CC-LAB MID

Submitted By:	**Talha Azeem**
Reg No:	**Sp20-Bcs-047**
Submitted To:	**Bilal Bukhari**
Date:	**05-04-2024**

Question 1: Briefly describe the regex library of C# ?

Answer: The regex library in C# provides a robust set of classes and methods for working with regular expressions. It is part of the .NET framework's System.Text.RegularExpressions namespace. Here's a brief overview of its features:

1. **Regex Class**: The Regex class is the primary entry point for working with regular expressions in C#. It provides methods for compiling regular expressions, matching input strings against patterns, and replacing occurrences of patterns in input strings.
2. **Pattern Syntax**: The library supports standard regular expression syntax, including metacharacters for specifying patterns, quantifiers for repetition, character classes, groups, and more.
3. **Match Object**: When a regex pattern matches a portion of an input string, it creates a Match object containing information about the match, such as the matched text, its index in the input string, and any captured groups.
4. **Grouping and Capturing**: Regular expressions can define groups within patterns using parentheses. These groups can be captured

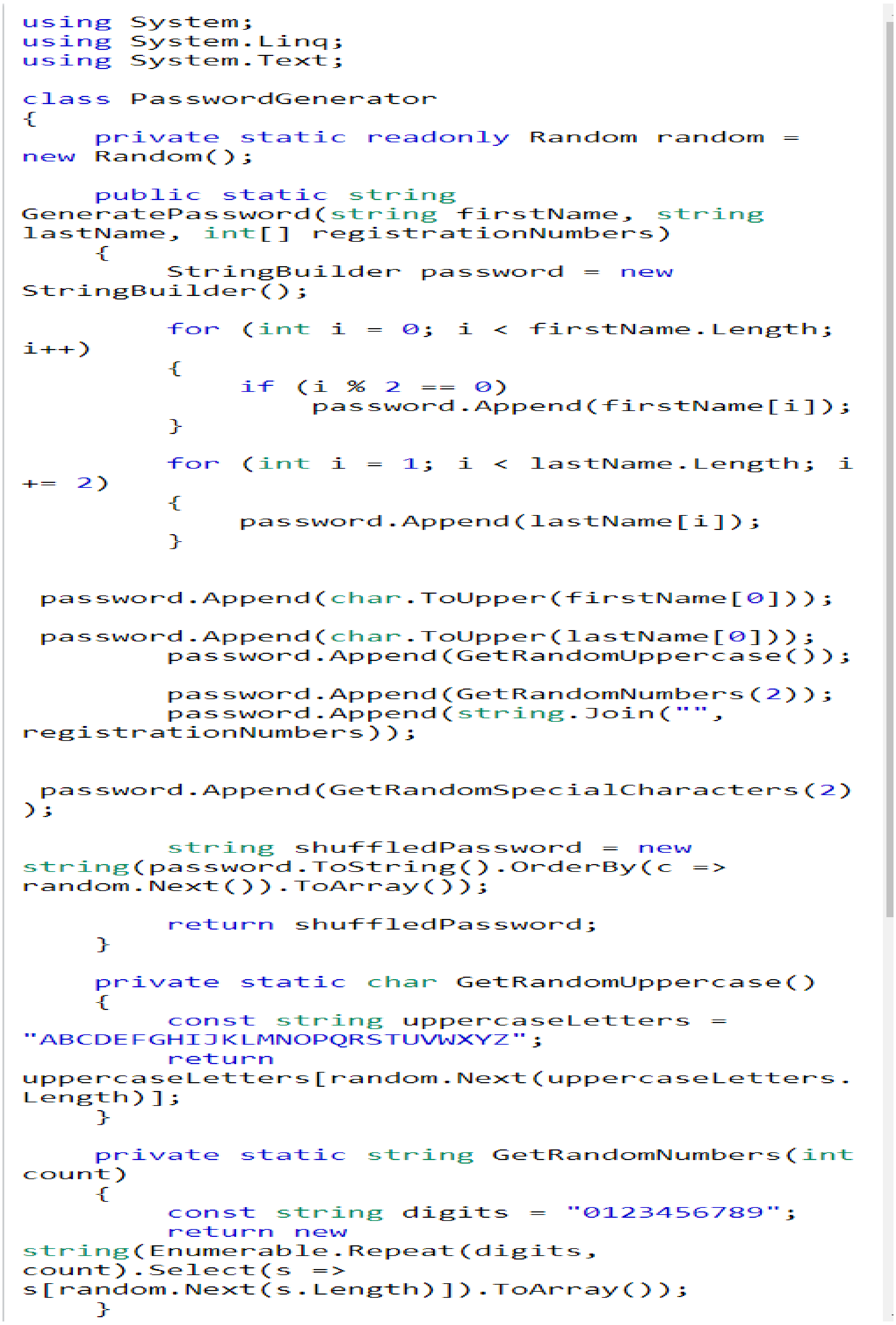
separately during matching, allowing access to specific parts of the matched text.

5. **Replacement**: The Regex class provides methods for replacing matched patterns in input strings with specified replacement text. It supports placeholders for referencing captured groups in the replacement text.
6. **Options and Modifiers**: Various options and modifiers can be applied to regex patterns to control matching behavior, such as case sensitivity, multiline mode, and single-line mode.
7. **Performance Optimization**: The library includes features for optimizing regex performance, such as compiled regex patterns that are cached for reuse across multiple matches.

Question 3: Make a Password generator according the following rules:

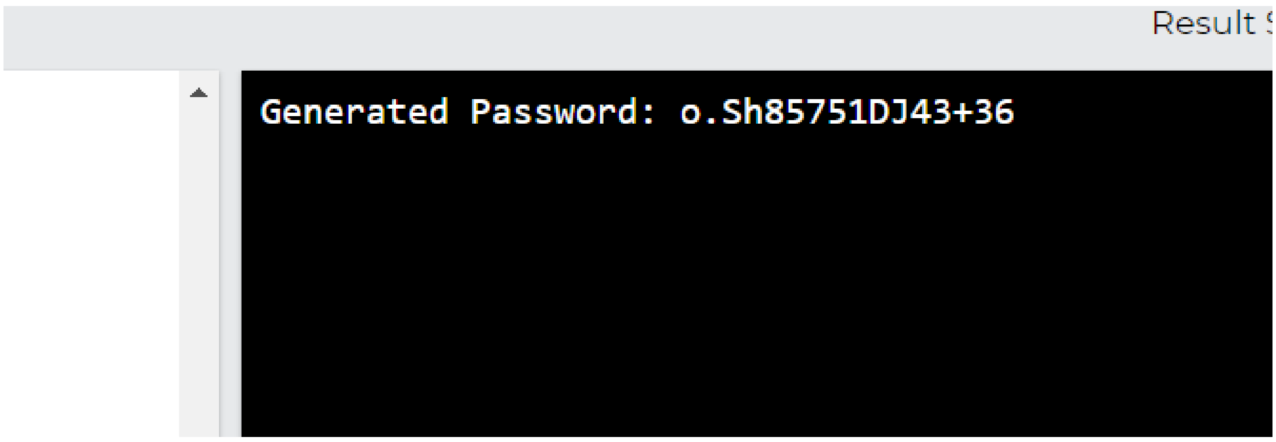
- Atleast one uppercase alphabet
- Atleast 4 numbers , two numbers must be your registration numbers
- Atleast 2 special characters
- Must contain initials of first and last name
- Must contain all odd letters of your first name.
- Must contain all even letters of your last name.

Answer:



```
using System;
using System.Linq;
using System.Text;

class PasswordGenerator
{
    private static readonly Random random =
new Random();

    public static string
GeneratePassword(string firstName, string
lastName, int[] registrationNumbers)
    {
        StringBuilder password = new
StringBuilder();

        for (int i = 0; i < firstName.Length;
i++)
        {
            if (i % 2 == 0)
                password.Append(firstName[i]);
        }

        for (int i = 1; i < lastName.Length; i
+= 2)
        {
            password.Append(lastName[i]);
        }


 password.Append(char.ToUpper(firstName[0]));

 password.Append(char.ToUpper(lastName[0]));
        password.Append(GetRandomUppercase());

        password.Append(GetRandomNumbers(2));
        password.Append(string.Join("",
registrationNumbers));


 password.Append(GetRandomSpecialCharacters(2)
);

        string shuffledPassword = new
string(password.ToString().OrderBy(c =>
random.Next()).ToArray());

        return shuffledPassword;
    }

    private static char GetRandomUppercase()
    {
        const string uppercaseLetters =
"ABCDEFGHIJKLMNOPQRSTUVWXYZ";
        return
uppercaseLetters[random.Next(uppercaseLetters.
Length)];
    }

    private static string GetRandomNumbers(int
count)
    {
        const string digits = "0123456789";
        return new
string(Enumerable.Repeat(digits,
count).Select(s =>
s[random.Next(s.Length)]).ToArray());
    }
string(Enumerable.Repeat(digits,
count).Select(s =>
s[random.Next(s.Length)]).ToArray());
    }

    private static string
GetRandomSpecialCharacters(int count)
    {
        const string specialCharacters =
"!@#$%^&*()-_=+[]{}|;:,.<>?";
        return new
string(Enumerable.Repeat(specialCharacters,
count).Select(s =>
s[random.Next(s.Length)]).ToArray());
    }
}

class Program
{
    static void Main(string[] args)
    {
        string firstName = "John";
        string lastName = "Doe";
        int[] registrationNumbers = { 1234,
5678 }; // Example registration numbers

        string password =
PasswordGenerator.GeneratePassword(firstName,
lastName, registrationNumbers);
        Console.WriteLine("Generated Password:
" + password);
    }
}
```

Output:



```
Generated Password: o.Sh85751DJ43+36
```

Code Of Question 3:

```
using System;

using System.Linq;

using System.Text;

class PasswordGenerator

{

    private static readonly Random random = new Random();

    public static string GeneratePassword(string firstName, string lastName, int[]
registrationNumbers)

    {

        StringBuilder password = new StringBuilder();

        for (int i = 0; i < firstName.Length; i++)

        {

            if (i % 2 == 0)

                password.Append(firstName[i]);

        }

        for (int i = 1; i < lastName.Length; i += 2)

        {

            password.Append(lastName[i]);

        }

        password.Append(char.ToUpper(firstName[0]));

        password.Append(char.ToUpper(lastName[0]));

        password.Append(GetRandomUppercase());


        password.Append(GetRandomNumbers(2));

        password.Append(string.Join("", registrationNumbers));

        password.Append(GetRandomSpecialCharacters(2));

        string shuffledPassword = new string(password.ToString().OrderBy(c =>
random.Next()).ToArray());

        return shuffledPassword;
    }

    private static char GetRandomUppercase()

    {

        const string uppercaseLetters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

        return uppercaseLetters[random.Next(uppercaseLetters.Length)];

    }

    private static string GetRandomNumbers(int count)

    {

        const string digits = "0123456789";

        return new string(Enumerable.Repeat(digits, count).Select(s =>
s[random.Next(s.Length)]).ToArray());

    }

    private static string GetRandomSpecialCharacters(int count)

    {

        const string specialCharacters = "!@#$%^&*()-_=+[]{}|;:,.<>?";

        return new string(Enumerable.Repeat(specialCharacters, count).Select(s =>
s[random.Next(s.Length)]).ToArray());

    }}

class Program

{

    static void Main(string[] args)

    {

        string firstName = "John";

        string lastName = "Doe";

        int[] registrationNumbers = { 1234, 5678 }; // Example registration numbers

        string password = PasswordGenerator.GeneratePassword(firstName, lastName,
registrationNumbers);

        Console.WriteLine("Generated Password: " + password);

    }

}
```

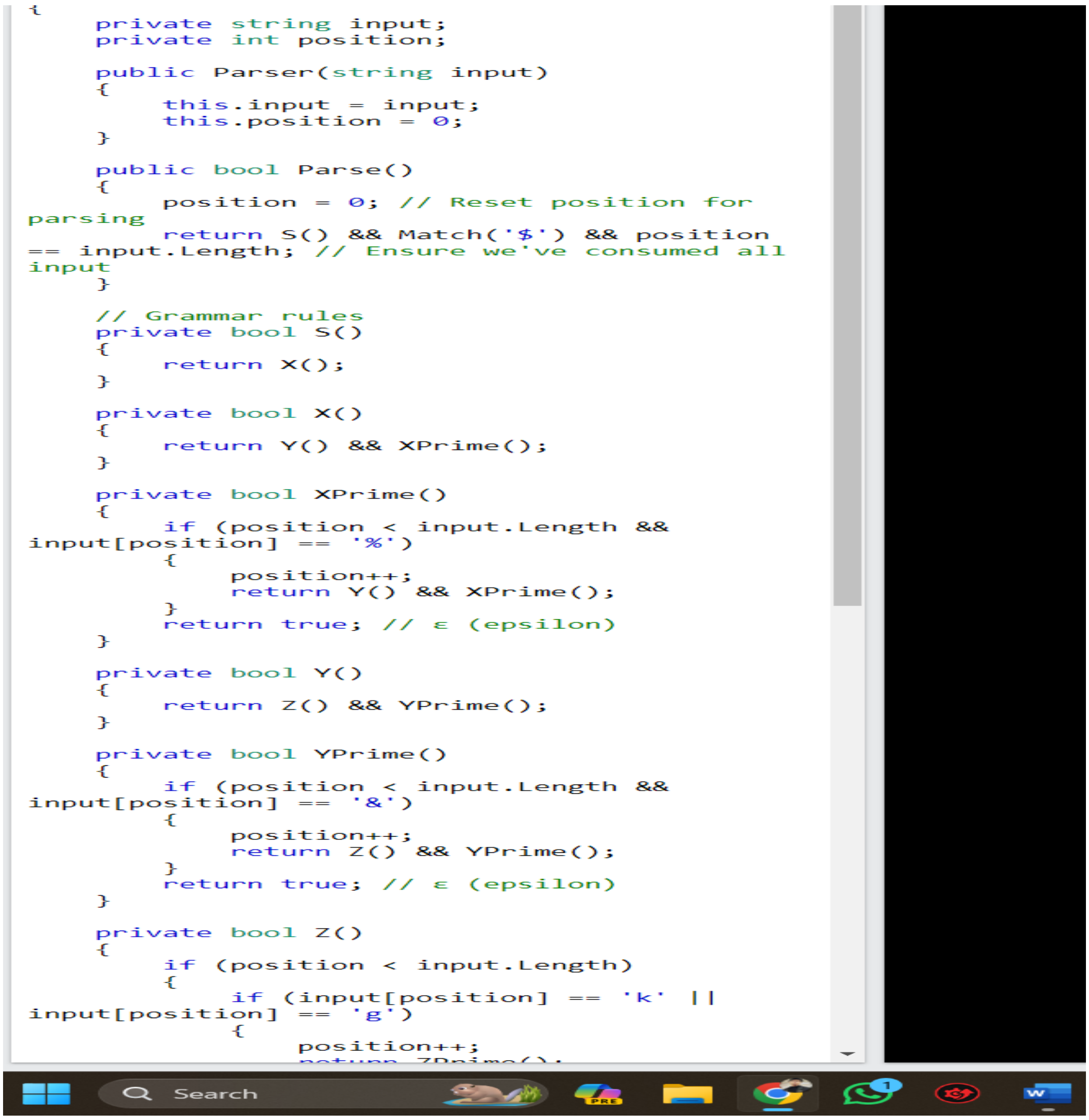
Question 2: Make recursive descent or LL1 parser or recursive descent parser for the following grammar:

S -> X$

X -> X % Y |Y

Y -> Y & Z |Z

Z -> k X k | g



```
{
        private string input;
        private int position;

        public Parser(string input)
        {
            this.input = input;
            this.position = 0;
        }

        public bool Parse()
        {
            position = 0; // Reset position for
parsing
            return S() && Match('$') && position
== input.Length; // Ensure we've consumed all
input
        }

        // Grammar rules
        private bool S()
        {
            return X();
        }

        private bool X()
        {
            return Y() && XPrime();
        }

        private bool XPrime()
        {
            if (position < input.Length &&
input[position] == '%')
            {
                position++;
                return Y() && XPrime();
            }
            return true; // ε (epsilon)
        }

        private bool Y()
        {
            return Z() && YPrime();
        }

        private bool YPrime()
        {
            if (position < input.Length &&
input[position] == '&')
            {
                position++;
                return Z() && YPrime();
            }
            return true; // ε (epsilon)
        }

        private bool Z()
        {
            if (position < input.Length)
            {
                if (input[position] == 'k' ||
input[position] == 'g')
                {
                    position++;
                    return ZPrime();
        {
            if (input[position] == 'k' ||
input[position] == 'g')
            {
                position++;
                return ZPrime();
            }
        }
        return false;
    }

    private bool ZPrime()
    {
        if (position < input.Length &&
input[position] == 'k')
        {
            position++;
            if (X() && input[position] == 'k')
            {
                position++;
                return ZPrime();
            }
            return false;
        }
        return true; // ε (epsilon)
    }

    // Helper function
    private bool Match(char expected)
    {
        if (position < input.Length &&
input[position] == expected)
        {
            position++;
            return true;
        }
        return false;
    }
}

class Program
{
    static void Main(string[] args)
    {
        string input = "g&g$";
        Parser parser = new Parser(input);
        if (parser.Parse())
            Console.WriteLine("Input is valid
according to the grammar.");
        else
            Console.WriteLine("Input is not
valid according to the grammar.");
    }
}
```

Code:

```
using System;
class Parser
{
    private string input;
    private int position;
    public Parser(string input)
    {
        this.input = input;
        this.position = 0;
    }

    public bool Parse()
    {
        position = 0; // Reset position for parsing
        return S() && Match('$') && position == input.Length; // Ensure
we've consumed all input
    }
    private bool S()
    {
        return X();
    }

    private bool X()
    {
```

```
    return Y() && XPrime();
}
private bool XPrime()
{
    if (position < input.Length && input[position] == '%')
    {
        position++;
        return Y() && XPrime();
    }
    return true; // ε (epsilon)
}

private bool Y()
{
    return Z() && YPrime();
}
private bool YPrime()
{
    if (position < input.Length && input[position] == '&')
    {
        position++;
        return Z() && YPrime();
    }
    return true; // ε (epsilon)
}
```

```
private bool Z()

{

   if (position < input.Length)

   {

      if (input[position] == 'k' || input[position] == 'g')

      {

         position++;

         return ZPrime();

      }

   }

   return false;

}


private bool ZPrime()

{

   if (position < input.Length && input[position] == 'k')

   {

      position++;

      if (X() && input[position] == 'k')

      {

         position++;

         return ZPrime();

      }

      return false;
```

```
        }
        return true; // ε (epsilon)
    }
    private bool Match(char expected)
    {
        if (position < input.Length && input[position] == expected)
        {
            position++;
            return true;
        }
        return false;
    }
}
class Program
{
    static void Main(string[] args)
    {
        string input = "g&g$";
        Parser parser = new Parser(input);
        if (parser.Parse())
            Console.WriteLine("Input is valid according to the grammar.");
        else
            Console.WriteLine("Input is not valid according to the grammar.");
    }}
```