Exercise: Associative Arrays, Lambda and Stream API

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

You can check your solutions in Judge.

1. Count Chars in a String

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

Print all 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

Until you receive the **"stop"** command, you will be given a sequence of strings, each on a new line. Every **odd** line on the console represents a **resource** (e.g., Gold, Silver, Copper, etc.) and every **even** - **quantity**. Your task is to collect the resources and print them on a new line.

Print the resources and their quantities in the 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

3. Orders

Write a program which keeps the information about **products** and their **prices**. Each product has a **name**, a **price,** and its **quantity**. If the product **doesn't exist** yet, **add** it with its **starting quantity**.

If you receive a product that **already exists, increases** 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. When you do receive the command "**buy**", print the items with their **names** and the **total price** of all the products with that name.

Input

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

Output

- Print information about **each product**, following the format: "**`{productName} -> {totalPrice}`**"
- **Format** the average total price to the **2nd decimal place.**

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

4. SoftUni Parking

*SoftUni just got a new **parking lot**. It's so fancy, it even has online **parking validation**. Except, the online service doesn't work. It can only receive users' data but 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 that validates parking for an online service. Users can **register** to park and **unregister** to leave.

The program **receives 2 commands**:

- "**`register {username} {licensePlateNumber}`**":
  - 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 {licensePlateNumber}`**"
  - If the aforementioned checks pass successfully, the plate can be registered, so the system should print:
    "**`{username} registered {licensePlateNumber} successfully`**"
- "**`unregister {username}`**":
  - If the user is **not present** in the database, the system should print:
    "**`ERROR: user {username} not found`**"
  - If the aforementioned check 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} => {licensePlateNumber}`**"

Input

- First line: **n** - **number of commands** – **integer.**
- Next **n** lines: **commands** in one of **two** possible formats:
  - Register: "**`register {username} {licensePlateNumber}`**"
  - 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	Jony registered AA4132BB successfully

register Jony AA4132BB register Jony AA4132BB register Linda AA9999BB unregister Jony	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

5. Courses

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

You will receive the **course name** and **student name** until you receive the command "**end**". **Check if such a course already exists and if not - add the course.** Register the user into the course. When you do receive the command "**end**", print the courses with their **names** and **total registered users**. For each contest, print the registered users.

Input

- Until you receive "**end**", the input come in the format: "**{courseName} : {studentName}**".
- The product data is **always** delimited by " **:** ".

Output

- Print information about **each course**, following the format:
  "{courseName}: {registeredStudents}"
- Print information about each student, following the format:
  "-- {studentName}"

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	Algorithms: 2 -- Jay Moore -- Bob Jackson Programming Basics: 1 -- Martin Taylor

Python Fundamentals : Clark Lewis end	Python Fundamentals: 3 -- John Anderson -- Andrew Robinson -- Clark Lewis

6. Student Academy

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

On the first line, you will receive number **n**. After that, you will receive **n pair of rows**. First, you will receive the **student's name**, after that, you will receive his **grade**. **Check if the student already exists and if not - add him**. Keep track of all grades for each student.

When you finish reading data, keep students with an **average grade higher or equal to 4.50**.

Print the students and their average grade in the 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

7. 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 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: {numberOfShards}

fragments: {numberOfFragments}

motes: {numberOfMotes}"

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

8. Company Users

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

You will receive company names and an employees' id until you receive the **"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 "**End**", the input come in the format: "**{companyName} -> {employeeId}**".
- 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

9. *ForceBook

The force users are struggling to remember which side is the different forceUsers from because they switch them too often. So you are tasked to create a web application to manage their profiles.

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** on 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** on 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 a 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 force 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: {forceSide}, Members: {forceUsers.Count}
! {forceUser}
! {forceUser}
! {forceUser}"
```

- In case there are **NO forceUsers**, don't print this side.

Examples

Input	Output	Comments

`Light | Peter`
`Dark | 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 **registered** him and added 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.**

10. *SoftUni Exam Results

Judge statistics on the last Programming Fundamentals exam were not working correctly, so you have the task to take all the submissions and analyze 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.

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

"**Results:**

{username} | {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 participant's points 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	Results: Peter | 91

Katy-Java-90 Katy-C#-50 Katy-banned exam finished	George | 84 Submissions: Java - 2 C# - 2