Logo, company name

Description automatically generated

Lab mid

NAME: MISBAH NAZIR

REG NO: FA20-BCS-015

SUBJECT: Compiler Construction

**Question 1**

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

Regex in C# defines a regular expression in C#. The Regex class offers methods and properties to parse a large text to find patterns of characters. In this article, you’ll learn how to use a Regex class in C#.

**Regex**

Regex, short for regular expression, is a sequence of characters that defines a search pattern. It is a versatile tool used for pattern matching and manipulation of text. With a regular expression, you can specify rules or patterns to search for specific sequences of characters within a larger text. These patterns can include literal characters, metacharacters, character classes, quantifiers, and more. Regular expressions are commonly used for various tasks, such as:

* Pattern matching: Finding and extracting specific sequences of characters in text.
* String validation: Checking if a string conforms to a specific pattern or format.
* Data extraction: Parsing and extracting data from structured text, such as log files or web pages.
* Text manipulation: Replacing or modifying specific patterns in a string.

1. **Pattern Compilation:**
   * Regular expressions are defined as patterns using a specific syntax. The **Regex** class compiles these patterns into a regex object that can be used for matching.

string pattern = @"\d+"; // Match one or more digits Regex regex = new Regex(pattern);

1. **Matching:**
   * The **Match** method is used to find the first occurrence of the pattern in a given input string.

string input = "The price is $100"; Match match = regex.Match(input);

* + The **Success** property of the **Match** object indicates whether a match was found.

1. **Matching All Occurrences:**
   * The **Matches** method returns a collection of all matches in the input string.

MatchCollection matches = regex.Matches(input);

1. **Replace:**
   * The **Replace** method is used to replace occurrences of the pattern with a specified replacement string.

string replaced = regex.Replace(input, "X");

1. **Split:**
   * The **Split** method is used to split a string into an array of substrings based on the pattern.

string[] parts = regex.Split(input);

1. **Options and Modifiers:**
   * Regex options can be set to control the matching behavior, such as case-insensitivity.

Regex regexWithOptions = new Regex(pattern, RegexOptions.IgnoreCase);

1. **Groups:**
   * Parentheses in the pattern define capturing groups, allowing you to extract specific parts of the matched string.

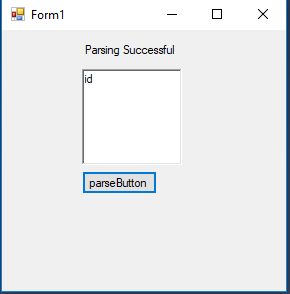
string patternWithGroups = @"(\d{2})-(\d{2})"; Regex regexWithGroups = new Regex(patternWithGroups); Match matchWithGroups = regexWithGroups.Match("22-45"); // Access captured groups string group1 = matchWithGroups.Groups[1].Value; // "22" string group2 = matchWithGroups.Groups[2].Value; // "45"

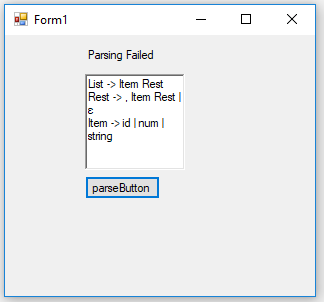
1. **Anchors:**
   * Anchors such as **^** (start of line) and **$** (end of line) help ensure that the pattern matches at specific positions in the string.

string startPattern = @"^start"; string endPattern = @"end$";

**Question 2**

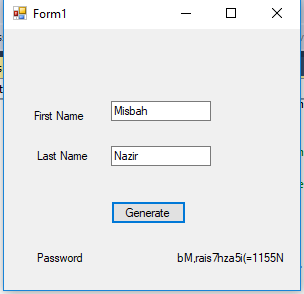
**Output:**

****

****

**Question 3**

**Output:**

****