# Exam Preparation

Part I: [judge.softuni.bg/Contests/476](https://judge.softuni.bg/Contests/Practice/Index/476#0)

Part II: [judge.softuni.bg/Contests/477](https://judge.softuni.bg/Contests/Practice/Index/477#0)

## **Hornet Wings**

The hornets are having a cardio contest. Your task is to calculate a contestant’s distance travelled, based upon the wing flaps he made. However some hornet contestants are faster and less durable, while others are slower but have more endurance.

You will be given **N** – an **integer** indicating the **wing flaps**, a contestant has chosen to do.

After that, you will receive **M** – a **floating-point number** indicating the **distance**, in **meters**, the hornet travels for **1000 wing flaps**.

Then you will receive **P** – an **integer** indicating the **endurance** of the contestant, or **how many wing flaps** he can make, before **he stops to take a break** and rest. A hornet **rests** for **5** **seconds**.

You can assume that a hornet makes **100** wing flaps **per** **second**.

Your task is to **calculate** how much **distance** will the hornet **travel**, after it **flaps** its **wings** **N times**, and how much **time** it **took him**, to travel it. The **distance** is measured in **meters** and the time – in **seconds**.

### Input

* On the first input line you will receive N – the wing flaps, the hornet has chosen to do.
* On the second input line you will receive M – the distance the hornet travels for 1000 wing flaps.
* On the third input line you will receive P – the endurance of the hornet.

### Output

* As output you must print the total distance the hornet contestant has travelled, and the amount of time it took him.
* The output must be in the format of two lines:
* On the first output line you must print the distance: “**{metersTraveled} m.**”
* On the second output line you must print the time: “**{secondsPassed} s.**
* The **distance** must be **printed** to the **second digit** after the **decimal point**.

### Constrains

* The integer **N** – the wing flaps, will be in **range [0; 1,000,000,000]**.
* The floating-point number **M** – the distance for 1000 wing flaps, will be in **range [0; 1,000,000]**.
* The integer **P** – the endurance, will be in range **[1; N]**.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 2000  5  200 | 10.00 m.  70 s. | The contestant has chosen to do **2000 wing flaps**.  He moves **5 meters** per **1000** wing flaps.  He rests every **200** **wing flaps** for **5** **seconds**.  The distance is **(2000 / 1000) \* 5 = 2 \* 5 = 10.00** meters.  The hornet flaps **100 times** for a **second**, so **2000 / 100 =** **20** seconds.  But it also rests for **5** seconds every **200** flaps.  **(2000 / 200) \* 5 = 10 \* 5** = **50**; **20** + **50** = **70** seconds. |
| 1000000  10  1500 | 10000.00 m.  13330 s. |

## Hornet Comm

The Hornet Ex-King – Horny, has established an innovative technology providing communication for his fellow hornets, called Hornet Comm. Inc. Hornet Comm. provides functionality from private messages to wide broadcasts.

You will be given data of several tracked comm. channels, which you must decrypt. The input data will come in the following format:

{firstQuery} <-> {secondQuery}

If the **first query** consists **only of digits** and the **second one** consists of **digits and / or letters**, it is a **private message**.

If the **first query** consists of **anything but digits**, and the **second one** consists of **letters and / or digits,** it is a **broadcast**.

Any input that **does** **NOT** **follow** the format, specified above, should be **IGNORED**.

If the **given data** is a **private message**, the first query is the **recipient’s code**, and the second query is the **message**. You must **reverse** the **recipient’s code** and **store** it along with the message.

If the **given data** is a **broadcast**, the first query is the message, and the second query is the **frequency**. You must take the **frequency** and make **all capital letters** – **small** and **all small letters** – **capital**. Then you must **store** it, along with the **message**.

You must **keep** all input broadcasts and messages after you **decrypt** them.

When you receive the command “**Hornet is Green**”, the input sequences **ends**, and you must print the stored broadcasts and messages.

### Input

* The input comes in the form of several input lines in the format specified above.
* The input ends when you receive the command “**Hornet is Green**”.

### Output

* As output you must print all broadcasts and messages, printing first the broadcasts, in the following format:
  + Broadcasts:
  + {frequency} -> {message}
  + . . .
  + Messages:
  + {recipient} -> {message}
  + . . .
* If there are **no messages**, or **no broadcasts**, print “**None**” in their place.

### Constrains

* The input lines may consist of **any ASCII** character.
* There will be **NO** more than 1000 lines of input.
* **All data** must be printed in **order of input**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 213094 <-> BeeQueenDown  213094 <-> Repeat  Foxtrot <-> 123321  213094 <-> BeeQueenDown  Unicorn <-> 871203  Charlie <-> 56210  Kilo <-> 423211  Hornet is Green | Broadcasts:  123321 -> Foxtrot  871203 -> Unicorn  56210 -> Charlie  423211 -> Kilo  Messages:  490312 ->BeeQueenDown  490312 ->Repeat  490312 ->BeeQueenDown |
| <+>.<+> <-> az13b6  <->.<-> <-> P2Z4x789  12345 <-> Pr1v@teM3ssage  Hornet is Green | Broadcasts:  AZ13B6 -> <+>.<+>  p2z4X789 -> <->.<->  Messages:  None |

## Hornet Assault

The hornets are preparing an assault on beehives. It takes very little amount of hornets to annihilate a beehive, but the bees are far from defenseless. You must calculate how many beehives, can the hornets annihilate, before they die.

You will be given a **sequence of integers**, separated by a **space**. The integers will represent the **beehives** and the **amount of bees** in them. You will then receive **another** sequence of integers, which will represent the **hornets**, and their **power**. The **power** indicates **how many bees** a hornet can **kill**.

The hornets must **start attacking** the beehives **one** by **one**. If the bees are **more** or **equal to** the **summed-up power** of the **hornets**, the **first** (**entered**) hornet, **currently alive**, **dies** in the assault of the **current beehive**.

When the hornets assault a beehive, they **kill** an **amount of bees**, **equal** to their **summed-up power**. This means that you must **decrease** the **integer** of the **beehive**, with the **value** of the summed-up power, of the **live hornets**.

After you’ve successfully assaulted all beehives, you must print the result of the **winning side**. If there are **ANY** **beehives** with **live bees** inside them, you must print them. If there aren’t, you must print the **live hornets**.

### Input

* On the first line of input you will receive a sequence of integers, separated by **spaces** – the **beehives**.
* On the second line of input you will receive a sequence of integers, separated by **spaces** – the **hornets**.

### Output

* Depending on the case of printing you must either print the **live** beehives, or the **live** hornets.
* They are sequences of integers, so in both cases you must print them **separated** by a **space**.

### Constrains

* The input will consist only of integers in **range [0; 1,000,000]**.
* There will be **NO** **STALEMATE** situations.
* The input sequences may consist of up to **3000** elements.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 20 10 20 30  5 10 5 3 | 7 | The **summed power** of the **hornets** is 23. They kill the first **3 beehives**, due to overwhelming power.  The last beehive has **higher value**, and its left with **7 bees** **alive** inside it. The **first hornet** (**5**) **dies**.  All other beehives **are dead**, so we print **only this one**. |
| 10 20 10 15  5 6 7 | 2 2 | The **summed power** is **18**. The first beehive dies. In the second one, 18 bees die, but due to higher power, the **first hornet** (**5**) **dies**.  Now the **summed power** is **13**. The third beehive dies, but the fourth one has **higher** value. **13 bees die** from the **fourth** beehive and the **current first hornet** (**6**) **dies**.  We have the **second** and the **fourth** beehive with **2 bees**, each, so we print them. |
| 20 100 40 45 20 10  40 10 5 40 5 | 1. 40 5 |

## Hornet Armada

The Hornet Overlord Nostalgia, who is famed for his absolute discipline and strict orders, owns the most sorted army in the Hornet history. Help Nostalgia “computerize” the process of sorting out his army.

You will be given **N** – an integer.  
On the next **N** lines you will be given input containing information about soldiers in the following format:

**{lastActivity} = {legionName} -> {soldierType}:{soldierCount}**

The **last activity** is an **integer**. The **legion name** and **soldier type**, will both be **strings**. The **soldier count** will be an **integer**. You must **store** **every** **legion** with its **activity**, and **every** **soldier type** with its **count**, in its **legion**.

If a **given legion already exists**, you must **add** the new **soldier type**, with its count. If the soldier type exists **ALSO**, you should just **add** the **soldier count**.

**IN** **BOTH** cases, stated above, you should **update** the **last** **activity**, with the newly entered one, **ONLY** if the **entered** **one** is **GREATER** than the **previous one**.

After you’ve read **all N** input lines, you will receive a line in one of the following formats:

* {activity}\{soldierType}
* {soldierType}

In the **first case,** you must print all **legions**, and the **count of soldiers** they have from the **given** **soldier** **type**, who’s **last activity** is **LOWER** than the **given activity**. The legions must be printed in **descending order** by **soldier count**.

In the **second case**, you must print all legions which **have** the **given soldier type**, with **last activity**, and **legion name**. The legions must be printed in **descending** **order** oftheir **activity**.

### Input

* On the first line you will receive **N** –the **integer**.
* On the next **N** lines you will receive data about **soldiers** and **legions**.
* On the last line you will receive **one** of the **two commands**, which will **determine** the **output**.

### Output

* If you are given the **last activity** and **soldier type** on the last command, you must print the legions in this format:
* {legionName} -> {soldierCount}
* If you are given **only** the **soldier type** on the last command, you must print the legions in this format:
* {lastActivity} : {legionName}

### Constrains

* The first integer – **N**, will be in **range [0; 10,000]**.
* The **legion names** and **soldier types** may consist of **any ASCII** character, except “**=**”, “**-**”, “**>**”, “**:**”, “ ”(**space**).
* The **soldier count** and **last activity** will be integers in **range [0; 1,000,000,000]**.
* All input data will be exactly as stated above. There will be **NO invalid** input lines.
* Data which has **NO specified order** must be sorted in **order of** **input**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 6  1 = BlackBeatles -> Soldier:2000  2 = BlackBeatles -> Worker:1000  1 = Red\_Ones -> Soldier:10000  5 = Rm -> Soldier:30000  2 = Red\_Ones -> Soldier:20000  10 = RND -> Soldier:100000  10\Soldier | Red\_Ones -> 30000  Rm -> 30000  BlackBeatles -> 2000 |
| 7  1000 = F1rstL3gion -> Aisers:15000  500 = F1rstL3gion -> Aisers:1000  200 = F1rstL3gion -> Guards:2000  2000 = Second!egion -> Guards:2000  1500 = Second!egion -> Aisers:15000  2500 = Second!egion -> Spies:2000  1000 = Forked\_Ones -> Guards:10000000  Guards | 2500 : Second!egion  1000 : F1rstL3gion  1000 : Forked\_Ones |