



LAIBA SULTAN

SP21-BCS-015

5th April, 2024

Submitted To: Sir Bilal Bukhari

QUESTION NO 1:

Briefly describe the regex library of C#

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. 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.

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:

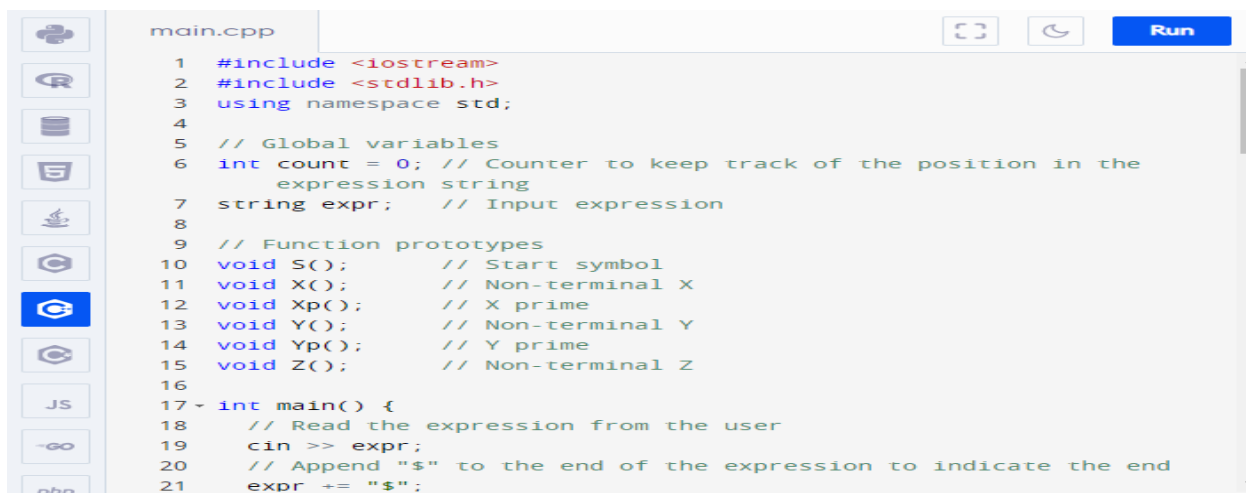
$S \rightarrow X\$$

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

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

$Z \rightarrow k X k \mid g$

Code:



```
main.cpp
1  #include <iostream>
2  #include <stdlib.h>
3  using namespace std;
4
5  // Global variables
6  int count = 0; // Counter to keep track of the position in the
   expression string
7  string expr;   // Input expression
8
9  // Function prototypes
10 void S();       // Start symbol
11 void X();       // Non-terminal X
12 void Xp();      // X prime
13 void Y();       // Non-terminal Y
14 void Yp();      // Y prime
15 void Z();       // Non-terminal Z
16
17 ~ int main() {
18     // Read the expression from the user
19     cin >> expr;
20     // Append "$" to the end of the expression to indicate the end
21     expr += "$";
```

```
main.cpp
21     expr += "$";
22
23     // Call the start symbol parsing function
24     S();
25
26     // Check if the entire expression has been parsed
27     if (expr.length() == count) {
28         cout << "Accepted" << endl;
29     } else {
30         cout << "Rejected" << endl;
31     }
32
33     cin.get(); // Wait for a character input
34 }
35
36 // Parse S
37 void S() {
38     cout << "S->X$" << endl;
39     X();
40     if (expr[count] == '$') {
41         count++;
42     } else {
```

```
main.cpp
40     if (expr[count] == '$') {
41         count++;
42     } else {
43         cout << "Rejected" << endl;
44         exit(0);
45     }
46 }
47
48 // Parse X
49 void X() {
50     cout << "X->YX'" << endl;
51     Y();
52     Xp();
53 }
54
55 // Parse X prime
56 void Xp() {
57     if (expr[count] == '%') {
58         count++;
59         cout << "X'->%YX'" << endl;
60         Y();
61         Xp();
```

```
main.cpp
62 } else {
63     cout << "X'->ε" << endl;
64 }
65 }
66
67 // Parse Y
68 void Y() {
69     cout << "Y->ZY'" << endl;
70     Z();
71     Yp();
72 }
73
74 // Parse Y prime
75 void Yp() {
76     if (expr[count] == '&') {
77         count++;
78         cout << "Y'->&ZY'" << endl;
79         Z();
80         Yp();
81     } else {
82         cout << "Y'->ε" << endl;
83     }
84 }
```

```
main.cpp
82     cout << "Y'->ε" << endl;
83 }
84 }
85
86 // Parse Z
87 void Z() {
88     if (expr[count] == 'k') {
89         count++;
90         cout << "Z->kXk" << endl;
91         X();
92     } else 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 }
```

Output:

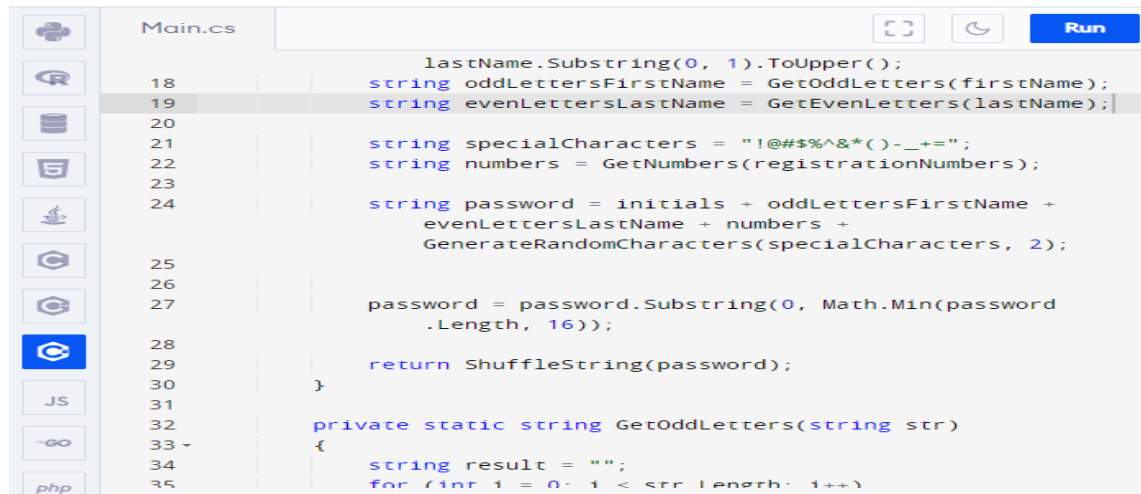
```
Output Clear
/tmp/SiWL68WNP9.o
&g
S->X$
X->YX'
Y->ZY'
Y'->&ZY'
Z->g
Y'->ε
X'->ε
Accepted

=== Code Execution Successful ===
```

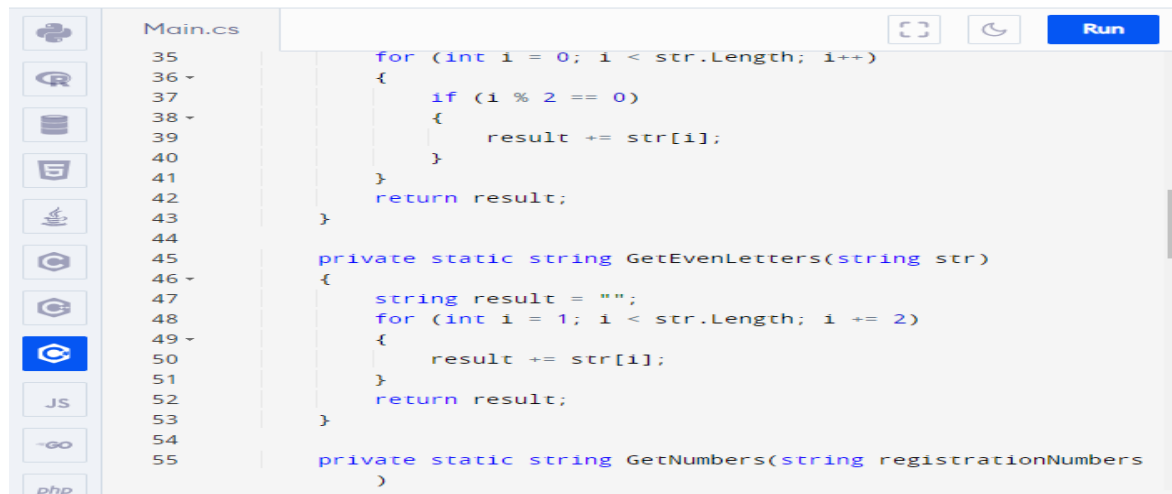
QUESTION NO 3:

Code:

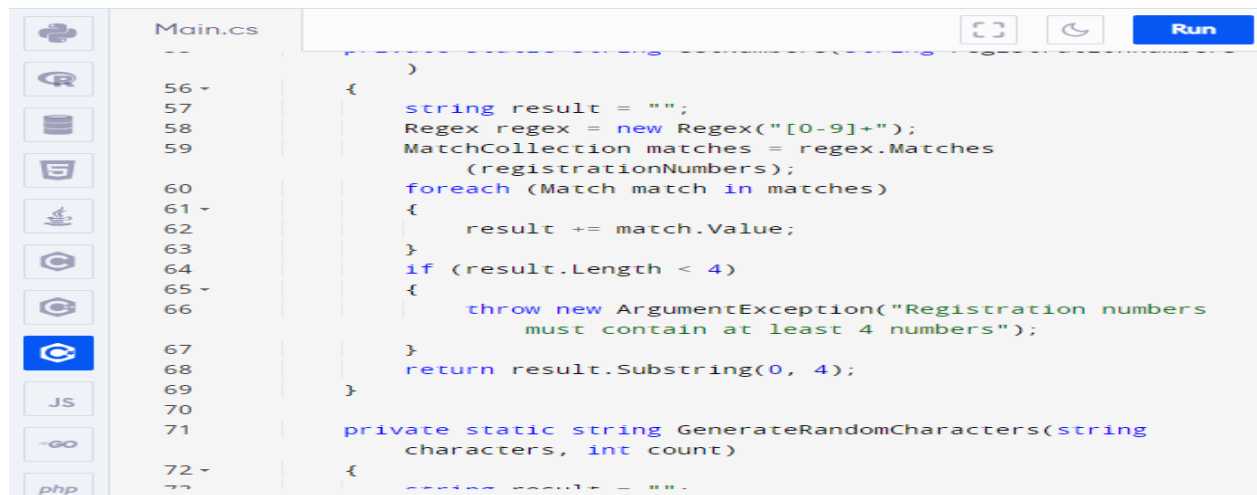
```
Main.cs Run
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Text;
5 using System.Threading.Tasks;
6 using System.Text.RegularExpressions;
7
8 namespace Question3
9 {
10     internal class Program
11     {
12         {
13             private static readonly Random random = new Random();
14
15             public static string GeneratePassword(string firstName,
16                 string lastName, string registrationNumbers)
17             {
18                 string initials = firstName.Substring(0, 1).ToUpper() +
19                     lastName.Substring(0, 1).ToUpper();
20                 string oddLettersFirstName = GetOddLetters(firstName);
21                 string evenLettersLastName = GetEvenLetters(lastName);
```



```
18         lastName.Substring(0, 1).ToUpper();
19         string oddLettersFirstName = GetOddLetters(firstName);
20         string evenLettersLastName = GetEvenLetters(lastName);
21
22         string specialCharacters = "!@#$%^&*()-_+=";
23         string numbers = GetNumbers(registrationNumbers);
24
25         string password = initials + oddLettersFirstName +
26             evenLettersLastName + numbers +
27             GenerateRandomCharacters(specialCharacters, 2);
28
29         password = password.Substring(0, Math.Min(password
30             .Length, 16));
31
32         return ShuffleString(password);
33     }
34     private static string GetOddLetters(string str)
35     {
36         string result = "";
37         for (int i = 0; i < str.Length; i++)
```

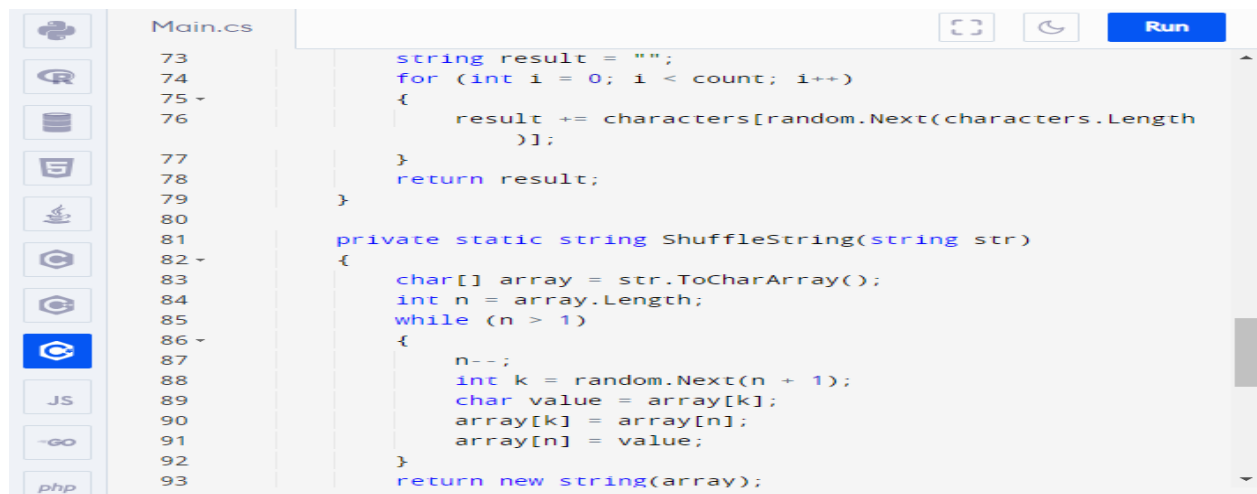


```
35         for (int i = 0; i < str.Length; i++)
36         {
37             if (i % 2 == 0)
38             {
39                 result += str[i];
40             }
41         }
42         return result;
43     }
44     private static string GetEvenLetters(string str)
45     {
46         string result = "";
47         for (int i = 1; i < str.Length; i += 2)
48         {
49             result += str[i];
50         }
51         return result;
52     }
53     private static string GetNumbers(string registrationNumbers
54     )
55     {
```

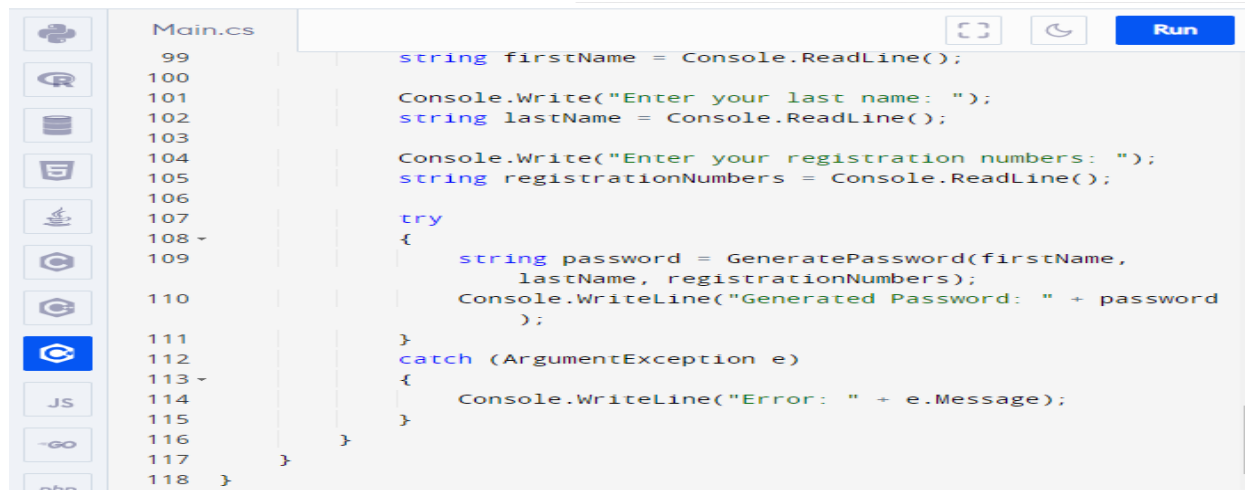
The screenshot shows a code editor with a sidebar on the left containing icons for various languages and a 'Run' button. The main editor area displays the following C# code in Main.cs:

```
--
56 ~
57 {
58     string result = "";
59     Regex regex = new Regex("[0-9]+");
60     MatchCollection matches = regex.Matches
61     (registrationNumbers);
62     foreach (Match match in matches)
63     {
64         result += match.Value;
65     }
66     if (result.Length < 4)
67     {
68         throw new ArgumentException("Registration numbers
69         must contain at least 4 numbers");
70     }
71     return result.Substring(0, 4);
72 }
73
74 private static string GenerateRandomCharacters(string
75 characters, int count)
76 {
77     string result = "";
```

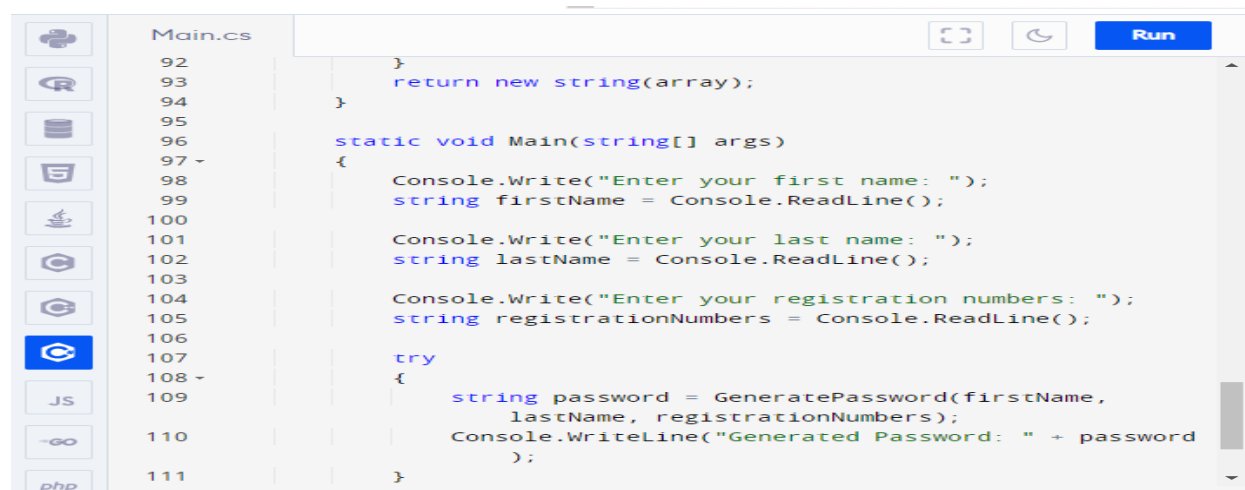


The screenshot shows a code editor with a sidebar on the left containing icons for various languages and a 'Run' button. The main editor area displays the following C# code in Main.cs:

```
73 string result = "";
74 for (int i = 0; i < count; i++)
75 {
76     result += characters[random.Next(characters.Length
77     )];
78 }
79 return result;
80 }
81
82 private static string ShuffleString(string str)
83 {
84     char[] array = str.ToCharArray();
85     int n = array.Length;
86     while (n > 1)
87     {
88         n--;
89         int k = random.Next(n + 1);
90         char value = array[k];
91         array[k] = array[n];
92         array[n] = value;
93     }
94     return new string(array);
```

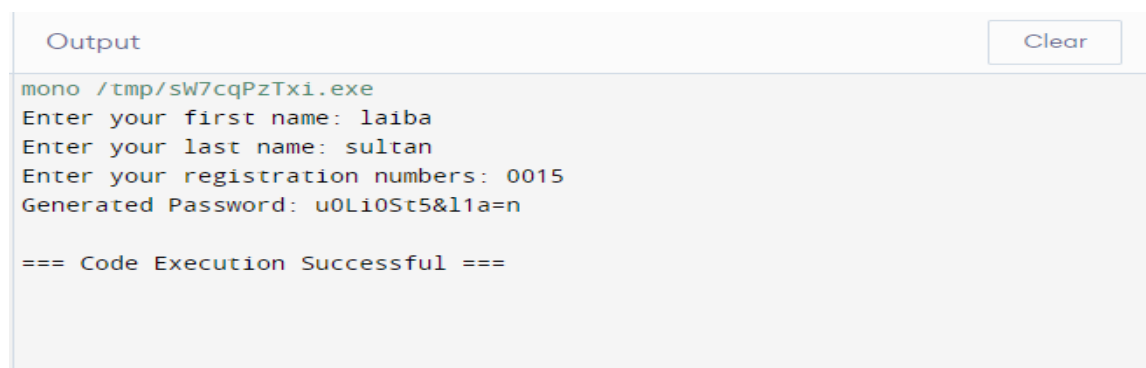


```
99     string firstName = Console.ReadLine();
100
101     Console.Write("Enter your last name: ");
102     string lastName = Console.ReadLine();
103
104     Console.Write("Enter your registration numbers: ");
105     string registrationNumbers = Console.ReadLine();
106
107     try
108     {
109         string password = GeneratePassword(firstName,
110             lastName, registrationNumbers);
111         Console.WriteLine("Generated Password: " + password
112             );
113     }
114     catch (ArgumentException e)
115     {
116         Console.WriteLine("Error: " + e.Message);
117     }
118 }
```



```
92     }
93     return new string(array);
94 }
95
96 static void Main(string[] args)
97 {
98     Console.Write("Enter your first name: ");
99     string firstName = Console.ReadLine();
100
101     Console.Write("Enter your last name: ");
102     string lastName = Console.ReadLine();
103
104     Console.Write("Enter your registration numbers: ");
105     string registrationNumbers = Console.ReadLine();
106
107     try
108     {
109         string password = GeneratePassword(firstName,
110             lastName, registrationNumbers);
111         Console.WriteLine("Generated Password: " + password
112             );
113     }
114 }
```

Output:



```
Output
mono /tmp/sw7cqPzTx1.exe
Enter your first name: laiba
Enter your last name: sultan
Enter your registration numbers: 0015
Generated Password: u0Li0St5&l1a=n

=== Code Execution Successful ===
```

