Name:Muhammad Awais Aslam

Reg No: FA20-BCS-067

COMPILER CONSTRUCTION MID LAB:

Question 1

Describe functioning of regex C# library , give examples of patterns,seperators and anchor e.t.c.

The regex library of C# is a powerful tool for working with regular expressions. It provides a set of classes and methods that allow you to match, search, and replace text based on regular expression patterns.

The most important class in the regex library is the Regex class. This class represents a regular expression and provides a variety of methods for working with it. For example, you can use the Match() method to find a match for the regular expression in a given string, and the Replace() method to replace all matches for the regular expression with a new string.

The regex library also includes several other classes and methods that can be useful for working with regular expressions. For example, the MatchCollection() class represents a collection of matches for a regular expression, and the Group class represents a group of characters within a match.

Key Classes in C# Regex Library:

Regex: The `Regex` class is the primary class in the library. It represents a compiled regular expression pattern and provides methods for pattern matching and replacement.

Match: The `Match` class represents a single match of a regular expression pattern in an input string. It provides information about the matched text and its position.

MatchCollection: This class represents a collection of `Match` objects. It is returned by methods like `Regex.Matches()` when you want to find all matches in an input string.

Basic Operations:

Pattern Matching (Regex.Match): You can use `Regex.Match()` to find the first occurrence of a regular expression pattern in an input string. It returns a `Match` object containing information about the first match.

Pattern Matching (Regex.Matches): The `Regex.Matches()` method finds all occurrences of a

pattern in an input string and returns a `MatchCollection` containing all the matches.

Pattern Replacement (Regex.Replace): You can use `Regex.Replace()` to replace all occurrences of a pattern in an input string with a specified replacement string.

The regex library in C# provides a powerful way to work with regular expressions. Regular expressions, also known as regex, allow you to search, match, and manipulate text patterns.

In C#, you can utilize the System.Text.RegularExpressions namespace to work with regular expressions. This namespace contains classes like Regex, Match, and MatchCollection that enable you to construct and apply regular expressions.

To create a regex pattern, you can use a combination of literal characters and special characters that define rules for matching specific patterns in text. Some common special characters in regex are:

Metacharacters: These characters have special meanings in regex patterns. For example:

* . (dot) matches any character except a newline.
* matches zero or more occurrences of the preceding element.
* + matches one or more occurrences of the preceding element.
* ? matches zero or one occurrence of the preceding element.
* [] matches any single character within the brackets.
* () creates a capturing group.

Anchors: These are special characters that represent positions in the text. For example:

* ^ matches the start of a line or input string.
* $ matches the end of a line or input string.
* \b matches a word boundary.
* Escape sequences: These are used to match specific characters. For example:
* \d matches any digit (0-9).
* \w matches any word character (alphanumeric and underscore).
* \s matches any whitespace character.

Here are some examples of regex patterns and their descriptions:

* Pattern: ^\d{3}-\d{3}-\d{4}$ Description: Matches a US phone number in the format XXX-XXX-XXXX.
* Pattern: ^[A-Za-z]+\d\*$ Description: Matches a string that starts with one or more letters, followed by zero or more digits.
* Pattern: \bcat\b Description: Matches the word "cat" as a whole word, not as part of another word like "category".
* Pattern: ^[A-Z][a-z]+\s[A-Z][a-z]+$ Description: Matches a string with two capitalized words separated by a space.

Remember that regex patterns can be much more complex depending on your requirements. You can combine different metacharacters, anchors, and escape sequences to build more sophisticated patterns.

Make sure to use the Regex class methods like Regex.Match or Regex.Matches to apply regex patterns to input text and retrieve matches or match collections.

I hope this clarifies how the regex library works in C# and provides you with some examples of patterns, separators, and anchors in regular expressions.

Here is an example of how to use the regex library to match and replace text:

C#

// Create a regular expression object

Regex regex = new Regex(@"\d+");

// Match the regular expression in the input string

Match match = regex.Match("This string contains 123 numbers");

// If there is a match, replace it with the string "numbers" if (match.Success)

{

string output = match.Result.Replace("123 numbers", "numbers");

Console.WriteLine(output); // This string contains numbers }

The regex library is a very powerful tool for working with text, and it can be used to solve a wide variety of problems.

Here are some of the benefits of using the regex library of C#:

* It is very efficient and can be used to process large amounts of text quickly.
* It is very flexible and can be used to create complex regular expression patterns.
* It is well-documented and there are many resources available to help you learn how to use it.

Question 2

For the given grammar create LL(1) or recursive descent parser

List -> Item Rest

Rest -> , Item Rest | ε

Item -> id | num | string

Code:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace LL1\_MID\_Task2

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private void ParseButton\_Click(object sender, EventArgs e)

{

string input = InputRichTextBox.Text;

Parser parser = new Parser(input);

if (parser.ParseList())

{

ResultLabel.Text = "Valid input!";

}

else

{

ResultLabel.Text = "Invalid input!";

}

}

public class Parser

{

private string[] tokens;

private int index;

public Parser(string input)

{

this.tokens = input.Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

this.index = 0;

}

public bool ParseList()

{

if (ParseItem())

{

return ParseRest();

}

return false;

}

private bool ParseRest()

{

if (index < tokens.Length && tokens[index] == ",")

{

index++;

if (ParseItem())

{

return ParseRest();

}

return false;

}

return true; // ε-production

}

private bool ParseItem()

{

if (index < tokens.Length)

{

string token = tokens[index];

if (token == "id" || token == "num" || token == "string")

{

index++;

return true;

}

}

return false;

}

}

private void Form1\_Load(object sender, EventArgs e)

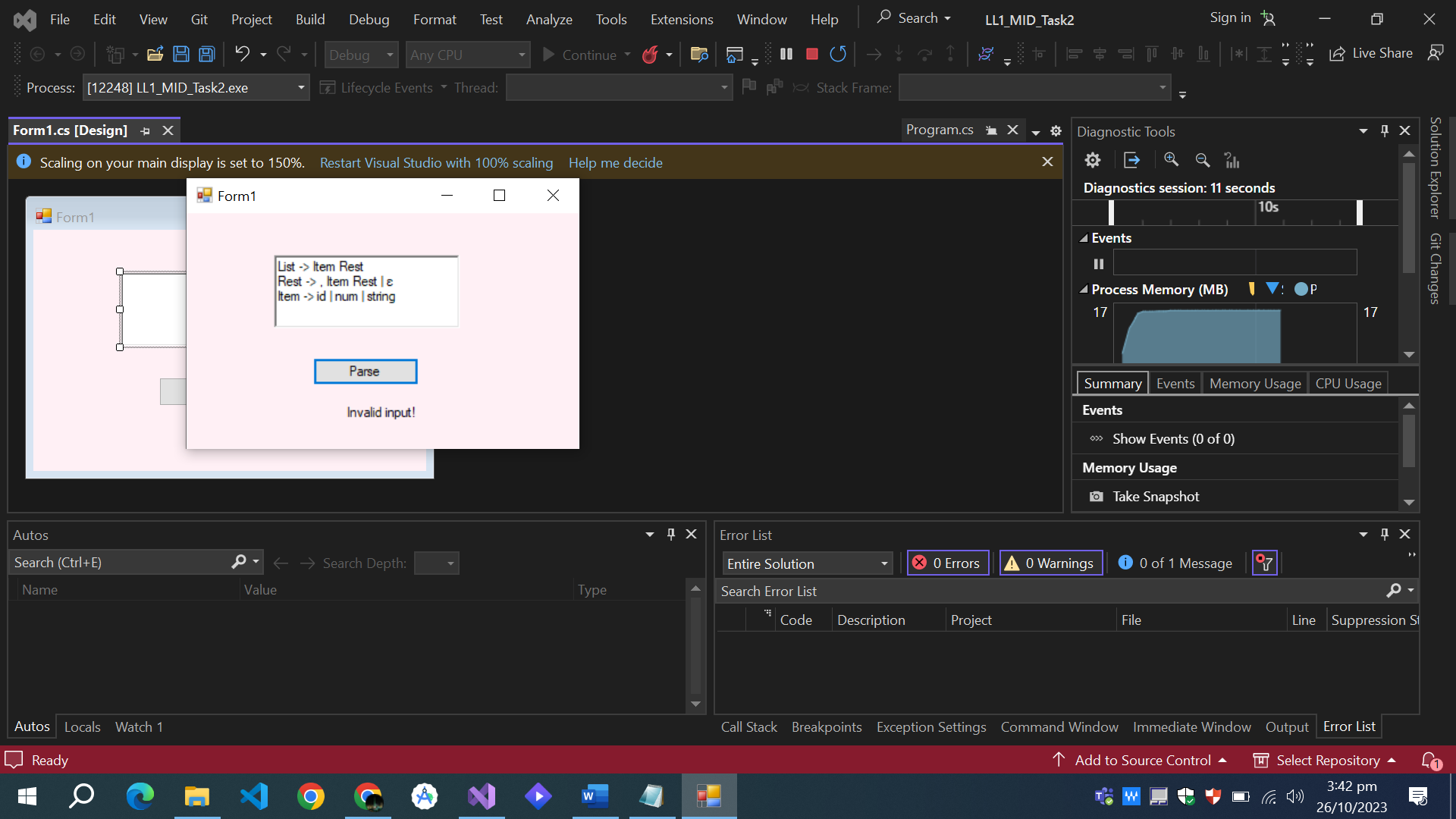
{

}

}

}

Output:



Question 3

Create a password generator which has the following rules

1. maximum length of 20

2. Atleast 2 special characters

3. Atleast 4 numbers

4. Intials of your first and last name (e.g. Babar Khan has intials B and K) in upper case

5. 2 of the numbers should be your last two digits of your registration number (e.g. if registration number is fa20-bcs-012)

then two of the numbers should be 1 and 2.

Code:

using System;

using System.Text;

using System.Windows.Forms;

namespace CC\_LabMid\_Question3

{

public partial class PasswordGenerator : Form

{

public PasswordGenerator()

{

InitializeComponent();

}

private void GeneratePasswordButton\_Click(object sender, EventArgs e)

{

string firstName = "Awais";

string lastName = "Aslam";

string regNumber = "67";

if (firstName.Length < 1 || lastName.Length < 1 || regNumber.Length < 1)

{

MessageBox.Show("Please enter your first name, last name, and registration number.");

return;

}

// Create a StringBuilder to build the password

StringBuilder password = new StringBuilder();

// Add initials of first and last name

password.Append(firstName[0]);

password.Append(lastName[0]);

// Add registration number

password.Append(regNumber);

// Generate random uppercase alphabet

Random random = new Random();

password.Append((char)random.Next('A', 'Z' + 1));

// Generate 3 random numbers (1 less than previously)

for (int i = 0; i < 2; i++)

{

password.Append((char)random.Next('0', '9' + 1));

}

// Generate 2 special characters (same as previously)

string specialCharacters = "!@#$%^&\*()\_-+=<>?";

for (int i = 0; i < 2; i++)

{

password.Append(specialCharacters[random.Next(specialCharacters.Length)]);

}

// Shuffle the password characters for better security

password = ShuffleString(password);

// Limit the password to a maximum length of 16 (1 less than previously)

if (password.Length > 20)

{

password.Length = 20;

}

// Display the generated password

GeneratedPasswordLabel.Text = password.ToString();

}

private StringBuilder ShuffleString(StringBuilder str)

{

Random random = new Random();

int n = str.Length;

while (n > 1)

{

n--;

int k = random.Next(n + 1);

char value = str[k];

str[k] = str[n];

str[n] = value;

}

return str;

}

private void label2\_Click(object sender, EventArgs e)

{

}

}

}

Output:

