Lab: Lists

Problems for in-class lab for the "Programming Fundamentals: Arrays and Lists" course from the official "Applied Programmer" curriculum.

You can check your solutions in https://judge.softuni.bg/Contests/2912.

1. 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

Solution

Read a list of numbers.

```
List<double> nums = Console.ReadLine()
    .Split()
    .Select(double.Parse)
    .ToList();
```

Iterate through the elements. Check if the number at the **current index** is **equal** to the **next** number. If it is, **aggregate the numbers** and **reset** the loop, otherwise **don't do anything**.

```
if (nums[i] == nums[i + 1])
{
    nums[i] += nums[i + 1];
    nums.RemoveAt(i + 1);
    i = -1;
}
```

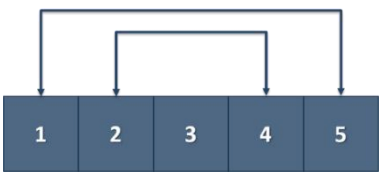
Finally, you have to print the numbers **joined by a single space**.

```
Console.WriteLine(string.Join(" ", nums));
```

2. Gauss' Trick

Write a program that **sums** all of the **numbers in a list** in the following order:

first + last, first + 1 + last - 1, first + 2 + last - 2, … first + n, last - n.



Example

Input	Output
1 2 3 4 5	6 6 3
1 2 3 4	5 5

3. Merging Lists

You are going to receive two lists with numbers. Create a result list, which contains the numbers from both of the lists. The **first element** should be **from the first list**, the **second** from the **second list** and so on. If the length of the two lists **are not equal**, just **add** the **remaining elements** at the **end** of the list.

Example

Input	Output
3 5 2 43 12 3 54 10 23 76 5 34 2 4 12	3 76 5 5 2 34 43 2 12 4 3 12 54 10 23
76 5 34 2 4 12 3 5 2 43 12 3 54 10 23	76 3 5 5 34 2 2 43 4 12 12 3 54 10 23

Hint

- Read the two lists.
- Create a result list.
- Start looping through them until you reach the end of the smallest one.
- Finally add the remaining elements (if there are any) to the end of the list.

4. List of Products

Read a number **n** and **n lines of products**. Print a **numbered list** of all the products **ordered by name**.

Examples

Input	Output
4 Potatoes Tomatoes Onions Apples	1.Apples 2.Onions 3.Potatoes 4.Tomatoes

Solution

First, we need to read the number **n** from the console.

```
int n = int.Parse(Console.ReadLine());
```

Then we need to create our **list of strings**, because the **products are strings.**

```
List<string> products = new List<string>();
```

Then we need to iterate **n times** and **read our current product**.

```
for (int i = 0; i < n; i++)
{
    string currentProduct = Console.ReadLine();
```

The next step is to **add** the current product to the list.

```
    products.Add(currentProduct);
}
```

After we finish reading the products, we **sort our list alphabetically.**

```
products.Sort();
```

- The **sort method** sorts the list in ascending order.

Finally, we have to **print our sorted** list. To do that we **loop through the list**.

```
for (int i = 0; i < products.Count; i++)
{
    Console.WriteLine($"{i + 1}.{products[i]}");
}
```

- We use **i + 1**, because we want to **start counting from 1**, we put the **'.'**, and **finally** we put **the actual product.**

5. 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 there are 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

Solution

Read a list of **integers**.

```
List<int> nums = Console.ReadLine()
    .Split()
    .Select(int.Parse)
    .ToList();
```

Remove **all negative** numbers.

```
nums.RemoveAll(n => n < 0);
```

If the list count is **equal to 0** print **"empty"**, otherwise print all numbers **joined** by **space**.

```
if (nums.Count == 0)
{
    Console.WriteLine("empty");
}
else
{
    Console.WriteLine(String.Join(" ", nums));
}
```

6. List Manipulation Basics

Write a program that reads a list of integers. Then until you receive **"end"**, you will receive different **commands:**

Add {number}: add a number to the end of the list.
Remove {number}: remove a number from the list.
RemoveAt {index}: remove a number at a given index.
Insert {number} {index}: insert a number at a given index.

Note: All the indices will be valid!

When you receive the **"end"** command, print the **final state** of the list (**separated by spaces**).

Example

Input	Output
4 19 2 53 6 43 Add 3 Remove 2 RemoveAt 1 Insert 8 3 end	4 53 6 8 43 3

Solution

First let us read the list from the console.

```
List<int> nums = Console.ReadLine()
    .Split()
    .Select(int.Parse)
    .ToList();
```

- We **split** the string we have received from the console, then we **loop through each of the elements** and parse them to **integers.**
- This returns **IEnumarable<int>** (a **collection** of integers) and we have to keep it in the form of a list.

Next, we go through the input using a **while loop** and a **switch case** statement for the **different** commands.

```
while (true)
{
    string command = Console.ReadLine();

    if (command == "end")
    {
        break;
    }

    string[] elements = command.Split();
```

- We stop the cycle if the line is end, otherwise we **split** the input string into **tokens**.

```
    switch (elements[0])
    {
        case "Add":
            break;
        case "Remove":
            break;
        case "RemoveAt":
            break;
        case "Insert":
            break;
    }
}
```

Now, let us implement **each** of the **commands**.

```
case "Add":
    int numberToAdd = int.Parse(elements[1]);
    nums.Add(numberToAdd);
    break;
case "Remove":
    int numberToRemove = int.Parse(elements[1]);
    nums.Remove(numberToRemove);
    break;
case "RemoveAt":
    int indexToRemove = int.Parse(elements[1]);
    nums.RemoveAt(indexToRemove);
    break;
case "Insert":
    int numberToInsert = int.Parse(elements[1]);
    int indexToInsert = int.Parse(elements[2]);
    nums.Insert(indexToInsert, numberToInsert);
    break;
```

- For each of the commands, **except "Insert", tokens[1]** is the **number/index.**
- For the **"Insert"** command we receive a **number and an index** (**tokens[1], tokens[2]**).

Finally, we **print** the numbers, joined by **a single space.**

```
Console.WriteLine(string.Join(" ", nums));
```

7. List Manipulation Advanced

Next, we are going to implement more complicated list commands, **extending the previous task**. Again, read a list and keep reading commands until you receive **"end"**:

Contains {number} – check if the list contains the number and if so - print **"Yes"**, **otherwise** print **"No such number"**.
PrintEven – print **all the even numbers**, **separated by a space**.
PrintOdd – print **all the odd numbers, separated by a space**.
GetSum – print the **sum of all the numbers**.
Filter {condition} {number} – print all the numbers that **fulfill the given condition**. The condition will be either '**<**', '**>**', "**>=**", "**<=**".

After the **end command,** print the list **only if** you have made some **changes** to the **original list**. **Changes** are made **only** from the commands from the **previous task**.

Example

Input	Output
2 13 43 876 342 23 543 Contains 100 Contains 543 PrintEven PrintOdd GetSum Filter >= 43 Filter < 100 end	No such number Yes 2 876 342 13 43 23 543 1842 43 876 342 543 2 13 43 23