**COMSATS UNIVERSITY ISLAMABAD**

**(ATTOCK CAMPUS)**

****

**LAB MID EXAM**

**Subject:**

**CC Lab**

**Submitted to:**

**Sir Bilal Haider**

**Submitted by:**

Muhammad Zubair

*Fa20-bcs-041*

***Dated****: 26nd Oct, 2023*

Question 1

Describe functioning of regex C# library, give examples of patterns, separators and anchors etc.

Answer:

The regular expression (regex) library in C#, provided by the `System.Text.RegularExpressions` namespace, allows you to work with regular expressions, which are powerful tools for pattern matching and text manipulation. Here's a description of the functioning of the regex library along with examples of patterns, separators, and anchors:

1. Creating a Regex Object:

Start by creating a `Regex` object with the desired pattern you want to match. For example:

using System.Text.RegularExpressions;

string pattern = @"\d+"; // Match one or more digits

Regex regex = new Regex(pattern);

2. Matching and Searching:

You can use the `Match` and `Matches` methods to find matches in a string:

string text = "There are 123 cats and 456 dogs.";

Match match = regex.Match(text); // Find the first match

MatchCollection matches = regex.Matches(text); // Find all matches

// Loop through matches

foreach (Match m in matches)

{

Console.WriteLine(m.Value); // Output: "123", "456"

}

```

3. Pattern Elements:

- `.`: Matches any character except a newline.

- `\d`: Matches any digit (equivalent to `[0-9]`).

- `\w`: Matches a word character (equivalent to `[a-zA-Z0-9\_]`).

- `\*`: Matches 0 or more occurrences of the preceding element.

- `+`: Matches 1 or more occurrences of the preceding element.

- `?`: Matches 0 or 1 occurrence of the preceding element.

- `[]`: Defines a character class to match any one of the characters inside.

- `()`: Groups elements together.

- `|`: Acts as a logical OR for alternatives.

4. Anchors:

- `^`: Matches the start of a line.

- `$`: Matches the end of a line.

- `\b`: Matches a word boundary.

5. Quantifiers:

- `{n}`: Matches exactly n occurrences of the preceding element.

- `{n,}`: Matches n or more occurrences of the preceding element.

- `{n, m}`: Matches between n and m occurrences of the preceding element.

6. Character Classes:

- `[abc]`: Matches any one of the characters a, b, or c.

- `[0-9]`: Matches any digit.

- `[^a-z]`: Matches any character except lowercase letters.

7. Escape Sequences:

- Use backslashes to escape metacharacters when you want to match them literally. For example, `\.`, `\\`, and `\(`.

8. Separators:

- `\s`: Matches any whitespace character.

- `\t`: Matches a tab character.

- `,`: Matches a comma.

9. Modifiers:

- `i`: Case-insensitive matching.

- `m`: Multiline mode (enables `^` and `$` to match the start and end of lines).

- `s`: Single-line mode (treats the input as a single line).

Examples of using regular expressions in C#:

- Matching email addresses:

```csharp

string pattern = @"\b[A-Za-z0-9.\_%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}\b";

```

- Extracting URLs from text:

```csharp

string pattern = @"https?://\S+";

```

- Extracting words that start with a capital letter:

string pattern = @"\b[A-Z][a-z]\*\b";

- Extracting all the hashtags from a social media post:

string pattern = @"#\w+";

Regular expressions in C# are versatile and powerful, allowing you to work with complex text patterns and perform tasks like validation, extraction, and transformation of data efficiently. However, they can be complex and require careful crafting and testing for desired results.

Question 2

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

List -> Item Rest

Rest -> , Item Rest | ε

Item -> id | num | string

using System;

using System.Collections.Generic;

class Parser

{

private List<string> tokens;

private int index;

public Parser(List<string> input)

{

tokens = input;

index = 0;

}

// Start the parsing process

public void Parse()

{

List();

if (index == tokens.Count)

Console.WriteLine("Parsing successful!");

else

Console.WriteLine("Parsing failed: Unexpected input.");

}

private void List()

{

Item();

Rest();

}

private void Rest()

{

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

{

index++;

Item();

Rest();

}

// epsilon (empty) case

}

private void Item()

{

if (index < tokens.Count && (tokens[index] == "id" || tokens[index] == "num" || tokens[index] == "string"))

{

index++;

}

else

{

Console.WriteLine("Parsing failed: Expected 'id', 'num', or 'string'.");

Environment.Exit(1);

}

}

}

class Program

{

static void Main()

{

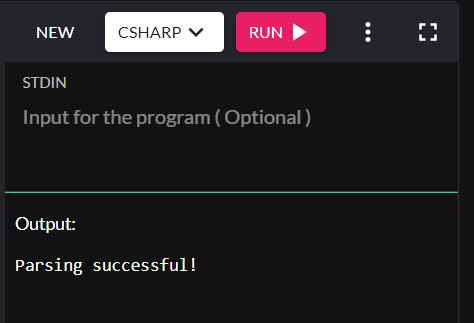
List<string> inputTokens = new List<string> { "id", ",", "num", ",", "string" };

Parser parser = new Parser(inputTokens);

parser.Parse();

}

}



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

using System;

using System.Text;

using System.Linq;

class PasswordGenerator

{

static void Main()

{

// Your personal information

string firstName = "Muhammad";

string lastName = "Zubair";

string registrationNumber = "fa20-bcs-041";

// Generate the password

string password = GeneratePassword(firstName, lastName, registrationNumber);

Console.WriteLine("Generated Password: " + password);

}

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

{

Random random = new Random();

StringBuilder password = new StringBuilder();

// Initials (Uppercase)

password.Append(char.ToUpper(firstName[0]));

password.Append(char.ToUpper(lastName[0]));

// Random letters (lowercase)

for (int i = 0; i < 14; i++)

{

password.Append((char)random.Next('a', 'z' + 1));

}

// Special characters

string specialCharacters = "!@#$%^&\*()-\_=+[]{}|;:'\",.<>?/";

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

{

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

}

// Numbers

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

{

password.Append(random.Next(10));

}

// Last two digits of registration number

password.Append(registrationNumber.Substring(registrationNumber.Length - 2));

// Shuffle the characters in the password for added security

string shuffledPassword = new string(password.ToString().ToCharArray().OrderBy(s => (random.Next(2) % 2) == 0).ToArray());

return shuffledPassword;

}

}

