# More Exercises: Lists Basics

Please, submit your source code solutions for the described problems to the [Judge System](https://judge.softuni.org/Contests/1726/Lists-Basics-More-Exercises).

***Note: These exercises are excluded from your homework!***

## Zeros to Back

Write a program that receives a **single string** (**integers** **separated** by a comma and space **", "**), finds all the **zeros,** and moves them **to the back** without messing up the other elements. Print the resulting **integer list**.

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1, 0, 1, 2, 0, 1, 3 | [1, 1, 2, 1, 3, 0, 0] |
| 0, 5, 0, 4, 0, 0, 5 | [5, 4, 5, 0, 0, 0, 0] |

## Messaging

On the **first line**, you will receive a **sequence of numbers** separated by a single space. On the **second line**, you will receive a **string**.

Your task is to write a program that sends a message only using chars from the given string. **Each** **char** the program adds to the message should be found by **its index**. The index you are looking for is the **sum of a number's digits from the first sequence**. If the index is **greater than the length of the text**, continue counting **from the beginning** (so that you always have a valid index). When you find a char, you should **add it** to the message and **remove it** from the string. It means that for the following index, the **text will contain one character less**.

**Print** **the final message**.

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| 9992 562 8933  This is some message for you | hey |
| 2 122 1123 1321 9 17211  87j973u59dg37e725! | judge! |

## Car Race

Write a program that announces the **winner of a car race**.

You will receive a **sequence of numbers**. Each number represents the **time the car needs to pass through that step** (the index). There will be **two cars**. **The first one** **starts** from the **left side**, and the **other one starts from the right** **side**. **The middle index of the sequence is the finish line**.

Calculate **the total time each racer needs to reach the finish** **line** and **print the winner with his total time** (the **racer with less time**). If you have a **zero in the list**, you should **reduce the racer's time that reached it by 20%** (**from his current time**).

The **number of elements** in the sequence **will always be odd**.

Print the result in the following format **"The winner is {left/right} with total time: {total\_time}"**.

The time should be **formatted** to the **first 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. | |
| **Input** | **Output** |
| 123 20 4 0 13 0 0 5 5 14 0 | The winner is right with total time: 19.2 |

## Josephus Permutation

*This problem takes its name from arguably the most important event in the life of the ancient historian Josephus. According to his tale, he and his 40 soldiers were trapped in a cave by the Romans during a siege. Refusing to surrender to the enemy, they instead opted for mass suicide, with a twist: they formed a circle and proceeded to kill one man out of every three until one last man was left (and that was supposed to kill himself to end the act). Well, Josephus and another man were the last, and, as we now know every detail of the story, you may have correctly guessed that they did not precisely follow through with the original idea.*

You are now to create a program that prints a **Josephus permutation**, receiving **two lines** of code:

* the list itself- **numbers separated by a single space** representing the people in the circle
* a number **k**

People are standing in a circle waiting to be executed. Counting begins **from the first one** in the circle and proceeds from **left to right**. After a specified number of people are skipped, the **k** person is executed. The procedure is repeated with the remaining people, **starting with the next person**, going in the same direction, and skipping the same number of people until **no one remains**.

Print the people by **order of executions** in the format: **"[{executed1},{executed2}, … {executedN}]"**

### Example

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comment** |
| 1 2 3 4 5 6 7  3 | [3,6,2,7,5,1,4] | [1,2,3,4,5,6,7] - initial sequence  [1,2,4,5,6,7] => 3 is counted out and goes into the result [3]  [1,2,4,5,7] => 6 is counted out and goes into the result [3,6]  [1,4,5,7] => 2 is counted out and goes into the result [3,6,2]  [1,4,5] => 7 is counted out and goes into the result [3,6,2,7]  [1,4] => 5 is counted out and goes into the result [3,6,2,7,5]  [4] => 1 is counted out and goes into the result [3,6,2,7,5,1]  [] => 4 is counted out and goes into the result [3,6,2,7,5,1,4] |
| 10 5 65 104 1 0 2  8 | [10,65,0,1,5,2,104] |  |

## Tic-Tac-Toe

You will receive a field of a tic-tac-toe game in **three lines** containing **numbers, separated by a single space**.

***Legend***:

* 0 - **empty** space
* 1 - **first** player move
* 2 - **second** player move

Find out who the **winner** is. If the **first** player **wins**,print **"First player won"**. If the **second** player **wins**, print **"Second player won"**. Otherwise, print **"Draw!"**.

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2 0 1  0 1 0  1 0 2 | First player won |
| 0 1 0  2 2 2  1 0 0 | Second player won |
| 1 0 2  0 1 2  1 2 0 | Draw! |

## List Manipulator

Trifon has finally become a junior developer and has received his first task. It is about manipulating a list of integers. He is not quite happy about it since he hates manipulating lists. They will pay him a lot of money, though, and he is willing to give somebody half of it if to help him do his job. On the other hand, you love lists (and money), so you decide to try your luck.

The list may be manipulated by one of the following commands:

* **"exchange {index}"** – splits the list **after** the given index and **exchanges the places** of the two resulting sub-lists. E.g., **[1, 2, 3, 4, 5]** -> **"exchange 2"** -> result: **[4, 5, 1, 2, 3]**
  + If the index is outside the boundaries of the list, print "**Invalid index**"
  + **A negative index is considered invalid**
* **"max** **even/odd"**– returns the **INDEX** of the **max even/odd element**. E.g., **[1, 4, 8, 2, 3]** -> **"max odd"** -> print: **4**
* **"min** **even/odd"** – returns the **INDEX** of the **min even/odd element**. E.g. **[1, 4, 8, 2, 3]** -> **"min even"** -> print: 3
  + If there are two or more equal **min/max** elements, return the index of the **rightmost** one
  + If a **min/max even/odd** element **cannot** be found, print **"No matches"**
* **"first {count} even/odd"** – returns the **first** **count** **even/odd** **elements**. E.g. **[1, 8, 2, 3]** -> **"first 2 even"** -> print **[8, 2]**
* **"last {count} even/odd"** – returns the **last count even/odd elements**. E.g. **[1, 8, 2, 3]** -> **"last 2 odd"** -> print **[1, 3]**
  + If the **count is greater** than the **list** **length**, print **"Invalid count"**
  + If there are **not** **enough** elements to satisfy the count, print **as many as you can**. If there are **zero** **even/odd** elements, print an empty list **"[]"**
* **"end"** - **stop** taking input and **print the final state** of the list

### Input

* The input data should be read from the console.
* On the first line, the initial list is received as a line of integers, separated by a single space.
* On the following lines, until the command "**end**" is received, you will receive the list manipulation commands.
* The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

* The output should be printed on the console.
* On a separate line, print the output of the corresponding command.
* On the last line, print the final list in **square brackets** with its elements separated by a comma and a space.
* See the examples below to get a better understanding of your task.

### Constraints

* The **number of input lines** will be in the range [2 … 50].
* The **list elements** will be integers in the range [0 … 1000].
* The **number of elements** will be in the range [1 .. 50].
* The **split index** will be an integer in the range [-231 … 231 – 1].
* The **first/last count** will be an integer in the range [1… 231 – 1].
* There will **not** be redundant whitespace anywhere in the input.
* Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 3 5 7 9  exchange 1  max odd  min even  first 2 odd  last 2 even  exchange 3  end | 2  No matches  [5, 7]  []  [3, 5, 7, 9, 1] |
| 1 10 100 1000  max even  first 5 even  exchange 10  min odd  exchange 0  max even  min even  end | 3  Invalid count  Invalid index  0  2  0  [10, 100, 1000, 1] |
| 1 10 100 1000  exchange 3  first 2 odd  last 4 odd  end | [1]  [1]  [1, 10, 100, 1000] |