Exercise: Dictionaries, Lamdba and LINQ

Tasks for exercise in class and for homework to the course "Programming Advanced for QA" @ SoftUni
Test your tasks in the Judge system: https://judge.softuni.org/Contests/4473

1. Count Chars in a String

Create 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. 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 the following format:

`"{resource} –> {quantity}"`

The quantities will be **in the range [1… 2 000000000].**

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

Create 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, which **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. 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 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:
  `"{productName} -> {totalPrice}"`
- **Format** the average grade 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

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 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 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 passes 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 of 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 the **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 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 Joshua => DD1111XX Anthony => AB1111XX Samantha => AA9999BB

5. Student Academy

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

You will receive **n pair of rows**. First, you will receive the **student's name**, **after that, you will receive their grade**. **Check if the student already exists and if not, add him.** Keep track of all grades for each student.

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

Print the 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

6. *Company Users

Create a program that keeps information about companies and their employees.

You will be receiving a **company name** and an **employee's 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.

When you finish reading the data, print the company's name and each employee's id in the following format:

`{companyName}`

`-- {id1}`

`-- {id2}`

`-- {idN}`

Input / Constraints

- Until you receive the "**End**" command, you will be receiving input 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