# Exercises: Sets and Dictionaries

You can check your solutions here: <https://judge.softuni.bg/Contests/3175/Sets-Dictionaries>.

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

## Sets of Elements

Write a program that prints a **set of elements**. On the first line you will receive two numbers - **n** and **m**, which represent the lengths of two separate sets. On the next **n** + **m** lines you will receive **n** numbers, which are the numbers in the **first** set, and **m** numbers, which are in the **second** set. Find all the **unique** **elements** that appear in **both of them** and **print** them in the order in which they appear in the **first** set - **n**.

**For example:**

* Set with length n = 4: {1, **3**, **5**, 7}
* Set with length m = 3: {**3**, 4, **5**}
* Set that contains all the **elements** that repeat in **both** **sets** -> {**3**, **5**}

### Examples

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Output** | **Input** | **Output** |
| 4 3  1  3  5  7  3  4  5 | 3 5 | 2 2  1  3  1  5 | 1 |

## Periodic Table

Write a program that keeps all the **unique** chemical **elements**. On the first line you will be given a number **n** - the **count** of input **lines** that you are going to receive. On the next **n** lines you will be receiving **chemical** **compounds**, separated by a **single** **space**. Your task is to print all the **unique ones** in **ascending** **order**:

### Examples

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Output** | **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 |

## Even Times

Write a program that **prints** a **number** from a collection, which appears an **even** **number** of **times** in it. On the first line, you will be given **n** – the **count** of **integers** you will receive. On the next n lines you will be receiving **the numbers**. It is **guaranteed** that **only** **one** of them **appears** an **even** **number** of times. Your task is to **find** that **number** and **print** it in the end.

### Examples

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Output** | **Input** | **Output** |
| 3  2  -1  2 | 2 | 5  1  2  3  1  5 | 1 |

## 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 |

## Courses

Write a program that keeps 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**". **Check if such course already exists, and if not, add the course.** Register the user into the course. When you receive the command "**end**", print the courses with their **names** and **total registered users**, ordered by the count of registered users in descending order. For each contest print the registered users **ordered by name** **in ascending order**.

### Input

* Until the "**end**" command is received, you will be receiving input in the format: "**{courseName} : {studentName}**".
* The product data is **always** delimited by **" : ".**

### Output

* Print the information about **each** **course** in the following the format:   
  **"{courseName}: {registeredStudents}"**
* Print the information about each student, in the 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  Python Fundamentals : Clark Lewis  end | Python Fundamentals: 3  -- Andrew Robinson  -- Clark Lewis  -- John Anderson  Algorithms: 2  -- Bob Jackson  -- Jay Moore  Programming Basics: 1  -- Martin Taylor |

## Company Users

Write 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 command "**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, **order the companies by the name in ascending order**.

Print the company 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** | **Input** | **Output** |
| SoftUni -> AA12345  SoftUni -> BB12345  Microsoft -> CC12345  HP -> BB12345  End | HP  -- BB12345  Microsoft  -- CC12345  SoftUni  -- AA12345  -- BB12345 | SoftUni -> AA12345  SoftUni -> CC12344  Lenovo -> XX23456  SoftUni -> AA12345  Movement -> DD11111  End | Lenovo  -- XX23456  Movement  -- DD11111  SoftUni  -- AA12345  -- CC12344 |

## \* ForceBook

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

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

{forceSide} | {forceUser}

{forceUser} -> {forceSide}

The forceUser and forceSide are strings, containing any character.

If you receive forceSide | forceUser, you should **check if such forceUser already exists**, and **if not**, **add** him/her to the corresponding side.

If you receive a forceUser -> forceSide, you should check if there is such a **forceUser** already and if so, **change his/her side**. If there is no such **forceUser**, add him/her to the corresponding forceSide, treating the command **as a new registered forceUser.**  
Then you should print on the console: "{forceUser} joins the {forceSide} side!"

You should end your program when you receive the command "Lumpawaroo". At that point you should print each force side, **ordered descending by forceUsers count, than ordered by name**. For each side print the **forceUsers**, **ordered by name**.

In case there are **no forceUsers in 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 forceSide, **ordered descending by forceUsers count**, **then by name**, you must print all the forceUsers, **ordered by name alphabetically**.
* 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 | Gosho  Dark | Pesho  Lumpawaroo | Side: Dark, Members: 1  ! Pesho  Side: Light, Members: 1  ! Gosho | We register Gosho in the Light side and Pesho in the Dark side. After receiving "Lumpawaroo" we print both sides, ordered by membersCount and then by name. |
| 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  ! DCay  ! Ivan Ivanov  ! Royal | Although Ivan Ivanov doesn`t have profile, we **register** him and add him to the Lighter side.  We **remove DCay** from Darker side and add him to Lighter side.  We print only Lighter side because Darker side **has no members.** |

## \* SoftUni Exam Results

Judge statistics on the last Programing Fundamentals exam was 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 that the participants submitted their solutions in.

You will be receiving lines in the following format: "{username}-{language}-{points}" **until you receive** "exam finished".You should store each username and his 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, ordered descending by their max points, then by username, in the following format:

Results:

{username} | {points}

…

After that print each language, used in the exam, ordered descending by total submission count and then by language name, in the following format:

Submissions:

{language} – {submissionsCount}

…

### 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, ordered descending by max points and then by username, in the following format:

Results:

{username} | {points}

…

* After that print each language, ordered descending by total submissions and then by language name, in the following format:

Submissions:

{language} – {submissionsCount}

…

* Allowed working **time** / **memory**: **100ms** / **16MB**.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comment** |
| Pesho-Java-84  Gosho-C#-84  Gosho-C#-70  Kiro-C#-94  exam finished | Results:  Kiro | 94  Gosho | 84  Pesho | 84  Submissions:  C# - 3  Java - 1 | We order the participant descending by max points and then by name, printing only the username and the max points.  After that we print each language along with the count of submissions, ordered descending by submissions count, and then by language name. |
| Pesho-Java-91  Gosho-C#-84  Kiro-Java-90  Kiro-C#-50  Kiro-banned  exam finished | Results:  Pesho | 91  Gosho | 84  Submissions:  C# - 2  Java - 2 | Kiro is banned so he is removed from the contest, but he`s submissions are still preserved in the languages submissions count.  So althou there are only 2 participants in the results, there are 4 submissions in total. |

## \* Wardrobe

Write a program that helps you decide what **clothes** to wear from your **wardrobe**. You will receive the **clothes**, which are currently in your wardrobe, sorted by their **color** in the following **format**:

"{color} -> {item1},{item2},{item3}…"

If you receive a certain color, which already **exists** in your wardrobe, just **add** the clothes to **its** **records**. You can also receive **repeating** **items** for a certain **color** and you have to keep their **count**.

In the end, you will receive a **color** and a piece of **clothing**, which you will **look for** in the wardrobe, separated by a space in the following format:

**"{color} {clothing}"**

Your task is to print all the **items** and their **count** for **each** **color** in the following format**:**

**"{color}** **clothes**:

\* **{item1}** - **{count}**

\* **{item2}** - **{count}**

\* **{item3}** - **{count}**

…

\* **{itemN}** - **{count}"**

If you find the **item** you are **looking for**, you need to print **"(found!)"** next to it:

"\* **{itemN}** - **{count} (found!)"**

### Input

* On the **first** **line**, you will receive n – the **number of lines** of clothes, which you will receive.
* On the next **n** lines, you will receive the **clothes** in the **format** **described** above.

### Output

* Print the **clothes** from your wardrobe in the **format** **described** above

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4  Blue -> dress,jeans,hat  Gold -> dress,t-shirt,boxers  White -> briefs,tanktop  Blue -> gloves  Blue dress | Blue clothes:  \* dress - 1 (found!)  \* jeans - 1  \* hat - 1  \* gloves - 1  Gold clothes:  \* dress - 1  \* t-shirt - 1  \* boxers - 1  White clothes:  \* briefs - 1  \* tanktop - 1 |
| 4  Red -> hat  Red -> dress,t-shirt,boxers  White -> briefs,tanktop  Blue -> gloves  White tanktop | Red clothes:  \* hat - 1  \* dress - 1  \* t-shirt - 1  \* boxers - 1  White clothes:  \* briefs - 1  \* tanktop - 1 (found!)  Blue clothes:  \* gloves - 1 |
| 5  Blue -> shoes  Blue -> shoes,shoes,shoes  Blue -> shoes,shoes  Blue -> shoes  Blue -> shoes,shoes  Red tanktop | Blue clothes:  \* shoes - 9 |

## \* The V-Logger

Create a program that keeps information about **vloggers** and their **followers**. The **input** will come as a sequence of strings, where each string will represent a **valid** command. The commands will be presented in the following format:

* "{vloggername}" **joined The V-Logger** – keep the vlogger in your records.
  + Vloggernames **consist** **of only one word**.
  + If the **given** vloggernamealready **exists, ignore** thatcommand**.**
* "{vloggername} followed {vloggername}" – The first vlogger followed the second vlogger.
  + If **any** of the **given vlogernames** **does not exist** in you collection, **ignore** thatcommand**.**
  + Vlogger **cannot** follow **himself**
  + Vlogger **cannot** follow someone he is **already a follower** **of**
* **"Statistics" -** Upon receiving this command, you have to print a statistic about the vloggers.

Each vlogger has an unique **vloggername**. **Vloggers** can **follow other vloggers** and a vlogger **can follow** **as many other vloggers** **as he wants**, but he **cannot** follow **himself** or follow someone he is **already a follower** **of**. You need to print the **total** **count** of **vloggers** in your collection. Then you have to print the **most** **famous** **vlogger** – the one with the most followers, with **his** **followers.** If more than one vloggers have the **same** **number** of **followers**, print theone **following less** people and his **followers** should be printed in **lexicographical order (**in case the vlogger has **no followers**, print just the first line, which is described **below**). Lastly, print the **rest** **vloggers**, ordered by the **count** of followers in **descending** order, then by the number of vloggers he follows in **ascending** **order.** The **whole output must be** in the following format:

**"****The V-Logger has a total of {registered vloggers} vloggers in its logs.**

**1.** {mostFamousVlogger} : {followers} followers, {followings} following

**\*** {follower1}

\* {follower2} …

**{No}.** {vlogger} : {followers} followers, {followings} following

**{No}.** {vlogger} : {followers} followers, {followings} following…"

### Input

* The input will come in the format described above.

### Output

* On the first line, print **the total count of vloggers** in the format described above.
* On the second line, print the **most** **famous** vlogger in the format described above.
* On the **next** lines, print all of the **rest** vloggers in the format described above.

### Constraints

* There will be **no** **invalid** input.
* There will be no situation where **two** **vloggers** have **equal** count of **followers** and **equal** count of **followings**
* Allowed time/memory: **100ms/16MB**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| EmilConrad joined The V-Logger  VenomTheDoctor joined The V-Logger  Saffrona joined The V-Logger  Saffrona followed EmilConrad  Saffrona followed VenomTheDoctor  EmilConrad followed VenomTheDoctor  VenomTheDoctor followed VenomTheDoctor  Saffrona followed EmilConrad  Statistics | The V-Logger has a total of 3 vloggers in its logs.  1. VenomTheDoctor : 2 followers, 0 following  \* EmilConrad  \* Saffrona  2. EmilConrad : 1 followers, 1 following  3. Saffrona : 0 followers, 2 following |
| JennaMarbles joined The V-Logger  JennaMarbles followed Zoella  AmazingPhil joined The V-Logger  JennaMarbles followed AmazingPhil  Zoella joined The V-Logger  JennaMarbles followed Zoella  Zoella followed AmazingPhil  Christy followed Zoella  Zoella followed Christy  JacksGap joined The V-Logger  JacksGap followed JennaMarbles  PewDiePie joined The V-Logger  Zoella joined The V-Logger  Statistics | The V-Logger has a total of 5 vloggers in its logs.  1. AmazingPhil : 2 followers, 0 following  \* JennaMarbles  \* Zoella  2. Zoella : 1 followers, 1 following  3. JennaMarbles : 1 followers, 2 following  4. PewDiePie : 0 followers, 0 following  5. JacksGap : 0 followers, 1 following |