Lab: Lists

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

You can check your solutions in Judge.

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
0.1 0.1 5 -5	0.2 5 -5	**0.1 0.1** 5 -5 → 0.2 5 -5

Solution

Read a list from numbers.

```
Scanner sc = new Scanner(System.in);

List<Double> numbers =
        Arrays.stream(sc.nextLine().split( regex: " "))
                .map(Double::parseDouble)
                .collect(Collectors.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(numbers.get(i).equals(numbers.get(i + 1))){
    numbers.set(i, (numbers.get(i) + numbers.get(i + 1)));
    numbers.remove( index: i + 1);
    i = -1;
}
```

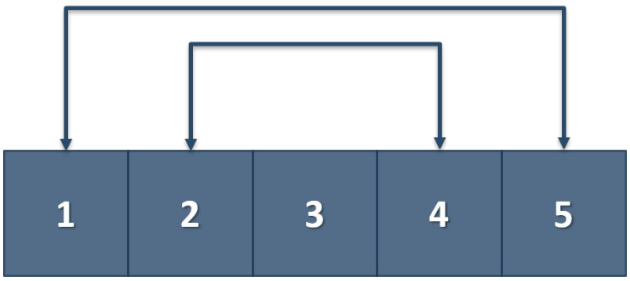
Finally, you have to print the numbers joined by space.

```
String output = joinElementsByDelimiter(numbers, delimiter: " ");
System.out.println(output);
```

```
static String joinElementsByDelimiter(List<Double> items,
                                      String delimiter) {
    String output = "";
    for (Double item : items)
        output += (new DecimalFormat( pattern: "0.#").format(item)
                + delimiter);
    return output;
}
```

2. Gauss' Trick

Write a program that **sum** all **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 any) to the end of the list

4. List Manipulation Basics

Write a program that reads a list of integers. Then until you receive **"end"**, you will be given 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.

```
public class ListManipulationBasics {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        List<Integer> numbers =
                Arrays.stream(sc.nextLine()
                        .split( regex: " "))
                        .map(Integer::parseInt)
                        .collect(Collectors.toList());

    }
}
```

Next we make the while loop for the commands and make switch statement for the commands

```
while (true){
    String line = sc.nextLine();

    if(line.equals("end")){
        break;
    }

    String[] tokens = line.split( regex: " ");
}
```

We break if the line is "end", otherwise we split it into tokens and process the command.

```
String[] tokens = line.split( regex: " ");

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

Now let's implement each command.

```
case "Add":
    int numberToAdd = Integer.parseInt(tokens[1]);
    numbers.add(numberToAdd);
    break;
case "Remove":
    int numberToRemove = Integer.parseInt(tokens[1]);
    numbers.remove(numberToRemove);
    break;
case "RemoveAt":
    int indexToRemove = Integer.parseInt(tokens[1]);
    numbers.remove(indexToRemove);
    break;
case "Insert":
    int numberToInsert = Integer.parseInt(tokens[1]);
    int indexToInsert = Integer.parseInt(tokens[2]);
    numbers.add(indexToInsert, numberToInsert);
    break;
```

For all commands **except from** the **"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**

```
System.out.println(numbers.toString()
        .replaceAll( regex: "[\\[\\],]",  replacement: ""));
```

5. List Manipulation Advanced

Now we will implement more complicated list commands. Again, read a list, and until you receive **"end"** read commands:

Contains {number} – check if the list contains the number. If **yes** print **"Yes"**, **otherwise** print **"No such number"**

Print even – print **all the numbers** that are **even separated by a space**

Print odd – print **all the numbers** that are **odd separated by a space**

Get sum – print the **sum of all the numbers**

Filter ({condition} {number}) – print all the numbers that **fulfill that condition**. The condition will be either **'<'**, **'>'**, **">="**, **"<="**

Example

Input	Output
2 13 43 876 342 23 543 Contains 100 Contains 543 Print even Print odd Get sum 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

6. 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

```
import java.util.Scanner;

public class ListOfProducts {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = Integer.parseInt(sc.nextLine());
    }
}
```

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

```
public class ListOfProducts {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        int n = Integer.parseInt(sc.nextLine());
        List<String> products = new ArrayList<>();
    }
}
```

Then we need to iterate **n times** and **read products**.

```
for (int i = 0; i < n; i++) {
    String currentProduct = sc.nextLine();
}
```

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

```
for (int i = 0; i < n; i++) {
    String currentProduct = sc.nextLine();
    products.add(currentProduct);
}
```

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

```
Collections.sort(products);
```

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.size(); i++) {
    System.out.printf("%d.%s%n", i + 1, products.get(i));
}
```

We use **i + 1**, because we want to **start counting from 1**.

7. 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

Solution

Read a list of integers.

```
Scanner sc = new Scanner(System.in);

List<Integer> numbers =
        Arrays.stream(sc.nextLine().split( regex: " "))
                .map(Integer::parseInt)
                .collect(Collectors.toList());
```

Remove all negative numbers and reverse the collection.

```
numbers.removeIf(n -> n < 0);
Collections.reverse(numbers);
```

If the list is empty print "empty", otherwise print all numbers joined by space.

```
if (numbers.isEmpty()) {
    System.out.println("empty");
} else {
    System.out.println(numbers.toString().replaceAll( regex: "[\\[\\],]",  replacement: ""));
}
```

Exercise: Lists

Problems for exercises and homework for the "Technology 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**)

At 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, which 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 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 the guests that are 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** in 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 have to remove the person if he/she **is** in the list. (If not print: "**{name} is not in the list!**"). At 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! Garry is going! Jerry is going!	Tom is already in the list! Tom Annie Garry Jerry

4. List Operations

You will be given numbers (list of integers) on the first input line. Until you receive **"End"** you will be given operations you have to 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: It is possible that the index given is 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`

5. 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 detonation 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 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 its 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 have to find out the winning deck and respectively the winner.

You start from the beginning of both hands. Compare the cards from the first deck to the cards from the second deck. The player, who has bigger card, takes both cards and puts them at 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 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: **`"Player {one/two} 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

The Anonymous have created a cyber hypervirus which steals data from the CIA. You, as the lead security developer in CIA, have been tasked to analyze the software of the virus and observe its actions on the data. The virus is known for his 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}**

Every time you receive the **merge command**, you must merge all elements from the **startIndex** till 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 elements of the array, **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 **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 **range [0, 100]**
- Allowed working **time/memory**: **100ms / 16MB**

Examples

Input	Output
Ivo Johny Tony Bony Mony	`IvoJohnyTonyBonyMony`

merge 0 3 merge 3 4 merge 0 3 3:1	
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. *Pokemon Don't Go

Ely likes to play Pokemon Go a lot. But Pokemon Go bankrupted... So, the developers made Pokemon Don't Go out of depression. And so, Ely now plays Pokemon Don't Go. In Pokemon Don't Go, when you walk to a certain pokemon, those closer 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 pokemons.
Then you will begin **receiving integers**, which will **correspond** 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 pokemon).

- 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 pokemons 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 on the console, the **summed up value** of **all REMOVED elements**

Constrains

- 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	14	The `array` is **{4, 5, 3}**. The index is **1**.

1 1 0		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**. There are **no elements 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 Programing Fundamentals course by keeping track of the lessons, that are going to 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 are going to be part of the next course, separated by **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** in 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 Databases lesson, because it doesn't exist. We Insert at 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.
Input	**Output**	**Comment**
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 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 lesson in our schedule.

More Exercise: Lists

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

You can check your solutions in Judge.

1. Messaging

You will be given some **list of numbers** and a **string**. For each element of the list you have to **take the sum of its digits** and take the **element corresponding to that index from the text**. If the index is **greater than the length of the text**, start counting **from the beginning** (so that you always have a valid index). After getting the element from the text, you have to **remove the character** you have taken from it (so for the next index, the text will be with one character less).

Example

Input	Output
9992 562 8933 This is some message for you	hey

2. Car Race

Write a program to calculate the **winner of a car race**. You will receive an **array of numbers**. Each element of the array represents the **time needed to pass through that step** (the index). There are going to be **two cars**. **One** of them **starts** from the **left side** and the **other one starts from the right side**. **The middle index of the array is the finish line**. (The **number of elements** of the array **will always be odd**). Calculate **the total time for each racer to reach the finish** (the **middle of the array**) and **print the winner with his total time**. (The **racer with less time**). If you have a **zero in the array**, you have to **reduce the time of the racer that reached it by 20%** (**from the time so far**).

Print the result in the following format **"The winner is {left/right} with total time: {total time}", formatted** with **one digit** after the decimal point.

Example

Input	Output
29 13 9 0 13 0 21 0 14 82 12	The winner is left with total time: 53.8
Comment	
The time of the left racer is (29 + 13 + 9) * 0.8 (because of the zero) + 13 = 53.8 The time of the right racer is (82 + 12 + 14) * 0.8 + 21 = 107.4 The winner is the left racer, so we print it	

3. Take/Skip Rope

Write a program, which reads a **string** and **skips** through it, extracting a **hidden message**. The algorithm you have to implement is as follows:

Let's take the string "**`skipTest_String044170`**" as an example.

Take every **digit** from the string and **store it** somewhere. After that, **remove** all the digits from the string. After this operation, you should have **two lists of items**: the **numbers list** and the **non-numbers list**:

- Numbers list: **`[0, 4, 4, 1, 7, 0]`**

- Non-numbers: `[s, k, i, p, T, e, s, t, _, S, t, r, i, n, g]`

After that, take every digit in the **numbers list** and split it up into a **take list** and a **skip list**, depending on whether the digit is in an **even** or an **odd** index:

- Numbers list: `[0, 4, 4, 1, 7, 0]`
- Take list: `[0, 4, 7]`
- Skip list: `[4, 1, 0]`

Afterwards, **iterate** over both of the lists and **skip** `{skipCount}` characters from the **non-numbers list**, then **take** `{takeCount}` characters and store it in a **result string**. Note that the skipped characters are **summed up** as they go. The process would look like this on the aforementioned **non-numbers list**:

`Example: "skipTest_String"`

1. Take **0** characters ➔ Taken: "", skip **4** characters ➔ Skipped: "**skip**"➔ Result: ""
2. Take **4** characters➔ Taken: "**Test**", skip **1** characters ➔ Skipped: "_" ➔ Result: "**Test**"
3. Take **7** characters➔ Taken: "**String**", skip **0** characters ➔ Skipped: "" ➔ Result: "**TestString**"

After that, just print the **result string** on the console.

Input

The **encrypted** message as a **string**

Output

The **decrypted** message as a **string**

Constraints

- The count of digits in the input string will **always be even**.
- The encrypted message will contain any printable ASCII character.

Examples

Input	Output
`T2exs15ti23ng1_3cT1h3e0_Roppe`	`TestingTheRope`
`O{1ne1T2021wf312o13Th111xreve!!@!`	`OneTwoThree!!!`
`this forbidden mess of an age rating 0127504740`	`hidden message`

4. *Mixed up Lists

Write a program that **mixes up two lists** by some rules. You will receive **two lines of input**, each one being a **list of numbers**. The **rules** for mixing are:

- Start from the **beginning of the first** list and from the **ending of the second**
- **Add** element **from the first** and element **from the second**
- At the end there will always be a list in which there are **2 elements remaining**
- These elements will be the **range of the elements you need to print**
- **Loop through the result list** and take **only the elements that fulfill the condition**
- Print the elements **ordered in ascending** order and **separated by a space**

Example

Input	Output
1 5 23 64 2 3 34 54 12 43 23 12 31 54 51 92	23 23 31 34 43 51
Comment	
After looping through the two of the arrays we get: 1 92 5 51 23 54 64 31 2 12 3 23 34 43 The constrains are 54 and 12 (so we take only the numbers between them): 51 23 31 23 34 43 We print the result sorted	

5. *Drum Set

Gabsy is Orgolt's Final Revenge charming drummer. She has a drum set but the different drums have different origins – some she bought, some are gifts, so they are all with **different quality**. Every day she practices on each of them, so she does damage and reduces the drum`s quality. Sometimes a drum brakes, so she needs to buy new one. Help her keep her drum set organized.

You will receive Gabsy's **savings**, the money she can spend on new drums. Next you receive a **sequence of integers** which represent the **initial quality** of each drum in Gabsy's drum set.

Until you receive the command **"Hit it again, Gabsy!"**, you will be receiving integer: the **hit power** Gabsy applies **on each drum** while practicing. When the power is applied you should **decrease** the value of the drum's quality with the current power.

When a certain drum **reaches 0 quality**, it breaks. Then Gabsy should buy a replacement. She needs to buy the exact same model. Therefore, its quality will be **the same as the initial quality** of the broken drum. The price is calculated by the formula: {initialQuality} * 3. Gabsy will always replace her broken drums **until the moment she can no longer afford it**. If she doesn't have enough money for a replacement, the broken drum is **removed** from the drum set.

When you receive the command **"Hit it again, Gabsy!"**, the program ends and you should print the current state of the drum set. On the second line you should print the **remaining money** in Gabsy's savings account.

Input

- On the **first line** you receive the **savings** – a floating-point number;
- On the **second line** you receive the **drum set**: **sequence** of **integers**, **separated** by **spaces**.
- Until you receive the command **"Hit it again, Gabsy!"** you will be receiving **integers** – the hit power Gabsy applies on each drum.

Output

- On the first line you should print **each drum** in the drum set, **separated** by **space**.
- Then you must print the **money** that are left on the **second line** in the format **`"Gabsy has {money left}lv."`**, formatted with two digits after the decimal point.

Constraints

- The **savings – floating-point number in the range [0.00, 10000.00]**
- The **quality of each drum in the drum set** – integer in the range **[1, 1000]**.
- The **hit power** will be in the **range [0, 1000]**
- Allowed working **time** / **memory**: **100ms** / **16MB**.

Examples

Input	Output	Comment
1000.00 58 65 33 11 12 18 10 Hit it again, Gabsy!	7 14 23 Gabsy has 901.00lv.	DrumSet – 58 65 33. Day 1: hit power applied = 11 => 47 54 22; Day 2: hit power applied = 12 => 35 42 10; Day 3: hit power applied = 18 => 17 24 -8; The third drum breaks. But Gabsy has enough savings so she replaces it => 17 24 33; Day 4: hit power applied = 10 => 7 14 23; We print the current state of the drum set and what's left in Gabsy's bank account.
154.00 55 111 3 5 8 50 2 50 8 23 1 Hit it again, Gabsy!	27 2 4 7 Gabsy has 10.00lv.	