ATTOCK CAMPUS

LAB MID

Submitted By	**:**	**Sadia Bibi**
Registration No.	**:**	**SP21-BCS-027**
Submitted To	**:**	**Sir Bilal Bukhari**
Course	**:**	**Compiler construction**

Date: 7th April, 2024

QUESTION NO 1:

Briefly describe the regex library of C# .

Regular Expressions:

A regular expression (or regex) is a sequence of characters that defines a search pattern. This search pattern can then find specific text strings within a larger string. Regexes are commonly used in programming languages such as C#, Java, JavaScript, and Python to perform tasks such as data validation, string manipulation, and text processing.

C# Regular Expressions Library:

The .NET C# Regular Expressions Library is a library of classes, methods, and objects that allows developers to create and use regular expressions easily. It is included in the .NET Framework and enables developers to write more efficient code when working with strings.

The .NET C# Regular Expressions Library can be used for various tasks. Some of the most common uses include:

- **Data validation -** You can use regexes to verify that user-supplied data is valid and conforms to the required format.
- **String manipulation -** You can use regexes to extract certain parts of a string, replace parts, or even split a string into multiple parts.
- **Text processing -** You can use regexes to search for specific patterns in a text document or manipulate the text itself.

How to Use the .NET C# Regular Expressions Library:

- Using the .NET C# Regular Expressions Library is relatively easy. To get started, you will need to first add a reference to the **System.Text.RegularExpressions** namespace. This can be done by adding the following line of code at the top of your source file:

  using System.Text.RegularExpressions;
- Once you have added the namespace reference, you can create regular expressions. For example, you want to create a regex that checks whether a string contains only alphabetic characters. You can do so by using the following line of code:

  Regex regex = new Regex("^[a-zA-Z]+$");
- Once you have created your regex, you can use it to test whether a given string matches your criteria. To do this, you can use the IsMatch method. For example:

  bool isValid = regex.IsMatch("abc123");
- In this case, the IsMatch method will return false because the string "abc123" contains numbers and letters.

Key features of the regex library in C# :

- **Regex Class:** The Regex class is the main class in the library. It represents a compiled regular expression pattern and provides methods for pattern matching operations such as searching, replacing, splitting, etc.
- **Regular Expression Patterns:** Developers can define regular expression patterns using a rich syntax to describe text patterns they want to match. This syntax includes metacharacters, quantifiers, character classes, groups, anchors, and more.
- **Match Objects:** The library provides Match objects to represent a single match of a regular expression pattern in a string. These objects contain information about the matched substring, captured groups, positions, etc.

- **MatchCollection:** For patterns that can have multiple matches within a string, the library offers MatchCollection objects. These collections hold multiple Match objects, each representing a separate match.
- **Replacement Strings:** In addition to searching for matches, the library allows developers to replace matched substrings with specified replacement strings using methods like Regex.Replace().
- **Options and Modifiers:** Developers can apply various options and modifiers to regex patterns to control matching behavior. Options include case sensitivity, single-line mode, and more.
- **Error Handling:** The library provides mechanisms for handling errors that may occur during regular expression operations, such as invalid patterns or unexpected behavior.

Overall, the regex library in C# provides a powerful and flexible way to work with text patterns, making it suitable for tasks such as text processing, data validation, parsing, and more.

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

After removing the left recursion, the following is the grammar:

S -> X$

X -> Y X'

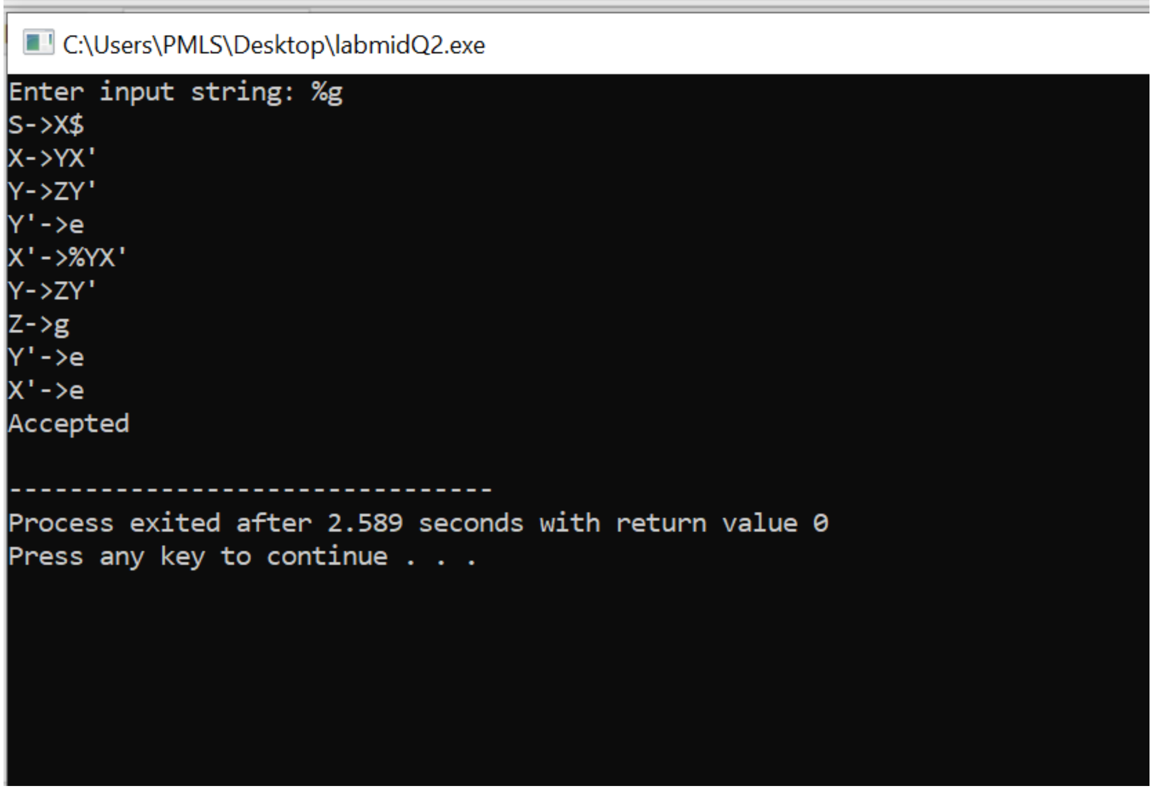
X' -> % Y X' | ε

Y -> Z Y'

Y' -> & Z Y' | ε

Z -> k X k | g

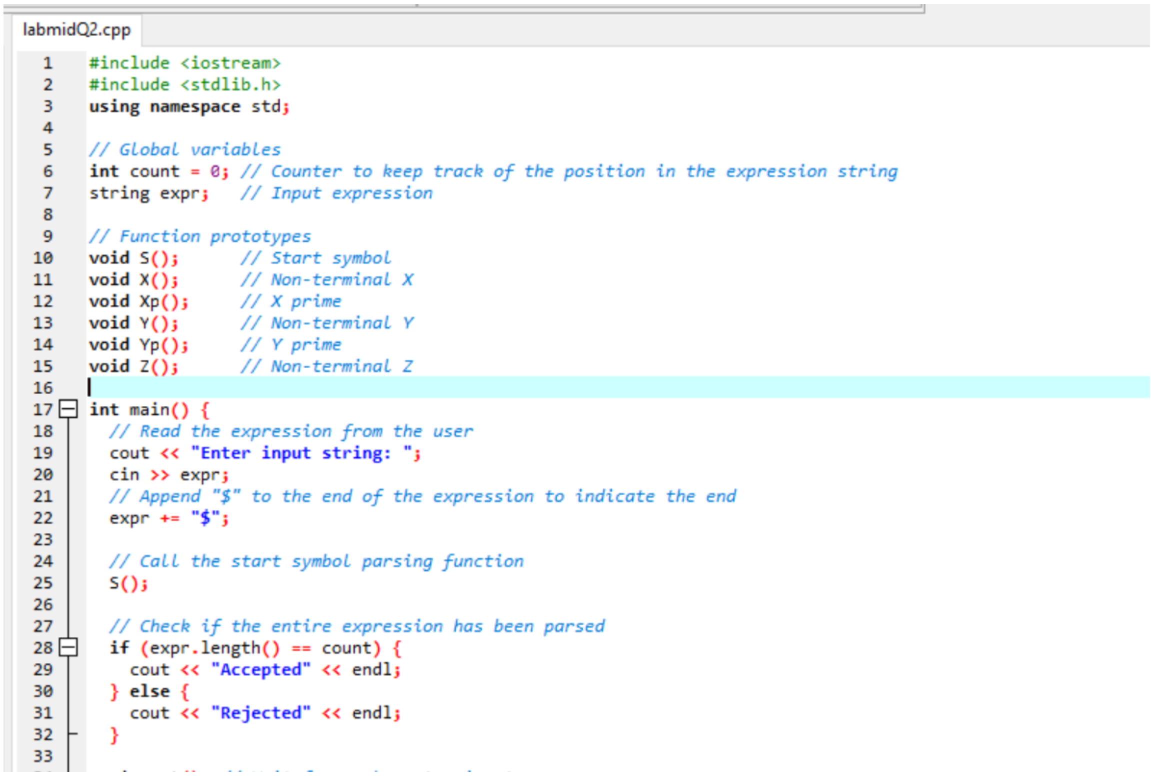
Output:



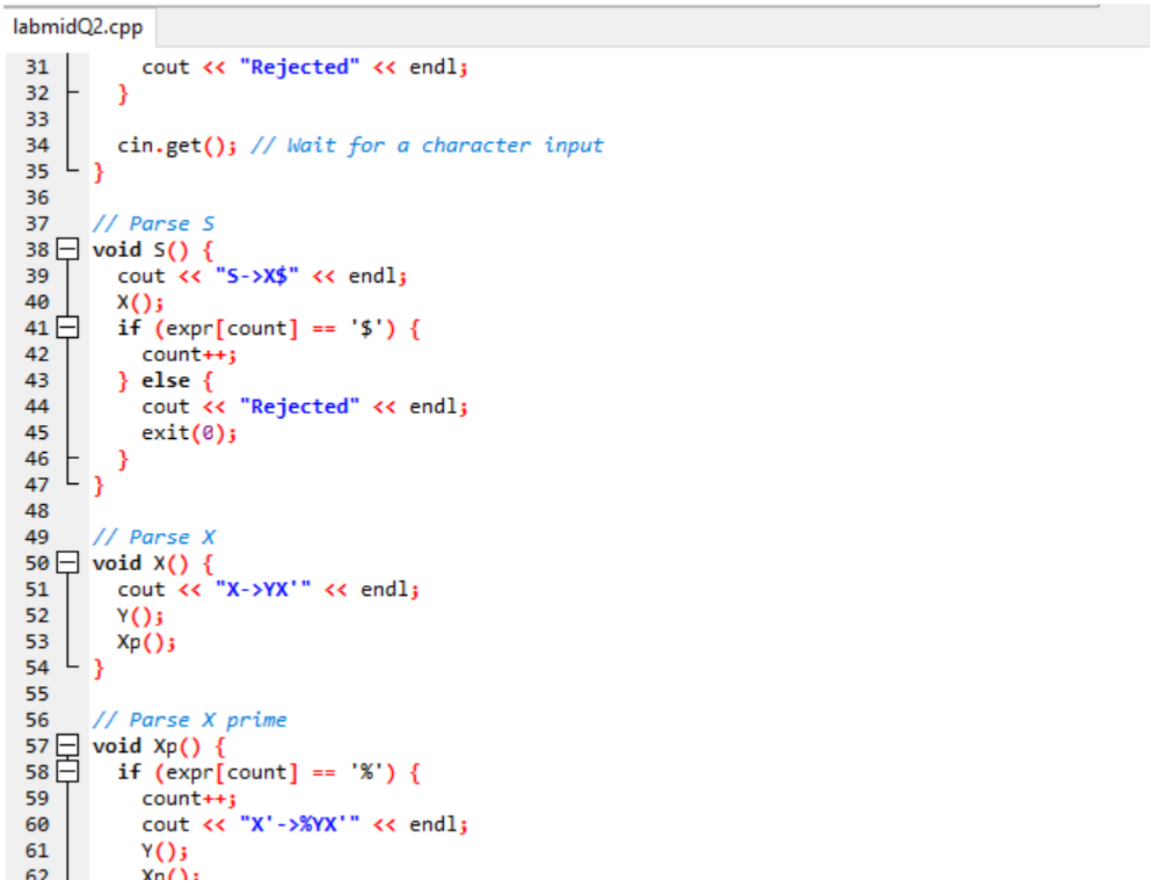
C:\Users\PMLS\Desktop\labmidQ2.exe

```
Enter input string: %g
S->X$
X->YX'
Y->ZY'
Y'->e
X'->%YX'
Y->ZY'
Z->g
Y'->e
X'->e
Accepted

--------------------------------
Process exited after 2.589 seconds with return value 0
Press any key to continue . . .
```

Code:

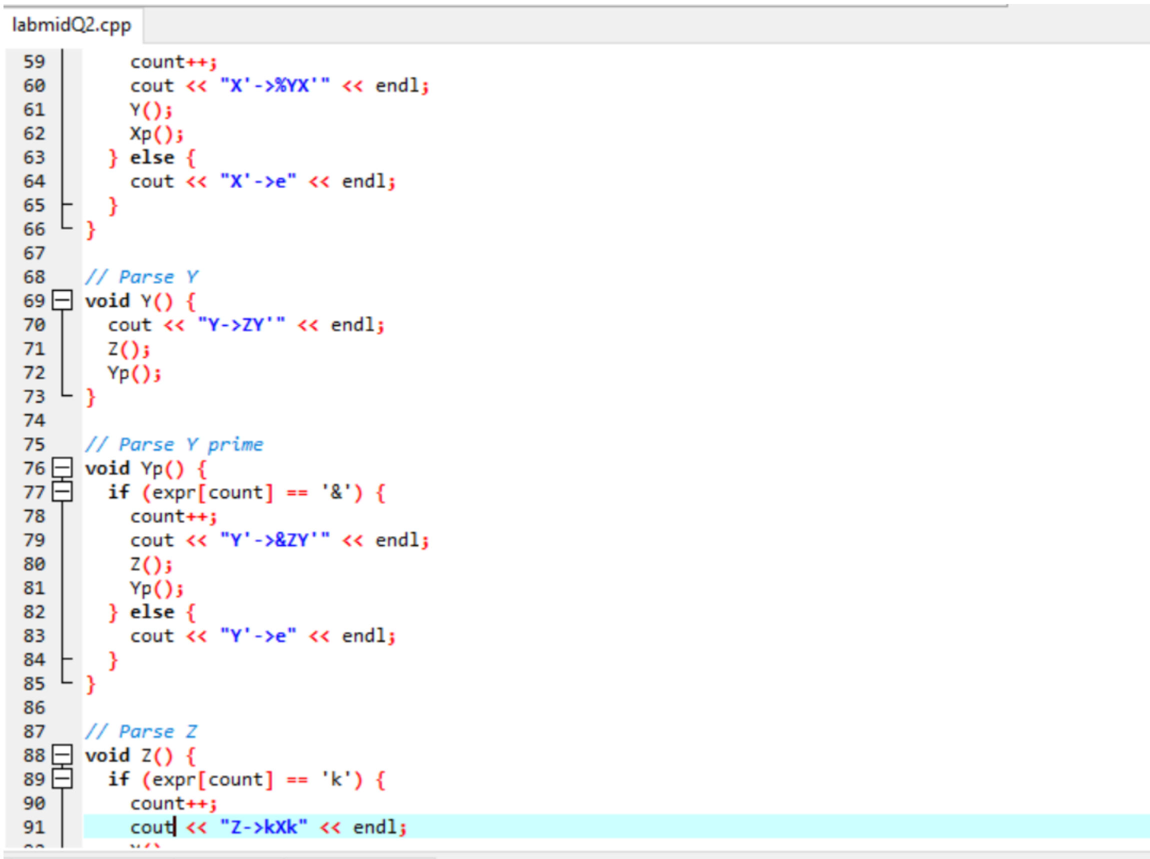


```cpp
#include <iostream>
#include <stdlib.h>
using namespace std;

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

// Function prototypes
void S();      // Start symbol
void X();      // Non-terminal X
void Xp();     // X prime
void Y();      // Non-terminal Y
void Yp();     // Y prime
void Z();      // Non-terminal Z

int main() {
  // Read the expression from the user
  cout << "Enter input string: ";
  cin >> expr;
  // Append "$" to the end of the expression to indicate the end
  expr += "$";

  // Call the start symbol parsing function
  S();

  // Check if the entire expression has been parsed
  if (expr.length() == count) {
    cout << "Accepted" << endl;
  } else {
    cout << "Rejected" << endl;
  }

```

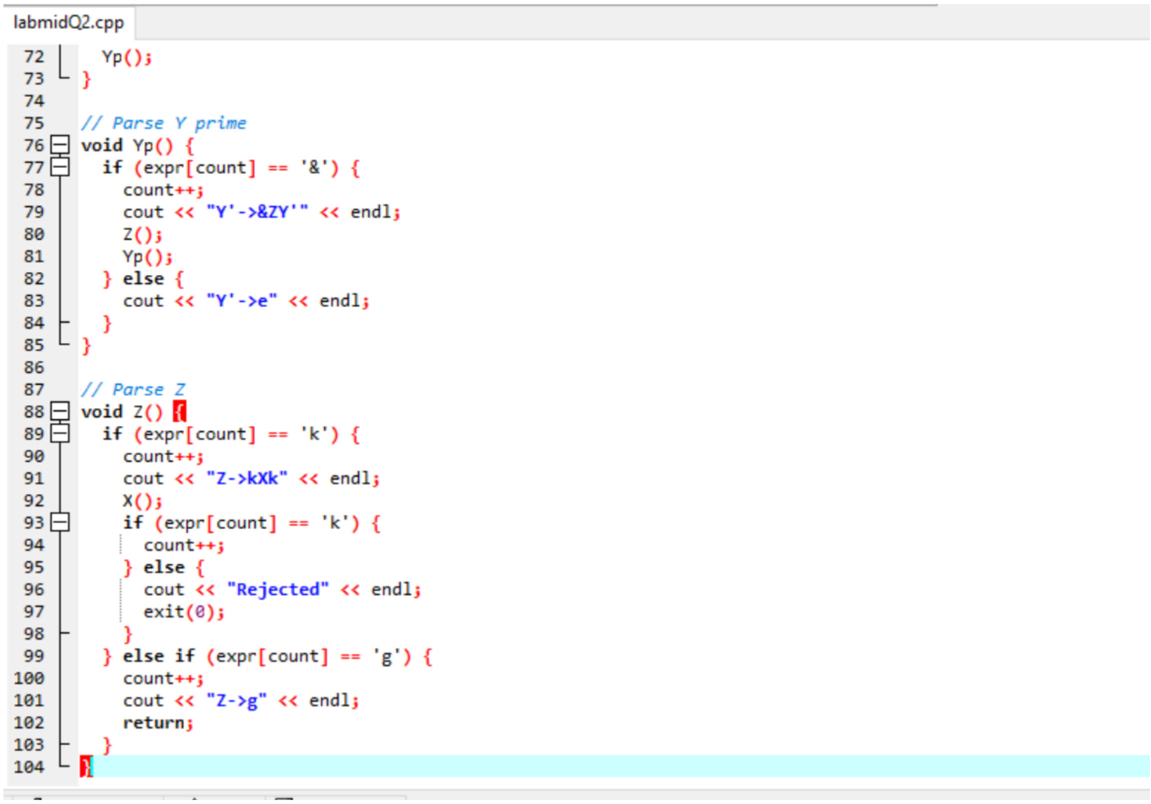


labmidQ2.cpp

```cpp
        cout << "Rejected" << endl;
    }

    cin.get(); // Wait for a character input
}

// Parse S
void S() {
    cout << "S->X$" << endl;
    X();
    if (expr[count] == '$') {
        count++;
    } else {
        cout << "Rejected" << endl;
        exit(0);
    }
}

// Parse X
void X() {
    cout << "X->YX'" << endl;
    Y();
    Xp();
}

// Parse X prime
void Xp() {
    if (expr[count] == '%') {
        count++;
        cout << "X'->%YX'" << endl;
        Y();
        Xp();
```



```cpp
    count++;
    cout << "X'->%YX'" << endl;
    Y();
    Xp();
  } else {
    cout << "X'->e" << endl;
  }
}

// Parse Y
void Y() {
  cout << "Y->ZY'" << endl;
  Z();
  Yp();
}

// Parse Y prime
void Yp() {
  if (expr[count] == '&') {
    count++;
    cout << "Y'->&ZY'" << endl;
    Z();
    Yp();
  } else {
    cout << "Y'->e" << endl;
  }
}

// Parse Z
void Z() {
  if (expr[count] == 'k') {
    count++;
    cout << "Z->kXk" << endl;
```



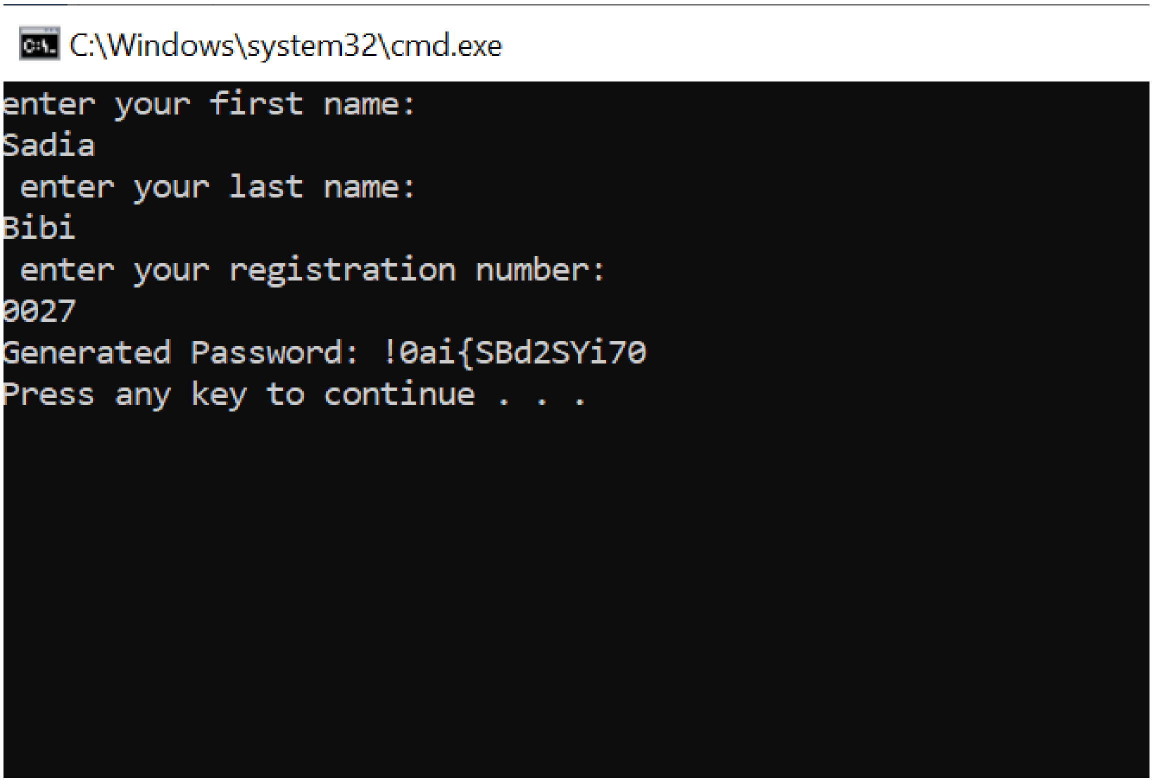
```
    Yp();
}

// Parse Y prime
void Yp() {
    if (expr[count] == '&') {
        count++;
        cout << "Y'->&ZY'" << endl;
        Z();
        Yp();
    } else {
        cout << "Y'->e" << endl;
    }
}

// Parse Z
void Z() {
    if (expr[count] == 'k') {
        count++;
        cout << "Z->kXk" << endl;
        X();
        if (expr[count] == 'k') {
            count++;
        } else {
            cout << "Rejected" << endl;
            exit(0);
        }
    } else if (expr[count] == 'g') {
        count++;
        cout << "Z->g" << endl;
        return;
    }
}
```

QUESTION NO 3:

Make a Password generator according the following rules:

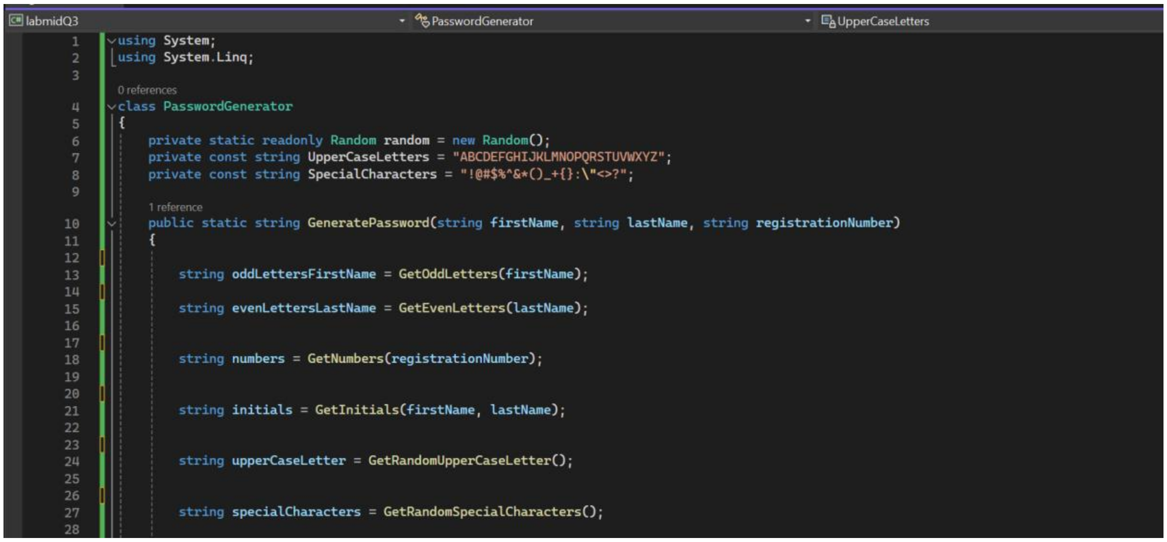
a) **Atleast one uppercase alphabet**
b) **Atleast 4 numbers , two numbers must be your registration numbers**
c) **Atleast 2 special characters**
d) **Must contain initials of first and last name**
e) **Must contain all odd letters of your first name.**
f) **Must contain all even letters of your last name.**
g) **maximum length of 16**

Output:

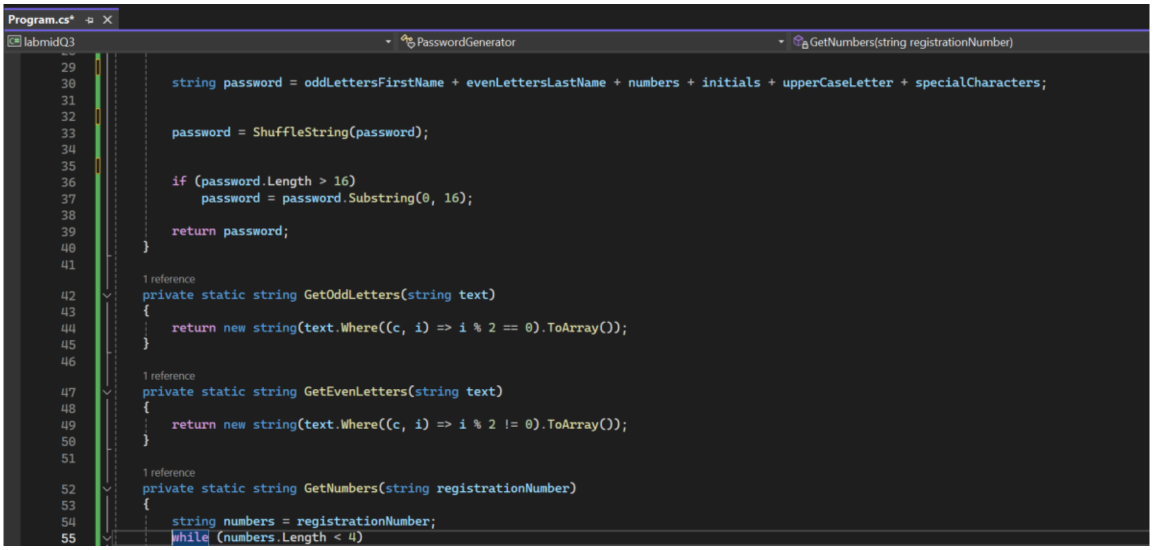


```
enter your first name:
Sadia
 enter your last name:
Bibi
 enter your registration number:
0027
Generated Password: !0ai{SBd2SYi70
Press any key to continue . . .
```

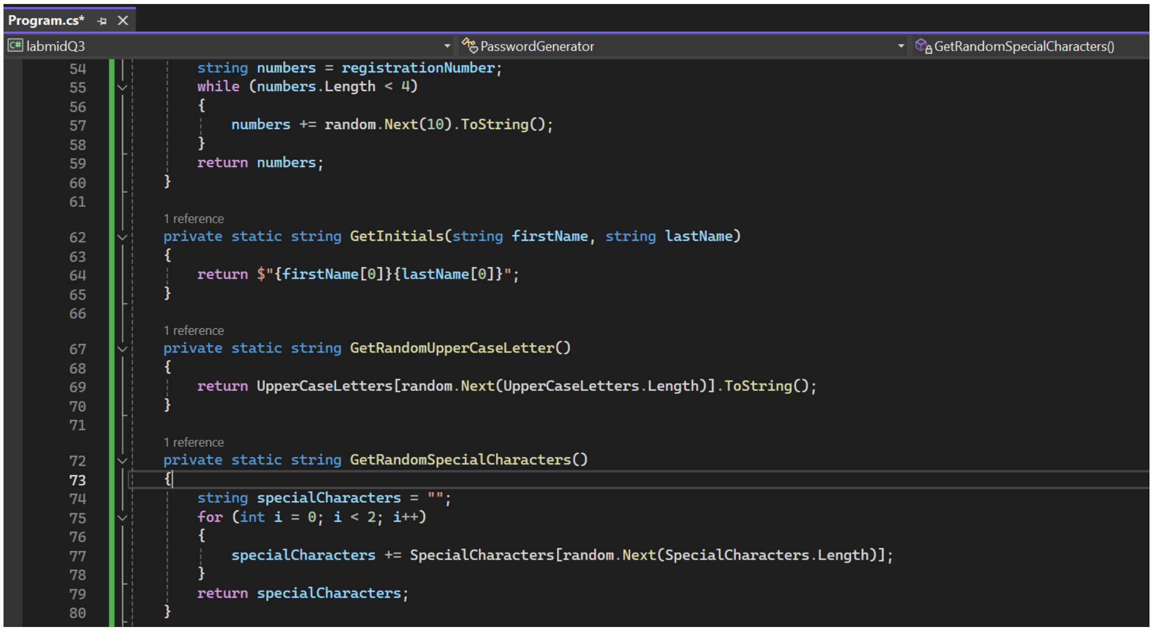
Code:



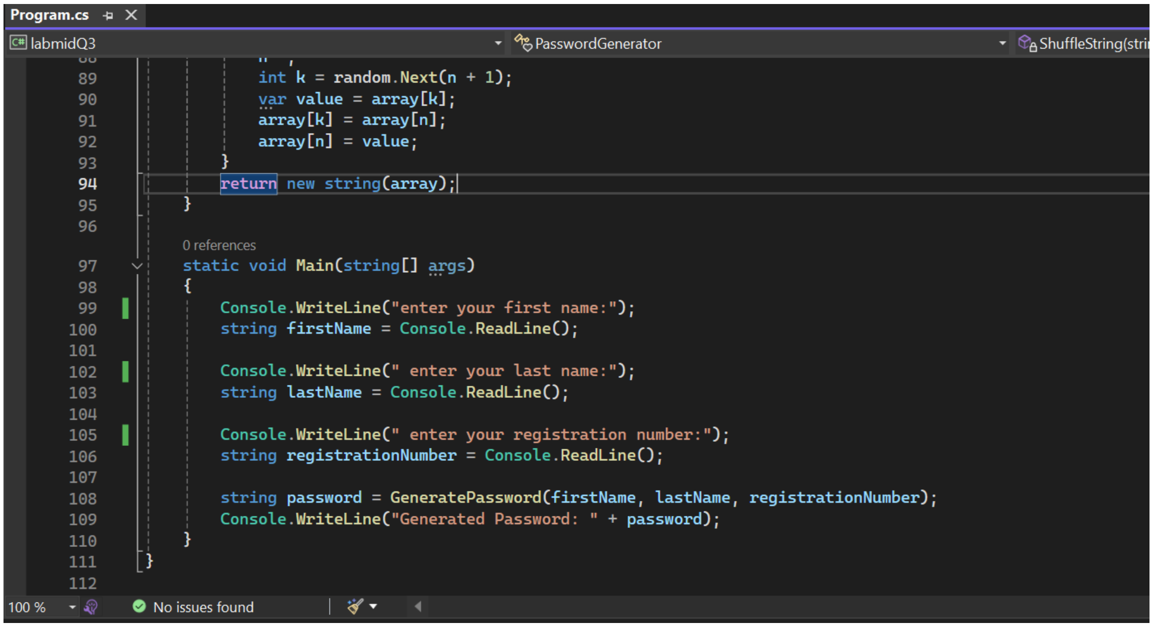
```
using System;
using System.Linq;

class PasswordGenerator
{
    private static readonly Random random = new Random();
    private const string UpperCaseLetters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    private const string SpecialCharacters = "!@#$%^&*()_+{}:\"<>?";

    public static string GeneratePassword(string firstName, string lastName, string registrationNumber)
    {

        string oddLettersFirstName = GetOddLetters(firstName);

        string evenLettersLastName = GetEvenLetters(lastName);


        string numbers = GetNumbers(registrationNumber);


        string initials = GetInitials(firstName, lastName);


        string upperCaseLetter = GetRandomUpperCaseLetter();


        string specialCharacters = GetRandomSpecialCharacters();

```



```
        string password = oddLettersFirstName + evenLettersLastName + numbers + initials + upperCaseLetter + specialCharacters;


        password = ShuffleString(password);


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

        return password;
    }

    1 reference
    private static string GetOddLetters(string text)
    {
        return new string(text.Where((c, i) => i % 2 == 0).ToArray());
    }

    1 reference
    private static string GetEvenLetters(string text)
    {
        return new string(text.Where((c, i) => i % 2 != 0).ToArray());
    }

    1 reference
    private static string GetNumbers(string registrationNumber)
    {
        string numbers = registrationNumber;
        while (numbers.Length < 4)
```



```
        string numbers = registrationNumber;
        while (numbers.Length < 4)
        {
            numbers += random.Next(10).ToString();
        }
        return numbers;
    }

    1 reference
    private static string GetInitials(string firstName, string lastName)
    {
        return $"{firstName[0]}{lastName[0]}";
    }

    1 reference
    private static string GetRandomUpperCaseLetter()
    {
        return UpperCaseLetters[random.Next(UpperCaseLetters.Length)].ToString();
    }

    1 reference
    private static string GetRandomSpecialCharacters()
    {
        string specialCharacters = "";
        for (int i = 0; i < 2; i++)
        {
            specialCharacters += SpecialCharacters[random.Next(SpecialCharacters.Length)];
        }
        return specialCharacters;
    }
```



```
            n--;
            int k = random.Next(n + 1);
            var value = array[k];
            array[k] = array[n];
            array[n] = value;
        }
        return new string(array);
    }

    static void Main(string[] args)
    {
        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 registrationNumber = Console.ReadLine();

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