Lab: Lists

Problems for exercises and homework for the [“Programming Fundamentals” course @ SoftUni](https://softuni.bg).

You can check your solutions here: https://judge.softuni.bg/Contests/397/Lists-Lab.

1. 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).

2. 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.

3. 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.

4. 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.

5. 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()`**.

6. 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));`**

7. Count Numbers

Read a **list of integers** in range [0…1000] and **print them in ascending order** along with their **number of occurrences**.

Examples

Input	Output

`8 2 2 8 2 2 3 7`	`2 -> 4`
	`3 -> 1`
	`7 -> 1`
	`8 -> 2`

`10 8 8 10 10`	`8 -> 2`
	`10 -> 3`

Hints

Several algorithms can solve this problem:

- Use an **array `count[0…1000]`** to count in **`counts[x]`** the occurrences of each element **x**.

- **Sort** the numbers and count occurrences of each number.
- Use a dictionary **`counts<int, int>`** to count in **`counts[x]`** the occurrences of each element **x**.

Counting Occurrences Using Array

1. Read the input elements in array of integers **`nums[]`**.
2. Assume **max** holds the largest element in **`nums[]`**.
    - **`max = nums.Max()`**
3. Allocate an array **`counts[0 … max+1]`**.
    - It will hold for each number **x** its number of occurrences **`counts[x]`**.
4. **Scan** the input elements and for each element **x** increase **`counts[x]`**.
    - For each value **v** in [**`0`**…**`max`**], print **v `->` `count[v]`**.
    - Skip all values which do not occur in **`nums[]`**, i.e. **`counts[v]`** == **`0`**.

This algorithm has a **serious drawback**:

- It depends on **mapping numbers to array indexes**.
- It will work well for input values in the range [0…1000].
- It will **not work** for very large and very small values, e.g. if the input holds -100 000 000 or 100 000 000.
- It will **not work** for real numbers, e.g. 3.14 or 2.5.

Counting Occurrences by After Sorting

1. Read the input elements in array of integers **`nums[]`**. Example: {8, 2, 2, 8, 2, 2, 3, 7}.
2. Sort **`nums[]`** in increasing order: {2, 2, 2, 2, 3, 7, 8, 8}. Now find all subsequences of equal numbers.
3. **Scan** the numbers from left to right. Count how many times each number occurs.
    - Start at **`count`** = **`1`**.
    - While the next number on the right is **the same** as the current number, **increase `count`** and proceed to the next number.
    - When the next number on the right is **different** (or there is no next number), **print** the current number and its count.
    - Continue scanning from the next number on the right.

This algorithm will work correctly for real numbers and very large numbers. It does not depend on mapping numbers to array indexes.

Counting Occurrences with Dictionary

Dictionaries map some input **key** (e.g. string, integer or real number) to some **value**. Using a dictionary, we can map an input number **x** to a dictionary value **`dict[x]`** and apply the first occurrence counting algorithm. Learn more about dictionaries at https://www.dotnetperls.com/dictionary. In the next lesson you will learn how to use dictionaries.