

LAB MID TERM



NAME : FATIMAH IJAZ

REG # SP21-BCS-006

COURSE : COMPILER CONSTRUCTION

SUBMITTED TO : SIR BILAL BUKHARI

DATE : 07-04-2024

QUESTION NO 1:

Briefly describe the regex library of C#.

ANSWER :

Regular Expression:

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.

Use Cases for the .NET C# Regular Expressions Library:

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.

Components and Features of the Regex library in C#:

1. **Regex Class:** The **Regex** class is central to working with regular expressions in C#. It provides methods for compiling and using regular expressions in various operations like pattern matching, replacement, and splitting.
2. **Pattern Syntax:** C# supports a rich syntax for regular expressions, including character classes (`[...]`), quantifiers (`*`, `+`, `?`, `{n}`, `{n,}`, `{n,m}`), anchors (`^`, `$`), groupings (`(...)`, `(?:...)`), alternation (`|`), and more.
3. **Options:** The **RegexOptions** enum allows specifying options such as case sensitivity, multiline mode, single-line mode, and others to control how the regular expression engine interprets patterns.
4. **Match Class:** The **Match** class represents a single match result obtained by applying a regular expression pattern to an input string. It provides properties and methods to access details about the match, captured groups, and more.

5. **MatchCollection Class:** When a regular expression can match multiple occurrences in an input string, the **MatchCollection** class is used to store and manage multiple **Match** instances.
6. **Replacement Operations:** The **Regex.Replace** method is used for replacing matched substrings in an input string with a specified replacement string or evaluated expression.
7. **Splitting Strings:** The **Regex.Split** method allows splitting a string into substrings based on a regular expression pattern, providing more flexibility than the standard **String.Split** method.
8. **Validation:** Regular expressions are commonly used for input validation tasks, such as validating email addresses, phone numbers, URLs, etc., by matching input strings against specific patterns.

C# Regular Expressions Library is an incredibly powerful library that allows developers to create and manipulate strings in various ways easily. With just a few lines of code, you can create powerful search patterns that can be used for data validation, string manipulation, and text processing.

QUESTION NO 2:

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

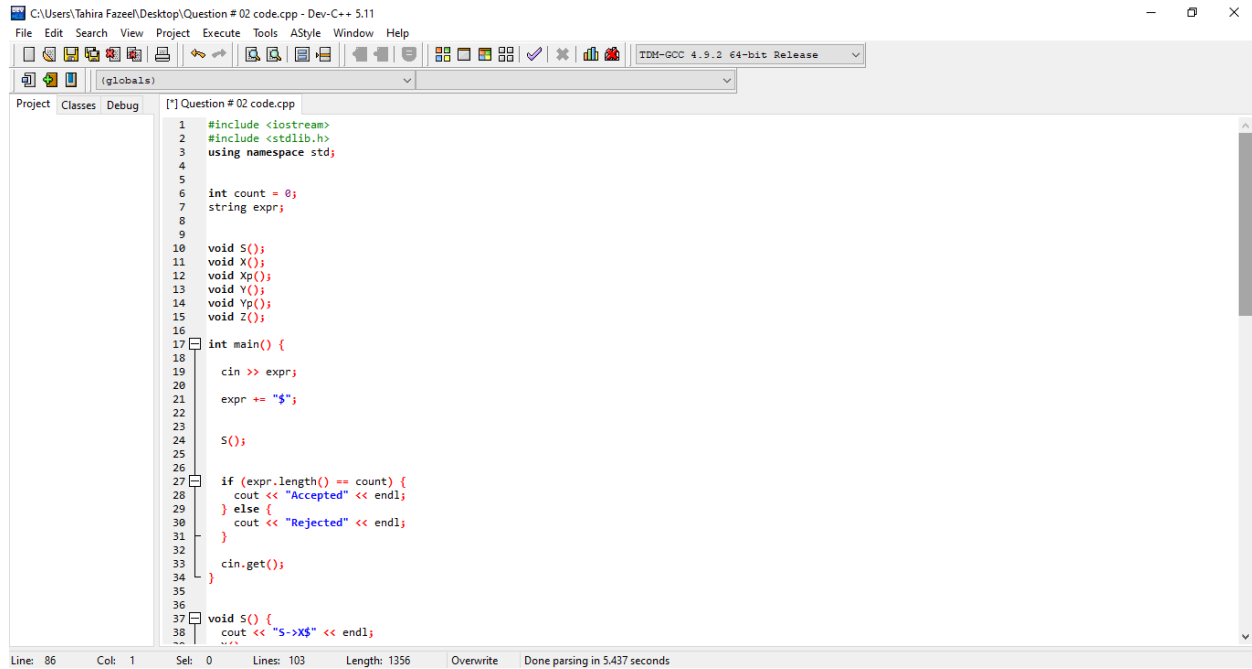
$S \rightarrow X\$$

$X \rightarrow X \% Y \mid Y$

$Y \rightarrow Y \& Z \mid Z$

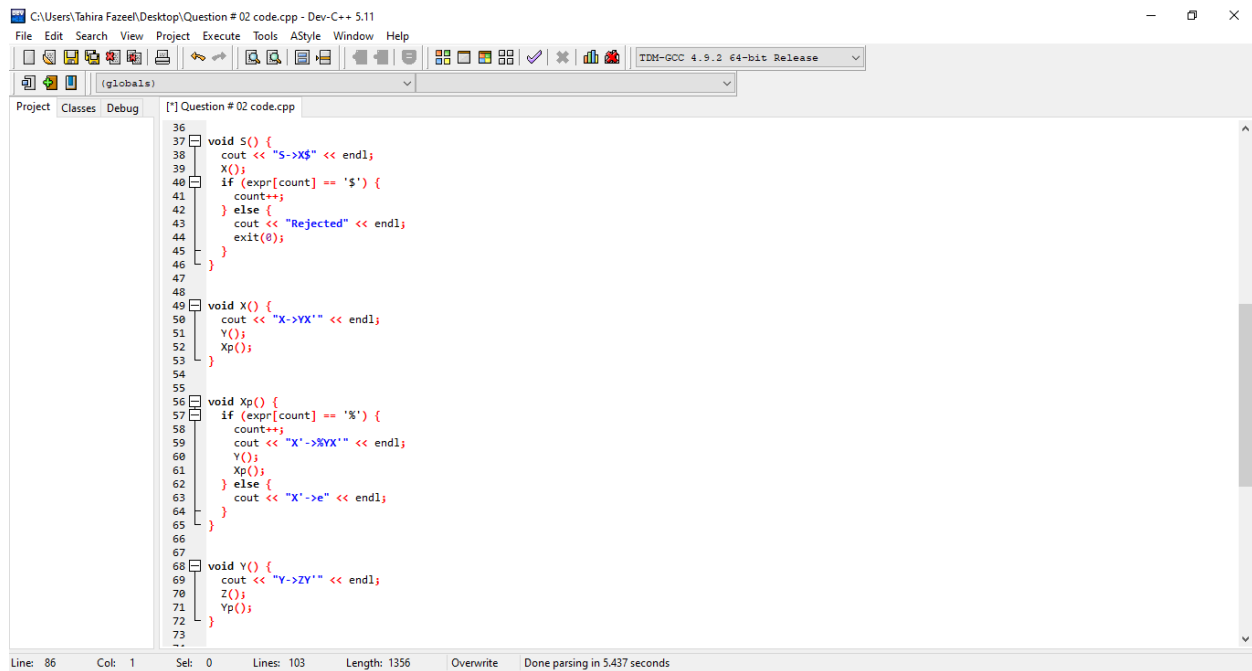
$Z \rightarrow k X k \mid g$

CODE :



```
1  #include <iostream>
2  #include <stdlib.h>
3  using namespace std;
4
5
6  int count = 0;
7  string expr;
8
9
10 void S();
11 void X();
12 void Xp();
13 void Y();
14 void Yp();
15 void Z();
16
17 int main() {
18
19     cin >> expr;
20
21     expr += "$";
22
23
24     S();
25
26
27     if (expr.length() == count) {
28         cout << "Accepted" << endl;
29     } else {
30         cout << "Rejected" << endl;
31     }
32
33     cin.get();
34 }
35
36 void S() {
37     cout << "S->X$" << endl;
38     ...
39 }
```

Line: 86 Col: 1 Sel: 0 Lines: 103 Length: 1356 Overwrite Done parsing in 5.437 seconds



```
36
37 void S() {
38     cout << "S->X$" << endl;
39     X();
40     if (expr[count] == '$') {
41         count++;
42     } else {
43         cout << "Rejected" << endl;
44         exit(0);
45     }
46 }
47
48 void X() {
49     cout << "X->YX" << endl;
50     Y();
51     Xp();
52 }
53
54
55 void Xp() {
56     if (expr[count] == 'X') {
57         count++;
58         cout << "X->YX" << endl;
59         Y();
60         Xp();
61     } else {
62         cout << "X->e" << endl;
63     }
64 }
65
66
67 void Y() {
68     cout << "Y->ZY" << endl;
69     Z();
70     Yp();
71 }
72
73
74 }
```

Line: 86 Col: 1 Sel: 0 Lines: 103 Length: 1356 Overwrite Done parsing in 5.437 seconds

```

C:\Users\Tahira Fazeel\Desktop\Question # 02 code.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
[global]
Project Classes Debug [*] Question # 02 code.cpp
66
67
68 void Y() {
69     cout << "Y->ZY" << endl;
70     Z();
71     Yp();
72 }
73
74
75 void Yp() {
76     if (expr[count] == 'k') {
77         count++;
78         cout << "Y'->&ZY" << endl;
79         Z();
80         Yp();
81     } else {
82         cout << "Y'->e" << endl;
83     }
84 }
85
86
87 void Z() {
88     if (expr[count] == 'k') {
89         count++;
90         cout << "Z->kXk" << endl;
91         X();
92         if (expr[count] == 'k') {
93             count++;
94         } else {
95             cout << "Rejected" << endl;
96             exit(0);
97         }
98     } else if (expr[count] == 'g') {
99         count++;
100         cout << "Z->g" << endl;
101         return;
102     }
103 }
Line: 86 Col: 1 Sel: 0 Lines: 103 Length: 1356 Overwrite Done parsing in 5.437 seconds

```

OUTPUT :

```

C:\Users\Tahira Fazeel\Desktop\Question # 02 code.exe
&g%kkk
S->Xk
X->YX'
Y->ZY'
Y'->&ZY'
Z->g
Y'->e
X'->%YX'
Y->ZY'
Z->kXk
X->YX'
Y->ZY'
Z->g
Y'->e
X'->e
Y'->e
X'->e
Accepted
-----
Process exited after 16.58 seconds with return value 0
Press any key to continue . . .

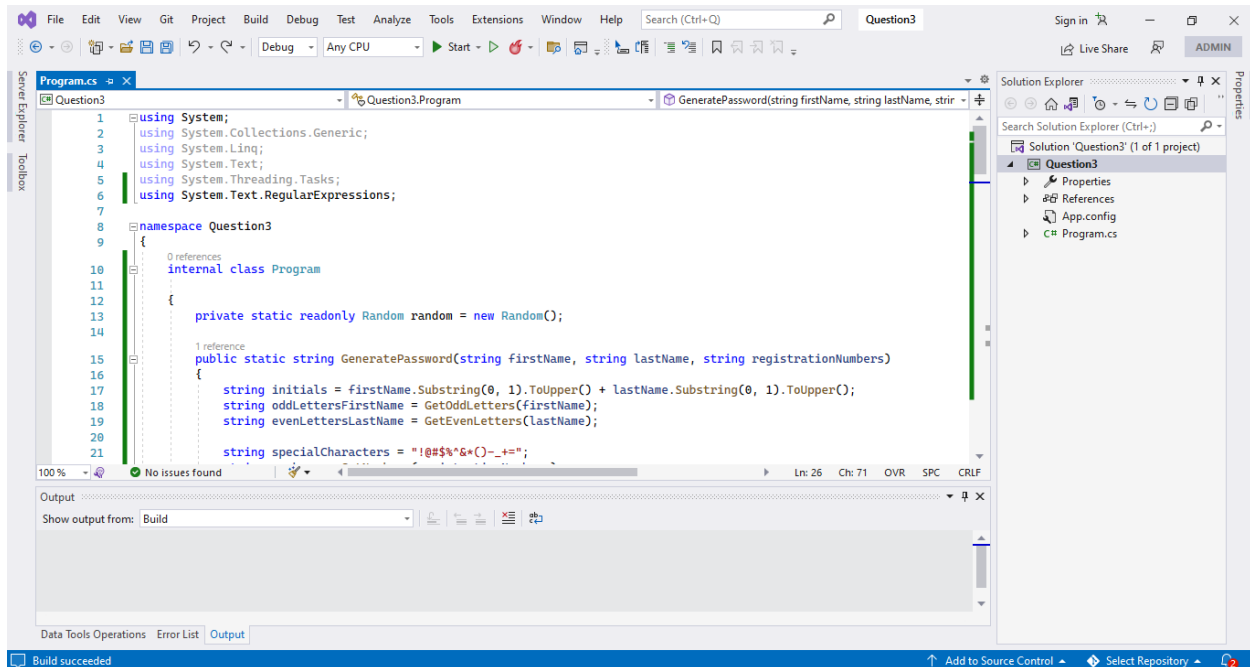
```

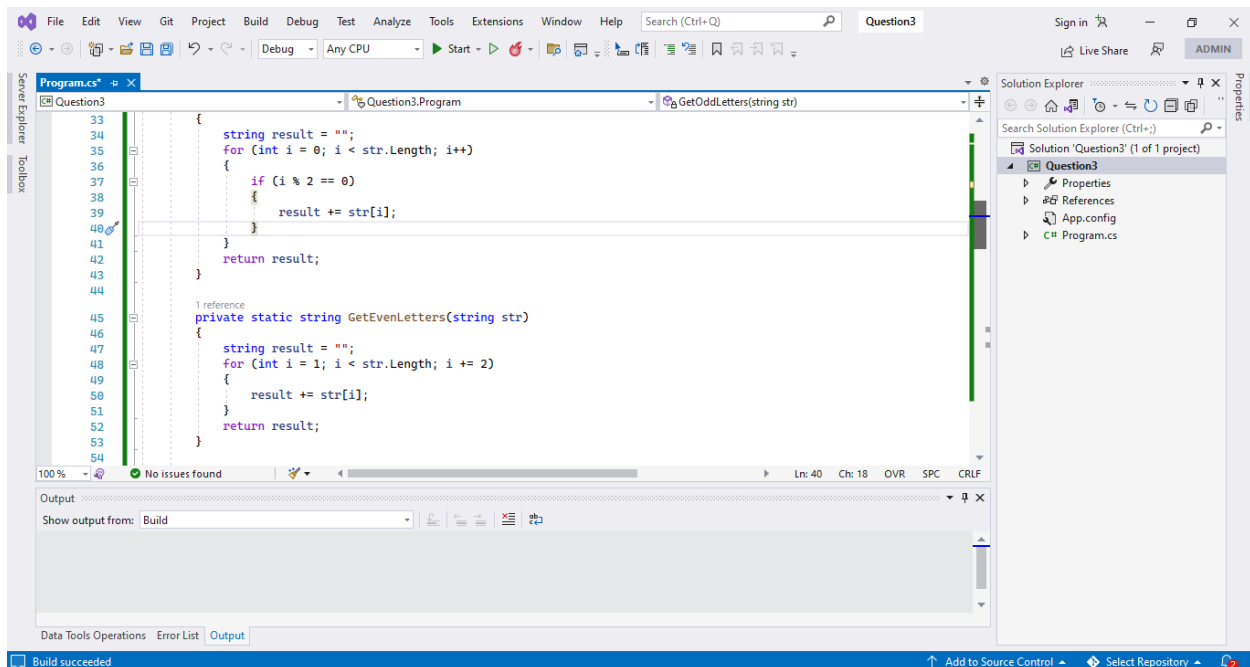
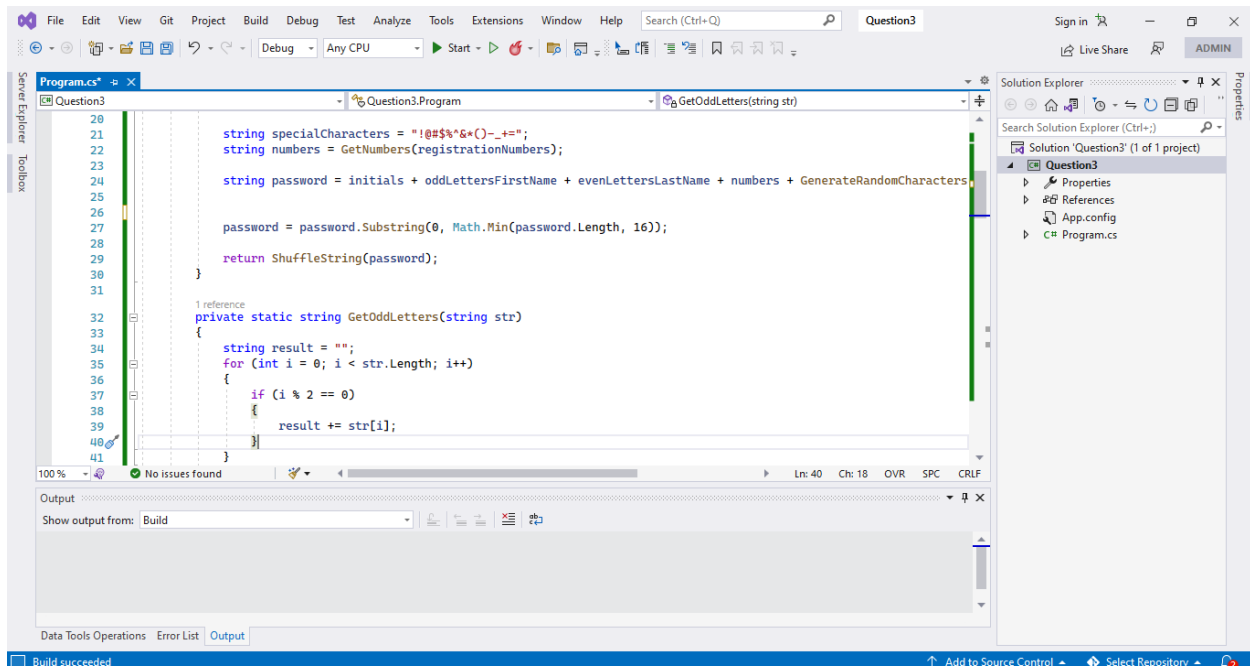
QUESTION NO 3:

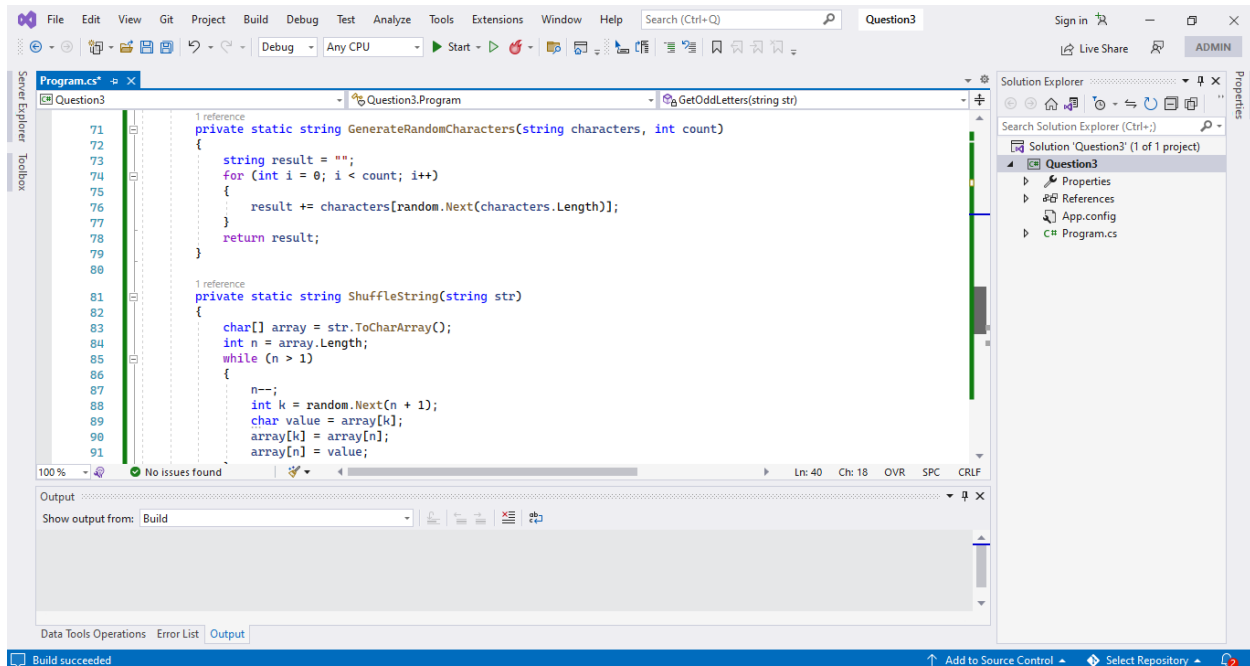
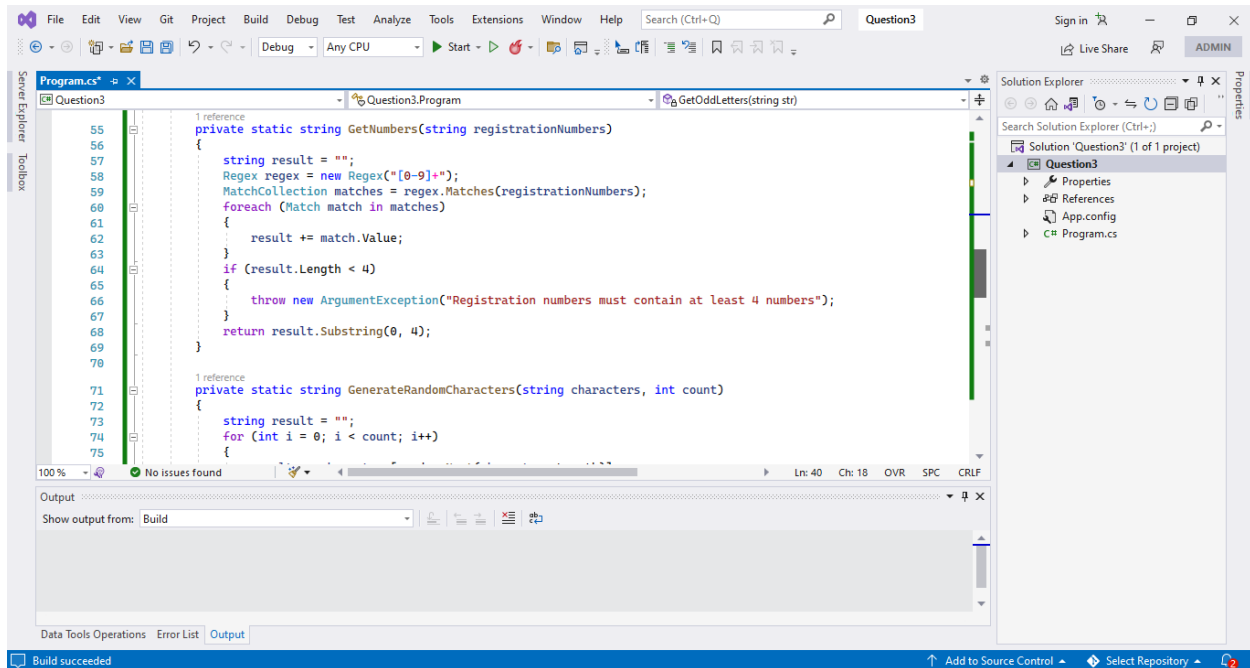
Make a Password generator according the following rules:

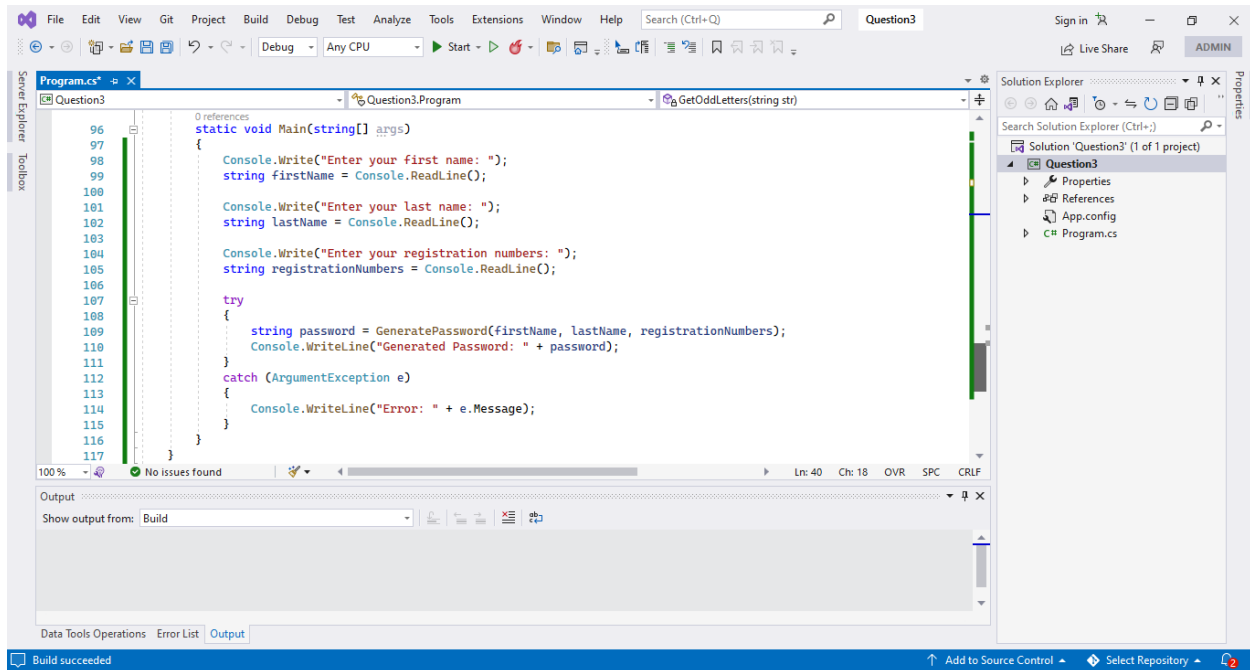
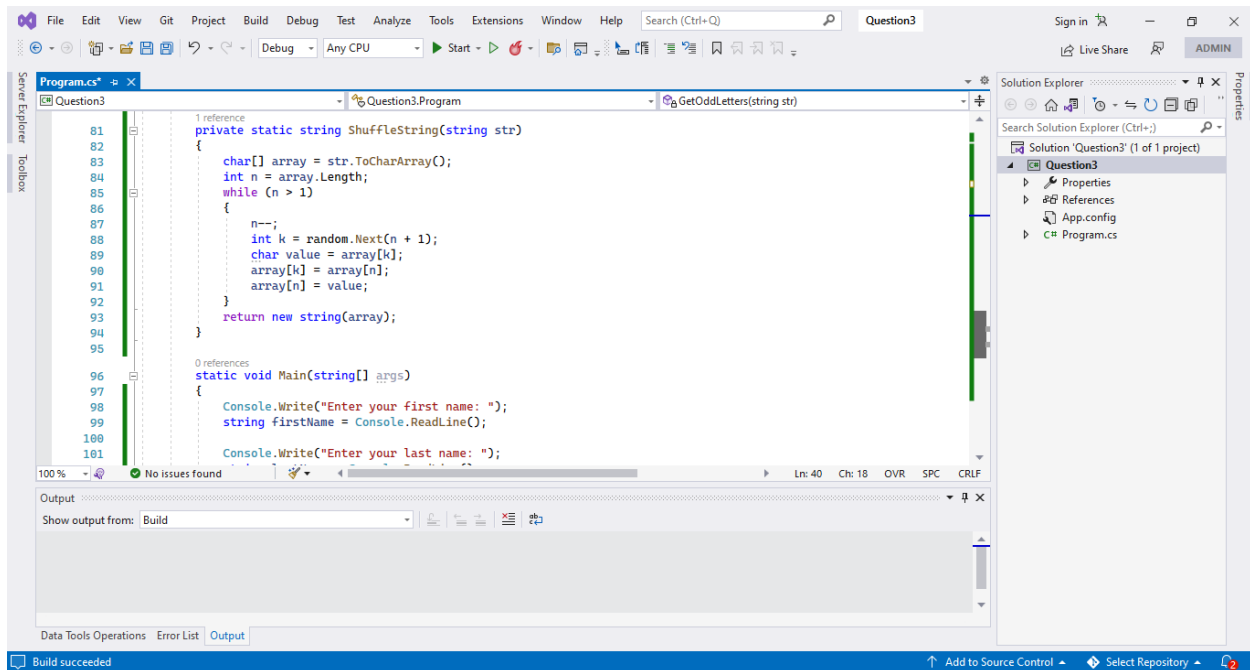
- (a) Atleast one uppercase alphabet
- (b) Atleast 4 numbers , two numbers must be your registration numbers
- (c) Atleast 2 special characters
- (d) Must contain initials of first and last name
- (e) Must contain all odd letters of your first name.
- (f) Must contain all even letters of your last name.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

CODE :









OUTPUT :

```
C:\WINDOWS\system32\cmd.exe
Enter your first name: fatimah
Enter your last name: ijaz
Enter your registration numbers: 0006
Generated Password: hj_6tz00fmF)l0
Press any key to continue . . .
```