# Problem 1. SoftUni Airline

Mary has finally become a junior developer and has received her first task. It’s about managing flights and you need to help her.

Your **income** is calculated based on **how many tickets you sell**. There are two types of tickets. The first is for **adult** and the second is for **youth**. Income is calculated by the formula:

(adult passengers count \* adult ticket price) + (youth passengers count \* youth ticket price)

You also have **expenses**. They are calculated based on the **fuel** **you are burning during a flight**. You will receive the **fuel consumption per hour**, the **fuel price** for 1-hour flight and the **flight duration**. To calculate the expenses, use the following formula:

flight duration \* fuel consumption \* fuel price

For **each flight** you need to **calculate your profit**, **subtracting** the **expense** from the **income**, and you need to print the result. After **all flights**, you need to **calculate** your **overall** and **average** **profit**.

## Input

On the first line you will receive an integer **N**, the number of flights you need to manage. For **each flight** you will receive **exactly 7 lines** of input:

* **adult passengers count**
* **adult ticket price**
* **youth passengers count**
* **youth ticket price**
* **fuel price per hour**
* **fuel consumption per hour**
* **flight duration**

## Output

* For **each flight** you need to **calculate the profit** and **print** the result in the following format:
  + If the **income** is **greater** **or equal** than the **expense**:

You are ahead with {profit}$.

* + If the **expenses** are **greater**:

We’ve got to sell more tickets! We’ve lost {profit}$.

* After **all of the flights** you need to **print** the **overall** **profit** in the following format:

Overall profit -> {overallProfit}$.  
Average profit -> {averageProfit}$.

The output **must be rounded** to **three** **decimal** **places** after the decimal point.

## Constraints

* The **adult and youth passengers count** will be integers in range **[0…1,000,000,000]**.
* The **adult and youth ticket price** will befloating-point numbers in range **[1…1,000,000,000.00]**.
* The **fuel** **price** will be **floating**-**point** number in range [**1…1,000,000.00**].
* The **fuel** **consumption** will be **floating**-**point** number in range [**1…1,000,000.00**].
* The **flight** **duration** will be **integer** in range **[0…72]**.

## Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1  6  20  11  10  15  5  3 | You are ahead with 5.000$.  Overall profit -> 5.000$.  Average profit -> 5.000$. |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  3  50  5  20  5  11  5  5  12  7  8  9  3  2 | We've got to sell more tickets! We've lost -25.000$.  You are ahead with 62.000$.  Overall profit -> 37.000$.  Average profit -> 18.500$. |

# Problem 2. SoftUni Water Supplies

In the SoftUni office we drink lots of water (and other drinks). That’s why instead of using glasses, we use bottles.

Everyone is using different bottles, but with the **same capacity**. You are given an **array of floating-point numbers**, in which **each index represents someone’s bottle**.

We have **some water in stock**, but sometimes it is **not enough to fill our bottles**. That’s why we need you. Your job is to create a program that fills our bottles and **tells us if there is enough water or not**. If there **is enough water** to fill all bottles, you need to **tell us how much water we have left**. If there are **some bottles** that are not **completely filled**, you need to **tell us how many are those bottles**, and **how much more water we need in order to fill them**. We also need to know which members of the team need to wait for more water, and instead of using names, we will use **the indexes of the bottles**.

On the **first line** you are given the **total amount of water** that we have. On the **second line** you are given **all bottles that you need to fill**. There is a tricky part. If the **total water** is an **even number**, you will start filling the bottles from the **beginning** **of the array**, **until you run out of water, or fill all bottles**. If the **number is odd**, you will **traverse** **the** **array** **from the** **end to the beginning**, filling the bottles that way. **In case you run out of water**, you need to **print the indexes** of the empty bottles in the **same order in which you are traversing the array**. On the **last line** you will receive the **maximum capacity** that **each bottle** has.

## Input

* On the **first line** you will receive integer, which represents the **total amount of water** that we have.
* On the **second line** you will receive an **array**, **representing the bottles** that you need to fill.
* On the **last line** you will receive the **capacity for each bottle** of the array.

## Output

* If there is enough water to fill all bottles print the following lines:
  + “Enough water!”
  + “Water left: {amountOfWaterLeft}l.”
* If there isn’t enough water print the following lines:
  + “We need more water!”
  + “Bottles left: {amountOfBottles}”
  + “With indexes: {index}, {index}, {index}”
    - The order of the indexes must be the same, as the order in which you are traversing the array.
  + “We need {amountOfWaterNeeded} more liters!”

## Constraints

* The **total amount of water** will be **integer** in range [0…2,147,483,647].
* The **items in the array** will be **floating-point numbers** in range [0…1.7\*10308].
* The **bottle capacity** will be **integer** in range [0…2,147,483,647].

## Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 30  2 5 1 3 2 2  5 | Enough water!  Water left: 15l. | The total water is 30 liters. That is an even number, so we start traversing the array from index 0. We fill the bottles and have 15 liters left. |

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 15  0 0 0 0 0  4 | We need more water!  Bottles left: **2**  With indexes: **1**, **0**  We need **5** more liters! | The total water is odd number, so we start traversing the array from its last element (array.lenght - 1).  We are filling the bottles one by one until we run out of water.  That happens on the cell with index=1. We need 1 more liter to fill the bottle thereforе we count it towards the empty bottles.  Now we continue to traverse the array till the end and we count how many bottles we have to fill, and we save the indexes of those bottles in the order we traverse the array (from start to end). |

# Problem 3. Matrix Operator

You are matrix operator chief, but you are tired of doing your job manually. You want to write a program, that will automate the process. The information you are processing comes from your boss, and he wants to see daily reports for the job.

You are given a **table of integers**. For that table you will need to execute some **commands**.

The **commands** are the following: remove, swap and insert. All commands come with 2 additional parameters.

* The **remove** command – **type** and **position**.
  + The **type** can be one of the following – **positive** / **negative** / **odd** / **even**. That means that you will need to **remove said elements** from the **given** **row** / **col**.
  + The position will be in the following format: {row / col index}.
* The **swap** command – the **2 rows** that you need to swap.
* The **insert** command – **row** and **number** that you need to **insert at the beginning** of the given row.
* The input stops once you receive the “**end**” command, and then you need to print the table after all operations.

### Input

* On the first line you will receive integer **r** – rows.
* On the next **r** lines, you will receive the elements for **each** **row**.
* On the next lines, you will receive commands in the **following** **format**:
  + remove {type} {position}
  + swap {firstRow} {secondRow}
  + insert {row} {element}
* The input stops when you receive the command “end”.

### Output

* The output should consist of the matrix after all commands have been executed.

### Constraints

* The rows of the table will be in range **[1…30]**.
* The columns of each row will be in range **[0…30]**.
* The elements of the table will be integers in the range **[-2,147,483,648…2,147,483,647]**.
* The commands will always be valid and in the given format.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 4  1 5 2 -6 -2 54 -1  1 7 -3  -5 -3 1 5 6  5 6 -8 -8 -3  remove even row 3  remove odd col 5  swap 1 3  end | 1 5 2 -6 -2 54 -1  5 -3  -5 -3 1 5 6  1 7 -3 | The first command **removes all even elements** from the **row** with **index=3**. That leaves the row with {5, -3}.  The next command **removes all odd elements** from the **col** with **index=5**. The only row **that has 6 or more elements** is the row with **index=0**. In that row on col-5 the **element is even**, so we **leave** **it** there.  The last command is **swap row 1 and row 3**, so we just switch their positions and **print** **the result**. |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  -1 1 2 -2 3 -3  -6 -4 -5 -2 3 -1  1  1 2 3  -5 2 6 24  remove even row 3  remove negative row 0  remove odd row 3  swap 1 3  remove positive col 4  insert 1 50  end | 1 2 3  50  1  -6 -4 -5 -2 -1  -5 2 6 24 |

# Problem 4. SoftUni Coffee Supplies

Alex is having hard time ordering coffee recently, because the new members of the team drink too much coffee. You need to help her. At the **beginning of the input**, you will receive the **keywords** for the week. Those keywords are delimiters, that Alex will use for the rest of the input. Alex will tell you **what coffee, everyone drinks**. She **should** also tell you **the quantity for each coffee type**, but **she might forget**, so you will have to assume that there in **none** left.

After you receive **all of the information** you need, you must **check how much coffee you have**, and if there in **none left from certain type**, you **must tell Alex, to order more**. Then you must monitor how much coffee each member of the team drinks. If a **coffee type ends**, you must **tell Alex to order** more. After the **end of the week**, you need to **print reports**. The **first report** should **print how much coffee is left**. It should be in the **following format**:

“coffeeType count”

This report **must** **be** **sorted** by the count for each **coffee** **type** in **descending** **order**.

The second report should tell us **which team members** have some coffee left, and what is the type of the coffee. The report **must be sorted** by coffee type **lexicographically**. If there are **team members** that drink the **same type of coffee**, their **names** **must** **be** **sorted** **lexicographically in descending order**.

## Input

* On the first line you are given the two delimiters separated by (space).
* On the next lines, until you receive “end of info” you are given one of the following lines:
  + “PersonName[firstDelimiter]CoffeeType”
  + “CoffeeType[secondDelimiter]Quantity”
* On the next lines, until you receive “end of week” you will receive how much coffee everyone drinks in the following format:
  + “PersonName count”

## Output

* When you run out of a coffee type you must print:
  + “Out of {coffeeType}”
* After you receive the command “end of week” you must print the following reports:
  + “Coffee Left:”
  + “{coffeeType} {quantity}” – All coffee types that have more than 0 quantity, sorted in **descending order by quantity**
  + “For:”
  + “{personName} {coffeeType}” – For each of the coffeeTypes from the previous report, print the team members drinking **each type of coffee**. This report must be **sorted in lexicographical order for each coffee type**. If there is **more than 1 team member** drinking the **same coffeeType**, **order them alphabetically in descending order**.

## Constraints

* The **two delimiters** will always be **different** strings
* The **Coffee Quantity** will be valid integer in the range **[0...231]**

## Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| . -  Nakov.Espresso Ricco  Trifon.Voix de la Terre  Voix de la Terre-2  Espresso Ricco-5  end of info  Nakov 3  Trifon 1  Nakov 2  end of week | Out of Espresso Ricco  Coffee Left:  Voix de la Terre 1  For:  Trifon Voix de la Terre |

|  |  |
| --- | --- |
| **Input** | **Output** |
| Bojo Slav  SimoBojoBevanda Bianca  PeshoBojoConsomme  ConsommeSlav50  ToshkoBojoConsomme  RoyalBojoCafe Crema Dolce  MeriBojoEspresso Amabile  AlexBojoEspresso Amabile  Espresso AmabileSlav10  Bevanda BiancaSlav2  Espresso AmabileSlav10  Espresso AmabileSlav13  RosiBojoEspresso Delicato  Espresso DelicatoSlav5  end of info  Simo 1  Pesho 20  Toshko 2  Rosi 3  Meri 2  Meri 2  Alex 5  Meri 5  Rosi 2  end of week | Out of Cafe Crema Dolce  Out of Espresso Delicato  Coffee Left:  Consomme 28  Espresso Amabile 19  Bevanda Bianca 1  For:  Simo Bevanda Bianca  Toshko Consomme  Pesho Consomme  Meri Espresso Amabile  Alex Espresso Amabile |