# Lab: Regular Expressions C#

Problems for in-class lab for the ["C# Fundamentals" course @ SoftUni](https://softuni.bg/trainings/3836/programming-fundamentals-with-csharp-september-2022)  
You can check your solutions in [Judge](https://judge.softuni.org/Contests/1667/Regular-Expressions-Lab)

## Match Full Name

Write a C# Program to **match full names** from a list of names and **print** them on the console.

### Writing the Regular Expression

First, create a regular expression to match a valid full name, according to these conditions:

* A valid full name has the following characteristics:
  + It consists of **two words**.
  + Each word **starts** with a **capital letter**.
  + After the first letter, it **only contains lowercase letters afterward**.
  + **Each** of the **two words** should be **at least two letters long**.
  + The **two words** are **separated** by a **single space**.

using System;

using System.Text.RegularExpressions;

namespace \_01.\_Match\_Full\_Name

{

class Program

{

static void Main(string[] args)

{

string pattern = @"\b[A-Z][a-z]+ [A-Z][a-z]+\b";

string input = Console.ReadLine();

//Creating regex pattern

Regex regex = new Regex(pattern);

//Find all matches with this pattern in the input

MatchCollection matches = regex.Matches(input);

//print all matches (type Match)

foreach (Match item in matches)

{

Console.Write(item.Value + " ");

}

}

}

}

To help you out, we've outlined several steps:

1. Use an online regex tester like <https://regex101.com/>.
2. Check out how to use **character sets** (denoted with square brackets - "[]").
3. Specify that you want **two words** with a space between them (the **space character '** **'** and **not** any whitespace symbol).
4. For each word, specify that it should begin with an uppercase letter using a **character set**. The desired characters are in a range – **from** '**A**' **to** '**Z**'.
5. For each word, specify that what follows the first letter are only **lowercase letters**, one or more – use another character set and the correct **quantifier**.
6. To prevent capturing of letters across new lines, put "\b" at the beginning and the end of your regex. This will ensure that what precedes and what follows the match is a word boundary (like a new line).

To check your RegEx, use these values for reference (paste all of them in the **Test String** field):

|  |  |
| --- | --- |
| **Match ALL of these** | **Match NONE of these** |
| Bethany Taylor John Smith | Bethany Taylor, Oliver miller, sophia Johnson, SARah Wilson, John Smith, Sam Smith |

By the end, the matches should look something like this:



After you've constructed your regular expression, it's time to write the solution in C#.

### Implementing the Solution in C#

Create a new C# project and copy your **regular expression** into a string variable:



Note: It's usually a good idea to use a **verbatim string** (**@** in front of the string literal) to store **regular expressions** since characters like the backslash **'**\**'** can clash with **string escaping**.

Now, it's time to **read the input** and **extract all the matches** from it. For this, we can use the MatchCollectionclass:



After we extract all the matches, we need to **iterate** over the MatchCollection and **print** every match that we found:



### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Bethany Taylor, Oliver miller, sophia Johnson, SARah Wilson, John Smith, Sam Smith | Bethany Taylor John Smith |
| Elvis Presley a a C C-Muhammad Ali EE PeterpeterJR-a a xi ban D D bb b b-B B-c c EE-Michael Jackson DD peter smith B B PETER BROWN IVAN DAVIES- | Elvis Presley Muhammad Ali Michael Jackson |

## Match Phone Number

Create a regular expression to match a **valid phone number** from **Sofia**. After you find all **valid phones**, **print** them on the console, separated by a **comma and a space** ", ".

using System;

using System.Linq;

using System.Text.RegularExpressions;

namespace \_02.\_Match\_Phone\_Number

{

class Program

{

static void Main(string[] args)

{

string pattern = @"\+\b359( |-)2\1\d{3}\1\d{4}\b";

string input = Console.ReadLine();

Regex regex = new Regex(pattern);

MatchCollection matches = regex.Matches(input);

//foreach (Match item in matches)

//{

// Console.Write(item.Value+", ");

//}

string[] result = matches.Select(x => x.Value).ToArray();

Console.WriteLine(string.Join(", ",result));

}

}

}

### Compose the Regular Expression

A valid number has the following characteristics:

* It starts with "**+359**"
* Then, it is followed by the area code (always **2**)
* After that, it's followed by the **number** itself:
  + The number consists of **7 digits** (separated into **two** **groups** of **3** and **4** **digits** respectively).
* The different **parts** are **separated** by **either a space or a hyphen** ('**-**').

You can use the following RegEx properties to **help** with the matching:

* Use **quantifiers** to match a **specific number** of **digits**
* Use a **capturing group** to make sure the delimiter is **only one of the allowed characters** **(space or hyphen)** and **not** a **combination** of both (e.g. +359 2-111 111 has **mixed delimiters**, it is **invalid**). Use a **group backreference** to achieve this.
* Add a **word boundary** at the **end** of the match to avoid **partial matches** (the last example on the right-hand side).
* Ensure that before the **'+'** sign there is either a **space** or the **beginning of the string**.

You can use the following table of values to test your RegEx against:

|  |  |
| --- | --- |
| **Match ALL of these** | **Match NONE of these** |
| +359 2 222 2222  +359-2-222-2222 | 359-2-222-2222, +359/2/222/2222, +359 2 222 2222  +359 2-222-2222, +359-2-222-222, +359-2-222-22222 |

### Implement the Solution in C#

Now it's time to write the solution, so let's start writing!

First, just like in the previous problem, put your RegEx in a variable:



After that, let's make a MatchCollection for our matches:



Let's try to print **all the matches**, using only a **single line** **of code**. Since MatchCollection is, as its name suggests, a **collection**, we can use LINQ methods on it.

To get all of the matches and put them into a string array, we need to perform several manipulations on the MatchCollection:

1. Cast every single element of the MatchCollection to the Match type using Cast<Match>().
2. Since every element is a Match now, we can extract just the Valueproperty of the match itself, which holds the **match value** as a **string**, using Select(). We can also Trim() **the value**, to get rid of any **leading** or **trailing spaces**.
3. After getting the match value, we can use ToArray() to **convert** the collection to an **array**.

Here's what that looks like as a LINQ query:



After that, just print the valid phone number array, using string.Join():



### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| +359 2 222 2222,359-2-222-2222, +359/2/222/2222, +359-2 222 2222 +359 2-222-2222, +359-2-222-222, +359-2-222-22222 +359-2-222-2222 | +359 2 222 2222, +359-2-222-2222 |
| +359 2 234 2324, 359-2-111-9876, +359/8/655/5432, +359-2 222 2222, +359 2-222-2222, +359-2-234-345, +359-2-123-45678, +359-2-222-2222, +359 2 654 1234 | +359 2 234 2324, +359-2-222-2222, +359 2 654 1234 |

## Match Dates

Write a program, which matches a date in the format "dd{separator}MMM{separator}yyyy". Use **named** **capturing groups** in your regular expression.

### Compose the Regular Expression

Every valid date has the following characteristics:

* Always starts with **two digits**, followed by a **separator.**
* After that, it has **one uppercase** and **two lowercase** letters (e.g. Jan, Mar).
* After that, it has a **separator** and **exactly 4 digits** (for the year).
* The separator could be either of three things: a period (**'**. **'**), a hyphen (**'**-**'**) or a forward-slash (**'**/**'**).
* The separator needs to be **the same** for the whole date (e.g. 13.03.2016 is valid, 13.03/2016 is **NOT**). Use a **group backreference** to check for this.

using System;

using System.Text.RegularExpressions;

namespace \_03.\_Match\_Dates

{

class Program

{

static void Main(string[] args)

{

string pattern = @"\b(?<day>\d{2})(\.|-|/)(?<month>[A-Z][a-z]{2})\1(?<Year>\d{4})\b";

string input = Console.ReadLine();

Regex regex = new Regex(pattern);

MatchCollection matches = regex.Matches(input);

foreach (Match item in matches)

{

string day = item.Groups[2].Value;

string month = item.Groups[3].Value;

string year= item.Groups[4].Value;

Console.WriteLine($"Day: {day}, Month: {month}, Year: {year}");

}

}

}

}

You can follow the table below to help with composing your RegEx:

|  |  |
| --- | --- |
| **Match ALL of these** | **Match NONE of these** |
| 13/Jul/1928, 10-Nov-1934, 25.Dec.1937 | 01/Jan-1951, 23/sept/1973, 1/Feb/2016 |

Use **named capturing groups** for the **day**, **month** and **year**.

Since this problem requires more complex RegEx, which includes **named capturing groups**, we'll take a look at how to construct it:

* First off, we don't want anything at the **start** of our date, so we're going to use a **word boundary** "\b":  
  
* Next, we're going to match the **day** by telling our RegEx to match **exactly two digits**,and since we want to **extract** the day from the match later, we're going to put it in a **capturing group**:  
    
  We're also going to give our group a **name** since it's easier to navigate by **group name** than by **group index**:  
  
* Next comes the separator – either a **hyphen**, **period** or **forward slash**. We can use a **character class** for this:  
    
  Since we want to use the separator we matched here, to match the **same separator** further into the date, we're going to put it in a **capturing group**:  
  
* Next comes the **month**, which consists of a **capital Latin letter** and **exactly two lowercase Latin letters**:
* Next, we're going to match the **same separator** **we matched earlier**. We can use a **backreference** for that:  
  
* Next up, we're going to match the year, which consists of **exactly 4 digits**:  
  
* Finally, since we don't want to match the date if there's anything else **glued to it**, we're going to use another **word boundary** for the end:  
  

Now it's time to find all the **valid dates** in the input and **print each date** in the following format: "Day: {day}, Month: {month}, Year: {year}", each on a **new line**.

### Implement the Solution in C#

First off, we're going to put our RegEx in a variable and get a MatchCollection from the string:



Since RegEx works differently across different languages, before we continue, we're going to **set our backreference from** \2 **to** \1. This is because **C# backreferences** don't count **named capture groups for backreferences**. So, **change it before we continue**.

Next, we're going to **iterate** over every single Match and **extract** the **day**, **month** and **year** from the **groups**. We can use a special syntax in C# to get a match's group **value** by its **key**, the **same way** as when we access a Dictionary's values:



### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 13/Jul/1928, 10-Nov-1934, , 01/Jan-1951,f 25.Dec.1937 23/09/1973, 1/Feb/2016 | Day: 13, Month: Jul, Year: 1928  Day: 10, Month: Nov, Year: 1934  Day: 25, Month: Dec, Year: 1937 |
| 03-Mar-1878, 25/Apr/1915, 31-May-1916, 22/Jun-1941, 25.Dec.1937, 23/09/1973 | Day: 03, Month: Mar, Year: 1878  Day: 25, Month: Apr, Year: 1915  Day: 31, Month: May, Year: 1916  Day: 25, Month: Dec, Year: 1937 |