Exercises: Sets and Maps Advanced

This document defines the exercises for the ["Java Advanced" course @ Software University](). Please submit your solutions (source code) to all below-described problems in [Judge]().

Problem 1. Unique Usernames

Write a simple program that reads a sequence of usernames from the console and keeps a collection with only the unique ones. Print the collection on the console in order of insertion:

Examples

Input	Output

`6`
`Hello`
`Hello`
`Hello`
`World`
`Hello`
`Greetings`
	`Hello`
	`World`
	`Greetings`

`10`
`Peter`
`Maria`
`Peter`
`George`
`Stephen`
`Maria`
`Alexander`
`Peter`
`Stephen`
`George`
	`Peter`
	`Maria`
	`George`
	`Stephen`
	`Alexander`

Problem 2. Sets of Elements

On the first line, you are given the length of two sets, **N** and **M**. On the next **N + M** lines, there are **N** numbers that are in the **first** set and **M** numbers that are in the **second** one. Find all non-repeating element that appears in both of them, and print them in the same order at the console:

Set with length N = 4: {1, **3**, **5**, 7}

Set with length M = 3: {**3**, 4, **5**}

Set that contains all repeating elements -> {**3**, **5**}

Examples

Input	Output

`4 3`
`1`
`3`
`5`
`7`
`3`
`4`
`5`
	`3 5`

`2 2`
`1`
`3`
`1`
`5`
	`1`

Problem 3. Periodic Table

You are given an **n** number of chemical compounds. You need to keep track of all chemical elements used in the compounds and at the end, print all unique ones in ascending order:

Examples

Input	Output

`4`
`Ce O`
`Mo O Ce`
`Ee`
`Mo`
	`Ce Ee Mo O`

`3`
`Ge Ch O Ne`
`Nb Mo Tc`
`O Ne`
	`Ch Ge Mo Nb Ne O Tc`

Problem 4. Count Symbols

Write a program that reads some text from the console and counts the occurrences of each character in it. Print the results in **alphabetical** (lexicographical) order.

Examples

Input	Output

`SoftUni rocks`	` : 1 time/s`
	`S: 1 time/s`
	`U: 1 time/s`
	`c: 1 time/s`
	`f: 1 time/s`

	`i: 1 time/s`
	`k: 1 time/s`
	`n: 1 time/s`
	`o: 2 time/s`
	`r: 1 time/s`
	`s: 1 time/s`
	`t: 1 time/s`

`Did you know Math.Round rounds to the nearest even integer?`	`  : 9 time/s`
	`.: 1 time/s`
	`?: 1 time/s`
	`D: 1 time/s`
	`M: 1 time/s`
	`R: 1 time/s`
	`a: 2 time/s`
	`d: 3 time/s`
	`e: 7 time/s`
	`g: 1 time/s`
	`h: 2 time/s`
	`i: 2 time/s`
	`k: 1 time/s`
	`n: 6 time/s`
	`o: 5 time/s`
	`r: 3 time/s`
	`s: 2 time/s`
	`t: 5 time/s`
	`u: 3 time/s`
	`v: 1 time/s`
	`w: 1 time/s`
	`y: 1 time/s`

Problem 5. Phonebook

Write a program that receives some info from the console about **people** and their **phone numbers**.

You are free to choose how the data is entered. Each **entry** should have just **one name** and **one number** (both of them strings). If you receive a name that **already exists** in the phonebook, simply update its number.

After filling this simple phonebook, upon receiving the command "**`search`**", your program should be able to perform a search of contact by name and print her details in the format "**`{name} -> {number}`**". In case the contact isn't found, print "**`Contact {name} does not exist.`**".

Examples

Input	Output

`John-0888080808`
`search`
	`Contact Maria does not exist.`
	`John -> 0888080808`

Maria John stop	
John-00359888001122 Peter-0040333111000 George-0049112233 Samuel-0047123123123 **search** Samuel samuel PeTeR Peter stop	Samuel -> 0047123123123 Contact samuel does not exist. Contact PeTeR does not exist. Peter -> 0040333111000

Problem 6. Fix Emails

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

"**`{name} – > {email}`**"

Examples

Input	Output
John johnDoe@softuni.org Peter Smith smith.peter@softuni.org Taylor Baker baker@gmail.com stop	John -> johnDoe@softuni.org Peter Smith -> smith.peter@softuni.org
Peter Adamas peter_adams@gmail.com Anna Foster foster.anna@yahoo.com Duke Jenkins jenkins.duke@softuni.org stop	Duke Jenkins -> jenkins.duke@softuni.org

Problem 7. Hands Of Cards

You are given a sequence of people and what cards he draws from the deck for every person. 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 he draws such a card, he **discards** 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** is **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

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

`Jonathan Davis: JD, JD, JD, JD`
`JOKER`
	`Jonathan Davis: 22`

Problem 8. *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 that 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 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 a **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:

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**
  - **Example: IPv4 -** 123.45.67.89 , **IPv6 -** 2001:db8::ff00:42:8329
- The messages will be in the format: **This&is&a&message.**
- The username will be a string with a length in the range **[3..50].**
- Time limit: **300ms/16MB.**

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`
`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`
	`child0:`
	`192.23.33.40 => 1.`
	`child1:`
	`192.23.30.40 => 1.`
	`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.`

Problem 9. *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 the 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 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]**.
- Time limit: **100ms/16MB**.

Examples

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

Problem 10. *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 the format "`{user}: {duration} [IP₁, IP₂, ...]`". 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} [{IP₁}, {IP₂}, ...]`". 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 a length of **[1…20]**.
- The `{duration}` is an integer number in the range **[1…1000]**.
- Time limit: **300ms/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]

Problem 11. *Legendary Farming

You've beaten all the content, and the last thing left to accomplish is to 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, and motes, ordered by **quantity** in **descending** 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 the 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 the range **[1..10].**
- All materials are case-insensitive.
- Allowed time: **250ms/16MB.**

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
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

Problem 12. ***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 a statistic 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 the 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 below format.
- Format for singer lines is "`# {singer} -> {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 time: **100ms/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`

`Lepa Brena @Sunny Beach 25 3500`
`Dragana@Belgrade23 3500`
`Ceca @Sunny Beach 28 3500`
`Mile Kitic @Sunny Beach 21 3500`
`Ceca @Belgrade 35 3500`
`Ceca @Sunny Beach 70 15000`
`Saban Saolic @Sunny Beach 120 35000`
`End`
	`Sunny Beach`
	`#  Saban Saolic -> 4200000`
	`#  Ceca -> 1148000`
	`#  Lepa Brena -> 87500`
	`#  Mile Kitic -> 73500`
	`Belgrade`
	`#  Ceca -> 122500`

Problem 13. ***Dragon Army

Heroes III is the best game ever. Everyone loves it, and everyone plays it all the time. John is not an exclusion from 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 input order, 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 stats.

There **may** be **missing** stats in the input, though. If a stat is missing, you should assign its 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 a null value. See the examples below for a 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 the 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 a 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 the range **[1...100].**
- The dragon type and name are one word only, starting with a capital letter.
- Damage health and armor are integers in the 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