**Lab mid**

**COMSATS UNIVERSITY ISLAMABAD**

**Logo, company name

Description automatically generated**

**ATTOCK CAMPUS**

**Submitted By**

Raja Muhammad Zeeshan (SP21-BCS-025)

**Submitted To**

**Mr Syed Bilal haider**

**Course Title**

Compiler construction

**Date**

7-4-2024

**QUESTION NO 1:**

Briefly describe the regex library of C#

Ans The regex library in C# is a powerful tool for working with text patterns. It's part of the .NET Framework and is found in the System.Text.RegularExpressions namespace. Here's a breakdown of its key components and functionalities:

**Regex Class:**

This is the primary class used for working with regular expressions in C#. You can create a Regex object by passing a pattern string to its constructor. This class provides methods for matching patterns against input strings, searching for multiple matches within a string, and replacing matched substrings with specified replacements.

**Pattern Syntax:**

C# regex patterns follow the standard syntax defined by the regular expression specification. This syntax allows you to define complex patterns to match specific sequences of characters in text. It includes metacharacters like ^, $, ., \*, +, ?, \, and many more, which have special meanings within the pattern.

**Match Class:**

When you use a Regex object to search for matches within a string, it returns instances of the Match class. Each Match object represents a single match found in the input string. You can access properties of the Match object to get information about the matched substring, its position in the input string, and any captured groups within the match.

**MatchCollection Class:**

If you're searching for multiple occurrences of a pattern within a string, the Regex class provides methods that return a MatchCollection object. This collection contains all the matches found in the input string. You can iterate over the MatchCollection to process each match individually.

**Grouping and Capturing:**

Regular expressions in C# support grouping and capturing of substrings within a match. You can use parentheses () to define groups within your pattern. These groups can be used to extract specific parts of a match or to apply quantifiers and modifiers to a group of characters.

**Replacement Operations**

: Along with searching for matches, the regex library also supports replacing matched substrings with specified replacement strings. You can use the Regex.Replace() method to perform these replacement operations. Additionally, you can use captured groups in the replacement string to dynamically generate replacements based on the content of the matched substring.

**Options and Modifiers:**

C# regex patterns can include options and modifiers that affect how the pattern is interpreted and applied. For example, you can use the IgnoreCase option to perform case-insensitive matching, or the Singleline option to make the dot . character match newline characters as well

**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

**Code**

using System;

class RecursiveDescentParser

{

static string input;

static int position;

static void Main()

{

Console.WriteLine("Enter the input string:");

input = Console.ReadLine();

position = 0;

Console.WriteLine("\nInput Action");

Console.WriteLine("---------------------");

if (S() && position == input.Length - 1)

Console.WriteLine("---------------------\nString is successfully parsed.");

else

Console.WriteLine("---------------------\nError in parsing string.");

}

static bool S()

{

Console.WriteLine("{0,-15} S -> X$", input[position]);

if (X() && position == input.Length - 1 && input[position] == '$')

return true;

return false;

}

static bool X()

{

Console.WriteLine("{0,-15} X -> Y X'", input[position]);

if (Y() && X\_())

return true;

return false;

}

static bool X\_()

{

if (input[position] == '%')

{

Console.WriteLine("{0,-15} X' -> % Y X'", input[position]);

position++;

if (Y() && X\_())

return true;

return false;

}

else

{

Console.WriteLine("{0,-15} X' -> ε", input[position]);

return true; // ε production

}

}

static bool Y()

{

Console.WriteLine("{0,-15} Y -> Z Y'", input[position]);

if (Z() && Y\_())

return true;

return false;

}

static bool Y\_()

{

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

{

Console.WriteLine("{0,-15} Y' -> & Z Y'", input[position]);

position++;

if (Z() && Y\_())

return true;

return false;

}

else

{

Console.WriteLine("{0,-15} Y' -> ε", input[position]);

return true; // ε production

}

}

static bool Z()

{

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

{

Console.WriteLine("{0,-15} Z -> k X k", input[position]);

position++;

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

{

position++;

return true;

}

}

else if (input[position] == 'g')

{

Console.WriteLine("{0,-15} Z -> g", input[position]);

position++;

return true;

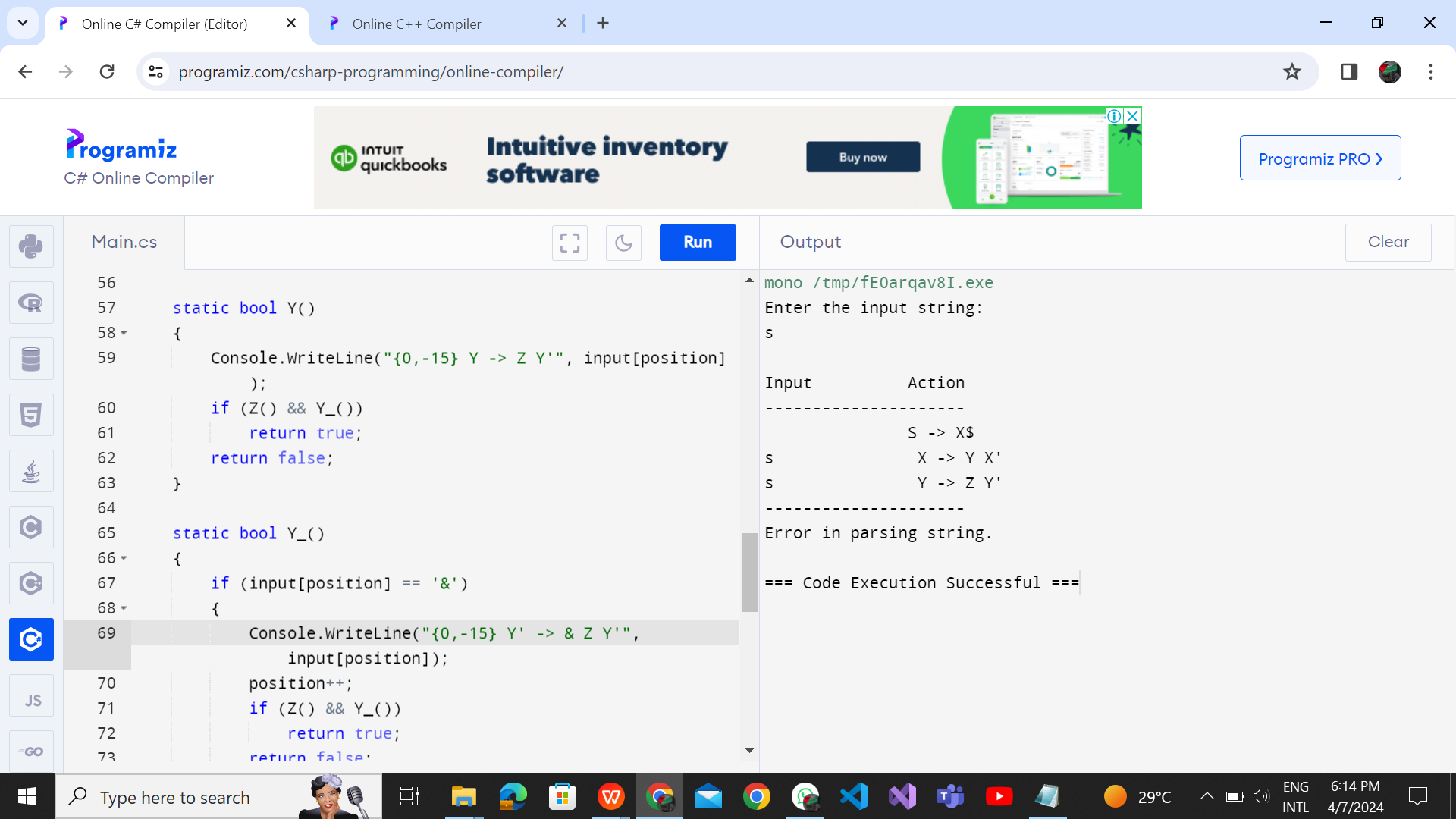
}

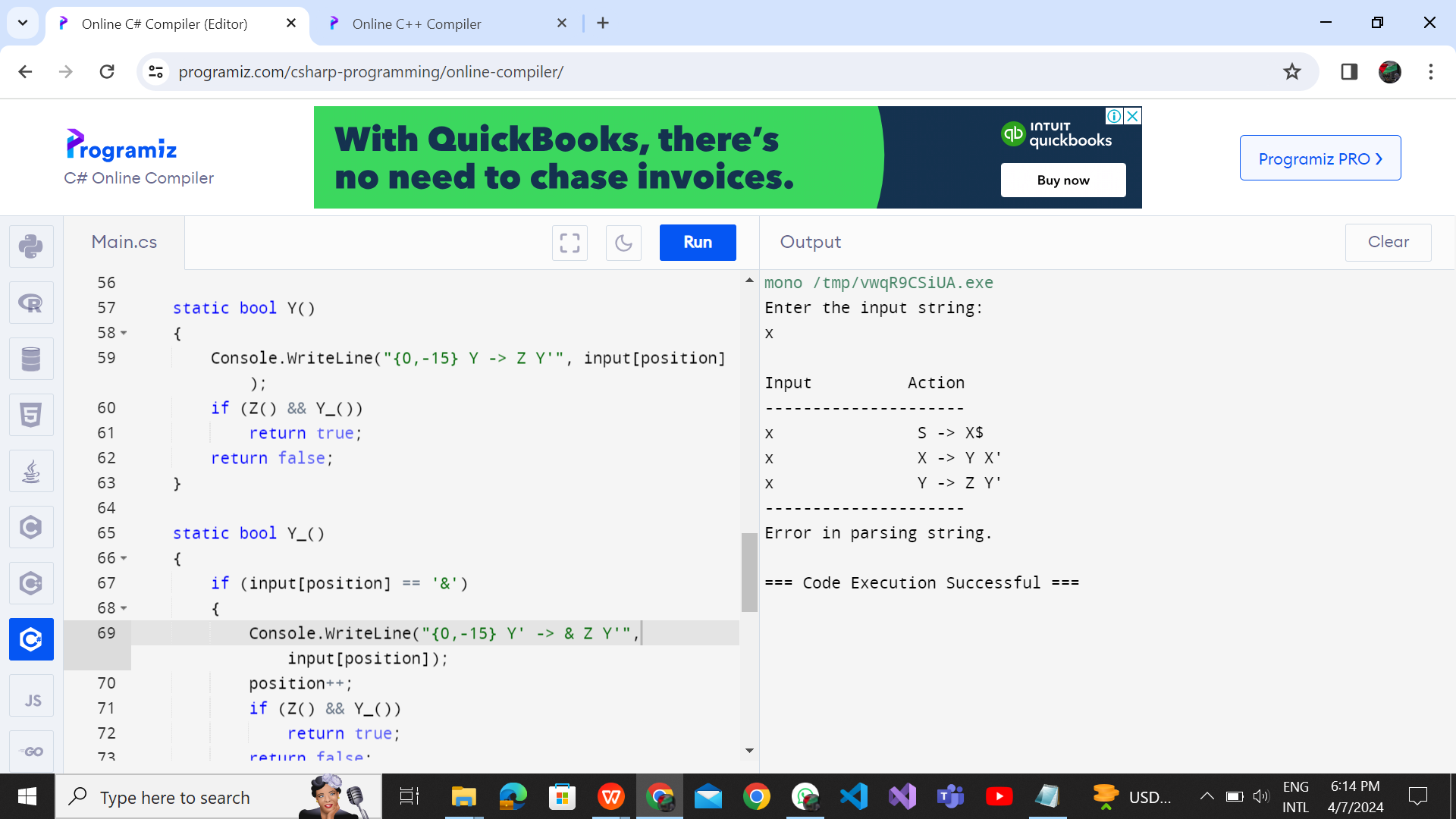
return false;

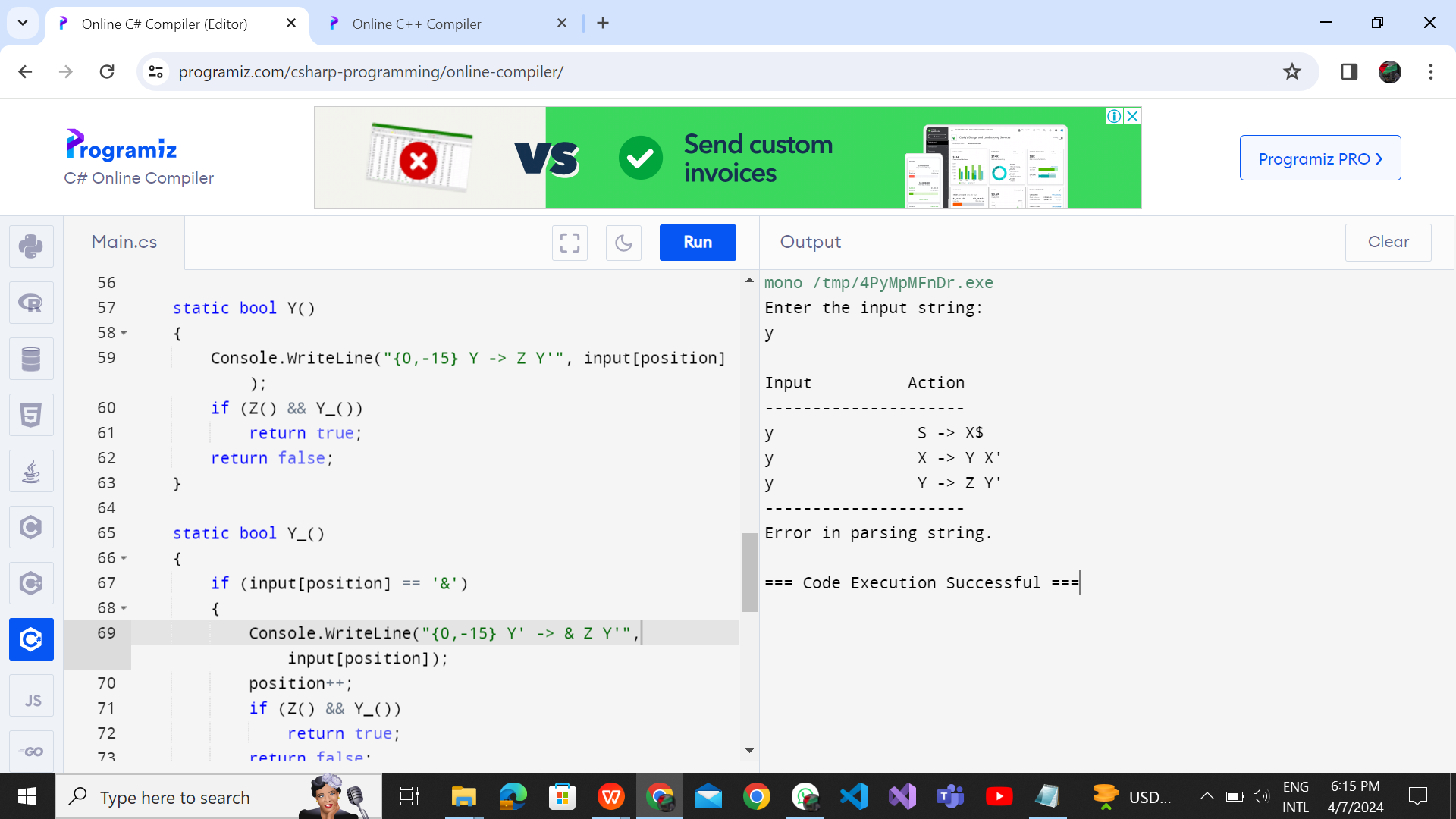
}

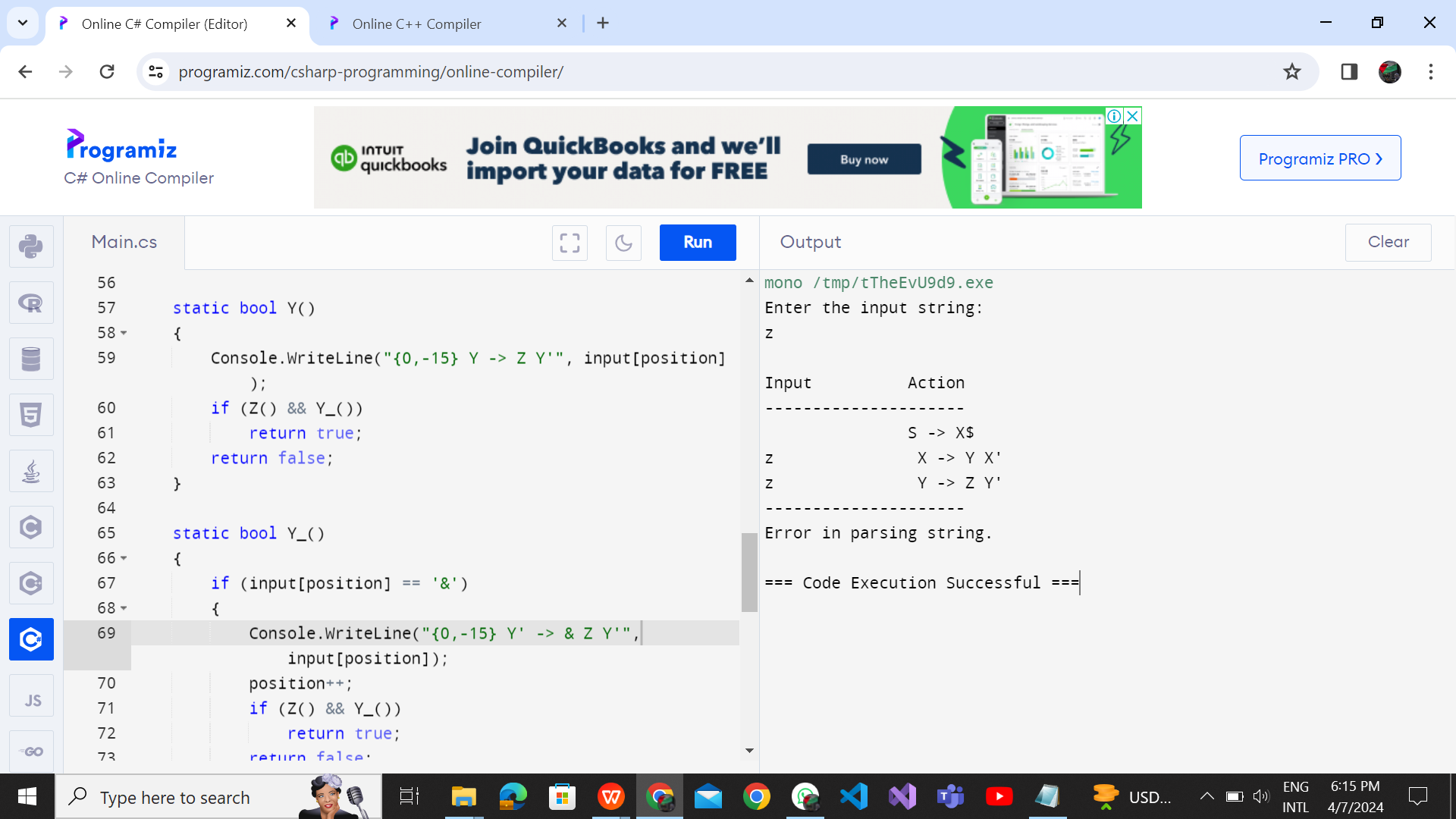
}

Output









**QUESTION NO 3:**

Make a Password generator according the following rules:

1. Atleast one uppercase alphabet
2. Atleast 4 numbers , two numbers must be your registration numbers
3. Atleast 2 special characters
4. Must contain initials of first and last name
5. Must contain all odd letters of your first name.
6. Must contain all even letters of your last name.
7. maximum length of 16

**Code**

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 <= chars; i++)

{

password += Char.ToUpper((char)('a' + random.Next(26)));

}

password += lastName[0];

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-025", "raja", "zeeshan");

Console.WriteLine(pass);

}

}

}

Output

