**Lab: LINQ**

Problems for exercises and homework for the [“C# Advanced” course @ SoftUni".](https://softuni.bg/courses/csharp-advanced)

Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/598/LINQ-Lab.>

**1.   Take Two**

Read a **sequence of integers**, given on a single line separated by a space.

Finds all **unique** elements, that are in range [**10, 20] inclusive** and print only **the first 2 elements**.

If there are **fewer than 2 elements**, print as much as there are. If there are **no elements**, print nothing.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 15 2 15 14 12 | 15 14 |
| 17 -2 3 | 17 |
| -2 3 | (*no output)* |

**Hints**

* Read the input and parse the strings to a list of numbers:
* Filter the numbers with Where() and lambda expression, take the unique ones with Distinct(), take only two from the collection with Take() and iterate over them while printing with ForEach():

**2.   Upper Strings**

Read a sequence of strings, given on a single line separated with a space.

Map each to upper case and print them, using LINQ.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| Pesho Gosho Stefan | PESHO GOSHO STEFAN |
| Soft Uni Rocks | SOFT UNI ROCKS |
| *(empty line)* | *(no output)* |

**Hints**

* Read the input into a list of strings List<string>:
* Use the Select() method to change every element of the list of words to uppercase. Iterate over the words and print the result:

**3.   First Name**

Read a **sequence of names**, given on a single line, separated by a space.

Read a **sequence of letters**, given on the next line, separated by a space.

Find the **names that start with one of the given letters** and print the first of them (**ordered lexicographically**).

If there is **no name** that conforms to the requirement, **print "No match"**.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| Rado Plamen Gosho  p r | Plamen |
| Plamen Gosho Rado  s c | No match |
| plamen gosho rado  r d e g | Gosho |

**Hints**

* You can use a hash set to store letter, for efficient searching. Or just simply sort the letters but it will perform poorly
* Make sure you are comparing letter with the same casing (lower or upper)
* Use Where() and First() or FirstOrDefault()

**4.   Average of Doubles**

Read a **sequence of double numbers**, given on a single line, separated by a space.

Find the **average of all elements**, using LINQ.

**Round** the output to the second digit after the decimal separator.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3 4 5 6 | 4.50 |
| 3.14 5.2 6.18 | 4.84 |

**5.   Min Even Number**

Read a sequence of numbers, given on a single line, separated by a space.

Find the smallest number of all even numbers, using LINQ.

If there are **no numbers** in the sequence, **print "No match"**.

Numbers in the output should be formatted with **2 decimal** **places** after floating point.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 2 3 4 5 6 | 2.00 |
| 3.14 -2.00 1.33 | -2.00 |
| *1 3* | No match |

**Hints**

* Use select function to map the objects to double
* Make sure to filter empty strings
* Filter the even numbers
* Get the smallest number using Min()

**6.   Find and Sum Integers**

Read a sequence of elements, given on a single line, separated by a space.

Filter all elements that are integers and calculate their sum, using LINQ.

If there are **no numbers** in the sequence, **print "No match"**.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| Sum 3 and 4 | 7 |
| Sum -3 and -4 | -7 |
| Sum three and four | No match |
| cat dog 1 catch | 1 |
| cat 1 dog -1 min | 0 |

**7.   Bounded Numbers**

On the first line, read two numbers, a **lower** and **an upper bound**, separated by a space. The bigger number is the upper bound and the smaller number is the lower bound.

On the second line, read a sequence of numbers, separated by a space.

Print all numbers, such that [lower bound] ≤ n ≤ [upper bound].

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5 7  1 2 3 4 5 6 7 8 9 | 5 6 7 |
| 7 5  9 5 7 2 6 8 | 5 7 6 |
| 3 4  5 6 7 8 | *(no output)* |

**8.   Map Districts**

On the first line, you are given the population of districts in different cities, separated by a single space in the format "city:district population".

On the second line, you are given the minimum population for filtering of the towns. The **population of a town** is the **sum of populations of all of its districts**.

**Print** all **cities** with population greater than a given number on the second line. **Sort** **cities** **and districts** by descending population and **print top 5 districts for a given city**.

For a better understanding, see the examples below.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| Pld:9 Pld:13 Has:7 Sof:20 Sof:10 Sof:15  10 | Sof: 20 15 10  Pld: 13 9 |
| Sof:10 Sof:12 Sof:15  10 | Sof: 15 12 10 |
| Sof:5  15 | *(no output)* |