Exercise: Arrays

Problems for exercise and homework for the "C# Fundamentals" course @ SoftUni
You can check your solutions in Judge

1. Train

A train has **n** number of wagons (integer, received as input). On the next **n lines,** you will receive the number of people that are going to get on each wagon. **Print out the number of passengers in each wagon** followed by the **total number of passengers on the train**.

Examples

Input	Output
3 13 24 8	13 24 8 45
6 3 52 71 13 65 4	3 52 71 13 65 4 208
1 100	100 100

2. Common Elements

Create a program that prints out all **common elements in two arrays**. You have to **compare the elements** of the **second array** to the elements of the **first**.

Examples

Input	Output
Hey hello 2 4 10 hey 4 hello	4 hello
S of t un i of i 10 un	of i un
i love to code code i love to	code i love to

3. Zig-Zag Arrays

Create a program that creates **2 arrays**. You will be given an **integer n**. On the **next n lines,** you will get **2 integers**. Form 2 new arrays in a **zig-zag** pattern as **shown below**.

Examples

Input	Output

4 1 5 9 10 31 81 41 20	1 10 31 20 5 9 81 41
2 80 23 31 19	80 19 23 31

4. Array Rotation

Create a program that **receives an array and several rotations** that you have to perform. The rotations are done by moving the first element of the array from the front to the back. **Print the resulting** array.

Examples

Input	Output
51 47 32 61 21 2	32 61 21 51 47
32 21 61 1 4	32 21 61 1
2 4 15 31 5	4 15 31 2

5. Top Integers

Create a program to **find all the top integers** in an array. A top integer is an integer that is **greater** than all the elements **to its right**.

Examples

Input	Output
1 4 3 2	4 3 2
14 24 3 19 15 17	24 19 17
27 19 42 2 13 45 48	48

6. Equal Sums

Create a program that determines if **an element exists in an array** for which the **sum of all elements to its left** is **equal** to **the sum of all elements to its right**. If there are **no elements to the left or right**, their **sum is considered to be 0**. **Print** the **index** of the element that satisfies the condition or "no" if there is no such element.

Examples

Input	Output	Comments
1 2 3 3	2	At a[2] -> left sum = 3, right sum = 3 a[0] + a[1] = a[3]
1 2	no	At a[0] -> left sum = 0, right sum = 2 At a[1] -> left sum = 1, right sum = 0

		No such index exists
1	0	At a[0] -> left sum = 0, right sum = 0
1 2 3	no	No such index exists
10 5 5 99 3 4 2 5 1 1 4	3	At a[3] -> left sum = 20, right sum = 20 a[0] + a[1] + a[2] = a[4] + a[5] + a[6] + a[7] + a[8] + a[9] + a[10]

7. Max Sequence of Equal Elements

Create a program that **finds the longest sequence of equal elements in an array** of integers. If **several equal sequences are present in the array**, print out the **leftmost** one.

Examples

Input	Output
2 1 1 2 3 3 **2 2 2** 1	2 2 2
1 1 1 2 3 1 3 3	1 1 1
4 4 4 4	4 4 4 4
0 **1 1** 5 2 2 6 3 3	1 1

8. Magic Sum

Create a program, which **prints all unique pairs** in an array of integers whose **sum is equal to a given number**.

Examples

Input	Output
1 7 6 2 19 23 8	1 7 6 2
14 20 60 13 7 19 8 27	14 13 20 7 19 8

9. *Kamino Factory

The clone factory in Kamino got another order to clone troops. But this time you are tasked to find **the best DNA** sequence to use in the production.

You will receive the **DNA length** and until you receive the command **"Clone them!"**, you will be receiving a **DNA sequence of ones and zeroes, split by '!' (one or several).**

You should select the sequence with the **longest subsequence of ones**. If there are several sequences with the **same length of** the **subsequence of ones**, print the one with the **leftmost starting index**, if there are several sequences with the same **length and starting index**, select the sequence with the **greater sum** of its elements.

After you receive the last command **"Clone them!"** you should print the collected information in the following format:

"Best DNA sample {bestSequenceIndex} with sum: {bestSequenceSum}."

"{DNA sequence, joined by space}"

Input / Constraints

- The **first line** holds the **length** of the **sequences** – **integer in the range [1…100].**
- On the next lines, until you receive **`"Clone them!"`,** you will be receiving sequences (at least one) of ones and zeroes, **split by** '**!**' (one or several).

Output

The output should be printed on the console and consists of two lines in the following format:

`"Best DNA sample {bestSequenceIndex} with sum: {bestSequenceSum}."`

`"{DNA sequence, joined by space}"`

Examples

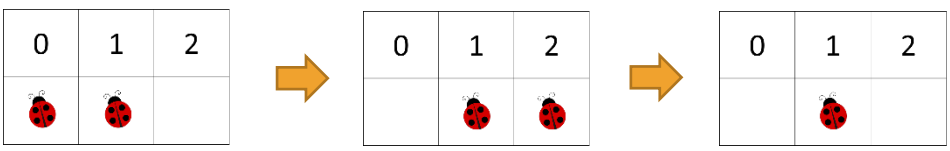
Input	Output	Comments
5 1!0!**1!1**!0 0!**1!1**!0!0 Clone them!	Best DNA sample 2 with sum: 2. 0 1 1 0 0	We receive 2 sequences with the **same length of the subsequence of ones**, but the second is printed because its subsequence starts at **`index[1]`**.

Input	Output	Comments
4 **1!1**!0!**1** 1!0!0!1 **1!1**!0!0 Clone them!	Best DNA sample 1 with sum: 3. 1 1 0 1	We receive 3 sequences. Both 1 and 3 **have the same length** of the subsequence of ones -> 2, **and both start from the `index[0]`**, but the first is printed, because its **sum is greater.**

10. *LadyBugs

You are given a **field size** and the **indexes where ladybugs** can be found on the field. On every new line, until the **"end"** command is given, a ladybug **changes its position** either to its **left** or to its **right** by a given **fly length**. A movement description **command** looks like this: `"0 right 1"`. This means that the little insect **placed on index 0 should fly one index to its right**. If the ladybug **lands on another ladybug**, it **continues to fly in the same direction repeating the specified flight length**. If the ladybug **flies out of the field**, it is **gone**.

For example, you are given a **field of size 3**, where there are ladybugs on indexes **0 and 1**. If the ladybug **on index 0 needs to fly to its right by the length of 1** (0 right 1), it will attempt to **land on index 1** but as there is **another ladybug** there, it will continue further to the right passing 1 index in length, landing on **index 2**. After that, if the same ladybug needs to fly to its right passing 1 index (2 right 1), it will land somewhere **outside** of the field, so it flies **away**:



If we receive an initial index that does not contain a ladybug, **nothing happens**. If you are given a ladybug index that is **outside the field**, **nothing** happens. In the end, **print all cells** of the field **separated** by **blank spaces**. For each cell that has a ladybug in it print '1' and for each empty cell print '0'. The output of the example above should be "0 1 0".

Input

- On the first line, you will receive an integer - the size of the field.

- On the second line, you will receive the initial indexes of all ladybugs separated by a blank space.
- On the next lines, until you get the **"end"** command, you will receive commands in the format: **"{ladybug index} {direction} {fly length}"**.

Output

- Print all field cells in format: **"{cell} {cell} … {cell}"**
  - If a cell has a ladybug in it, print '**1**'.
  - If a cell is empty, print '**0**'.

Constrains

- The size of the field will be in the range [**0…1000**].
- The ladybug indexes will be in the range [**-2147483647…2147483647**].
- The number of commands will be in the range [**0…100**].
- The fly length will be in the range [**-2147483647…2147483647**].

Examples

Input	Output	Comments
3 0 1 0 right 1 2 right 1 end	0 1 0	1 1 0 – initial field 0 1 1 – field after "0 right 1" 0 1 0 – field after "2 right 1"

Input	Output
3 0 1 2 0 right 1 1 right 1 2 right 1 end	0 0 0

Input	Output
5 3 3 left 2 1 left -2 end	0 0 0 1 0