# Exercises: Arrays

Problems for exercises and homework for the "Programming Fundamentals: Arrays and Lists" course from the official "Applied Programmer" curriculum.

You can check your solutions here: https://judge.softuni.bg/Contests/2907.

## 1. Train

On the **first line**, you will receive **n** – the number of **lines**, which will **follow**. You will be given a count of wagons in a train which is represented by **n**. On the next **n** lines you will receive **how many people** are going to **get on each wagon**. At the end, **print the whole train** and after that, on the **next line**, the **sum of the people** in the train.

### Examples

| Input | Output |
|-------|--------|
| 3<br>13<br>24<br>8 | 13 24 8<br>45 |
| 6<br>3<br>52<br>71<br>13<br>65<br>4 | 3 52 71 13 65 4<br>208 |
| 1<br>100 | 100<br>100 |

## 2. Zig-Zag Arrays

Write a program which creates **2 arrays**. On the **first line**, you will receive **n** – the number of **lines**, which will **follow**. On the next **n** lines you will get **2 integers**. **Form 2 arrays** as shown below. Derive two arrays by separating the zigzag numbers from the given rows and thus compose two arrays.

### Examples

| Input | Output |
|-------|--------|
| 4<br>1  5<br>9  10<br>31  81<br>41  20 | 1 10 31 20<br>5 9 81 41 |
| 2<br>80  23<br>31  19 | 80 19<br>23 31 |

## 3. Array Rotation

Write a program that receives an **array** and single **number.** The number represents the **count** of **elements** that **need** to be **moved** from the beginning of the array to the end **(first element goes0 at the end)**. Print the resulting array.

### Examples



| Input | Output |
|-------|--------|
| 51 47 32 61 21<br>2 | 32 61 21 51 47 |
| 32 21 61 1<br>4 | 32 21 61 1 |
| 2 4 15 31<br>5 | 4 15 31 2 |

## 4. Top Integers

Write a program to **find all the top integers** in an array. **A top integer** is an integer which is **bigger 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 |

## 5. Equal Sums

Write a program that determines if there **exists an element in the array** such that the **sum of the elements on its left** is **equal** to the **sum of the elements on its right (there will never be more than 1 element like that)**. If there are **no elements to the left / right**, their **sum is considered to be 0**. Print the **index** that satisfies the required condition or **"no"** if there is no such index.

### Examples

| Input | Output | Comments |
|-------|--------|----------|
| 1 2 3 3 | 2 | At a[2] -> left sum = 3, right sum = 3<br>a[0] + a[1] = a[3] |
| 1 2 | no | At a[0] -> left sum = 0, right sum = 2<br>At a[1] -> left sum = 1, right sum = 0<br>No such index exists |
| 1 | 0 | At a[0] -> left sum = 0, right sum = 0 |
| 1 2 3 | no | No such index exists |

| | | 3 | At a[3] -> left sum = 20, right sum = 20 |
|---|---|---|---|
| 10 5 5 99 3 4 2 5 1 1 4 | | | a[0] + a[1] + a[2] = a[4] + a[5] + a[6] + a[7] + a[8] + a[9] + a[10] |

## 6. Max Sequence of Equal Elements

Write a program that finds the **longest sequence of equal elements** in an array of integers. If several longest sequences exist, print 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 |

## 7. *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 sequences 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 **same length of subsequence of ones**, print the one with the **leftmost starting index**
- if there are several sequences with 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 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

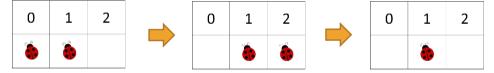| Input | Output | Comments |
|---|---|---|
| | | |

| 5<br>1!0!**1!1**!0<br>0!**1!1**!0!0<br>Clone them! | Best DNA sample 2 with sum: 2.<br>0 1 1 0 0 | We receive 2 sequences with **same length of subsequence of ones**, but the second is printed, because its subsequence starts at **index[1].** |
|---|---|---|
| **Input** | **Output** | **Comments** |
| 4<br>**1!1**!0!**1**<br>1!0!0!1<br>**1!1**!0!0<br>Clone them! | Best DNA sample 1 with sum: 3.<br>1 1 0 1 | We receive 3 sequences. Both 1 and 3 **have same length** of subsequence of ones -> 2, **and both start from index[0]**, but the first is printed, because its **sum is greater.** |

## 8. *LadyBugs

You are **given a field size** and the **indexes of ladybugs** inside the field. After that 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 **command to a ladybug** 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 a fellow ladybug**, it **continues to fly** in the same direction **by the same fly length**. If the ladybug **flies out of the field, it is gone**.

For example, imagine you are given a **field** with **size 3** and 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 by additional **length of 1**, landing on **index 2**. After that, if the same ladybug needs to fly to its right by the length of **1** (2 right 1), it will land somewhere outside of the field, so it flies away:



If you are given ladybug index that does not have ladybug there, nothing happens. If you are given ladybug index that is outside the field, nothing happens.

Your job is to create the program, simulating the ladybugs flying around doing nothing. At the end, **print all cells in the field separated by blank spaces**. For each cell that has a ladybug on it print '**1**' and for each empty cells print '**0**'. For the example above, the output 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. **The given indexes** may or may not be inside the field range.
- On the next lines, until you get the "end" command you will receive commands in the format: "**{ladybug index} {direction} {fly length}**".

### Output

- Print the **all cells within the field in format: "{cell} {cell} ... {cell}"**.
  - If a cell has 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 **[-2,147,483,647 ... 2,147,483,647]**.
- The number of commands will be in the range **[0 ... 100]** .

- The fly length will be in the range **[-2,147,483,647 … 2,147,483,647]**.

# Examples

| Input | Output | Comments |
|---|---|---|
| 3<br>0 1<br>0 right 1<br>2 right 1<br>end | 0 1 0 | 1 1 0 - Initial field<br>0 1 1 - field after "0 right 1"<br>0 1 0 - field after "2 right 1" |

| Input | Output |
|---|---|
| 3<br>0 1 2<br>0 right 1<br>1 right 1<br>2 right 1<br>end | 0 0 0 |

| Input | Output |
|---|---|
| 5<br>3<br>3 left 2<br>1 left -2<br>end | 0 0 0 1 0 |