# Exercise: Lists

Test your solutions in the **judge system**: <https://alpha.judge.softuni.org/contests/lists-exercise/5090>

## Train

On the first line, we will receive a **list of wagons** (integers). Each **integer** represents the **number of passengers** that are currently in each wagon of the passenger train. On the next line, you will receive the **max capacity of a wagon**, represented as a **single integer**. **Until** you receive the "**end**" command, you will be receiving two types of input:

* **Add** {**passengers**} – add a wagon to the end of the train with the given number of passengers.
* {**passengers**} – **find a single wagon** to fit all the incoming passengers (starting from the first wagon).

In the end, **print** the final state of the train (all the wagons separated by a space).

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| 32 54 21 12 4 0 23  75  Add 10  Add 0  30  10  75  end | 72 54 21 12 4 75 23 10 0 |
| 0 0 0 10 2 4  10  Add 10  10  10  10  8  6  end | 10 10 10 10 10 10 10 |

## House Party

Create a program that keeps track of the guests that are going to a house party.

On the first line, of input you are going to receive **the number of commands that will follow**.

On the next lines, you are going to receive some of the following:

"**{name} is going!**"

* You have to **add the person, if they are not on the guestlist already.**
* If **the person is on the list** print the following to the console: "**{name} is already in the list!**"

"**{name} is not going!**"

* You have to remove the person, if they are on the list.
* If not, print out: "**{name} is not in the list!**"

Finally, print all of the guests, each on a new line.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4  Allie is going!  George is going!  John is not going!  George is not going! | John is not in the list!  Allie |
| 5  Tom is going!  Annie is going!  Tom is going!  Garry is going!  Jerry is going! | Tom is already in the list!  Tom  Annie  Garry  Jerry |

## List Operations

The first input line will hold a list of **integers**. Until we receive the "**End**" command, we will be given **operations** we have to apply to the list.

The **possible commands** are:

* **Add {number}** – add the given number to the end of the list
* **Insert {number} {index} –** insert the number at the given index
* **Remove {index} –** remove the number at the given index
* **Shift left {count} –** first number becomes last. This has to be repeated the specified number of times
* **Shift right {count} –** last number becomes first. To be repeated the specified number of times

### Note: the index given may be outside of the bounds of the array. In that case print: "Invalid index".

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 23 29 18 43 21 20  Add 5  Remove 5  Shift left 3  Shift left 1  End | 43 20 5 1 23 29 18 |
| 5 12 42 95 32 1  Insert 3 0  Remove 10  Insert 8 6  Shift right 1  Shift left 2  End | Invalid index  5 12 42 95 32 8 1 3 |

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

**Note: All the indices will be valid!**

### 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 |
| 23 1 456 63 32 87 9 32  Remove 5  Add 1  Insert 14 2  RemoveAt 3  Add 34  end | 1. 14 63 32 87 9 32 1 34 |

## Bomb Numbers

Write a program that:

* Reads a **sequence of integer numbers** from the first line of the console
* Read a **special bomb number (integer)** and its **power (integer)** from the second line of the console
* 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
* Print the **sum of the remaining elements** in the sequence

### Example

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 1 2 2 4 2 2 2 9  4 2 | 12 | The special number is 4 with power 2. After detonation, we left with the sequence [1, 2, 9] with sum 12. |
| 1 4 4 2 8 9 1  9 3 | 5 | The special number is 9 with power 3. After detonation, we left with the sequence [1, 4] with sum 5. Since the 9 has only 1 neighbor 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 occurrence of 7 because it's already destroyed by the first occurrence. 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. |
| 1 2 3 4 5 6 7  2 2 | 18 | [5, 6, 7] |
| 1 2 3 4 5 6 7  1 4 | 13 | [6, 7] |