* **Problem 6. \*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** |
| 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. |

* **Problem 7. \*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.** |