

**MID LAB**

**FA20-BCS-009**

**Maryam Amjad**

**Compiler Construction**

**BCS 7A**

**25/OCT/2023**

**Submitted to :**

**MR. Bilal Bukhari**

# Question :1

**Introduction to Regex in C#:**

Regular expressions (regex) are a powerful tool for pattern matching and manipulation of strings. In C#, the **System.Text.RegularExpressions** namespace provides a comprehensive library for working with regular expressions. Let's dive into the fundamentals and various components of this library:

**1. Pattern Elements:**

* **Literal Characters:** You can use literal characters to match exact text. For example, the pattern **cat** will match the word "cat" in each text.
* **Metacharacters:** These are special characters with predefined meanings. For instance, **.** (dot) matches any character, and **\*** (asterisk) matches 0 or more occurrences.

**2. Character Classes:**

* **[ ]:** Character classes allow you to define a set of characters to match against. For example, **[a,e,I,o,u]** matches any vowel, and **[0-9]** matches any digit.
* **[^ ]:** The caret symbol within square brackets denotes negation. **[^0-9]** matches any character that is not a digit.

**3. Quantifiers:**

* **\*:** The asterisk quantifier matches 0 or more occurrences of the preceding element. For example, **a\*** matches "a," "aa," "aaa," and so on.
* **+:** The plus sign matches 1 or more occurrences of the preceding element. **b+** matches "b," "bb," "bbb," and more.
* **?:** The question mark matches 0 or 1 occurrence of the preceding element. **c?** matches "c" or an empty string.

**4. Anchors:**

* **^:** The caret symbol at the beginning of a pattern matches the start of a line or string. For example, **^abc** matches "abc" only if it appears at the start of a line.
* **$:** The dollar sign at the end of a pattern matches the end of a line or string. **xyz$** matches "xyz" only if it appears at the end of a line.

**5. Groups:**

* **( ):** Parentheses allow you to create capture groups for extracting matched substrings. For instance, **(abc)+** matches "abc," "abcabc," and so on, creating a capture group.

**6. Separators:**

* **\s:** The backslash followed by "s" matches any whitespace character, including spaces, tabs, and line breaks.
* **\b:** The backslash followed by "b" matches a word boundary. For example, **\bword\b** matches "word" but not "subword."

**7. Alternation:**

* **|:** The vertical bar allows you to specify multiple alternatives in a pattern. For instance, **apple|banana** matches either "apple" or "banana."

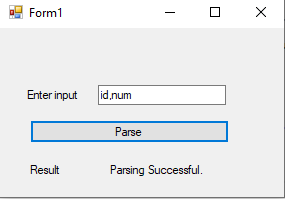
**8. Escape Characters:**

* **:** The backslash can be used to escape metacharacters, allowing you to match them literally. For example, **\\** matches a backslash.

**9. Match Evaluation:**

* **Regex.Match()**: This method searches for the first match in a given string.
* **Regex.Matches()**: It finds all matches in a given string.
* **Regex.Replace()**: This method replaces matched patterns in a string with a specified replacement.
* **Regex.Split()**: It splits a string into an array using a specified pattern as the separator.

Question 2:



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 Q2WF

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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

{

}

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

{

string input = textBoxInput.Text;

List<string> tokens = new List<string>(input.Split(new char[] { ' ', ',' }, StringSplitOptions.RemoveEmptyEntries));

int tokenIndex = 0;

if (ParseList(tokens, ref tokenIndex))

{

labelResult.Text = "Parsing Successfull.";

}

else

{

labelResult.Text = "Parssing Failed.";

}

}

private bool ParseList(List<string> tokens, ref int tokenIndex)

{

return ParseItem(tokens, ref tokenIndex) && ParseRest(tokens, ref tokenIndex);

}

private bool ParseRest(List<string> tokens, ref int tokenIndex)

{

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

{

tokenIndex++;

return ParseItem(tokens, ref tokenIndex) && ParseRest(tokens, ref tokenIndex);

}

return true; // ε (empty) production

}

private bool ParseItem(List<string> tokens, ref int tokenIndex)

{

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

{

tokenIndex++;

return true;

}

return false;

}

private void buttonValidate\_Click\_1(object sender, EventArgs e)

{

string input = textBoxInput.Text;

List<string> tokens = new List<string>(input.Split(new char[] { ' ', ',' }, StringSplitOptions.RemoveEmptyEntries));

int tokenIndex = 0;

if (ParseList(tokens, ref tokenIndex))

{

labelResult.Text = "Parsing Successful.";

}

else

{

labelResult.Text = "Parsing Failed.";

}

}

}

}

Question:3

# **Output:**

A screenshot of a computer

Description automatically generated

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Text;

using System.Text.RegularExpressions;

namespace Q3

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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

{

}

private void button1\_Click(object sender, EventArgs e)

{

string firstName = "Maryam,";

string lastName = "Amjad";

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

{

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

return;

}

// Create a StringBuilder to build the password

StringBuilder password = new StringBuilder();

// Add initials of first and last name

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

password.Append(firstName.Substring(1));

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

password.Append(lastName.Substring(1));

// Generate random uppercase alphabet

Random random = new Random();

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

// Generate 4 random numbers

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

{

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

}

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

{

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

}

// Generate 2 more random numbers between '0' and '9' (excluding 0 and 9)

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

{

password.Append((char)random.Next('1', '8' + 1));

}

// Generate 2 special characters

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 20

if (password.Length > 20)

{

password.Length = 20;

}

// Display the generated password

label1.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)

{

}

}

}