

**COMSATS Institute of Information Technology**

**Department of Computer Science, Attock Campus**

**“ ASSIGNMENT 04(Thoery)”**

Name**: ASMA AKHTAR**

**M.SAIFULLAH**

Reg No**: Fa20-BCS-003**

**Fa20-BCS-039**

Course**: HCI(theory)**

PROGRAM**: BS(CS)-6A**

Course Instructor**: Dr.Farman Ali Khan**

Date**:9th -April-2023**

**Q=Describe functioning of regex c# library,give examples of patterns, separators and anchors?**

**Ans=**Regular expressions, often abbreviated as regex or regexp, are powerful tools for pattern matching and manipulation of text in various programming languages, including C#.

In C#, you can use the **System.Text.RegularExpressions** namespace to work with regular expressions. Here's a basic overview of how the regex library in C# functions:

* **Namespace and Classes:**

The **System.Text.RegularExpressions** namespace in C# provides the classes and methods necessary for working with regular expressions. The primary class you'll use is Regex.

* **Creating a Regex Object:**

To work with regular expressions, you first create a Regex object. You pass the pattern you want to match as a parameter to the constructor. For example:

**C# code:**

Regex regex = new Regex(@"\d{3}-\d{2}-\d{4}");

* **Matching:**

You can use the Match method of the Regex object to check if a string matches the regular expression pattern. The Match method returns a Match object, which contains information about the first match found in the input string.

**C# code:**

Match match = regex.Match("123-45-6789");

if (match.Success)

{

// Match found

}

* **Searching for All Matches:**

You can also use the Matches method to find all matches in a string. It returns a MatchCollection containing all the matches.

**C# code:**

MatchCollection matches = regex.Matches("123-45-6789 456-78-9012");

foreach (Match match in matches)

{

// Process each match

}

* **Replacement:**

Regular expressions can be used for replacing text as well. The Regex.Replace method allows you to replace matches with a specified replacement string.

**C# code:**

string input = "My phone number is 123-45-6789.";

string pattern = @"\d{3}-\d{2}-\d{4}";

string replacement = "[REDACTED]";

string result = Regex.Replace(input, pattern, replacement);

**Result**: "My phone number is [REDACTED]."

* **Splitting:**

The Regex.Split method can be used to split a string into an array of substrings based on the regular expression pattern.

**C# code:**

string input = "apple,banana,orange";

string pattern = ",";

string[] parts = Regex.Split(input, pattern);

parts[0] = "apple", parts[1] = "banana", parts[2] = "orange"

* **Regular Expression Patterns:**

C# supports a wide range of regular expression patterns. You can use metacharacters, quantifiers, character classes, and more to define complex patterns. For example, "\d" matches a digit, "\w" matches a word character, and "[A-Za-z]" matches any uppercase or lowercase letter.

**Options:**

You can specify regex options such as case-insensitivity or multiline matching using the RegexOptions enumeration when creating the Regex object.

**C# code:**

Regex regex = new Regex (@"abc", RegexOptions.IgnoreCase);

*The C# regex library is quite powerful and flexible, allowing you to perform various text-processing tasks with ease. However, it's important to note that regular expressions can become quite complex, and writing efficient patterns can be challenging. It's essential to test your regex patterns thoroughly and consider performance implications when working with large input data.*

**Examples of Patterns Anchors and Separators:**

**2. Patterns:**

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

\w: Matches any word character (alphanumeric character plus underscore).

.: Matches any character except for a newline.

[]: Matches any one of the characters inside the brackets. For example, [aeiou] matches any vowel.

+: Matches one or more occurrences of the preceding pattern.

\*: Matches zero or more occurrences of the preceding pattern.

?: Matches zero or one occurrence of the preceding pattern.

^: Matches the start of a string.

$: Matches the end of a string.

**3. Anchors:**

^: Anchors the regex at the start of the string. For example, ^Hello matches a string that starts with "Hello".

$: Anchors the regex at the end of the string. For example, World$ matches a string that ends with "World".

**4. Separators:**

\s: Matches any whitespace character (space, tab, newline).

\S: Matches any non-whitespace character.

\b: Matches a word boundary.

\B: Matches a non-word boundary.