched)
54             {
55                 tokens.Add(input, "UNKNOWN");
56                 input = "";
57             }
58         }
59     }
60 }
```

```
43 {
44     Regex regex = new Regex("^" + tokenDefinition.Key);
45     Match match = regex.Match(input);
46
47     if (match.Success)
48     {
49         string value = match.Value;
50         string tokenType = tokenDefinition.Value;
51
52         if (tokenType != "WHITESPACE") // Ignore whitespace tokens
53         {
54             tokens.Add((tokenType, value));
55         }
56
57         input = input.Substring(value.Length); // Move forward in the input
58         matched = true;
59         break;
60     }
61 }
62
63 if (!matched)
64 {
65     Console.WriteLine($"Error: Unrecognized token at '{input[0]}');
66     input = input.Substring(1); // Skip the unknown character
67 }
68 }
69
70 return tokens;
71 }
72
73 0 references
74 public static void Main()
75 {
76     Console.WriteLine("Enter the source code:");
77     string input = Console.ReadLine();
78
79     List<(string, string)> tokens = Tokenize(input);
80
81     Console.WriteLine("\nTokens:");
82     foreach (var token in tokens)
83     {
84         Console.WriteLine($"{token.Item1}: {token.Item2}");
85     }
86 }
```

Output:

```
Enter the source code:
int x = 10;

Tokens:
KEYWORD: int
IDENTIFIER: x
ASSIGNMENT_OPERATOR: =
NUMBER: 10
SEMICOLON: ;

C:\Users\HP\source\repos\lab5 activity1\lab5 activity1\bin\Debug\net8.0\lab5 activity1.exe (process 15928) exited with code 0 (0x
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when d
ging stops.
Press any key to close this window . . .
```

Graded Task 1:

```
Program.cs  x
symboltable  SymbolTable  tableSize
{
1  using System;
2  using System.Collections.Generic;
3
4  class SymbolTable
5  {
6      private Dictionary<int, List<string>> table;
7      private int tableSize;
8
9      public SymbolTable(int size)
10     {
11         tableSize = size;
12         table = new Dictionary<int, List<string>>(size);
13
14         for (int i = 0; i < size; i++)
15             table[i] = new List<string>(); // Initialize empty lists
16     }
17
18     // Simple Hash Function
19     private int HashFunction(string identifier)
20     {
21         int hash = 0;
22         foreach (char ch in identifier)
23         {
24             hash += (int)ch; // Sum ASCII values of characters
25         }
26         return hash % tableSize; // Modulo to fit in table
27     }
28
29     // Insert into Symbol Table
30     public void Insert(string identifier)
31     {
32         int index = HashFunction(identifier);
33
34         if (!table[index].Contains(identifier))
35         {
36             table[index].Add(identifier);
37             Console.WriteLine($"Inserted: {identifier} at Index {index}");
38         }
39         else
40         {
41             Console.WriteLine($"Duplicate Entry: {identifier} already exists.");
42         }
43     }
44
45     // Search for an Identifier
}
```

```
Program.cs  X
symboltable SymbolTable tableSize
41 Console.WriteLine($"Duplicate Entry: {identifier} already exists.");
42 }
43 }
44
45 // Search for an Identifier
46 1 reference
47 public bool Search(string identifier)
48 {
49     int index = HashFunction(identifier);
50     return table[index].Contains(identifier);
51 }
52
53 // Display the Symbol Table
54 1 reference
55 public void Display()
56 {
57     Console.WriteLine("\nSymbol Table:");
58     foreach (var entry in table)
59     {
60         Console.Write($"Index {entry.Key}: ");
61         foreach (var id in entry.Value)
62         {
63             Console.Write($"{id} ");
64         }
65     }
66     Console.WriteLine();
67 }
```

```
Program.cs  X
symboltable SymbolTable tableSize
0 references
70 static void Main()
71 {
72     SymbolTable symbolTable = new SymbolTable(10); // Hash table size: 10
73
74     while (true)
75     {
76         Console.WriteLine("\n1. Insert Identifier\n2. Search Identifier\n3. Display Symbol Table\n4. Exit\n");
77         Console.Write("Enter Choice: ");
78         int choice = Convert.ToInt32(Console.ReadLine());
79
80         switch (choice)
81         {
82             case 1:
83                 Console.Write("Enter Identifier: ");
84                 string id = Console.ReadLine();
85                 symbolTable.Insert(id);
86                 break;
87
88             case 2:
89                 Console.Write("Enter Identifier to Search: ");
90                 string searchId = Console.ReadLine();
91                 if (symbolTable.Search(searchId))
92                     Console.WriteLine($"Found: {searchId}");
93                 else
94                     Console.WriteLine($"Not Found: {searchId}");
95             default:
96                 break;
97         }
98     }
99 }
```

The screenshot shows a C# program in Visual Studio. The file is named `Program.cs`. The code defines a `switch` statement with four cases. Case 2 handles searching for an identifier, case 3 handles displaying the symbol table, case 4 handles exiting the program, and the default case handles invalid choices. The code is as follows:

```
88  
89  
90  
91  
92  
93  
94  
95  
96  
97  
98  
99  
100  
101  
102  
103  
104  
105  
106  
107  
108  
109  
110  
111  
112  
113
```

```
case 2:  
    Console.WriteLine("Enter Identifier to Search: ");  
    string searchId = Console.ReadLine();  
    if (symbolTable.Search(searchId))  
        Console.WriteLine($"Found: {searchId}");  
    else  
        Console.WriteLine($"Not Found: {searchId}");  
    break;  
  
case 3:  
    symbolTable.Display();  
    break;  
  
case 4:  
    Console.WriteLine("Exiting...");  
    return;  
  
default:  
    Console.WriteLine("Invalid Choice! Try Again.");  
    break;
```

The status bar at the bottom indicates the current position is Line 112, Column 1, with SPC and CRLF line endings.

Output:

1. Insert Identifier
2. Search Identifier
3. Display Symbol Table
4. Exit

Enter Choice: 1

Enter Identifier: x

Inserted: x at Index 0

1. Insert Identifier
2. Search Identifier
3. Display Symbol Table
4. Exit

Enter Choice: 1

Enter Identifier: y

Inserted: y at Index 1

1. Insert Identifier
2. Search Identifier
3. Display Symbol Table
4. Exit

Enter Choice: 3

Symbol Table:

Index 0: x

Index 1: y

Index 2:

Index 3:

Index 4:

Index 5:

Index 6:

Index 7:

Index 8:

Index 7:

Index 8:

Index 9:

1. Insert Identifier
2. Search Identifier
3. Display Symbol Table
4. Exit

Enter Choice: 4

Exiting...

C:\Users\HP\source\repos\symbolsymbol\bin\Debug\net8.0\symbolsymbol.exe (process 18772) exited with code 0 (0x0).

To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.

Press any key to close this window . . .