# Problem 1. Charity Marathon

Every year a charity marathon takes place in your town in which all major companies are obliged to **make donations depending on the total kilometers ran by runners in a number of days**. And this year you have been chosen to create the software for it.

The **marathon can last for variable number days** and a **variable number of runners can participate** in it on a **track that can have a variable length**. However, the **track that can take only a limited number of runners per day**. If the runners that want to take part are more than the capacity, then the number of runners that will run will be **equal to the maximum capacity of the track**.

The **amount of money raised per kilometer is voted** in advance by all companies and the final money per kilometer is **calculated by an average of all votes**.

The goal is simple, create a program that calculates the total money raised through the marathon.

## Input

* On the first line of input you will get the **length of the marathon in days**
* On the second line of input you will get **the number of runners that will participate**
* On the third line you will get the **average number of laps every runner** makes
* On the fourth line you will get the **length of the track**
* On the fifth line you will get the **capacity of the track**
* On the sixth line you will get **the amount of money donated per kilometer**

## Output

* Print the money raised, **rounded by the second digit after the decimal point** from the charity marathon in the format: "**Money raised: {money}**"

## Constraints

* Marathon day count will be an integer in the range [0 … 365]
* Runner count will be an integer in the range [0 … 2,147,483,647]
* Average number of laps will be an integer in the range [0 … 100]
* Lap length will be an integer in the range [0 … 2,147,483,647]
* Track capacity will be an integer in the range [0 … 1000]
* Money per kilometer will all be a floating point number

## Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| 2  50  2  400  30  3 | Money raised: 120.00 | The marathon runs for 2 days, with 50 runners that will make an average of 2 laps on a track that is long 400 meters.  The capacity of the track is 30 runners per day. So a total of 30 \* 2 = 60 maximum runners. But only 50 runners are listed, so 50 will run.  Total meters = 50 runners \* 2 laps \* 400 m = 40,000 m  Total kilometers = 40,000 m / 1,000 = 40 km  Money raised by kilometer = 3  Money raised for the marathon = 40 \* 3 = 120 |
| 1  50  10  400  1  2.5 | Money raised: 10.00 | The listed runners are 50, but the maximum capacity of the track is 1 runner per day and the marathon will last for 1 day. So 1 runner will run in total. |

# Problem 2. 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  0 1  0 right 1  2 right 1  end | 0 1 0 | 1 1 0 - Initial field  0 1 1 - field after "0 right 1"  0 1 0 - field after "2 right 1" |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 3  0 1 2  0 right 1  1 right 1  2 right 1  end | 0 0 0 |  | 5  3  3 left 2  1 left -2  end | 0 0 0 1 0 |

**Problem 3 – Nether Realms**

Mighty battle is coming. In the stormy nether realms, demons are fighting against each other for supremacy in a duel from which only one will survive.

Your job, however is not so exciting. You are assigned to **sign in all the participants** in the nether realm's mighty battle's demon book, which of course is **sorted alphabetically**.

A demon's **name contains his health and his damage**.

The **sum of the asci codes** of **all characters** (excluding numbers (0-9), arithmetic symbols (**'+', '-', '\*', '/'**) and delimiter dot (**'.'**)) gives a **demon's total health**.

**The sum of all numbers** in his name forms his base damage. Note that you should consider the plus **'+'** and minus **'-'** signs (e.g. **+10 is 10** and **-10 is -10**). However, there are some symbols (**'\*'** and **'/'**) that can further **alter the base damage by multiplying or dividing it by 2** (e.g. in the name "m**15**\*/c**-5.0**", the base damage is **15 + (-5.0) = 10** and then you need to multiply it by 2 (e.g. 10 \* 2 = 20) and then divide it by 2 (e.g. 20 / 2 = 10)).

So, **multiplication and division** are applied **only after all numbers are included** in the calculation and **in the order they appear in the name**.

You will get all demons **on a single line**, separated by commas and zero or more blank spaces. Sort them in **alphabetical order** and print their names **along their health and damage**.

## Input

The input will be read from the console. The input consists of a **single line** containing all demon names **separated by commas and zero or more spaces** in the format: **"{demon name}, {demon name}, … {demon name}"**

## Output

Print all demons **sorted by their name in ascending order**, each on a separate line in the format:

* **"{demon name} - {health points} health, {damage points} damage"**

## Constraints

* A demon's name will contain **at least one character**
* A demon's name **cannot contain** blank spaces ' ' or commas ','
* A **floating point number will always have digits before and after its decimal separator**
* **Number** in a demon's name **is considere**d everything that is a valid integer or floating point number (with dot '.' used as separator). For example, all these are valid numbers: '4', '+4', '-4', '3.5', '+3.5', '-3.5'

## Examples

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Output** | **Comments** | |
| M3ph-0.5s-0.5t0.0\*\* | M3ph-0.5s-0.5t0.0\*\* - 524 health, 8.00 damage | M3ph-0.5s-0.5t0.0\*\*:  Health = 'M' + 'p' + 'h' + 's' + 't' = 524 health.  Damage = (3 + (-0.5) + (-0.5) + 0.0) \* 2 \* 2 = 8 damage. | |
| **Input** | **Output** | | **Comments** |
| M3ph1st0\*\*, Azazel | Azazel - 615 health, 0.00 damage  M3ph1st0\*\* - 524 health, 16.00 damage | | Azazel:  Health - 'A' + 'z' + 'a' + 'z' + 'e' + 'l' = 615 health. Damage - no digits = 0 damage.  M3ph1st0\*\*:  Health - 'M' + 'p' + 'h' + 's' + 't' = 524 health.  Damage - (3 + 1 + 0) \* 2 \* 2 = 16 damage. |
| Gos/ho | Gos/ho - 512 health, 0.00 damage | |  |

# Problem 4. Roli - The Coder

Roli is really busy with the recently started JS Core and DB Fundamentals modules at SoftUni. She is used to going out with friends on various events. However, when the times comes, you need to tell her to start coding.

Roli is the organizer of those events, so she needs to **keep track of the unique participants for each event**. She saves the events by '**ID**', which is the **unique code** for each event. **For each ID**, she **keeps the event name and the participants** for it.

She receives request in the following format:

* {**id**} #{**eventName**} @{**participant1**} @{**participant2**} … @{**participantN**}

If she is given event in an **invalid format** (such as without a hashtag), she **ignores that line of input**. If she is given ID that **already exists** she needs to check if the **eventName** is the same. If it is, she **adds** the **participants** from the **request** to the other registered participants. If the event id exists but the name doesn’t, it is invalid and you need to ignore it.

After you receive "**Time for Code**", you need to **print** the results. All events must **be sorted by participant count in descending order and then by alphabetical order**. For each event, the **participants** must be sorted in **alphabetical order**.

## Input / Constrains

* Unil you receive “**Time for Code**” you will get lines of input in which everything is separated by one or more blank spaces
* Until you receive "**Time for Code**", you will be receiving events in the following format:
  + **{id} #{eventName} @{participant1} @{participant2} … @{participantN}**
* The participant names may contain **Latin letters**, **digits**, **single quotes** (') and **dashes** (-).

## Output

* All events must be sorted in **descending** order by participant count and **then by alphabetical order**. For each event, you need to print:
* {eventName} – {participantCount}
* On the next lines you need to print all participants. All participants for an event must be sorted alphabetically.

## Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 #Beers @roli @trophon @alice  2 #GameDevMeetup @sino @valyo  3 #Karaoke @nakov @royal @ROYAL @ivo  Time for Code | Karaoke - 4  @ivo  @nakov  @royal  @ROYAL  Beers - 3  @alice  @roli  @trophon  GameDevMeetup - 2  @sino  @valyo |
| **Input** | **Output** |
| 1 #Beers @roli @trophon @alice  2 #GameDevMeetup @sino @valyo  1 #Beers2 @nakov @royal @ROYAL @ivo  1 #Beers @roli @trophon @alice @sino  3 #Rakia  Time for Code | Beers - 4  @alice  @roli  @sino  @trophon  GameDevMeetup - 2  @sino  @valyo  Rakia - 0 |