# Exercises: Lists

Problems for exercises and homework for the [“Programming Fundamentals with C#” course @ SoftUni](https://softuni.bg/trainings/1764/fundamentals-of-programming-with-csharp).

You can check your solutions here: <https://judge.softuni.bg/Contests/Compete/Index/867#0>.

## Remove Negatives and Reverse

Read a **list of integers**, **remove all negative numbers** from it and print the remaining elements in **reversed order**. In case of no elements left in the list, print “empty”.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 10 -5 7 9 -33 50 | 50 9 7 10 |
| 7 -2 -10 1 | 1 7 |
| -1 -2 -3 | empty |

### Hints

* Read a text line from the console, split it by space, parse the obtained items as integers and convert them to list of integers.
* Create a new empty list for the results.
* Scan the input list from the end to the beginning. Check each element and append all non-negative elements to the result list.
* Finally, print the results list (at a single line holding space-separated numbers).

## Append Lists

Write a program to **append several lists** of numbers.

* Lists are separated by ‘|’.
* Values are separated by spaces (‘ ’, one or several)
* Order the lists from the **last** to the **first**, and their values from **left** to **right**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 2 3 |4 5 6 | 7 8 | 7 8 4 5 6 1 2 3 |
| 7 | 4 5|1 0| 2 5 |3 | 3 2 5 1 0 4 5 7 |
| 1| 4 5 6 7 | 8 9 | 8 9 4 5 6 7 1 |

### Hints

* Create a new empty list for the results.
* Split the input by ‘|’ into array of tokens.
* Pass through each of the obtained tokens from tight to left.
  + For each token, split it by space and append all non-empty tokens to the results.
* Print the results.

## Sum Adjacent Equal Numbers

Write a program to **sum all adjacent equal numbers** in a list of decimal numbers, starting from **left to right**.

* After two numbers are summed, the obtained result could be equal to some of its neighbors and should be summed as well (see the examples below).
* Always sum the **leftmost** two equal neighbors (if several couples of equal neighbors are available).

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Explanation** |
| 3 3 6 1 | 12 1 | **3 3** 6 1 🡪 **6 6** 1 🡪 12 1 |
| 8 2 2 4 8 16 | 16 8 16 | 8 **2 2** 4 8 16 🡪 8 **4 4** 8 16 🡪 **8 8** 8 16 🡪 16 8 16 |
| 5 4 2 1 1 4 | 5 8 4 | 5 4 2 **1 1** 4 🡪 5 4 **2 2** 4 🡪 5 **4 4** 4 🡪 5 8 4 |

### Hints

1. Read the **input** and parse it to **list of numbers**.
2. Find the **leftmost** two **adjacent equal cells**.
3. **Replace** them with their **sum**.
4. **Repeat** (1) and (2) until no two equal adjacent cells survive.
5. **Print** the processed list of numbers.

## Split by Word Casing

Read a **text**, split it into words and distribute them into **3 lists**.

* **Lower-case words** like “programming”, “at” and “databases” – consist of lowercase letters only.
* **Upper-case words** like “PHP”, “JS” and “SQL” – consist of uppercase letters only.
* **Mixed-case words** like “C#”, “SoftUni” and “Java” – all others.

Use the following **separators** between the words: , ; : . ! ( ) " ' \ / [ ] space

Print the 3 lists as shown in the example below.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Learn programming at SoftUni: Java, PHP, JS, HTML 5, CSS, Web, C#, SQL, databases, AJAX, etc. | Lower-case: programming, at, databases, etc  Mixed-case: Learn, SoftUni, Java, 5, Web, C#  Upper-case: PHP, JS, HTML, CSS, SQL, AJAX |

### Hints

* **Split** the input text using the above described **separators**.
* **Process** the obtained **list of words** one by one.
* Create 3 lists of words (initially empty): lowercase words, mixed-case words and uppercase words.
* Check each word and append it to one of the above 3 lists:
  + Count the **lowercase letters** and **uppercase letters**.
  + If all letters are **lowercase**, append the word to the lowercase list.
  + If all letters are **uppercase**, append the word to the uppercase list.
  + Otherwise the word is considered mixed-case 🡪 append it to the mixed-case list.
* Print the obtained 3 lists as shown in the example above.

## Sort Numbers

Read a **list of decimal numbers** and **sort** them in increasing order. Print the output as shown in the examples below.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 8 2 7 3 | 2 <= 3 <= 7 <= 8 |
| 2 4 -9 | -9 <= 2 <= 4 |

### Hints

* Use the built-in method List<T>.Sort().

## Square Numbers

Read a **list of integers** and **extract all square numbers** from it and print them in **descending order**. A **square number** is an integer which is the square of any integer. For example, 1, 4, 9, 16 are square numbers.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3 **16 4** 5 6 8 **9** | 16 9 4 |
| 12 **1 9 4 16** 8 **25 49 16** | 49 25 16 16 9 4 1 |

### Hints

* To find out whether an integer is “**square number**”, check whether its square root is integer number (has no fractional part):
  + if (√num == (int)√num) …
* To order the results list in descending order use sorting by lambda function:
  + squareNums.Sort((a, b) => b.CompareTo(a));

## Search for a Number

On the **first** **line**, you will receive a **list** of **integers**. On the **next** line, you will receive an **array** with exactly **three** **numbers**. **First** number represents the **number** of **elements** you have to **take** from the **list** (**starting** from the **first** **one**). **Second** number represents the **number** of **elements** you have to **delete** from the numbers you took (**starting** from the **first** **one**). **Last** **number** is the **number** we search in our **collection** after the manipulations. If it is present print: “**YES!**”, otherwise print “**NO!**”

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 1 2 3 4 5 6  5 2 3 | YES! | 12 412 123 21 654 34 65 3 23  7 4 21 | NO! |

## Sum Reversed Numbers

Write a program that reads sequence of numbers, reverses their digits, and prints their sum.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 123 234 12 | 774 | 321 + 432 + 21 = 774 |
| 12 12 34 84 66 12 | 220 | 21 + 21 + 43+ 48 + 66 + 21 = 220 |
| 120 1200 12000 | 63 | 21 + 21 + 21 = 63 |

## Bomb Numbers

Write a program that **reads sequence of numbers** and **special bomb number** with a certain **power**. Your task is to **detonate every occurrence of the special bomb number** and according to its power **his neighbors from left and right**. Detonations are performed from left to right and all detonated numbers disappear. Finally print the **sum of the remaining elements** in the sequence.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 1 2 2 4 2 2 2 9  4 2 | 12 | Special number is **4** with power 2. After detontaion we left with the sequence [1, 2, 9] with sum 12. |
| 1 4 4 2 8 9 1  9 3 | 5 | Special number is **9** with power 3. After detontaion we left with the sequence [1, 4] with sum 5. Since the 9 has only 1 neighbour from the right we remove just it (one number instead of 3). |
| 1 7 7 1 2 3  7 1 | 6 | Detonations are performed from left to right. We could not detonate the second occurance of 7 because its already destroyed by the first occurance. The numbers [1, 2, 3] survive. Their sum is 6. |
| 1 1 2 1 1 1 2 1 1 1  2 1 | 4 | The red and yellow numbers disappear in two sequential detonations. The result is the sequence [1, 1, 1, 1]. Sum = 4. |

## \*Array Manipulator

Write a program that **reads an array of integers** from the console and **set of commands** and **executes them over the array**. The commands are as follows:

* **add <index> <element>** – adds element at the specified index (elements right from this position inclusively are shifted to the right).
* **addMany <index> <element 1> <element 2> … <element n>** – adds a set of elements at the specified index.
* **contains <element>** – prints the index of the first occurrence of the specified element (if exists) in the array or **-1** if the element is not found.
* **remove <index>** – removes the element at the specified index.
* **shift <positions>** – **shifts every element** of the array the number of positions **to the** **left** (with rotation).
  + For example, [1, 2, 3, 4, 5] -> shift 2 -> [3, 4, 5, 1, 2]
* **sumPairs** – sums the elements in the array by pairs (first + second, third + fourth, …).
  + For example, [1, 2, 4, 5, 6, 7, 8] -> [3, 9, 13, 8].
* **print** – stop receiving more commands and print the last state of the array.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 2 4 5 6 7  add 1 8  contains 1  contains -3  print | 0  -1  [1, 8, 2, 4, 5, 6, 7] |
| 1 2 3 4 5  addMany 5 9 8 7 6 5  contains 15  remove 3  shift 1  print | -1  [2, 3, 5, 9, 8, 7, 6, 5, 1] |
| 2 2 4 2 4  add 1 4  sumPairs  print | [6, 6, 6] |
| 1 2 1 2 1 2 1 2 1 2 1 2  sumPairs  sumPairs  addMany 0 -1 -2 -3  print | [-1, -2, -3, 6, 6, 6] |