## Problem 1. Sweet Dessert

Ivancho and his girlfriend are **throwing a party**. She plans **to cook her favorite dessert**. She asks Ivancho to **buy** the **needed products**. The **number of desserts** depends on **how many people will be coming**. She can prepare the dessert **in portions of 6**. If there are **5 guests** coming, she will still **cook 6 portions**, for **10 guests** – will **cook 12**. The products for the dessert are **bananas**, **eggs** and **berries**. For **a set of 6** she needs **2 bananas**, **4 eggs** and **0.2 kilos berries**.

You will be given **the amount of money Ivancho has**, the **number of guests** and the **prices of the products**. You have to help Ivancho **calculate** if the **cash** he has is **enough to buy all of the products**, or how much more money he needs.

### Input

The input data should be read from the console. It will consist of **exactly 5 lines**:

* The **amount of cash** Ivancho has – **floating-point number** in **range [0.00…1,000,000,000.00]**
* The **number of guests – integer in range [0…1,000,000,000]**
* The **price of bananas** for a **single unit – floating-point number** in **range [0.00…1,000.00]**
* The **price of eggs** for a **single unit – floating-point number** in **range [0.00…1,000.00]**
* The **price of berries** for a **kilo – floating-point number** in **range [0.00…1,000.00]**

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.

* **If the calculated price of the products is less or equal to the money Ivancho has:**
  + “Ivancho has enough money - it would cost {the cost of the products}lv.”
* **If the calculated price of the products is more than the money Ivancho has:**
  + “Ivancho will have to withdraw money - he will need {neededMoney}lv more.”
* **All prices** must be **rounded to two digits after the decimal point.**

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 10  12  0.35  0.20  4.50 | Ivancho has enough money - it would cost 4.80lv. | For 12 guests – 2 sets of 6 portions  Needed product:  2\*(2 bananas), 2\*(4 eggs), 2\*(0.2 kilos berries)  2\*(2\*0.35) + 2\*(4\*0.20) + 2\*(0.2\*4.50) = 4.80  4.80 <= 10 – the money will be enough. |
| **Input** | **Output** | **Comments** |
| 20  33  0.60  0.50  10 | Ivancho will have to withdraw money - he will need 11.20lv more. | For 33 guests – 6 sets of 6 portions  Needed product:  6\*(2 bananas), 6\*(4 eggs), 6\*(0.2 kilos berries)  6\*(2\*0.60) + 6\*(4\*0.50) + 6\*(0.2\*10.00) = 31.20  31.20 > 20 – need 11.20 lv. more. |

## Problem 2. Array Modifier

You are given **an array with integers**. Write a program to **modify the array elements** after **processing a sequence of commands** “**swap**”, “**multiply**” or “**decrease**”. The commands are as follows:

* “swap {index1} {index2}” takes **two elements** and **swaps them**.
* “multiply {index1} {index2}” takes **element at the 1st index** and **multiplies** it with the element at **2nd index**. Save the **product at the 1st index**.
* “decrease” **decreases** **all elements** in the array **with 1**.

### Input

On the **first input line** you will be given **the initial array values** separated by a single space.

On the **next lines** you will receive commands **until** you receive the **command “end”**. The **commands are** as follow:

* “swap {index1} {index2}”
* “multiply {index1} {index2}”
* “decrease”

### Output

**The output** should be printed on the console and consist **element** **of the** **modified array** – separated by “, “ (comma and single space).

### Constraints

* Commands will be: “**swap**”, “**multiply**”, “**decrease**” and “**end**”.
* **Elements of the array** will be **integer numbers** in the range **[-231**...**231]**.
* **Count of the array elements** will be in the range **[2**...**100]**.
* **Indexes** **will be always in the range of the array**.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 23 -2 321 87 42 90 -123  swap 1 3  swap 3 6  swap 1 0  multiply 1 2  multiply 2 1  decrease  end | 86, 7382, 2369942, -124, 41, 89, -3 | 23 -2 321 87 42 90 -123 – initial values  swap 1(-2) and 3(87) ▼  23 87 321 -2 42 90 -123  swap 3(-2) and 6(-123) ▼  23 87 321 -123 42 90 -2  swap 1(87) and 0(23) ▼  87 23 321 -123 42 90 -2  multiply 1(23) 2(321) = 7383 ▼  87 7383 321 -123 42 290 -2  multiply 2(321) 1(7383) = 2369943 ▼  87 7383 2369943 -123 42 90 -2  decrease – all - 1 ▼  86 7383 2369942 -124 41 89 -3 |

## Problem 3. Target Multiplier

Write a program which reads from the console **dimensions of a matrix** and **matrix elements values**. Get a **targeted cell** and **multiply** **its value** with **the sum of all neighboring cells**. The **neighboring cells** must **change their values too**. **Each one** should be **the product** of **its initial value** and **the initial value of the targeted cell**. Then **print on the console updated matrix**.

### Input

The input data should be read from the console:

* The **first line** holds the number of **rows – R** and **columns – C,** separated by space.
* The **next R lines** hold the **matrix values**. Each line holds **C** **integers**, separated by space.
* The **last line** holds **the position** (targeted **row** and targeted **col**) **of the targeted cell**, separated by space

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. The elements of each row should be separated by space.

### Constraints

* The **dimensions** of the matrix **(R and C)** will be a **positive integer numbers** in the range **[3...20]**.
* The **values of the cells** will be an **integer numbers** in range **[-16,300... 16,300]**.
* The **targeted row** will be an **integer number** in the range **[1...R-2]**.
* The **targeted column** will be an **integer number** in the range **[1...C-2]**.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 5 5  10 12 14 16 17  10 12 14 16 17  10 12 14 16 17  10 12 14 16 17  10 12 14 16 17  2 2 | 10 12 14 16 17  10 168 196 224 17  10 168 1568 224 17  10 168 196 224 17  10 12 14 16 17 | Targeted cell is [2,2] = 14  The sum all neighboring cells is:  12 + 14 + 16 + 12 + 16 + 12 + 14 + 16 = 112  The targeted cell new value = 14 \* 112 = 1568  Neighboring cells new values:  [1,1]=12\*14=168; [1,2]=14\*14=196; [1,3]=16\*14=224;  [2,1]=12\*14=168; [2,3]=14\*14=224;  [3,1]=12\*14=168; [3,2]=14\*14=196; [3,3]=16\*14=224 |
| **Input** | **Output** | |
| 6 4  0 105 420 480  1 143 624 744  2 182 628 488  3 226 326 538  4 263 376 406  5 -1 -2 -3  4 2 | 0 105 420 480  1 143 624 744  2 182 628 488  3 84976 122576 202288  4 98888 659128 152656  5 -376 -752 -1128 | |

# Problem 4 – Population Aggregation

Write a program that receives as input information about **country, city and** its **population** and prints an **aggregated statistic**. There are **2 types of input** lines

* **{Country}**\**{city}**\**{population}**
* **{city}**\**{Country}**\**{population}**

The **country name always starts with a capital letter** and the **city name always starts with a lowercase letter**. The country name and the city name are both polluted with **prohibited symbols (@, #, $, & and digits from 0 to 9)**. Your task is to **clear all prohibited symbols** and **print aggregated data** about the **all the** **countries ordered alphabetically** in format:

**{Country} -> {number of cities}**

And **top 3 cities with biggest population** ordered in **descending** **order** **by population** in format:

**{city} -> {population}**

In case of **repeating towns**, count the last seen population for each town (ignore the others).

**Count all towns** in each country. In case of repeating towns for certain country, **count the duplicated towns**.

### Input

* The input data should be read from the console.
* It consists of a variable number of lines and ends when the command "**stop**" is received.
* 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.
* Print the aggregated data for each country and city in the described format.

### Constraints

* The name of the city, country and the population will be separated from each other by **a back slash ('\')**.
* The **number of input lines** will be in the range [2 … 50].
* The **population count** of each city will be an integer in the range [0 … 263 − 1].
* Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| Bulgaria\sofia\123000  burgas\Bulgaria\4456576  stop | Bulgaria -> 2  burgas -> 4456576  sofia -> 123000 | Bulgaria\sofia\100  sofia\Bulgaria\200  stop | Bulgaria -> 2  sofia -> 200 |

|  |  |
| --- | --- |
| **Input** | **Output** |
| G$er&m@an@y\berlin\1234333  pa$r###is\F&r&a&n&c&e\30000000  Bulg@aria\varn@a@#$#\32145535  Bulgaria\pom$#or$ie\3131231  l$#ond$32on\U$#434565K43\98686644  ham$#bu4300r43g\Ger$man2@y\1324  stop | Bulgaria -> 2  France -> 1  Germany -> 2  UK -> 1  london -> 98686644  varna -> 32145535  paris -> 30000000 |