

**COMSATS UNIVERSITY ISLAMABAD, ATTOCK CAMPUS**

**DEPARTMENT OF COMPUTER SCIENCE**

**LAB MID**

**Submitted to :** Mr.Bilal Haider

**Submitted by :** **Munaza(Fa20-Bcs-077)**

**Question 1**

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

In C#, you can compile regular expressions using the **Regex.CompileToAssembly** method, which generates a dynamic assembly with regular expressions. This feature allows you to store and reuse compiled regex patterns for better performance. It's particularly useful when you have a set of predefined regular expressions that are used frequently.

Here is an example of how to use the **Regex.CompileToAssembly** method:

using System;

using System.Text.RegularExpressions;

class Program

{

static void Main()

{

string pattern = @"\b\d{3}\b"; // Example pattern to match 3-digit numbers

RegexCompilationInfo regexCompilationInfo = new RegexCompilationInfo(

pattern,

RegexOptions.IgnoreCase,

"ThreeDigitRegex",

"Regexes",

true

);

RegexCompilationInfo[] compilationArray = { regexCompilationInfo };

Regex.CompileToAssembly(compilationArray, new System.Reflection.AssemblyName("RegexAssembly"));

}

}

In this example, the **Regex.CompileToAssembly** method is used to compile a regular expression pattern that matches three-digit numbers. Here's a breakdown of the parameters used:

* **pattern**: The regular expression pattern.
* **RegexOptions**: Options that modify the pattern matching behavior (e.g., case insensitivity).
* **assemblyName**: The name of the assembly to be generated.
* **groupName**: The name of the group within the assembly that contains the regular expression.

After compiling the regular expression to an assembly, you can then use it in your C# projects without having to recreate the regex pattern each time, which can improve performance.

Regular expressions in the compile construction follow the same syntax rules as other regular expressions in C#. Here are some common elements:

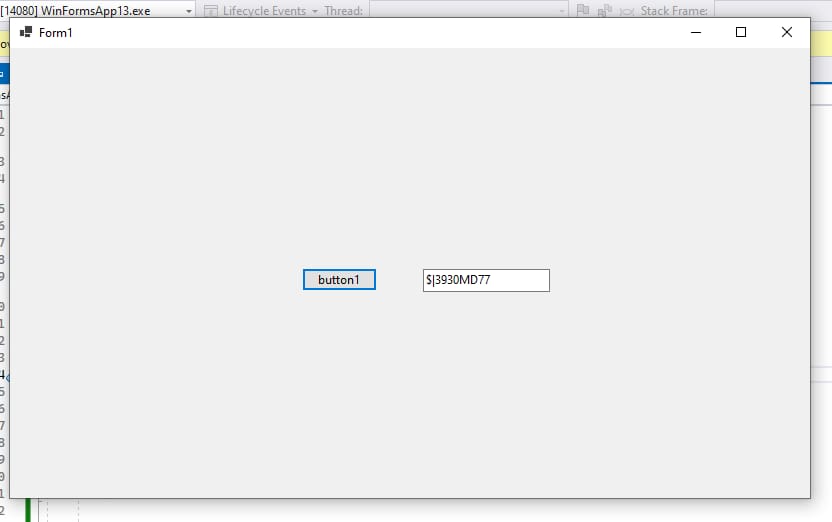
* **Anchors**: **^** for the start of a line, **$** for the end of a line.
* **Quantifiers**: **\*** for zero or more occurrences, **+** for one or more occurrences, **?** for zero or one occurrence.
* **Character Classes**: **[...]** to match any one of the characters within the brackets.
* **Grouping Constructs**: **( )** for creating groups, allowing quantifiers to be applied to them.
* **Escape Characters**: **\** to escape special characters.

Example patterns can be like:

* Matching a date in the format "YYYY-MM-DD": **@"^\d{4}-\d{2}-\d{2}$"**
* Matching a valid email address: **@"^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$"**
* Matching a phone number with a specific format: **@"^\(\d{3}\) \d{3}-\d{4}$"**

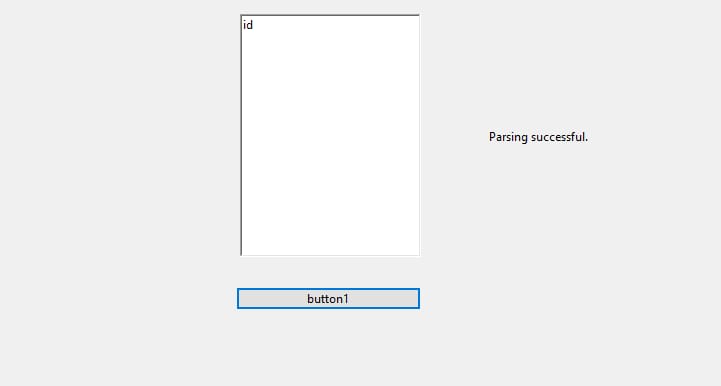
**Question:03**

**Output:**

****

**Question:02**

**Output:**

****