

Name: Moazzam Azam

Registration: SP22-BCS-010

Graded Lab Taks

Lab 04

Implement lexical analyzer using two buffers.

```
using System;
```

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

```
class LexicalAnalyzer
```

```
{
```

```
    private string input;
```

```
    private int currentPosition = 0;
```

```
    private char currentChar;
```

```
    private char lookAheadChar;
```

```
    public LexicalAnalyzer(string input)
```

```
    {
```

```
        this.input = input;
```

```
        currentChar = input[currentPosition];
```

```
        lookAheadChar = input.Length > currentPosition + 1 ? input[currentPosition + 1] : '\0';
```

```
    }
```

```
    private void Advance()
```

```
    {
```

```

        currentPosition++;
        if (currentPosition < input.Length)
        {
            currentChar = input[currentPosition];
            lookAheadChar = currentPosition + 1 < input.Length ? input[currentPosition + 1] : '\0';
        }
        else
        {
            currentChar = '\0';
            lookAheadChar = '\0';
        }
    }
}

```

```

private bool IsDigit(char c) => c >= '0' && c <= '9';

```

```

private bool IsLetter(char c) => (c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z') || c == '_';

```

```

private void SkipWhitespace()
{
    while (currentChar == ' ' || currentChar == '\n' || currentChar == '\t' || currentChar == '\r')
    {
        Advance();
    }
}

```

```

public List<string> Tokenize()
{

```

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

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

```
{
```

```
    SkipWhitespace();
```

```
    if (IsLetter(currentChar))
```

```
    {
```

```
        string identifier = "";
```

```
        while (IsLetter(currentChar) || IsDigit(currentChar))
```

```
        {
```

```
            identifier += currentChar;
```

```
            Advance();
```

```
        }
```

```
        tokens.Add(identifier);
```

```
    }
```

```
    else if (IsDigit(currentChar))
```

```
    {
```

```
        string number = "";
```

```
        while (IsDigit(currentChar))
```

```
        {
```

```
            number += currentChar;
```

```
            Advance();
```

```
        }
```

```
        tokens.Add(number);
```

```
    }
```

```
    else if (currentChar == '+' || currentChar == '-' || currentChar == '*' || currentChar == '/')
```

```

    {
        tokens.Add(currentChar.ToString());
        Advance();
    }
    else if (currentChar == '=' || currentChar == ';' || currentChar == '(' || currentChar == ')')
    {
        tokens.Add(currentChar.ToString());
        Advance();
    }
    else
    {
        tokens.Add("UNKNOWN");
        Advance();
    }
}

return tokens;
}
}

```

```

class Program
{
    static void Main(string[] args)
    {
        string input = "int x = 10 + 2;";

        LexicalAnalyzer lexer = new LexicalAnalyzer(input);
    }
}

```

```
List<string> tokens = lexer.Tokenize();

Console.WriteLine("Tokens:");
foreach (var token in tokens)
{
    Console.WriteLine(token);
}
}
```

Lab 05

Implement symbol table using hash function

using System;

```
class SymbolTable
```

```
{
```

```
    private const int TableSize = 10;
```

```
    private (string, int)?[] table;
```

```
    public SymbolTable()
```

```
    {
```

```
        table = new (string, int)?[TableSize];
```

```
    }
```

```
    private int Hash(string key)
```

```
    {
```

```
        int hashValue = 0;
```

```
        foreach (char c in key)
```

```
        {
```

```
            hashValue = (hashValue * 31 + c) % TableSize;
```

```
        }
```

```

        return hashValue;
    }

    public void Insert(string key, int value)
    {
        int index = Hash(key);

        while (table[index] != null)
        {
            if (table[index].Value.Item1 == key)
            {
                table[index] = (key, value);
                return;
            }

            index = (index + 1) % TableSize;
        }

        table[index] = (key, value);
    }

```

```

    public int? Find(string key)
    {
        int index = Hash(key);

        while (table[index] != null)
        {
            if (table[index].Value.Item1 == key)
            {
                return table[index].Value.Item2;
            }

            index = (index + 1) % TableSize;
        }
    }

```

```

    }

    return null;
}

public void Delete(string key)
{
    int index = Hash(key);

    while (table[index] != null)
    {
        if (table[index].Value.Item1 == key)
        {
            table[index] = null;
            return;
        }
        index = (index + 1) % TableSize;
    }

    Console.WriteLine($"Key '{key}' not found.");
}

public void Display()
{
    for (int i = 0; i < TableSize; i++)
    {
        if (table[i] != null)
        {
            Console.WriteLine($"Index {i}: {table[i].Value.Item1} -> {table[i].Value.Item2}");
        }
        else
    }

```



```

        {
            Console.WriteLine($"Index {i}: Empty");
        }
    }
}

```

class Program

```

{
    static void Main(string[] args)
    {
        SymbolTable symbolTable = new SymbolTable();

        symbolTable.Insert("x", 10);
        symbolTable.Insert("y", 20);
        symbolTable.Insert("z", 30);

        Console.WriteLine("Value of x: " + symbolTable.Find("x"));
        Console.WriteLine("Value of y: " + symbolTable.Find("y"));
        Console.WriteLine("Value of z: " + symbolTable.Find("z"));

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

        symbolTable.Delete("y");

        Console.WriteLine("\nSymbol Table after deletion:");
        symbolTable.Display();
    }
}

```

