

LAB: SETS AND DICTIONARIES ADVANCED

Lab: Sets and Dictionaries Advanced.....	1
Dictionaries.....	1
1. Count Same Values in Array	1
2. Average Student Grades.....	1
3. Largest 3 Numbers	2
4. Product Shop.....	2
5. Cities by Continent and Country	3
Sets	3
6. Record Unique Names.....	3
7. Parking Lot.....	4
8. SoftUni Party	5
Exercises: Sets and Dictionaries Advanced.....	6
1. Unique Usernames	6
2. Sets of Elements	6
3. Periodic Table	7
4. Even Times	7
5. Count Symbols.....	7
6. Wardrobe	8
7. *The V-Logger.....	9
8. *Ranking.....	10
9. *SoftUni Exam Results.....	12
10. *ForceBook	13

You can check your solutions in [Judge](#)

DICTIONARIES

1. COUNT SAME VALUES IN ARRAY

Create a program that counts in a given array of double values the number of occurrences of each value.

EXAMPLES

Input	Output
-2.5 4 3 -2.5 -5.5 4 3 3 -2.5 3	-2.5 - 3 times 4 - 2 times 3 - 4 times -5.5 - 1 times
2 4 4 5 5 2 3 3 4 4 3 3 4 3 5 3 2 5 4 3	2 - 3 times 4 - 6 times 5 - 4 times 3 - 7 times

2. AVERAGE STUDENT GRADES

Create a program, which reads a **name** of a student and his/her **grades** and **adds** them to the **student record**, then **prints the students' names with their grades and their average grade**.

EXAMPLES

Input	Output
7 John 5.20 Maria 5.50 John 3.20 Maria 2.50 Sam 2.00 Maria 3.46 Sam 3.00	John -> 5.20 3.20 (avg: 4.20) Maria -> 5.50 2.50 3.46 (avg: 3.82) Sam -> 2.00 3.00 (avg: 2.50)
4 Vlad 4.50 Peter 3.00 Vlad 5.00 Peter 3.66	Vlad -> 4.50 5.00 (avg: 4.75) Peter -> 3.00 3.66 (avg: 3.33)
5 George 6.00 George 5.50 George 6.00 John 4.40 Peter 3.30	George -> 6.00 5.50 6.00 (avg: 5.83) John -> 4.40 (avg: 4.40) Peter -> 3.30 (avg: 3.30)

HINTS

- Use a **dictionary (string → List<decimal>)**
- Check if the name **exists** before adding the grade. If it doesn't, add it to the dictionary.
- Pass through all **key-value pairs** in the dictionary and print the results. You can use the **.Average()** method to quickly calculate the average value from a list.

3. LARGEST 3 NUMBERS

Read a **list of integers** and **print the largest 3 of them**. If there are less than 3, print all of them.

EXAMPLES

Input	Output	Input	Output
10 30 15 20 50 5	50 30 20	20 30	30 20

HINTS

- Read an array of integers.
- **Order the array using a LINQ query.**

```
int[] sorted = numbers.OrderByDescending(n => n)
    .ToArray();
```

- **Print top 3 numbers with a for loop.**

4. PRODUCT SHOP

Create a program that prints information about **food shops** in Sofia and the **products** they **store**. Until the "**Revision**" command is received, you will be receiving input in the format: "**{shop}**, **{product}**, **{price}**". Keep in mind that if you receive a **shop** you already **have received**, you must **collect** its **product information**.

Your output must be **ordered** by shop **name** and must be in the format:

"{shop}->
Product: {product}, Price: {price}"
Note: The price **should not** be rounded or formatted.

EXAMPLES

Input	Output
lidl, juice, 2.30 fantastico, apple, 1.20 kaufland, banana, 1.10 fantastico, grape, 2.20 Revision	fantastico-> Product: apple, Price: 1.2 Product: grape, Price: 2.2 kaufland-> Product: banana, Price: 1.1 lidl-> Product: juice, Price: 2.3
tmarket, peanuts, 2.20 GoGrill, meatballs, 3.30 GoGrill, HotDog, 1.40 tmarket, sweets, 2.20 Revision	GoGrill-> Product: meatballs, Price: 3.3 Product: HotDog, Price: 1.4 tmarket-> Product: peanuts, Price: 2.2 Product: sweets, Price: 2.2

5. CITIES BY CONTINENT AND COUNTRY

Create a program that reads **continents**, **countries** and their **cities** put them in a **nested dictionary** and **prints** them.

EXAMPLES

Input	Output
9 Europe Bulgaria Sofia Asia China Beijing Asia Japan Tokyo Europe Poland Warsaw Europe Germany Berlin Europe Poland Poznan Europe Bulgaria Plovdiv Africa Nigeria Abuja Asia China Shanghai	Europe: Bulgaria -> Sofia, Plovdiv Poland -> Warsaw, Poznan Germany -> Berlin Asia: China -> Beijing, Shanghai Japan -> Tokyo Africa: Nigeria -> Abuja
3 Europe Germany Berlin Europe Bulgaria Varna Africa Egypt Cairo	Europe: Germany -> Berlin Bulgaria -> Varna Africa: Egypt -> Cairo
8 Africa Somalia Mogadishu Asia India Mumbai Asia India Delhi Europe France Paris Asia India Nagpur Europe Germany Hamburg Europe Poland Gdansk Europe Germany Danzig	Africa: Somalia -> Mogadishu Asia: India -> Mumbai, Delhi, Nagpur Europe: France -> Paris Germany -> Hamburg, Danzig Poland -> Gdansk

HINTS

- Use a **nested dictionary** (`string → (Dictionary → List<string>)`).
- Check if the continent exists before adding the country. If it doesn't, add it to the dictionary.
- Check if the country exists before adding the city. If it doesn't, add it to the dictionary.
- Pass through all **key-value pairs** in the dictionary and the values' key-value pairs and print the results.

SETS

6. RECORD UNIQUE NAMES

Create a program, which will take a list of **names** and print **only** the **unique** names in the list.

EXAMPLES

Input	Output	Input	Output	Input	Output
8		7	Lyle	6	Roki
John	Alex	Lyle	Bruce	Roki	
Alex	Sam	Bruce	Alice	Roki	
John	Alice	Alice	Easton	Roki	
Sam	Peter	Easton	Shawn	Roki	
Alex		Shawn	Alice	Roki	
Alice		Alice	Shawn	Roki	
Peter		Shawn	Peter		
Alex					

HINTS

You can store the names in a **HashSet<string>** to extract only the unique ones.

7. PARKING LOT

Create a program that:

- Records a **car number** for every car that enters the **parking lot**.
- Removes a **car number** when the car leaves the **parking lot**.

The input will be a string in the format: "**direction, carNumber**". You will be receiving commands until the "**END**" command is given.

Print the car numbers of the cars, which are still in the parking lot:

EXAMPLES

Input	Output
IN, CA2844AA IN, CA1234TA OUT, CA2844AA IN, CA9999TT IN, CA2866HI OUT, CA1234TA IN, CA2844AA OUT, CA2866HI IN, CA9876HH IN, CA2822UU END	CA9999TT CA2844AA CA9876HH CA2822UU
IN, CA2844AA IN, CA1234TA OUT, CA2844AA OUT, CA1234TA END	Parking Lot is Empty

HINTS

- Car numbers are **unique**.
- Before printing, **first check** if the set has any elements.

SOLUTION

You can help yourself with the code below:

```

var input = Console.ReadLine();
var parking = new HashSet<string>();
while (input != "END")
{
    var inputParams = Regex.Split(input, ", ");
    if (inputParams[0] == "IN")
    {
        parking.Add(inputParams[1]);
    }
    else
    {
        parking.Remove(inputParams[1]);
    }

    input = Console.ReadLine();
}

```

8. SOFTUNI PARTY

There is a party in SoftUni. Many guests are invited and there are two types of them: VIP and Regular. When a guest comes, check if he/she exists in any of the two reservation lists.

All reservation numbers will be with the length of 8 chars.

All VIP numbers start with a digit.

First, you will be receiving the reservation numbers of the guests. You can also receive 2 possible commands:

- "**PARTY**" – After this command, you will begin receiving the reservation numbers of the people, who came to the party.
- "**END**" – The party is over and you have to stop the program and print the appropriate output.

In the end, print the count of the guests who didn't come to the party, and afterward, print their reservation numbers. the VIP guests must be first.

EXAMPLES

Input	Output	Input	Output
7IK9Yo0h 9NoBUajQ Ce8vwPmE SVQXQCbc tSzE5t0p PARTY 9NoBUajQ Ce8vwPmE SVQXQCbc END	2 7IK9Yo0h tSzE5t0p	m8rfQBv1 fc1oZCE0 UgffRkOn 7ugX7bm0 9CQBGUEJ 2FQZT3uC dziNz78I mdSGyQCJ LjcVpmDL fPXNHpm1 HTTbwRmM B5yTkMQi 8N0FTThqG xys2FYzn MDzcM9ZK PARTY 2FQZT3uC dziNz78I mdSGyQCJ LjcVpmDL fPXNHpm1	2 xys2FYzn MDzcM9ZK

		HTTbwRmM B5yTkMQi 8N0FTThqG m8rfQBv1 fc1oZCE0 UgffRkOn 7ugX7bm0 9CQBGUeJ END	
--	--	--	--

EXERCISES: SETS AND DICTIONARIES ADVANCED

- You can check your solutions in [Judge](#)
- Ask your questions here <https://www.slido.com/> by entering the code **#csharp-advanced**

1. UNIQUE USERNAME

Create a program that reads from the console a sequence of **N usernames** and keeps a collection only of the **unique** ones. On the **first** line, you will be given an integer **N**. On the next **N** lines, you will receive **one** username **per line**. Print the collection on the console in **order of insertion**:

EXAMPLES

Input	Output
6 John John John Peter John Boy1234	John Peter Boy1234
10 Peter Maria Peter George Sam Maria Sara Peter Sam George	Peter Maria George Sam Sara

2. SETS OF ELEMENTS

Create 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
4 3 1 3	3 5

5	
7	
3	
4	
5	
2 2	1
1	
3	
1	
5	

3. PERIODIC TABLE

Create 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
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

4. EVEN TIMES

Create 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
3 2 -1 2	2
5 1 2 3 1 5	1

5. COUNT SYMBOLS

Create a program that reads some **text** from the console and **counts** the **occurrences** of **each character** in it. Print the results in **alphabetical** (lexicographical) order.

EXAMPLES

Input	Output
SoftUni rocks	: 1 time/s

	S: 1 time/s U: 1 time/s c: 1 time/s f: 1 time/s i: 1 time/s k: 1 time/s n: 1 time/s o: 2 time/s r: 1 time/s s: 1 time/s t: 1 time/s
Did you know Math.Round rounds to the nearest even integer?	: 9 time/s . : 1 time/s ? : 1 time/s D: 1 time/s M: 1 time/s R: 1 time/s a: 2 time/s d: 3 time/s e: 7 time/s g: 1 time/s h: 2 time/s i: 2 time/s k: 1 time/s n: 6 time/s o: 5 time/s r: 3 time/s s: 2 time/s t: 5 time/s u: 3 time/s v: 1 time/s w: 1 time/s y: 1 time/s

6. WARDROBE

Create 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

7. *THE V-LOGGER

Create a program that keeps the 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 vloggername already exists, ignore that command.
- "{vloggername} followed {vloggername}" – The first vlogger followed the second vlogger.
 - If any of the given vloggernames does not exist in your collection, ignore that command.
 - Vlogger cannot follow himself.
 - Vloggers 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 a 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 the one **following fewer** 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** of the **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 an **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

8. *RANKING



Create a program that **ranks** candidate-interns, depending on the **points** from the **interview tasks** and their **exam results** in SoftUni. You will receive some lines of **input** in the format "**{contest}:{password for contest}**" until you receive "**end of contests**". Save that data because **you will need it later**. After that you will receive another type of inputs in the format "**{contest}=>{password}>{username}>{points}**" until you receive "**end of submissions**". Here is what you need to do:

- Check if the **contest is valid** (if you received it in the first type of input).
- Check if the **password is correct for the given contest**.
- Save the user with the **contest** they take part in (**a user can take part in many contests**) and the **points** the user has in the **given contest**. If you receive the **same contest** and the **same user**, **update the points only if the new ones are more than the older ones**.

At the end you have to print the info for the user with the **most points** in the format:

"Best candidate is {user} with total {total points} points.". After that print **all students ordered by their names**. For each user, print **each contest** with the **points** in **descending order** in the following format:

```
{user1 name}
# {contest1} -> {points}
# {contest2} -> {points}
{user2 name}
...
"
```

INPUT

- You will be receiving strings in the formats described above, until the appropriate commands, also described above, are given.

OUTPUT

- On the **first** line, print the best user in the format **described** above.
- On the **next** lines, print all students ordered as mentioned above in format.

CONSTRAINTS

- There will be **no** case with two **equal contests**.
- The **strings** may contain any ASCII character except **(:, =, >)**.
- The **numbers** will be in the range **[0...10000]**.
- The **second** input is always **valid**.
- There will be no case with **2 or more** users with the **same total points**.

EXAMPLES

Input	Output
Part One Interview:success Js Fundamentals:JSFundPass C# Fundamentals:fundPass Algorithms:fun end of contests C# Fundamentals=>fundPass=>Tanya=>350 Algorithms=>fun=>Tanya=>380 Part One Interview=>success=>Nikola=>120 Java Basics Exam=>JSFundPass=>Parker=>400 Part One Interview=>success=>Tanya=>220 OOP Advanced=>password123=>BaiIvan=>231 C# Fundamentals=>fundPass=>Tanya=>250 C# Fundamentals=>fundPass=>Nikola=>200 Js Fundamentals=>JSFundPass=>Tanya=>400	Best candidate is Tanya with total 1350 points. Ranking: Nikola # C# Fundamentals -> 200 # Part One Interview -> 120 Tanya # Js Fundamentals -> 400 # Algorithms -> 380 # C# Fundamentals -> 350 # Part One Interview -> 220

end of submissions	
Java Advanced:funpass	Best candidate is Simon with total 880 points.
Part Two Interview:success	Ranking:
Math Concept:asdasd	Drago
Java Web Basics:forrF	# Math Concept -> 250
end of contests	# Part Two Interview -> 120
Math Concept=>ispass=>Monika=>290	Petyr
Java Advanced=>funpass=>Simon=>400	# Java Advanced -> 90
Part Two Interview=>success=>Drago=>120	# Part Two Interview -> 0
Java Advanced=>funpass=>Petyr=>90	Simon
Java Web Basics=>forrF=>Simon=>280	# Java Advanced -> 400
Part Two Interview=>success=>Petyr=>0	# Java Web Basics -> 280
Math Concept=>asdasd=>Drago=>250	# Part Two Interview -> 200
Part Two Interview=>success=>Simon=>200	
end of submissions	

9. *SOFTUNI EXAM RESULTS



Judge statistics on the last Programming 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
Peter-Java-84 George-C#-70 George-C#-84 Sam-C#-94 exam finished	Results: Sam 94 George 84 Peter 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.
Peter-Java-91 George-C#-84 Sam-JavaScript-90 Sam-C#-50 Sam-banned exam finished	Results: Peter 91 George 84 Submissions: C# - 2 Java - 1 JavaScript - 1	Sam is banned so he is removed from the contest, but his submissions are still preserved in the languages submissions count. So although there are only 2 participants in the results, there are 4 submissions in total.

10. *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 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 **newly 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 then ordered by name**. For each side print the **forceUsers**, **ordered by name**.

In case there are **no forceUsers in the 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 George Dark Peter Lumpawaroo	Side: Dark, Members: 1 ! Peter Side: Light, Members: 1 ! George	We register George 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 John Johnys -> Lighter DCay -> Lighter Lumpawaroo	John Johnys joins the Lighter side! DCay joins the Lighter side! Side: Lighter, Members: 3 ! DCay ! John Johnys ! Royal	Although John Johnys 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.