Exercises: Dictionaries, Lambda and LINQ

Problems for exercises and homework for the "Programming Fundamentals" course @ SoftUni.

Check your solutions here: https://judge.softuni.bg/Contests/209/Strings-Dictionaries-Lambda-and-LINQ-Exercises.

1. Phonebook

Write a program that receives some info from the console about **people** and their **phone numbers**. Each **entry** should have just **one name** and **one number** (both of them strings).

On each line, you will receive some of the following commands:

- **A {name} {phone}** – adds entry to the phonebook. In case of trying to add a name that is already in the phonebook you should change the existing phone number with the new one provided.
- **S {name}** – searches for a contact by given name and prints it in format "**`{name} -> {number}`**". In case the contact isn't found, print "**`Contact {name} does not exist.`**".
- **END** – stop receiving more commands.

Examples

Input	Output
A Nakov 0888080808 S Mariika S Nakov END	Contact Mariika does not exist. Nakov -> 0888080808
A Nakov +359888001122 A RoYaL(Ivan) 666 A Gero 5559393 A Simo 02/987665544 S Simo S simo S RoYaL S RoYaL(Ivan) END	Simo -> 02/987665544 Contact simo does not exist. Contact RoYaL does not exist. RoYaL(Ivan) -> 666
A Misho +359883123 A Misho 02/3123 S Misho END	Misho -> 02/3123

Hints

- **Parse the commands** by splitting by space. Execute the commands until "**END**" is reached.
- Store the **phonebook entries** in **Dictionary<string, string>** with key **{name}** and value **{phone number}**.

2. Phonebook Upgrade

Add functionality to the phonebook from the previous task to **print all contacts ordered lexicographically** when receive the command "**`ListAll`**".

Examples

Input	Output
A Nakov +359888001122 A RoYaL(Ivan) 666 A Gero 5559393 A Simo 02/987665544 ListAll END	Gero -> 5559393 Nakov -> +359888001122 RoYaL(Ivan) -> 666 Simo -> 02/987665544

Hints

1. **Variant I (slower):** Sort all entries in the dictionary by key and print them.
2. **Variant II (faster):** Keep the entries in more appropriate data structure that will keep them in sorted order for better performance.

3. A Miner Task

You are given a sequence of strings, each on a new line. Every odd line on the console is representing a resource (e.g. Gold, Silver, Copper, and so on), and every even – quantity. Your task is to collect the resources and print them each on a new line.

Print the resources and their quantities in format:

{resource} –> {quantity}

The quantities inputs will be in the range [1 … 2 000 000 000]

Examples

Input	Output
Gold 155 Silver 10 Copper 17 stop	Gold -> 155 Silver -> 10 Copper -> 17

Input	Output
gold 155 silver 10 copper 17 gold 15 stop	gold -> 170 silver -> 10 copper -> 17

4. Fix Emails

You are given a sequence of strings, each on a new line, **until you receive the "stop" command**. The first string is the **name** of a person. On the second line, you will receive their **email**. Your task is to **collect** their **names** and **emails**, and **remove** emails whose domain **ends with** "**us**" or "**uk**" (case insensitive). Print:

{name} – > {email}

Examples

Input	Output
Ivan ivanivan@abv.bg Petar Ivanov petartudjarov@abv.bg Mike Tyson myke@gmail.us stop	Ivan -> ivanivan@abv.bg Petar Ivanov -> petartudjarov@abv.bg

5. Hands of Cards

You are given a sequence of people and for every person what cards he draws from the deck. The input will be separate lines in the format:

- **{personName}: {PT, PT, PT,… PT}**

Where **P (2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A**) is the power of the card and **T (S, H, D, C)** is the type. The input ends when a **"JOKER"** is drawn. The name can contain any ASCII symbol except **':'**. The input will always be valid and in the format described, there is no need to check it.

A single person **cannot have more than one** card with the **same power and type**, if they draw such a card they **discard** it. The people are playing with **multiple decks**. Each card has a value that is calculated by the **power multiplied by the type**. Powers **2 to 10** have the same value and **J to A** are **11 to 14**. Types are mapped to multipliers the following way (**S -> 4, H-> 3, D -> 2, C -> 1**).

Finally print out the total value each player has in his hand in the format:

- **{personName}: {value}**

Examples

Input	Output
Pesho: 2C, 4H, 9H, AS, QS Slav: 3H, 10S, JC, KD, 5S, 10S Peshoslav: QH, QC, QS, QD Slav: 6H, 7S, KC, KD, 5S, 10C Peshoslav: QH, QC, JS, JD, JC Pesho: JD, JD, JD, JD, JD, JD JOKER	Pesho: 167 Slav: 175 Peshoslav: 197

6. User Logs

Marian is a famous system administrator. The person to overcome the security of his servers has not yet been born. However, there is a new type of threat where users flood the server with messages and are hard to be detected since they change their IP address all the time. Well, Marian is a system administrator and is not so into programming. Therefore, he needs a skillful programmer to track the user logs of his servers. You are the chosen one to help him!

You are given an input in the format:

- **IP=(IP.Address) message=(A&sample&message) user=(username)**

Your task is to parse the IP and the username from the input and for **every user**, you have to display **every IP** from which the corresponding user has sent a message and the **count of the messages** sent with the corresponding IP. In the output, the usernames must be **sorted alphabetically** while their IP addresses should be displayed in the **order of their first appearance.** The output should be in the following format:

```
username:
IP => count, IP => count.
```

For example, given the following input:

- "**IP=192.23.30.40 message='Hello&derps.' user=destroyer**",

You will have to get the username **destroyer** and the IP **192.23.30.40** and display it in the following format:

```
destroyer:
192.23.30.40 => 1.
```

The username **destroyer** has sent a message from IP **192.23.30.40** once.

Check the examples below. They will further clarify the assignment.

Input

The input comes from the console as **varying number** of lines. You have to parse every command until the command that follows is **end.** The input will be in the format displayed above, there is no need to check it explicitly.

Output

For every user found, you have to display each log in the format:

```
username:
IP => count, IP => count…
```

The IP addresses must be split with a comma, and each line of IP addresses must end with a dot.

Constraints

- The number of commands will be in the range **[1..50]**
- The IP addresses will be in the format of either **IPv4** or **IPv6.**
- The messages will be in the format: **`This&is&a&message`**
- The username will be a string with length in the range **[3..50]**
- Time limit: 0.3 sec. Memory limit: 16 MB.

Examples

Input	Output
IP=192.23.30.40 message='Hello&derps.' user=destroyer IP=192.23.30.41 message='Hello&yall.' user=destroyer IP=192.23.30.40 message='Hello&hi.' user=destroyer IP=192.23.30.42 message='Hello&Dudes.' user=destroyer end	destroyer: 192.23.30.40 => 2, 192.23.30.41 => 1, 192.23.30.42 => 1.
IP=FE80:0000:0000:0000:0202:B3FF:FE1E:8329 message='Hey&son' user=mother IP=192.23.33.40 message='Hi&mom!' user=child0 IP=192.23.30.40 message='Hi&from&me&too' user=child1	child0: 192.23.33.40 => 1. child1: 192.23.30.40 => 1.

IP=192.23.30.42 message='spam' user=destroyer IP=192.23.30.42 message='spam' user=destroyer IP=192.23.50.40 message='' user=yetAnotherUsername IP=192.23.50.40 message='comment' user=yetAnotherUsername IP=192.23.155.40 message='Hello.' user=unknown end	destroyer: 192.23.30.42 => 2. mother: FE80:0000:0000:0000:0202:B3FF:FE1E:8329 => 1. unknown: 192.23.155.40 => 1. yetAnotherUsername: 192.23.50.40 => 2.

7. Population Counter

So many people! It's hard to count them all. But that's your job as a statistician. You get raw data for a given city and you need to aggregate it.

On each input line, you'll be given data in format: **"city|country|population"**. There will be **no redundant whitespaces anywhere** in the input. Aggregate the data **by country and by city** and print it on the console.

For each country, print its **total population** and on separate lines, the data for each of its **cities**. **Countries should be ordered by their total population in descending order** and within each country, the **cities should be ordered by the same criterion**.

If two countries/cities have the same population, keep them **in the order in which they were entered.** Check out the examples; follow the output format strictly!

Input

- The input data should be read from the console.
- It consists of a variable number of lines and ends when the command "**report**" 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 format shown below.

Constraints

- The name of the city, country and the population count will be separated from each other by **a pipe ('|')**.
- The **number of input lines** will be in the range [2 … 50].
- A city-country pair will not be repeated.
- The **population count** of each city will be an integer in the range [0 … 2 000 000 000].
- Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

Examples

Input	Output
Sofia|Bulgaria|1000000 report	Bulgaria (total population: 1000000) =>Sofia: 1000000

Input	Output
Sofia|Bulgaria|1 Veliko Tarnovo|Bulgaria|2 London|UK|4 Rome|Italy|3 report	UK (total population: 4) =>London: 4 Bulgaria (total population: 3) =>Veliko Tarnovo: 2 =>Sofia: 1 Italy (total population: 3) =>Rome: 3

8. Logs Aggregator

You are given a sequence of access logs in format **<IP> <user> <duration>**. For example:

- `192.168.0.11 peter 33`
- `10.10.17.33 alex 12`
- `10.10.17.35 peter 30`
- `10.10.17.34 peter 120`
- `10.10.17.34 peter 120`
- `212.50.118.81 alex 46`
- `212.50.118.81 alex 4`

Write a program to aggregate the logs data and print **for each user** the **total duration** of his sessions and a **list of unique IP addresses** in format "**<user>: <duration> [<IP$_1$>, <IP$_2$>, …]**". Order the **users alphabetically**. Order the **IPs alphabetically**. In our example, the output should be the following:

- `alex: 62 [10.10.17.33, 212.50.118.81]`
- `peter: 303 [10.10.17.34, 10.10.17.35, 192.168.0.11]`

Input

The input comes from the console. At the first line a number **n** stays which says how many log lines will follow. Each of the next n lines holds a log information in format **<IP> <user> <duration>**. The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

Print **one line for each user** (order users alphabetically). For each user print its sum of durations and all of his sessions' IPs, ordered alphabetically in format **<user>: <duration> [<IP$_1$>, <IP$_2$>, …]**. Remove any duplicated values in the IP addresses and order them alphabetically (like we order strings).

Constraints

- The **count** of the order lines **n** is in the range [1…1000].
- The **<IP>** is a standard IP address in format **a.b.c.d** where **a**, **b**, **c** and **d** are integers in the range [0…255].
- The **<user>** consists of only of **Latin characters**, with length of [1…20].
- The **<duration>** is an integer number in the range [1…1000].
- Time limit: 0.3 sec. Memory limit: 16 MB.

Examples

Input	Output
7 192.168.0.11 peter 33 10.10.17.33 alex 12 10.10.17.35 peter 30 10.10.17.34 peter 120 10.10.17.34 peter 120 212.50.118.81 alex 46 212.50.118.81 alex 4	alex: 62 [10.10.17.33, 212.50.118.81] peter: 303 [10.10.17.34, 10.10.17.35, 192.168.0.11]
2 84.238.140.178 nakov 25 84.238.140.178 nakov 35	nakov: 60 [84.238.140.178]

9. Legendary Farming

You've beaten all the content and the last thing left to accomplish is own a legendary item. However, it's a tedious process and requires quite a bit of farming. Anyway, you are not too pretentious – any legendary will do. The possible items are:

- **Shadowmourne** – requires **250 Shards**;
- **Valanyr** – requires **250 Fragments**;
- **Dragonwrath** – requires **250 Motes**;

Shards, Fragments and **Motes** are the **key materials**, all else is **junk.** You will be given lines of input, such as **2 motes 3 ores 15 stones.** Keep track of the **key materials -** the **first** that reaches the **250 mark wins** the **race**. At that point, print the corresponding legendary obtained. Then, print the **remaining** shards, fragments, motes, ordered by **quantity** in **descending** order, then by **name** in **ascending** order, each on a new line. Finally, print the collected **junk** items, in **alphabetical** order.

Input

- Each line of input is in format **{quantity} {material} {quantity} {material} … {quantity} {material}**

Output

- On the first line, print the obtained item in format **{Legendary item} obtained!**
- On the next three lines, print the remaining key materials in descending order by quantity
  - If two key materials have the same quantity, print them in alphabetical order
- On the final several lines, print the junk items in alphabetical order
  - All materials are printed in format **{material}: {quantity}**
  - All output should be **lowercase**, except the first letter of the legendary

Constraints

- The quantity-material pairs are between 1 and 25 per line.
- The number of lines is in range [1..10]
- All materials are case-insensitive.
- Allowed working time: 0.25s
- Allowed memory: 16 MB

Examples

Input	Output
3 Motes 5 stones 5 Shards 6 leathers 255 fragments 7 Shards	Valanyr obtained! fragments: 5 shards: 5 motes: 3 leathers: 6 stones: 5

Input	Output
123 silver 6 shards 8 shards 5 motes 9 fangs 75 motes 103 MOTES 8 Shards 86 Motes 7 stones 19 silver	Dragonwrath obtained! shards: 22 motes: 19 fragments: 0 fangs: 9 silver: 123

10. *Сръбско Unleashed

Admit it – the СРЪБСКО is your favorite sort of music. You never miss a concert and you have become quite the geek concerning everything involved with СРЪБСКО. You can't decide between all the singers you know who your favorite one is. One way to find out is to keep statistics of how much money their concerts make. Write a program that does all the boring calculations for you.

On each input line you'll be given data in format: **"`singer @venue ticketsPrice ticketsCount`"**. There will be **no redundant whitespaces anywhere** in the input. Aggregate the data **by venue and by singer**. For each venue, print the singer and the total amount of money his/her concerts have made on a separate line. **Venues** should be kept in the **same order** they were entered; the **singers** should be **sorted by how much money** they have made in **descending order**. If two singers have made the same amount of money, keep them **in the order** in which **they were entered.**

Keep in mind that if a line is in incorrect format, it should be skipped and its data should not be added to the output. Each of the four tokens must be separated by **a space**, everything else is invalid. The venue should be denoted with **@** in front of it, such as @Sunny Beach

SKIP THOSE: Ceca@Belgrade125 12378, Ceca @Belgrade12312 123

The singer and town name may consist of one to three words.

Input

- The input data should be read from the console.
- It consists of a variable number of lines and ends when the command "**End**" 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 venue and singer in the format shown below.
- Format for singer lines is **#{2*space}{singer}{space}->{space}{total money}**

Constraints

- The **number of input lines** will be in the range [2 … 50].
- The **ticket price** will be an integer in the range [0 … 200].
- The **ticket count** will be an integer in the range [0 … 100 000]
- **Singers** and **venues** are case sensitive
- Allowed working time for your program: 0.1 seconds. Allowed memory: 16 MB.

Examples

Input	Output
Lepa Brena @Sunny Beach 25 3500 Dragana @Sunny Beach 23 3500 Ceca @Sunny Beach 28 3500 Mile Kitic @Sunny Beach 21 3500 Ceca @Sunny Beach 35 3500 Ceca @Sunny Beach 70 15000 Saban Saolic @Sunny Beach 120 35000 End	Sunny Beach # Saban Saolic -> 4200000 # Ceca -> 1270500 # Lepa Brena -> 87500 # Dragana -> 80500 # Mile Kitic -> 73500

Input	Output
Lepa Brena @Sunny Beach 25 3500 Dragana@Belgrade23 3500 Ceca @Sunny Beach 28 3500 Mile Kitic @Sunny Beach 21 3500	Sunny Beach # Saban Saolic -> 4200000 # Ceca -> 1148000 # Lepa Brena -> 87500

Ceca @Belgrade 35 3500 Ceca @Sunny Beach 70 15000 Saban Saolic @Sunny Beach 120 35000 End	# Mile Kitic -> 73500 Belgrade # Ceca -> 122500

11. *** 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

Input	Output
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