

**Mid Lab**

**Name : Iqra Basharat**

**Registration Number: fa20-bcs-007**

**Subject: CC**

**Submitted To: Sir Bilal**

**Q 01:**

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

**Solution:**

Regular Expressions, often referred to as regex or regexp, are a powerful tool for pattern matching and text manipulation in C# and many other programming languages. In C#, you can use the `System.Text.RegularExpressions` namespace to work with regular expressions. Here's an overview of how the regex library in C# functions, along with some examples of patterns, separators, anchors, and more.

**1. Import the namespace:**

To use regular expressions in C#, you need to import the `System.Text.RegularExpressions` namespace at the beginning of your code file:

using System.Text.RegularExpressions;

**2. Create a regular expression pattern:**

A regular expression pattern is a sequence of characters that defines a search pattern. You can create a pattern as a string and use it to search for matches within a given input text.

string pattern = "abc";

**3. Match regular expressions:**

You can use the `Regex.Match` method to find the first occurrence of a pattern in a given input string.

string input = "abcdefg";

Match match = Regex.Match(input, pattern);

if (match.Success)

{

Console.WriteLine("Match found: " + match.Value);

}

**4. Match regular expressions globally:**

To find all occurrences of a pattern in a string, you can use the `Regex.Matches` method.

string input = "abcdefgabcabc";

MatchCollection matches = Regex.Matches(input, pattern);

foreach (Match match in matches)

{

Console.WriteLine("Match found: " + match.Value);

}

**5. Pattern elements:**

Regex patterns can consist of various elements, including:

- \*\*Literal characters:\*\* These are characters that match themselves. For example, the pattern `"abc"` matches the string `"abc"`.

**Metacharacters:** Special characters with predefined meanings, such as `.` (matches any character), `\*` (matches 0 or more repetitions), `+` (matches 1 or more repetitions), `?` (matches 0 or 1 repetition), `[]` (character class), `()` (capturing group), and more.

**Anchors**:Anchors are used to specify the position in the input where a match should occur. Common anchors are `^` (start of the string) and `$` (end of the string).

Escape sequences: You can use escape sequences like `\d` (matches a digit), `\w` (matches a word character), `\s` (matches whitespace), etc.

**Example Patterns:**

- Matching an email address:

string pattern = @"\b[A-Z0-9.\_%+-]+@[A-Z0-9.-]+\.[A-Z]{2,}\b";

- Matching a date in the format "dd/mm/yyyy":

csharp

string pattern = @"^\d{2}/\d{2}/\d{4}$";

**6. Separators:**

Regex patterns can include separators for capturing specific parts of the match. For example, you can use parentheses to capture parts of the match for later use:

string pattern = @"(\d{2})/(\d{2})/(\d{4})";

Match match = Regex.Match("26/10/2023", pattern);

if (match.Success)

{

string day = match.Groups[1].Value;

string month = match.Groups[2].Value;

string year = match.Groups[3].Value;

}

```

This code will capture the day, month, and year separately from the date string.

Regular expressions in C# offer a flexible and powerful way to search for and manipulate text using patterns. They can be as simple or as complex as needed, depending on the requirements of your specific task.