Name: Moazzam Azam

Registration:SP22-BCS-010

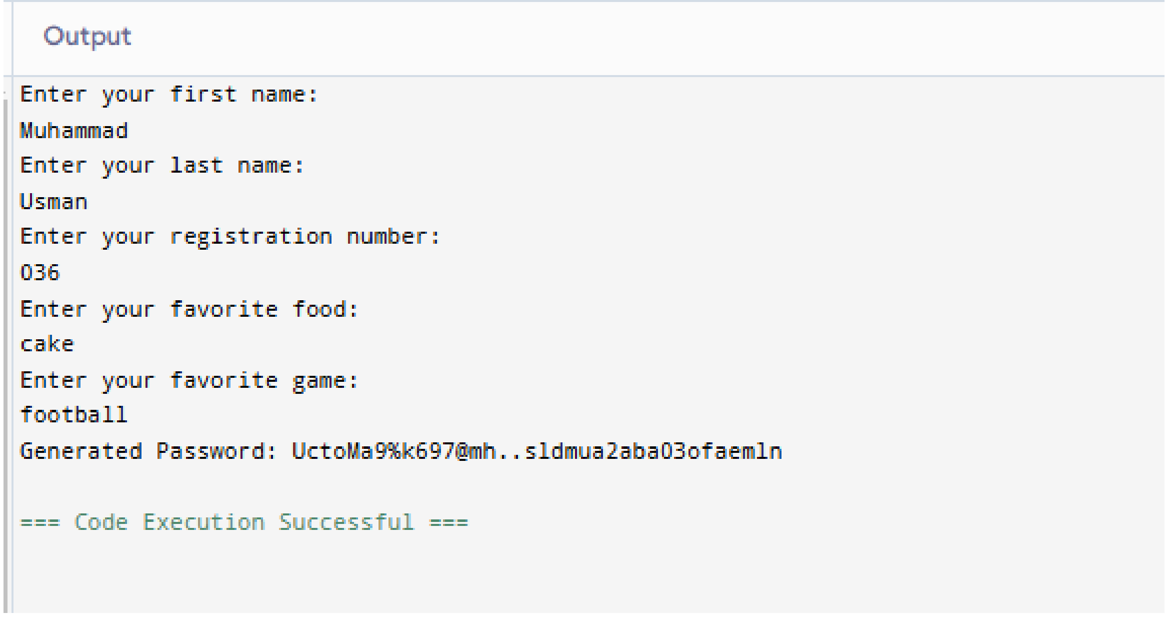
Lab 01

task

```
using System;
using System.Text;
using System.Text.RegularExpressions;

class Program
{
    static void Main(string[] args)
    {
        // Sample inputs
        Console.WriteLine("Enter your first name: ");
        string firstName = Console.ReadLine();

        Console.WriteLine("Enter your last name: ");
        string lastName = Console.ReadLine();

        Console.WriteLine("Enter your registration number: ");
        string regNumber = Console.ReadLine();
        Console.WriteLine("Enter your favorite food: ");
        string food = Console.ReadLine();

        Console.WriteLine("Enter your favorite game: ");
        string game = Console.ReadLine();

        // Generate the password
        string password = GeneratePassword(firstName, lastName, regNumber, food, game);

        // Display the generated password
        Console.WriteLine("Generated Password: " + password);
    }

    static string GeneratePassword(string firstName, string lastName, string regNumber, string food, string game)
    {
        // Combine all input values
        string combined = firstName + lastName + regNumber + food + game;

        // Regular expression to remove any unwanted characters (non-alphanumeric)
        string sanitized = Regex.Replace(combined, @"[^a-zA-Z0-9]", "");

        // Make the string more complex by adding special characters and digits
        string complexPassword = sanitized;

        // Add some random numbers and special characters
        Random rand = new Random();
```

```
string specialChars = "!@#$%^&*()_+[]{}|;:,.<>?/~`";
for (int i = 0; i < 4; i++)
{
    // Add random number
    complexPassword += rand.Next(0, 10).ToString();

    // Add random special character
    complexPassword += specialChars[rand.Next(specialChars.Length)];
}

// Ensure password length is at least 12 characters
if (complexPassword.Length < 12)
{
    complexPassword = complexPassword.PadLeft(12, 'X'); // Add filler 'X' if too short
}

// Randomly shuffle the password to increase complexity
StringBuilder shuffledPassword = new StringBuilder();
while (complexPassword.Length > 0)
{
    int index = rand.Next(complexPassword.Length);
    shuffledPassword.Append(complexPassword[index]);
    complexPassword = complexPassword.Remove(index, 1);
}

return shuffledPassword.ToString();
```

```
    }

}
```

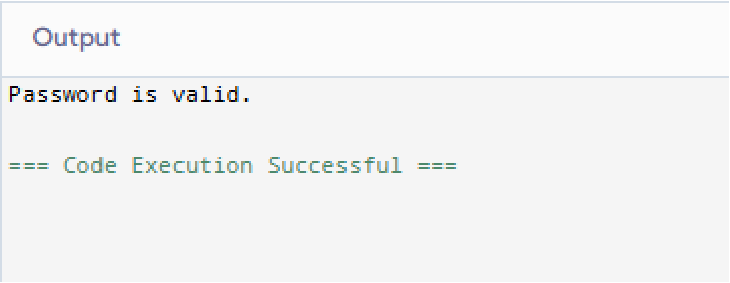


```
Enter your first name:
Muhammad
Enter your last name:
Usman
Enter your registration number:
036
Enter your favorite food:
cake
Enter your favorite game:
football
Generated Password: UctoMa9%k697@mh..sldmua2aba03ofaemln

=== Code Execution Successful ===
```

Lab 1:

task 2

```
using System;

using System.Text.RegularExpressions;


public class Program

{

    public static void Main()

    {

        // Hardcoded password for validation

        string password = "Sp22-bcs-036"; // Example password


        // Regular expression pattern for the requirements
        string pattern = @"^(?=(.*\d.*){2})(?=.*[A-Z])(?=(.*[a-z]){4})(?=(.*[-!@#$%^&*(),.?\""{}|<>]){2}).{1,12}$";


        // Check if the password matches the pattern
        if (Regex.IsMatch(password, pattern))
        {
            Console.WriteLine("Password is valid.");
        }
        else
        {
            Console.WriteLine("Password is invalid.");
        }
    }
}
```

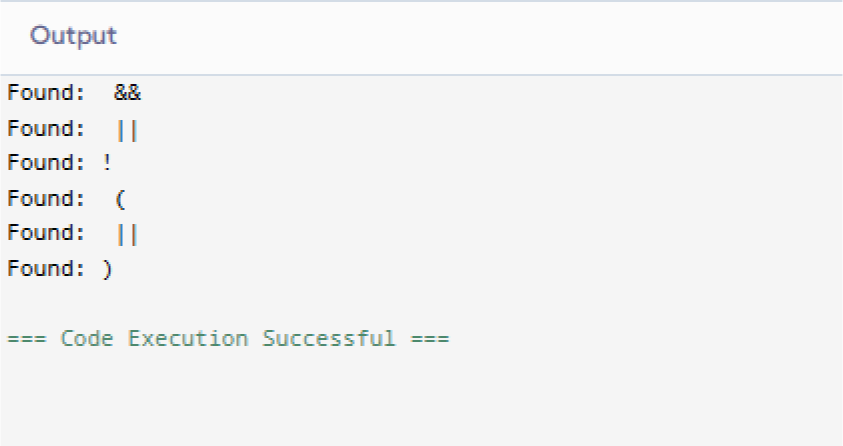


```
Password is valid.

=== Code Execution Successful ===
```

lab 02

task 1

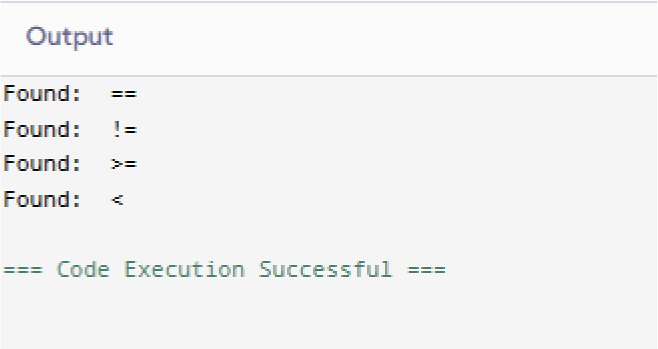
```
using System;
using System.Text.RegularExpressions;


class Program
{
    static void Main()
    {
        // The regular expression for logical operators and parentheses
        string pattern = @"\s*(&&|\|\||!|\(|\))\s*";

        // Test string with logical operators and parentheses
        string input = "x && y || !z (x || y)";

        // Create a Regex object with the pattern
        Regex regex = new Regex(pattern);

        // Find all matches
        MatchCollection matches = regex.Matches(input);

        // Output the matches
        foreach (Match match in matches)
        {
            Console.WriteLine($"Found: {match.Value}");
        }
    }
}
```



```
Found:  &&
Found:  ||
Found: !
Found:  (
Found:  ||
Found: )

=== Code Execution Successful ===
```

lab 2:task 2

```
using System;
using System.Text.RegularExpressions;

class Program
{
    static void Main()
    {
        // The regular expression for relational operators
        string pattern = @"\s*(==|!=|>=|<=|>|<)\s*";

        // Test string with relational operators
        string input = "a == b && c != d || e >= f && g < h";

        // Create a Regex object with the pattern
        Regex regex = new Regex(pattern);

        // Find all matches
        MatchCollection matches = regex.Matches(input);

        // Output the matches
        foreach (Match match in matches)
        {
            Console.WriteLine($"Found: {match.Value}");
        }
    }
}
```



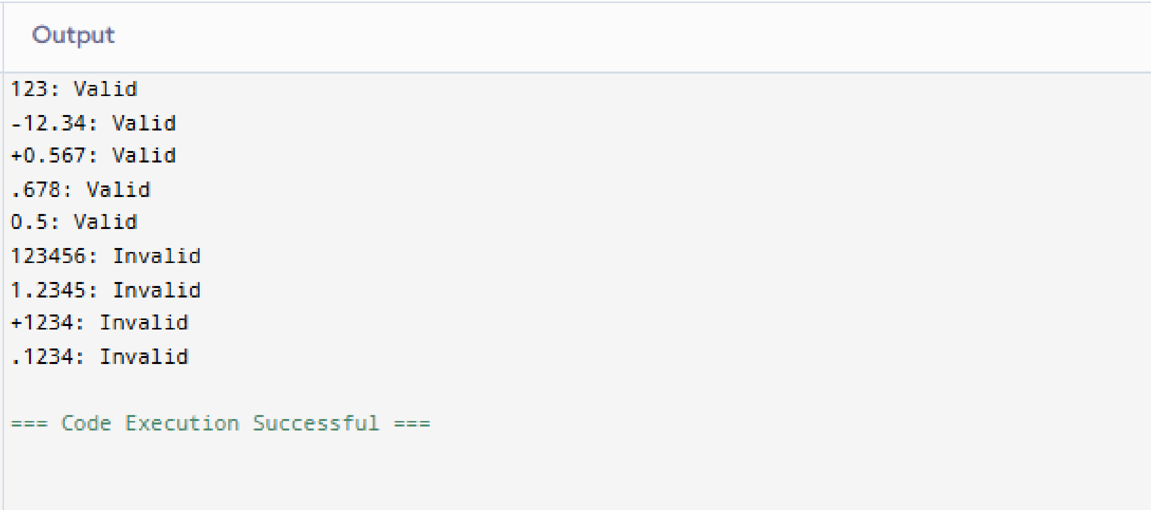
Output

```
Found:  ==
Found:  !=
Found:  >=
Found:  <

=== Code Execution Successful ===
```

Lab 03

task 1

```
using System;
using System.Text.RegularExpressions;

class Program
{
    static void Main()
    {
        // Regular expression for floating point numbers with length <= 6
        string pattern = @"^[+-]?\d{1,3}(\.\d{1,3})?$|^[+-]?\.\d{1,3}$";

        // Test strings
        string[] testStrings = {
            "123",        // valid
            "-12.34",     // valid
            "+0.567",     // valid
            ".678",       // valid
            "0.5",        // valid
            "123456",     // invalid
            "1.2345",     // invalid
            "+1234",      // invalid
            ".1234"       // invalid
        };

        // Check each string against the regex
        foreach (var test in testStrings)
        {
            bool isMatch = Regex.IsMatch(test, pattern);
            Console.WriteLine($"{test}: {(isMatch ? "Valid" : "Invalid")}");
        }
    }
}
```



```
123: Valid
-12.34: Valid
+0.567: Valid
.678: Valid
0.5: Valid
123456: Invalid
1.2345: Invalid
+1234: Invalid
.1234: Invalid

=== Code Execution Successful ===
```

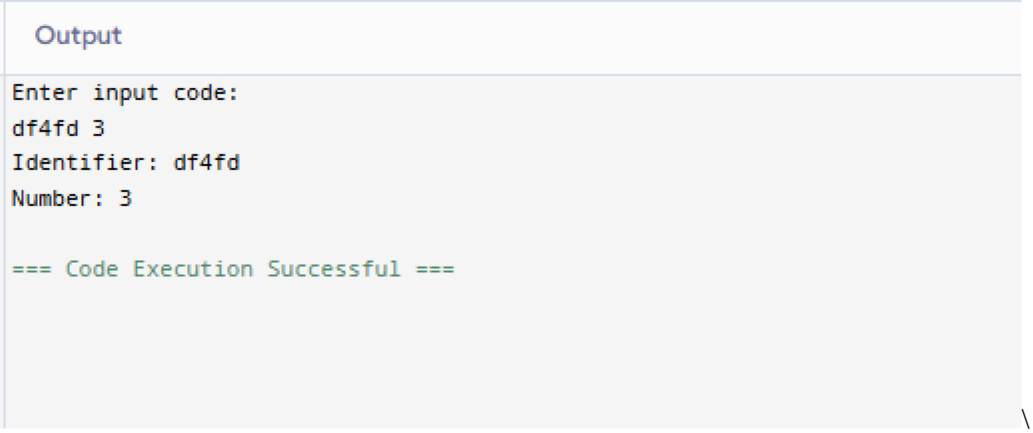
Lab 04

task 1

```
using System;

class LexicalAnalyzer
{
    const int BUFFER_SIZE = 1024;
    const int KEYWORD_COUNT = 3;
```

```
static string[] keywords = { "int", "if", "else" };
static char[] buffer = new char[BUFFER_SIZE];
static int bufferIndex = 0;

static bool IsKeyword(string lexeme)
{
    for (int i = 0; i < KEYWORD_COUNT; i++)
    {
        if (lexeme.Equals(keywords[i]))
            return true;
    }
    return false;
}

static void LexicalAnalyzerFunc()
{
    string lexeme = "";
    while (bufferIndex < buffer.Length && buffer[bufferIndex] != '\0')
    {
        char currentChar = buffer[bufferIndex];
        if (char.IsWhiteSpace(currentChar))
        {
            bufferIndex++;
            continue;
        }

        lexeme = "";
        if (char.IsLetter(currentChar)) // Identifier or Keyword
        {
            while (bufferIndex < buffer.Length && (char.IsLetterOrDigit(buffer[bufferIndex])))
            {
                lexeme += buffer[bufferIndex];
                bufferIndex++;
            }

            if (IsKeyword(lexeme))
                Console.WriteLine($"Keyword: {lexeme}");
            else
                Console.WriteLine($"Identifier: {lexeme}");
        }
        else if (char.IsDigit(currentChar)) // Number
        {
            while (bufferIndex < buffer.Length && char.IsDigit(buffer[bufferIndex]))
            {
                lexeme += buffer[bufferIndex];
                bufferIndex++;
            }
            Console.WriteLine($"Number: {lexeme}");
        }
        else // Operator or special character
        {
            Console.WriteLine($"Operator: {currentChar}");
            bufferIndex++;
        }
    }
}
```

```
    static void Main()
    {
        Console.WriteLine("Enter input code: ");
        string input = Console.ReadLine();
        buffer = input.ToCharArray();

        LexicalAnalyzerFunc();
    }
}
```



```
Enter input code:
df4fd 3
Identifier: df4fd
Number: 3

=== Code Execution Successful ===
```

\

Lab 05

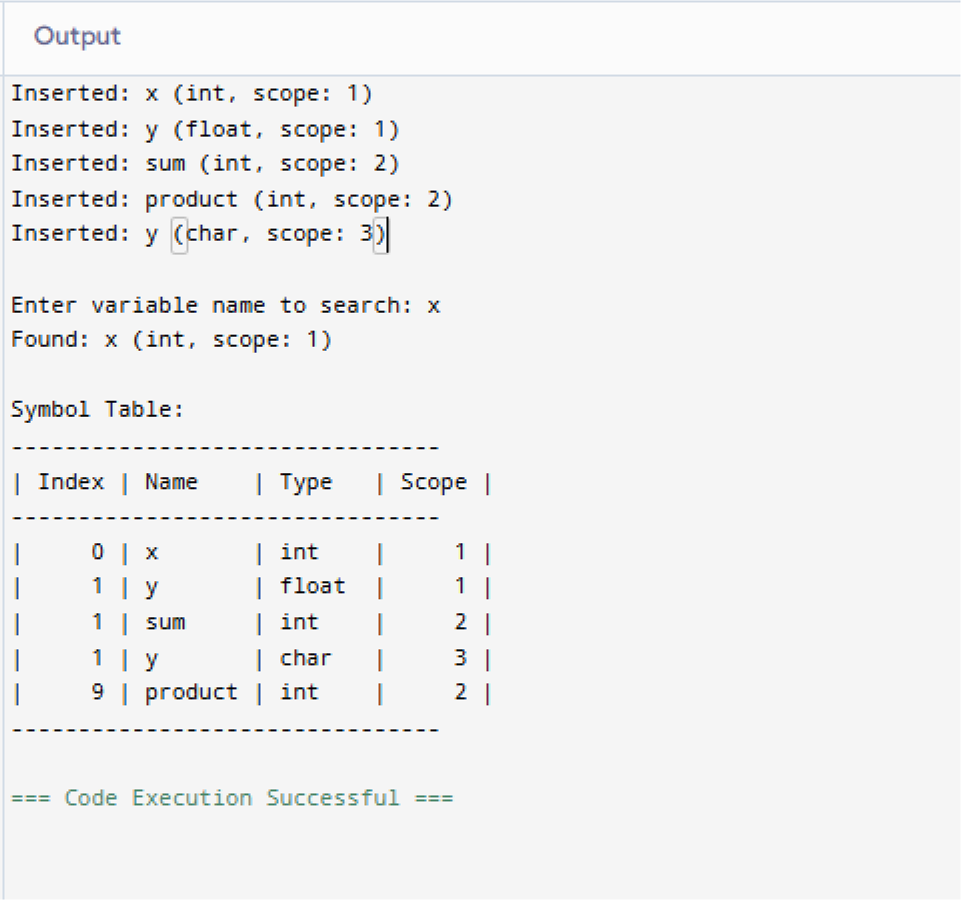
TASK 1

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

class Symbol
{
    public string Name { get; set; }
    public string Type { get; set; }
    public int Scope { get; set; }
    public Symbol Next { get; set; } // Linked list chain
```

```
}

class SymbolTable
{
    private const int TABLE_SIZE = 10;
    private List<Symbol>[] symbolTable;

    public SymbolTable()
    {
        symbolTable = new List<Symbol>[TABLE_SIZE];
        for (int i = 0; i < TABLE_SIZE; i++)
        {
            symbolTable[i] = new List<Symbol>();
        }
    }

    // Hash function (Sum of ASCII values modulo table size)
    private int HashFunction(string name)
    {
        int sum = 0;
        foreach (char c in name)
        {
            sum += c;
        }
        return sum % TABLE_SIZE;
    }

    // Insert a symbol into the table
    public void InsertSymbol(string name, string type, int scope)
```

```
{
    int index = HashFunction(name);

    // Create a new symbol
    Symbol newSymbol = new Symbol
    {
        Name = name,
        Type = type,
        Scope = scope,
        Next = null
    };

    // Insert at the beginning of the linked list (chaining)
    symbolTable[index].Add(newSymbol);
    Console.WriteLine($"Inserted: {name} ({type}, scope: {scope})");
}

// Search for a symbol in the table
public Symbol SearchSymbol(string name)
{
    int index = HashFunction(name);
    foreach (var symbol in symbolTable[index])
    {
        if (symbol.Name.Equals(name, StringComparison.Ordinal))
        {
            return symbol; // Found
        }
    }
    return null; // Not found
    }

    // Display the symbol table
    public void DisplaySymbolTable()
    {
        Console.WriteLine("\nSymbol Table:");
        Console.WriteLine("--------------------------------");
        Console.WriteLine("| Index | Name    | Type   | Scope |");
        Console.WriteLine("--------------------------------");

        for (int i = 0; i < TABLE_SIZE; i++)
        {
            foreach (var symbol in symbolTable[i])
            {
                Console.WriteLine($"| {i,5} | {symbol.Name,-7} | {symbol.Type,-6} | {symbol.Scope,5} |");
            }
        }
        Console.WriteLine("--------------------------------");
    }
}

class Program
{
    static void Main()
    {
        SymbolTable table = new SymbolTable();

        // Insert some symbols
        table.InsertSymbol("x", "int", 1);
        table.InsertSymbol("y", "float", 1);
        table.InsertSymbol("sum", "int", 2);
        table.InsertSymbol("product", "int", 2);
        table.InsertSymbol("y", "char", 3);  // Different scope

        // Search for a symbol
        Console.Write("\nEnter variable name to search: ");
        string searchName = Console.ReadLine();

        Symbol result = table.SearchSymbol(searchName);
        if (result != null)
        {
            Console.WriteLine($"Found: {result.Name} ({result.Type}, scope: {result.Scope})");
        }
        else
        {
            Console.WriteLine("Symbol not found.");
        }

        // Display the symbol table
        table.DisplaySymbolTable();
    }
```

```
}
```



```
Inserted: x (int, scope: 1)
Inserted: y (float, scope: 1)
Inserted: sum (int, scope: 2)
Inserted: product (int, scope: 2)
Inserted: y (char, scope: 3)

Enter variable name to search: x
Found: x (int, scope: 1)

Symbol Table:
--------------------------------
| Index | Name    | Type   | Scope |
--------------------------------
|     0 | x       | int    |     1 |
|     1 | y       | float  |     1 |
|     1 | sum     | int    |     2 |
|     1 | y       | char   |     3 |
|     9 | product | int    |     2 |
--------------------------------

=== Code Execution Successful ===
```