**More Exercise: Associative arrays**

Problems for more exercises and homework for the ["Technology Fundamentals" course @ SoftUni](https://softuni.bg/courses/technology-fundamentals).

You can check your solutions in [Judge](https://judge.softuni.bg/Contests/1310/).

## Cappy Juice

You will be given different juices, as **strings**. You will also **receive quantity** as a **number**. If you receive a juice, you already have, **you must sum** the **current quantity** of that juice, with the **given one**. When a juice reaches **1000 quantity**, it produces a bottle. You must **store all produced bottles** and you must **print them** at the end.

**Note:** **1000 quantity** of juice is **one bottle**. If you happen to have **more than 1000**, you must make **as much bottles as you can**, and store **what** **is** **left** from the juice.

**Example:** **You have 2643 quantity** of Orange Juice – this is **2 bottles** of Orange Juice and **643 quantity left**.

Each element holds data about a juice and quantity in the following format:

"{juiceName} => {juiceQuantity}"

The program must stop reading data from the console when the “**End**” command is received!

The **output** is the produced bottles. The bottles must be printed in **order of obtaining the bottles**. Check the second example bellow - even though we receive the Kiwi juice first, we don’t form a bottle of Kiwi juice until the 4th line, at which point we have already create Pear and Watermelon juice bottles, thus the Kiwi bottles appear last in the output.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Orange => 2000  Peach => 1432  Banana => 450  Peach => 600  Strawberry => 549  End | Orange => 2  Peach => 2 |
| Kiwi => 234  Pear => 2345  Watermelon => 3456  Kiwi => 4567  Pear => 5678  Watermelon => 6789  End | Pear => 8  Watermelon => 10  Kiwi => 4 |

## Store Catalogue

You have to create a sorted catalogue of store products. You will be given the product names and prices. You need to order them by **alphabetical order**.

The input is separated on 2 parts:

The **first part** is one single line which is integer **n**, the number of how many lines are in the **2nd part**.

The 2nd part is **n** number of **strings**. Each **string** holds info about a product in the following format:

"{productName} : {productPrice}"

The **product’s name** will be a **string**, which will **always** **start with a capital letter**, and the **price** will be **a number**. You can safely assume there will be **NO duplicate product input**. The comparison for alphabetical order is **case-insensitive**.

As **output** you must print all the products in a specified format. They must be ordered **exactly as specified above**. The products must be **divided into groups**, by the **initial of their name**. The **group’s initial should be printed**, and after that the products should be printed with **2 spaces before their names**. For more info check the examples.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 8  Appricot : 20.4  Fridge : 1500  TV : 1499  Deodorant : 10  Boiler : 300  Apple : 1.25  Anti-Bug Spray : 15  T-Shirt : 10 | A  Anti-Bug Spray: 15  Apple: 1.25  Appricot: 20.4  B  Boiler: 300  D  Deodorant: 10  F  Fridge: 1500  T  T-Shirt: 10  TV: 1499 |
| 9  Banana : 2  Rubic's Cube : 5  Raspberry P : 4999  Rolex : 100000  Rollon : 10  Rali Car : 2000000  Pesho : 0.01  Barrel : 10  Weapon : 3 | B  Banana: 2  Barrel: 10  P  Pesho: 0.01  R  Rali Car: 2000000  Raspberry P: 4999  Rolex: 100000  Rollon: 10  Rubic's Cube: 5  W  Weapon: 3 |

## MOBA Challenger

Pesho is a pro MOBA player, he is struggling to become master of the Challenger tier. So he watches carefully the statistics in the tier.

You will receive **several input lines** in one of the following formats:

"{player} -> {position} -> {skill}"

"{player} vs {player}"

The player and position are strings, the given **skill** will be an integer number. You need to keep track of **every player**.

When you receive a **player and his position and skill**, add him to the player pool, if he isn`t present, **else add** his position and skill **or update** his skill, only if the current position skill is lower than the new value.

If you receive **"{player} vs {player}"** and **both players exist** in the tier, **they duel** with the following rules:

Compare their positions, **if they got at least one in common**, the player with better **total skill points** wins and the other is **demoted** from the tier -> remove him. If they have same total skill points, **the duel is tie** and they both continue in the Season.

If they don`t have positions in common, **the duel isn`t happening** and both continue in the Season.

You should end your program when you receive the command "Season end". At that point you should print the players, **ordered by total skill in desecending order, then ordered by player name in ascending order**. **Foreach** player print their position and skill, **ordered desecending by skill, then ordered by position name in ascending order.**

### Input / Constraints

* The input comes in the form of commands in one of the formats specified above.
* Player and position **will always be one word string, containing no whitespaces**.
* Skill will be an **integer** in the **range [0, 1000]**.
* There will be **no invalid** input lines.
* The programm ends when you receive the command "Season end".

### Output

* The output format for each player is:

"{player}: {totalSkill} skill"

"- {position} <::> {skill}"

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| Pesho -> Adc -> 400  Gosho -> Jungle -> 300  Stamat -> Mid -> 200  Stamat -> Support -> 250  Season end | Stamat: 450 skill  - Support <::> 250  - Mid <::> 200  Pesho: 400 skill  - Adc <::> 400  Gosho: 300 skill  - Jungle <::> 300 | We order the players by total skill points descending, then by name. We print every position along its skill ordered descending by skill, then by position name. |
| Pesho -> Adc -> 400  Bush -> Tank -> 150  Faker -> Mid -> 200  Faker -> Support -> 250  Faker -> Tank -> 250  Pesho vs Faker  Faker vs Bush  Faker vs Hide  Season end | Faker: 700 skill  - Support <::> 250  - Tank <::> 250  - Mid <::> 200  Pesho: 400 skill  - Adc <::> 400 | Faker and Pesho don`t have common position, so the duel isn`t valid.  Faker wins vs Bush /common position: "Tank". Bush is demoted.  Hide doesn`t exist so the duel isn`t valid.  We print every player left in the tier. |

## Snowwhite

Snow White loves her dwarfs, but there are so many and she doesn’t know how to order them. Does she order them by name? Or by color of their hat? Or by physics? She can’t decide, so its up to you to write a program that does it for her.

You will be receiving **several input lines** which contain **data** about **dwarfs** in the following format:

{dwarfName} <:> {dwarfHatColor} <:> {dwarfPhysics}

The dwarfName and the dwarfHatColor are **strings**. The dwarfPhysics is an **integer**.

You must **store** the **dwarfs** in your program. There are several rules though:

* If **2 dwarfs** have the **same name** but **different color**, they should be **considered different dwarfs**, and you should store **both** of them.
* If **2 dwarfs** have the **same name** and the **same color**, **store** the **one** with the **higher physics**.

When you receive the command “Once upon a time”, the input ends. You must **order** the **dwarfs** by **physics** in **descending order** and thenby **total** **count** of **dwarfs** with the **same hat color** in **descending order**.   
Then you must print them all.

### Input

* The input will consists of **several input lines**, containing **dwarf data** in the format, specified above.
* The input **ends** when you receive the command “Once upon a time”.

### Output

* As output you must print the **dwarfs**, **ordered** in the way , specified above.
* The output format is: ({hatColor}) {name} <-> {physics}

### Constraints

* The dwarfName will be a **string** which may contain **any ASCII** character except ‘ ’ (space), ‘<’, ‘:’, ‘>’.
* The dwarfHatColor will be a **string** which may contain **any ASCII** character except ‘ ’ (space), ‘<’, ‘:’, ‘>’.
* The dwarfPhysics will be an **integer** in **range [0, 231 – 1]**.
* There will be **no invalid** input lines.
* If **all sorting criteria fail**, the order should be by **order** of **input**.
* Allowed working **time** / **memory**: **100ms** / **16MB**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Pesho <:> Red <:> 2000  Tosho <:> Blue <:> 1000  Gosho <:> Green <:> 1000  Sasho <:> Yellow <:> 4500  Prakasho <:> Stamat <:> 1000  Once upon a time | (Yellow) Sasho <-> 4500  (Red) Pesho <-> 2000  (Blue) Tosho <-> 1000  (Green) Gosho <-> 1000  (Stamat) Prakasho <-> 1000 |
| Pesho <:> Red <:> 5000  Pesho <:> Blue <:> 10000  Pesho <:> Red <:> 10000  Gosho <:> Blue <:> 10000  Once upon a time | (Blue) Gosho <-> 10000  (Blue) Pesho <-> 10000  (Red) Pesho <-> 10000 |

## Dragon Army

Heroes III is the best game ever. Everyone loves it and everyone plays it all the time. Stamat is no exclusion to this rule. His favorite units in the game are all **types** of dragons – black, red, gold, azure etc… He likes them so much that he gives them **names** and keeps logs of their **stats**: **damage, health** and **armor**. The process of aggregating all the data is quite tedious, so he would like to have a program doing it. Since he is no programmer, it’s your task to help him

You need to categorize dragons by their **type**. For each dragon, identified by **name,** keep information about his **stats.** Type is **preserved** as in the order of input, but dragons are **sorted** alphabetically by name. For each type, you should also print the average **damage**, **health** and **armor** of the dragons. For each dragon, print his own stats.

There **may** be **missing** stats in the input, though. If a stat is missing you should assign it default values. Default values are as follows: health **250**, damage **45**, and armor **10**. Missing stat will be marked by **null**.

The input is in the following format **{type} {name} {damage} {health} {armor}.** Any of the integers may be assigned null value. See the examples below for better understanding of your task.

If the same dragon is added a second time, the new stats should **overwrite** the previous ones. Two dragons are considered **equal** if they match by **both** name and type.

### Input

* On the first line, you are given number N -> the number of dragons to follow
* On the next N lines, you are given input in the above described format. There will be single space separating each element.

### Output

* Print the aggregated data on the console
* For each type, print average stats of its dragons in format **{Type}::({damage}/{health}/{armor})**
* Damage, health and armor should be rounded to two digits after the decimal separator
* For each dragon, print its stats in format **-{Name} -> damage: {damage}, health: {health}, armor: {armor}**

### Constraints

* N is in range [1…100]
* The dragon type and name are one word only, starting with capital letter.
* Damage health and armor are integers in range [0 … 100000] or **null**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  Red Bazgargal 100 2500 25  Black Dargonax 200 3500 18  Red Obsidion 220 2200 35  Blue Kerizsa 60 2100 20  Blue Algordox 65 1800 50 | Red::(160.00/2350.00/30.00)  -Bazgargal -> damage: 100, health: 2500, armor: 25  -Obsidion -> damage: 220, health: 2200, armor: 35  Black::(200.00/3500.00/18.00)  -Dargonax -> damage: 200, health: 3500, armor: 18  Blue::(62.50/1950.00/35.00)  -Algordox -> damage: 65, health: 1800, armor: 50  -Kerizsa -> damage: 60, health: 2100, armor: 20 |
| 4  Gold Zzazx null 1000 10  Gold Traxx 500 null 0  Gold Xaarxx 250 1000 null  Gold Ardrax 100 1055 50 | Gold::(223.75/826.25/17.50)  -Ardrax -> damage: 100, health: 1055, armor: 50  -Traxx -> damage: 500, health: 250, armor: 0  -Xaarxx -> damage: 250, health: 1000, armor: 10  -Zzazx -> damage: 45, health: 1000, armor: 10 |