COMPILER CONSTRUCTION (CSC 441)

LAB MID TERM

QUESTION NO 1:

Briefly describe the regex library of C#.

ANS:

The System.Text.RegularExpressions namespace in C# provides a powerful framework for working with regular expressions. Regular expressions (regex) are patterns used to match character combinations in strings.

Functionality of the Regex Class:

The Regex class provides methods for working with regular expressions in C#. Some of the key methods include:

IsMatch:

Checks if a regular expression matches a specified input string.

Match:

Searches an input string for a substring that matches a regular expression pattern. Returns the first match found.

Matches:

Searches an input string for all occurrences of a substring that matches a regular expression pattern. Returns a collection of matches.

Replace:

Replaces all occurrences of a specified regular expression pattern with a specified replacement string.

Split:

Splits an input string into an array of substrings based on a regular expression pattern.

Common Patterns:

- \d: Matches any digit (0-9).
- \w: Matches any word character (alphanumeric + underscore).
- \s: Matches any whitespace character (spaces, tabs, line breaks).
- . Matches any character except a newline.

Quantifiers:

- *: Matches 0 or more occurrences of the preceding pattern.
- +: Matches 1 or more occurrences of the preceding pattern.
- ?: Matches 0 or 1 occurrence of the preceding pattern.

Character Classes:

- [abc]: Matches any single character a, b, or c.
- [^abc]: Matches any single character except a, b, or c.
- [a-z]: Matches any lowercase letter.
- [0-9]: Matches any digit.

Anchors:

- ^ (Caret): Matches the start of a string.
- $ (Dollar Sign): Matches the end of a string.

QUESTION NO 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

Eliminating Left Recursion:

S -> X$

X -> YX'

X' -> %YX' | e

Y -> Z Y'

Y' -> & Z Y' | e

Z -> k X k | g

CODE:

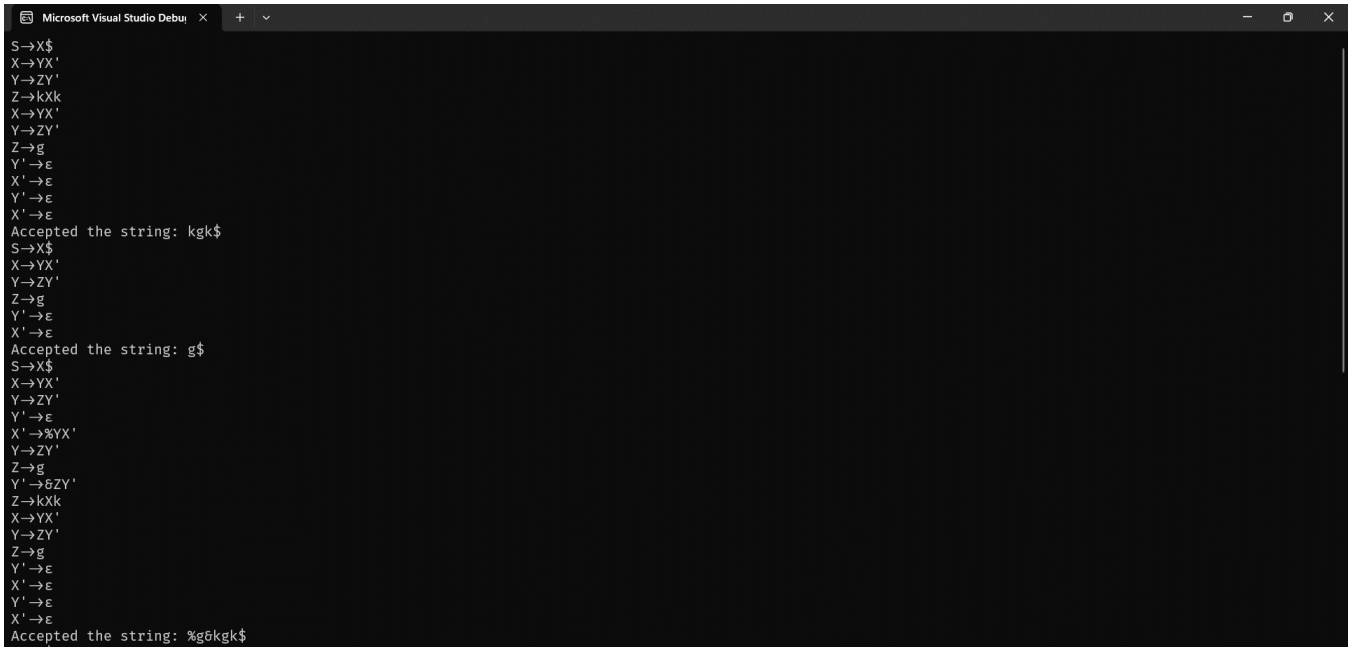
```
using System;

class MainClass
{
    // Global variables
    static int count = 0; // Counter to keep track of the position in the expression
string
    static string expr;   // Input expression

    // Main method
    static void Main(string[] args)
    {
        string[] arr = { "kgk", "g", "%g&kgk", "gkg", "ggk", "%&&k" };
```

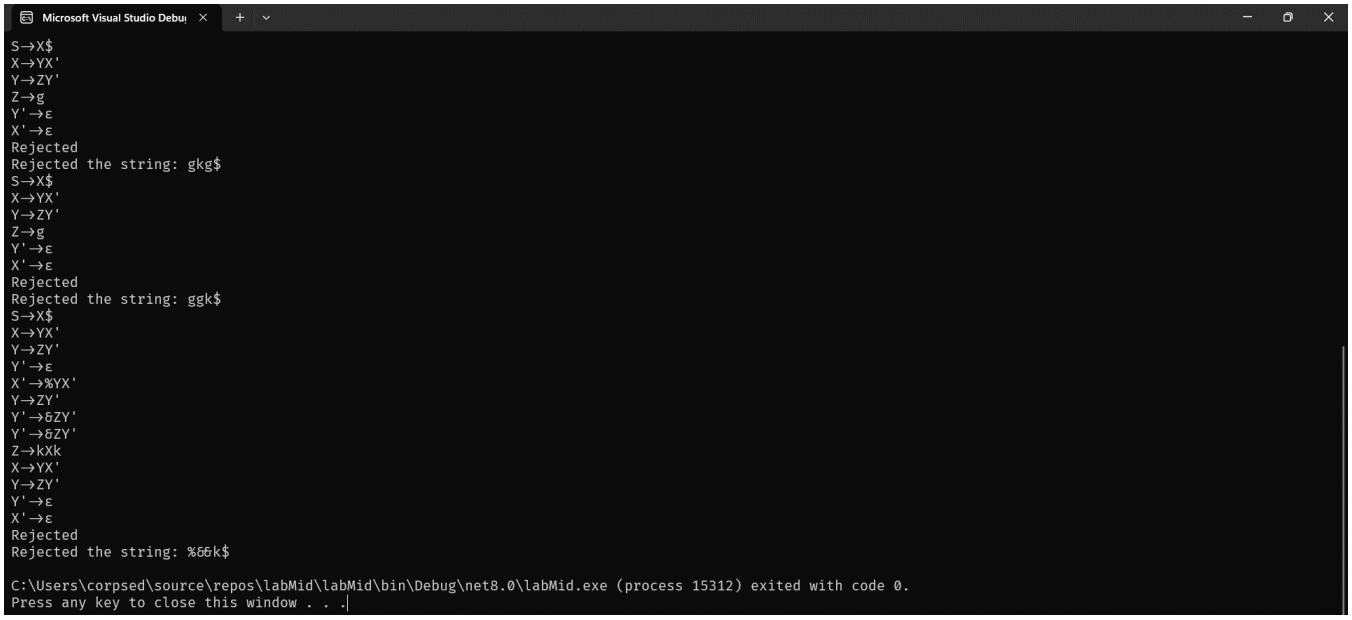
```
        // Append "$" to the end of the expression to indicate the end
        foreach (string s in arr)
        {
            expr = s;
            expr += "$";

            try
            {
                S();
            } catch (Exception)
            {
                Console.WriteLine("Rejected");
            } finally
            {
                // Check if the entire expression has been parsed
                Console.Write(expr.Length == count ? "Accepted" : "Rejected");
                Console.WriteLine(" the string: " + expr);
                count = 0;
            }
        }
    }

    // Parse S
    static void S()
    {
        Console.WriteLine("S->X$");
        X();
        Validate('$');
    }

    // Parse X
    static void X()
    {
        Console.WriteLine("X->YX'");
        Y();
        Xp();
    }

    // Parse X prime
    static void Xp()
    {
        if (MatchAndMove('%'))
        {
            Console.WriteLine("X'->%YX'");
```

```
            Y();
            Xp();
        }
        else
        {
            Console.WriteLine("X'->ε");
        }
    }

    // Parse Y
    static void Y()
    {
        Console.WriteLine("Y->ZY'");
        Z();
        Yp();
    }

    // Parse Y prime
    static void Yp()
    {
        if (MatchAndMove('&'))
        {
            Console.WriteLine("Y'->&ZY'");
            Z();
            Yp();
        }
        else
        {
            Console.WriteLine("Y'->ε");
        }
    }

    // Parse Z
    static void Z()
    {
        if (MatchAndMove('k'))
        {
            Console.WriteLine("Z->kXk");
            X();
            Validate('k');
        }
        else if (MatchAndMove('g'))
        {
            Console.WriteLine("Z->g");
```

```
        }
    }

    // Helper function to check if the next character in the expression matches the
expected character and move the counter
    static bool MatchAndMove(char expected)
    {
        if (expr[count] == expected)
        {
            count++;
            return true;
        }
        return false;
    }

    // Helper function to validate if the next character in the expression matches the
expected character, otherwise reject
    static void Validate(char expected)
    {
        if (!MatchAndMove(expected))
        {
            throw new Exception("Rejected");
        }
    }
}
```

OUTPUT:





QUESTION NO 3:

Make a Password generator according the following rules:

(a) Atleast one uppercase alphabet

(b) Atleast 4 numbers , two numbers must be your registration numbers

(c) Atleast 2 special characters

(d) Must contain initials of first and last name

(e) Must contain all odd letters of your first name.

(f) Must contain all even letters of your last name.

(g) maximum length of 16

ANS:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace labMid
{
    internal class PasswordGenerator
    {
        public static string GeneratePassword(string registrationNumber, string
firstName, string lastName)
        {
            if (registrationNumber.Length < 2)
            {
                return null;
            }

            Random random = new Random();

            int positionOfNum1 = random.Next(0, 4);
            int positionOfNum2 = random.Next(0, 4);
            while (positionOfNum1 == positionOfNum2)
            {
                positionOfNum2 = random.Next(0, 5);
            }
            string password = "";

            string lastTwoDigits =
registrationNumber.Substring(registrationNumber.Length - 2);

            int nums = random.Next(4, 8);
            for (int i = 1; i <= nums; i++)
            {
                if (i == positionOfNum1)
                {
                    password += lastTwoDigits[0];
                }
                else if (i == positionOfNum2)
                {
                    password += lastTwoDigits[1];
                }
                else if (i == 3)
                {
                    password += "1";
                }
                else if (i == 4)
                {
                    password += "2";
```

```
                }
                else
                {
                    password += random.Next(0, 10).ToString();
                }
            }

            string specialCharacters = "!@#$%^&*()_-+=<>?";
            int numberOfSpecialChars = random.Next(2, 5);
            for (int i = 1; i <= numberOfSpecialChars; i++)
            {
                password += specialCharacters[random.Next(0,
specialCharacters.Length)];
            }

            int chars = random.Next(1, 4);
            password += firstName[0];
            for (int i = 1; i < firstName.Length; i++)
            {
                if (i % 2 != 0)
                {
                    password += firstName[i];
                }
            }
            password += lastName[0];

            for (int i = 1; i < lastName.Length; i++)
            {
                if (i % 2 == 0)
                {
                    password += lastName[i];
                }
            }

            if (password.Length > 16)
            {
                password = password.Substring(0, 16);
            }

            return ShufflePassword(password);
        }

        static string ShufflePassword(string input)
        {
```

```
            char[] characters = input.ToCharArray();
            Random random = new Random();

            for (int i = characters.Length - 1; i > 0; i--)
            {
                int j = random.Next(0, i + 1);
                char temp = characters[i];
                characters[i] = characters[j];
                characters[j] = temp;
            }

            return new string(characters);
        }

    public static void Main(string[] args)
    {
            string pass = GeneratePassword("sp21-bcs-009", "hamid", "ali");
            Console.WriteLine(pass);
    }
    }
}
```