Exercise: Lists

Problems for exercises and homework for the "Programming Fundamentals" course @ SoftUni.

You can check your solutions in Judge.

1. Train

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

- **`Add {passengers}`** - add a wagon to the end with the given number of passengers
- **`{passengers}`** - find an existing wagon to **fit all the 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

2. Change List

Write a program that reads a **list** of **integers** from the console and receives **commands** which **manipulate** the list. Your program may receive the following commands:

- **`Delete {element}`** - delete all elements in the array which are equal to the given element
- **`Insert {element} {position}`** - insert element at the given position

You should stop the program when you receive the command **"end"**. Print all numbers in the array, separated with a **single** whitespace.

Examples

Input	Output
1 2 3 4 5 5 5 6 Delete 5 Insert 10 1 Delete 5 end	1 10 2 3 4 6
20 12 4 319 21 31234 2 41 23 4 Insert 50 2 Insert 50 5 Delete 4 end	20 12 50 319 50 21 31234 2 41 23

3. House Party

Write a program that keeps track of guests going to a house party.

On the **first** input line, you are going to receive how many commands you are going to have. On the **next lines** you are going to receive some of the following inputs:

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

If you receive the first type of input, you have to add the person if he/she **is not** on the list. If he/she is in the list, print on the console: "**{name} is already in the list!**". If you receive the second type of input, you must remove the person if he/she **is** on the list. If not, print: "**{name} is not in the list!**". In the end, print all guests.

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!	Tom is already in the list! Tom Annie Garry

Garry is going! Jerry is going!	Jerry

4. List Operations

You will be given numbers (a list of integers) on the first input line. Until you receive **"End"** you will be given operations you must apply on the list. The possible commands are:

- **Add {number}** - add number at the end
- **Insert {number} {index}** - insert number at given index
- **Remove {index}** - remove that index
- **Shift left {count}** - first number becomes last 'count' times
- **Shift right {count}** - last number becomes first 'count' times

Note: The index given may be outside the array's bounds. 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

5. Bomb Numbers

Write a program that **reads a sequence of numbers** and a **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	12	The special number is 4 with power 2. After detonation, we left with the sequence [1, 2, 9] with sum 12.

4 2		
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.

6. Cards Game

You will be given two hands of cards, which will be **integer** numbers. Assume that you have two players. You must find the winning deck and, respectively, the winner.

You start from the beginning of both hands. Compare the cards from the first deck to those from the second. The player, who has a bigger card, takes both cards and puts them on the **back** of his hand - **the second player's card is last, and the first person's card (the winning one) is before it (second to last),** and the player with the smaller card must **remove** the **card** from his deck. If both players' cards have the same values - no one wins, and the two cards must be **removed from** the **decks**. The game is over when one of the decks is left without any cards. You have to print the winner on the console and the sum of the left cards: **"{First/Second} player wins! Sum: {sum}".**

Examples

Input	Output
20 30 40 50 10 20 30 40	First player wins! Sum: 240
10 20 30 40 50 50 40 30 30 10	Second player wins! Sum: 50

7. Append Arrays

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

- Arrays are separated by "|".
- Values are separated by spaces (" ", **one or several**).
- Order the arrays 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

8. *Anonymous Threat

Anonymous has created a cyber-hyper virus that steals data from the CIA. As the lead security developer in the CIA, you have been tasked to analyze the software of the virus and observe its actions on the data. The virus is known for its innovative and unbelievably clever technique of merging and dividing data into partitions.

You will receive a **single input line** containing **STRINGS** separated by **spaces**.
The strings may contain **any ASCII** character except **whitespace**.

You will then begin receiving commands in one of the following formats:

- `merge {startIndex} {endIndex}`
- `divide {index} {partitions}`

Whenever you receive the **merge command**, you must merge all elements from the **startIndex** to the **endIndex**. In other words, you should concatenate them.
Example: **{abc, def, ghi} -> merge 0 1 -> {abcdef, ghi}**

If **any** of the **given indexes** is **out of the array**, you must take **ONLY** the **range** that is **INSIDE** the **array** and **merge** it.

Every time you receive the **divide command**, you must **DIVIDE** the **element** at the **given index** into **several small substrings** with **equal length**. The **count** of the **substrings** should be **equal** to the **given partitions**.

Example: **{abcdef, ghi, jkl} -> divide 0 3 -> {ab, cd, ef, ghi, jkl}**

If the string **CANNOT** be **exactly divided** into the **given partitions**, **make all partitions except** the **LAST** with **EQUAL LENGTHS**, and make the **LAST one** – **the LONGEST**.

Example: **{abcd, efgh, ijkl} -> divide 0 3 -> {a, b, cd, efgh, ijkl}**

The **input ends** when you receive the command "**`3:1`**". At that point, you must print the **resulting elements**, **joined** by a **space**.

Input

- The **first input line** will contain the **array** of **data.**
- On the **next several input** lines, you will **receive commands** in the **format specified above.**
- The **input ends** when you receive the command "**`3:1`**".

Output

- As output, you must print a single line containing the array elements, **joined** by a **space**.

Constrains

- The **strings** in the **array** may contain any **ASCII character** except **whitespace.**
- The **startIndex** and the **endIndex** will be in the **range [-1000, 1000].**
- The **endIndex** will **ALWAYS** be **GREATER** than the **startIndex.**
- The **`index`** in the **`divide command`** will **ALWAYS** be **INSIDE** the array.
- The **partitions** will be in the **range [0, 100].**
- Allowed working **time/memory**: **100ms / 16MB.**

Examples

Input	Output
Ivo Johny Tony Bony Mony merge 0 3 merge 3 4 merge 0 3 3:1	IvoJohnyTonyBonyMony
abcd efgh ijkl mnop qrst uvwx yz merge 4 10 divide 4 5 3:1	abcd efgh ijkl mnop qr st uv wx yz

9. *Pokémon Don't Go

Ely likes to play Pokémon Go a lot. But Pokémon Go went bankrupt… So, the developers made Pokémon Don't Go out of depression. And so, Ely now plays Pokémon Don't Go. In Pokémon Don't Go, when you walk to a certain Pokémon, those closest to you naturally get further, and those further from you get closer.

You will receive a **sequence** of **integers**, separated by **spaces** - the distances to the Pokémons.
Then you will begin **receiving integers** corresponding to **indexes** in **that sequence**.

When you **receive** an **index**, you must **remove** the **element** at **that index** from the **sequence** (as if you've captured the Pokémon).

- You must **INCREASE** the **value** of **all elements** in the sequence which are **LESS** or **EQUAL** to the **removed element** with the **value** of the **removed element**.
- You must **DECREASE** the **value** of **all elements** in the sequence which are **GREATER** than the **removed element** with the **value** of the **removed element**.

If the **given index** is **LESS** than **0**, **remove** the **first element** of the **sequence**, and **COPY** the **last element** to its place.

If the **given index** is **GREATER** than the **last index** of the **sequence**, **remove** the **last element** from the sequence, and **COPY** the **first element** to its place.

The **increasing** and **decreasing** of elements should be done in these cases, **also**. The **element** whose value you should use is the **REMOVED** element.

The program **ends** when the **sequence** has **no elements** (there are no Pokémons left for Ely to catch).

Input

- On the **first line** of input, you will receive a **sequence** of **integers**, **separated** by **spaces.**
- On the **next several** lines, you will receive **integers** – the **indexes.**

Output

- When the program ends, you must print the summed up value of all REMOVED elements on the console**.**

Constraints

- The input data will consist **ONLY** of **valid integers** in the **range [-2.147.483.648, 2.147.483.647].**

Examples

Input	Output	Comments
4 5 3 1 1 0	14	The **array** is **{4, 5, 3}**. The index is **1**. We **remove 5**, and we **increase all lower** than it and **decrease all higher** than it. In this case, there are **no higher** than **5**. The result is **{9, 8}**. The **index** is **1**. So we remove **8** and **decrease all higher** than it. The result is **{1}**. The **index** is **0**. So we remove **1**. No elements are **left**, so we print the **sum** of **all removed elements**. **5 + 8 + 1 = 14**.
5 10 6 3 5 2 4 1 1 3 0 0	51	**Step 1**: **{11, 4, 9, 11}** **Step 2**: **{22, 15, 20, 22}** **Step 3**: **{7, 5, 7}** **Step 4**: **{2, 2}** **Step 5**: **{4, 4}** **Step 6**: **{8}** **Step 7**: **{} (empty).** **Result** = **6 + 11 + 15 + 5 + 2 + 4 + 8 = 51**.

10. *SoftUni Course Planning

You are tasked to help plan the next Programming Fundamentals course by keeping track of the lessons that will be included in the course, as well as all the exercises for the lessons.

On the first input line, you will **receive the initial schedule of lessons and exercises** that will be part of the next course, separated by **a comma and space ", "**. But before the course starts, there are some changes to be made. Until you receive **"course start"**, you will be given **some commands to modify the course schedule**. The possible commands are:

- **`Add:{lessonTitle}`** - add the lesson to the end of the schedule, **if it does not exist**
- **`Insert:{lessonTitle}:{index}`** - insert the lesson to the given index, **if it does not exist**
- **`Remove:{lessonTitle}`** - remove the lesson, **if it exists**
- **`Swap:{lessonTitle}:{lessonTitle}`** - change the place of the two lessons, **if they exist**
- **`Exercise:{lessonTitle}`** - add Exercise in the schedule right after the lesson index, **if the lesson exists and there is no exercise already**, in the following format: **`"{lessonTitle}-Exercise"`**. **If the lesson doesn't exist**, **add the lesson** at the end of the course schedule**, followed by the Exercise**.

Each time you Swap or Remove a lesson, you should do the same with the Exercises, if there are any, which follow the lessons.

Input

- On the first line -the initial schedule lessons -strings, separated by comma and space **", "**.
- Until **"course start"**, you will receive commands in the format described above.

Output

- Print the whole course schedule, each lesson on a new line with its number(index) in the schedule: **"{lesson index}.{lessonTitle}".**
- Allowed working **time** / **memory**: **100ms** / **16MB.**

Examples

Input	Output	Comment
Data Types, Objects, Lists Add:Databases Insert:Arrays:0 Remove:Lists course start	1.Arrays 2.Data Types 3.Objects 4.Databases	We receive the initial schedule. Next, we add the Databases lesson because it doesn't exist. We Insert the given index lesson Arrays because it's not present in the schedule. After receiving the last command and removing lesson Lists, we print the whole schedule.
Arrays, Lists, Methods Swap:Arrays:Methods Exercise:Databases Swap:Lists:Databases Insert:Arrays:0 course start	1.Methods 2.Databases 3.Databases-Exercise 4.Arrays 5.Lists	We swap the given lessons because both exist. After receiving the Exercise command, we see that such a lesson doesn't exist, so we add the lesson at the end, followed by the exercise. We swap Lists and Databases lessons. The Databases-Exercise is also moved after the Databases lesson. We skip the next command because we already have such a lesson in our schedule.