**Exercise: Associative Arrays**

Problems for exercise and homework for the ["C# Fundamentals" course @ SoftUni](https://softuni.bg/trainings/2438/csharp-fundamentals-september-2019)  
You can check your solutions in [Judge](https://judge.softuni.bg/Contests/1213)

* **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 |

* **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 … 2 000 000 000]**

**Examples**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| Gold  155  Silver  10  Copper  17  stop | Gold -> 155  Silver -> 10  Copper -> 17 |  | gold  155  silver  10  copper  17  gold  15  stop | gold -> 170  silver -> 10  copper -> 17 |

* **Legendary Farming**

You’ve done all the work and the last thing left to accomplish is to own a **legendary item**. However, it’s a tedious process and it requires quite a bit of farming. Anyway, you are not too pretentious - any legendary item 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** and everything else is **junk.** You will be given lines of input, in the format:

**2 motes 3 ores 15 stones**

Keep track of the **key materials** -the **first** one that reaches the **250 mark,** **wins** the **race**. At that point you have to print that the corresponding legendary item is 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 comes in the following 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}**
* 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!  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 |

* **Orders**

Write a program that keeps 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 **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 |

* **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 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 |