Exercise: Dictionaries

Please, submit your source code solutions for the described problems to the Judge System.

Ask your questions here: https://www.slido.com by entering the course code **#fund-python**

1. Count Chars in a String

Write a program that **counts all characters** in a string **except for space** (" ").

Print all the occurrences in the following format:

"{char} -> {occurrences}"

Examples

Input	Output
text	t -> 2 e -> 1 x -> 1
text text text	t -> 6 e -> 3 x -> 3

2. A Miner Task

You will be given a **sequence of strings**, each **on a new line** until you receive the **"stop"** command. Every **odd line** on the console represents 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 the following format:

"{resource} -> {quantity}"

The quantities 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

3. Capitals

Using **dictionary comprehension**, write a program that receives **country names** on the **first line,** separated by **comma and space `", "`**, and their corresponding **capital cities** on the second line (again separated by **comma and space `", "`**). **Print each country** with its **capital** on a **separate line** in the following format: **`"{country} -> {capital}"`**.

Hints

- You could use the **`zip()`** method.

Examples

Input	Output

`Bulgaria, Romania, Germany, England`
`Sofia, Bucharest, Berlin, London`
	`Bulgaria -> Sofia`
	`Romania -> Bucharest`
	`Germany -> Berlin`
	`England -> London`

`Bulgaria, Germany, France`
`Varna, Frankfurt, Paris`
	`Bulgaria -> Varna`
	`Germany -> Frankfurt`
	`France -> Paris`

4. Phonebook

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

Each **entry** should have **a name** and **a number** (both strings) separated by a **`"-"`**. If you receive a name that **already exists** in the phonebook, **update its number**.

After filling out the phonebook, you will receive a number – **N**. Your program should be able to perform a **search of contact by name** and print its details in the format: **`"{name} -> {number}"`**. In case the contact isn't found, print: **`"Contact {name} does not exist."`**

Examples

Input	Output

`Adam-0888080808`
`2`
`Mery`
`Adam`
	`Contact Mery does not exist.`
	`Adam -> 0888080808`

`Adam-+359888001122`
`Ralf-666`
`George-5559393`
`Silvester-02/987665544`
`4`
`Silvester`
`silvester`
`Rolf`
`Ralf`
	`Silvester -> 02/987665544`
	`Contact silvester does not exist.`
	`Contact Rolf does not exist.`
	`Ralf -> 666`

5. Legendary Farming

You are playing a game, and your goal is to **win a legendary item** - any legendary item will be good enough. However, it's a tedious process, and it requires quite a bit of farming. The possible **items** are:

- **`"Shadowmourne"`** - requires **250 Shards**
- **`"Valanyr"`** - requires **250 Fragments**
- **`"Dragonwrath"`** - requires **250 Motes**

`"Shards"`, **`"Fragments"`**, and **`"Motes"`** are the **key materials** (**case-insensitive**), and everything else is **junk.**

You will be given lines of input in the format:

`"{quantity1} {material1} {quantity2} {material2} … {quantityN} {materialN}"`

Keep track of the **key materials - the first** one that reaches **250, wins** the **race**. At that point, you have to print that the corresponding legendary item is obtained.

In the end, print the **remaining shards, fragments, and motes** in the format:

`"shards: {number_of_shards}`

`fragments: {number_of_fragments}`

`motes: {number_of_motes}"`

Finally, **print** the collected **junk** items in the order of appearance.

Input

- Each line comes in the following format: **`"{quantity1} {material1} {quantity2} {material2} … {quantityN} {materialN}"`**

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
- On the **several final lines**, print the **junk** items
- All materials should be printed in the format: **`"{material}: {quantity}"`**
- The output should be **lowercase**, except for the first letter of the legendary

Examples

Input	**Output**

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

`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`
	`fragments: 0`
	`motes: 19`
	`silver: 123`
	`fangs: 9`

6. Orders

Write a program that keeps the information about **products** and their **prices**. Each product has a **name**, a **price,** and a **quantity**:

- If the product **doesn't exist** yet, **add** it with its **starting quantity**.
- If you receive a product, that **already exists, increase** its quantity by the input quantity and if its **price** is different, **replace** the price as well.

You will receive products' **names**, **prices,** and **quantities** on **new lines**. Until you receive the command **"buy"**, keep adding items. Finally, print all items with their **names** and the **total price** of each product.

Input

- Until you receive **"buy"**, the products will be coming in the format: **"{name} {price} {quantity}"**.
- The product data is **always** delimited by a **single space**.

Output

- Print information about **each product** in the following format:
  "{product_name} -> {total_price}"
- **Format** the total price to the **2nd digit after the decimal separator**.

Examples

Input	Output
Beer 2.20 100 IceTea 1.50 50 NukaCola 3.30 80 Water 1.00 500 buy	Beer -> 220.00 IceTea -> 75.00 NukaCola -> 264.00 Water -> 500.00
Beer 2.40 350 Water 1.25 200 IceTea 5.20 100 Beer 1.20 200 IceTea 0.50 120 buy	Beer -> 660.00 Water -> 250.00 IceTea -> 110.00
CesarSalad 10.20 25 SuperEnergy 0.80 400 Beer 1.35 350 IceCream 1.50 25 buy	CesarSalad -> 255.00 SuperEnergy -> 320.00 Beer -> 472.50 IceCream -> 37.50

7. SoftUni Parking

SoftUni just got a new fancy **parking lot**. It even has online **parking validation**, except the online service doesn't work. It can only receive users' data, but it doesn't know what to do with it. Good thing you're on the dev team and know how to fix it, right?

Write a program, which validates a parking place - users can **register** to enter the park and **unregister** to leave.

The program **receives 2 types of commands**:

- **"register {username} {license_plate_number}"**:

- The system only supports **one car per user** at the moment, so if a user tries to register **another license plate** using the **same username**, the system should print:
  `"ERROR: already registered with plate number {license_plate_number}"`
- If the check above **passes successfully**, the user should be registered, so the system should print:
  `"{username} registered {license_plate_number} successfully"`

- **`"unregister {username}"`**:
  - If the user is **not present** in the database, the system should print:
    "**`ERROR: user {username} not found`**"
  - If the check above **passes successfully**, the system should print:
    "**`{username} unregistered successfully`**"

After you execute all of the commands, **print** all the currently **registered users** and their **license plates** in the format:

- **`"{username} => {license_plate_number}"`**

Input

- First line: **n** - **number of commands** - **integer**
- Next **n** lines: **commands** in one of the **two** possible formats:
  - Register: "**`register {username} {license_plate_number}`**"
  - Unregister: "**`unregister {username}`**"

The input will **always** be **valid,** and you **do not need** to check it explicitly.

Examples

Input	Output

`5`
`register John CS1234JS`
`register George JAVA123S`
`register Andy AB4142CD`
`register Jesica VR1223EE`
`unregister Andy`
	`John registered CS1234JS successfully`
	`George registered JAVA123S successfully`
	`Andy registered AB4142CD successfully`
	`Jesica registered VR1223EE successfully`
	`Andy unregistered successfully`
	`John => CS1234JS`
	`George => JAVA123S`
	`Jesica => VR1223EE`

`4`
`register Jony AA4132BB`
`register Jony AA4132BB`
`register Linda AA9999BB`
`unregister Jony`
	`Jony registered AA4132BB successfully`
	`ERROR: already registered with plate number AA4132BB`
	`Linda registered AA9999BB successfully`
	`Jony unregistered successfully`
	`Linda => AA9999BB`

`6`
`register Jacob MM1111XX`
`register Anthony AB1111XX`
`unregister Jacob`
`register Joshua DD1111XX`
`unregister Lily`
`register Samantha AA9999BB`
	`Jacob registered MM1111XX successfully`
	`Anthony registered AB1111XX successfully`
	`Jacob unregistered successfully`
	`Joshua registered DD1111XX successfully`
	`ERROR: user Lily not found`
	`Samantha registered AA9999BB successfully`
	`Anthony => AB1111XX`
	`Joshua => DD1111XX`
	`Samantha => AA9999BB`

8. Courses

Write a program that keeps the information about **courses**. Each course has a **name** and **registered students**.

You will be receiving a **course name** and a **student name** until you receive the command **"end"**.

You should register each user into the corresponding course. If the given course does not exist, add it.

When you receive the command **"end"**, print the courses with their **names** and **total registered users**. For each course, print the registered users.

Input

- Until the **"end"** command is received, you will be receiving input lines in the format:
  `"{course_name} : {student_name}"`
- The product data is **always** delimited by **`" : "`**

Output

- Print the information about **each course** in the following format:
  `"{course_name}: {registered_students}"`
- Print the information about **each student** in the following format:
  `"-- {student_name}"`

Examples

Input	Output

`Programming Fundamentals : John Smith`
`Programming Fundamentals : Linda Johnson`
`JS Core : Will Wilson`
`Java Advanced : Harrison White`
`end`
	`Programming Fundamentals: 2`
	`-- John Smith`
	`-- Linda Johnson`
	`JS Core: 1`
	`-- Will Wilson`
	`Java Advanced: 1`
	`-- Harrison White`

`Algorithms : Jay Moore`
`Programming Basics : Martin Taylor`
`Python Fundamentals : John Anderson`
`Python Fundamentals : Andrew Robinson`
`Algorithms : Bob Jackson`
`Python Fundamentals : Clark Lewis`
`end`
	`Algorithms: 2`
	`-- Jay Moore`
	`-- Bob Jackson`
	`Programming Basics: 1`
	`-- Martin Taylor`
	`Python Fundamentals: 3`
	`-- John Anderson`
	`-- Andrew Robinson`
	`-- Clark Lewis`

9. Student Academy

Write a program that keeps the information about **students** and **their grades**.

On the first line, you will receive an integer number representing **the next pair of rows**. On the next lines, you will be receiving each **student's name** and their **grade**.

Keep track of all grades for each student and keep only the students with an **average grade higher than or equal to 4.50**.

Print the final dictionary with students and their average grade in the following format:

`"{name} -> {averageGrade}"`

Format the average grade to the **2nd decimal place**.

Examples

Input	Output
5 John 5.5 John 4.5 Alice 6 Alice 3 George 5	John -> 5.00 Alice -> 4.50 George -> 5.00

Input	Output
5 Amanda 3.5 Amanda 4 Rob 5.5 Christian 5 Robert 6	Rob -> 5.50 Christian -> 5.00 Robert -> 6.00

10. Company Users

Write a program that keeps the information about companies and their employees.

You will be receiving **company names** and an **employees' id** until you receive the command **"End"** command. **Add each employee** to the given company. Keep in mind that a **company cannot have two employees with the same id**.

Print the **company name** and **each employee's id** in the following format:

```
"{company_name}
-- {id1}
-- {id2}
…
-- {idN}"
```

Input / Constraints

- Until you receive the **"End"** command, you will be receiving input in the format: **"{company_name} -> {employee_id}"**.
- The input always will be valid.

Examples

Input	Output
SoftUni -> AA12345 SoftUni -> BB12345 Microsoft -> CC12345 HP -> BB12345 End	SoftUni -- AA12345 -- BB12345 Microsoft -- CC12345 HP -- BB12345

SoftUni -> AA12345 SoftUni -> CC12344 Lenovo -> XX23456 SoftUni -> AA12345 Movement -> DD11111 End	SoftUni -- AA12345 -- CC12344 Lenovo -- XX23456 Movement -- DD11111

11. *Force Book

The force users struggle to remember which side is the different force users from because they switch them too often. So you are tasked to create a web application to manage their profiles. You should store information for every **unique force user** registered in the application.

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

`"{force_side} | {force_user}"`

`"{force_user} -> {force_side}"`

The **`"force_user"`** and **`"force_side"`** are strings, containing any character.

If you receive **`"force_side | force_user"`**:

- If there is **no such force user** and **no such force side -> create a new force side** and **add** the **force user** to the corresponding side.
- Only **if there is no such force user** in any **force side** -> **add** the **force user** to the corresponding side.
- If there is such **force user** already -> **skip** the command and continue to the next operation.

If you receive a **`"force_user -> force_side"`**:

- If there is such **force user** already -> **change their side**.
- If there is no such **force user** in any **force side** -> add the **force user** to the corresponding **force side**.
- If there is no such **force user** and no such **force side -> create new force side** and **add** the **force user** to the corresponding side.
- **Then you should print on the console: `"{force_user} joins the {force_side} side!"`**.

You should **end your program** when you receive the command **`"Lumpawaroo"`**. At that point, you should print each force side. For each side, print the **force users**.

In case there are **no forced users on a side**, you **shouldn't print** the side information.

Input / Constraints

- The input comes in the form of commands in one of the formats specified above.
- The input ends when you receive the command **`"Lumpawaroo"`.**

Output

- As output for each force side, you must print all the force users.
- The output format is:

```
"Side: {force_side}, Members: {force_users_count}
! {force_user1}
```

! {force_user2}

…

! {force_userN}"

- In case there are **NO force users on a side**, don't print this side.

Examples

Input	Output	Comments

`Light | Peter`
`Dark | Kim`
`Light | Kim`
`Lumpawaroo`
	`Side: Light, Members: 1`
	`! Peter`
	`Side: Dark, Members: 1`
	`! Kim`
		We register Peter on the Light side and Kim on the Dark side. After receiving "Lumpawaroo", we print both sides.

`Lighter | Royal`
`Darker | DCay`
`Ivan Ivanov -> Lighter`
`DCay -> Lighter`
`Lumpawaroo`
	`Ivan Ivanov joins the Lighter side!`
	`DCay joins the Lighter side!`
	`Side: Lighter, Members: 3`
	`! Royal`
	`! Ivan Ivanov`
	`! DCay`
		Although Ivan Ivanov doesn't have a profile, we **register** him and add him to the Lighter side.
		We **remove DCay** from the Darker side and add him to the Lighter side.
		We print only the Lighter side because the Darker side **has no members.**

12. *SoftUni Exam Results

Judge statistics on the last Programming Fundamentals exam were not working correctly, so you have the task of taking all the submissions and analyzing them properly. You should collect all the submissions and print the final results and statistics about each language in which the participants submitted their solutions.

You will be receiving lines in the following format: **"{username}-{language}-{points}" until you receive "exam finished"**. You should store each username and their submissions and points. If a student has **two or more** submissions for the **same language**, save only his **maximum points**.

You can receive a **command to ban** a user for cheating in the following format: **"{username}-banned"**. In that case, you should **remove** the user from the contest but **preserve his submissions in the total count of submissions for each language**.

After receiving the **"exam finished"**, print each of the participants in the following format:

"Results:

{username1} | {points}

{username2} | {points}

…

{usernameN} | {points}"

After that, **print each language** used in the exam in the following format:

```
"Submissions:
{language1} - {submissions_count}
{language2} - {submissions_count}
…
{language3} - {submissions_count}"
```

Input / Constraints

Until you receive **"exam finished"** you will be receiving participant submissions in the following format: **"{username}-{language}-{points}"**

You can receive a ban command -> **"{username}-banned"**

The points of the participant will always be a **valid integer in the range [0-100];**

Output

- Print the exam results for each participant
- After that, print each language in the format shown above
- Allowed working **time** / **memory**: **100ms** / **16MB**

Examples

Input	Output
Peter-Java-84 George-C#-84 George-C#-70 Katy-C#-94 exam finished	Results: Peter | 84 George | 84 Katy | 94 Submissions: Java - 1 C# - 3
Peter-Java-91 George-C#-84 Katy-Java-90 Katy-C#-50 Katy-banned exam finished	Results: Peter | 91 George | 84 Submissions: Java - 2 C# - 2