More Exercise: Dictionaries

This document defines the additional exercises for the "Python Fundamentals" course at @Software University.

Please submit your solutions (source code) to all the below-described problems in Judge.

Note: All the exercises are excluded from your homework!

1. Ranking

Here comes the final and the most interesting part – the Final ranking of the candidate-interns. The final ranking is determined by the points of the interview tasks and by the points from the exams in SoftUni. Here is your final task. 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 input 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 (It is considered valid if you received it in the first type of input)**
- Check if the **password is correct for the given contest**
- If the contest and the password are valid, 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.**

In the end, you should print the info for the user with the **most points** (total points for all contents they participated in) 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**. See the examples.

Input

- Strings in format **`"{contest}:{password for contest}"`** until the **`"end of contests"`** command. There will be no case with two equal contests
- Strings in format **`"{contest}=>{password}=>{username}=>{points}"`** until the **`"end of submissions"`** command.
- **There will be no case with 2 or more users with the same total points!**

Output

- On the first line, print the best user in the format **`"Best candidate is {user} with total {total points} points."`**.
- Then print all students ordered as mentioned above in the format:

```
"{user_name1}
#  {contest1} -> {points}
#  {contest2} -> {points}
…
#  {contestN} -> {points}"
```

Constraints

- The strings may contain any ASCII character except from **(:, =, >).**

- The numbers will be in range **[0 - 10000].**
- Second input is always valid.

Examples

Input	Output
Part One Interview:success JS Fundamentals:fundExam 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=>wrong_pass=>Teo=>400 Part One Interview=>success=>Tanya=>220 OOP Advanced=>password123=>Kay=>231 C# Fundamentals=>fundPass=>Tanya=>250 C# Fundamentals=>fundPass=>Nikola=>200 JS Fundamentals=>fundExam=>Tanya=>400 end of submissions	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
Java Advanced:funpass Part Two Interview:success Math Concept:asdasd Java Web Basics:forrF end of contests Math Concept=>ispass=>Monika=>290 Java Advanced=>funpass=>Simona=>400 Part Two Interview=>success=>Drago=>120 Java Advanced=>funpass=>Petyr=>90 Java Web Basics=>forrF=>Simona=>280 Part Two Interview=>success=>Petyr=>0 Math Concept=>asdasd=>Drago=>250 Part Two Interview=>success=>Simona=>200 end of submissions	Best candidate is Simona with total 880 points. Ranking: Drago # Math Concept -> 250 # Part Two Interview -> 120 Petyr # Java Advanced -> 90 # Part Two Interview -> 0 Simona # Java Advanced -> 400 # Java Web Basics -> 280 # Part Two Interview -> 200

2. Judge

You know the judge system, right?! Your job is to create a program similar to the Judge system.

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

"{username} -> {contest} -> {points}"

The "**contest**" and "**username**" are strings, the given "**points**" will be an integer number. You need to keep track of **every contest** and **points of each user**:

- If the **user** has **already participated** in the contest, **update their points only if the new ones are more than the older ones**.
- Otherwise, just **save the data** - contest, username, and points.

Also, you need to keep **individual statistics** for each **user** - his/her final **total points for all contests.**

You should end your program when you receive the command "**no more time**". At that point, you should print each contest in **order of input**, for each contest print the participants **ordered by points in descending order, then ordered by name in ascending order**. After that, you should print **individual statistics for every participant** ordered by **total points** in **descending order**, and **then by alphabetical order**.

Input / Constraints

- The input comes in the form of commands in one of the formats specified above.
- Username and contest name **always will be one word**.
- Points will be an integer in the **range [0, 1000]**.
- There will be **no invalid** input lines.
- If **all sorting criteria fail**, the order should be by **order** of **input**.
- The input ends when you receive the command **"no more time"**.

Output

- The output format for the contests is:

  "{constest_name}: {number_participants} participants

  1. {username1} <::> {points}

  2. {username2} <::> {points}

  …

  {N}. {usernameN} <::> {points}"

- After you print all contests, print the **individual statistics for every participant.**
- The output format is:

  "Individual standings:

  1. {username1} -> {total_points}

  2. {username} -> {total_points}

  …

  {N}. {username} -> {total_points}"

Examples

Input	Output
Peter -> Algo -> 400 George -> Algo -> 300 Simo -> Algo -> 200 Peter -> DS -> 150 Mariya -> DS -> 600 no more time	Algo: 3 participants 1. Peter <::> 400 2. George <::> 300 3. Simo <::> 200 DS: 2 participants 1. Mariya <::> 600 2. Peter <::> 150 Individual standings: 1. Mariya -> 600 2. Peter -> 550 3. George -> 300 4. Simo -> 200
Peter -> OOP -> 350 George -> OOP -> 250 Simo -> Advanced -> 600	OOP: 3 participants 1. Peter <::> 350 2. George <::> 300

George -> OOP -> 300 Prakash -> OOP -> 300 Prakash -> Advanced -> 250 Ani -> JSCore -> 400 no more time	3. Prakash <::> 300 Advanced: 2 participants 1. Simo <::> 600 2. Prakash <::> 250 JSCore: 1 participants 1. Ani <::> 400 Individual standings: 1. Simo -> 600 2. Prakash -> 550 3. Ani -> 400 4. Peter -> 350 5. George -> 300

3. MOBA Challenger

You are a pro MOBA player, you are struggling to become a master of the Challenger tier. So, you carefully watch the statistics in the tier.

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

"{player} -> {position} -> {skill}"

"{player} vs {player}"

The **"player"** and **"position"** are strings, and the given **"skill"** will be an integer number. You need to keep track of **every player**.

When you receive a **player with his position and skill**, add him to the players' pool, if he isn`t present, **else add** his position and skill **or update** his skill, only if the current position skill is lower than the new value.

If you receive **"{player} vs {player}"** and **both players exist** in the tier, **they duel** with the following rules:

- **If they have at least one position in common**, the player with better **total skill points** wins and the other is **demoted** from the tier -> remove him.
- If they have the same total skill points at the same positions, **the duel is tied** and they both continue in the Season.
- If they don`t have positions in common, **the duel isn`t happening** and both continue in the Season.

You should end your program when you receive the command **"Season end"**. At that point you should print the players, **ordered by total skill in descending order, then ordered by player name in ascending order**. **For each** player print their position and skill, **ordered descending by skill, then ordered by position name in ascending order.**

Input / Constraints

- The input comes in the form of commands in one of the formats specified above.
- Player and position **will always be one word string, containing no whitespaces**.
- Skill will be an **integer** in the **range [0, 1000]**.
- There will be **no invalid** input lines.
- The program ends when you receive the command **"Season end"**.

Output

- The output format for each player is:

```
"{player}: {total_skills} skill"
- {position1} <::> {skill}
- {position2} <::> {skill}
…
- {positionN} <::> {skill}"
```

Examples

Input	Output	Comments
Peter -> Adc -> 400 George -> Jungle -> 300 Simon -> Mid -> 200 Simon -> Support -> 250 Season end	Simon: 450 skill - Support <::> 250 - Mid <::> 200 Peter: 400 skill - Adc <::> 400 George: 300 skill - Jungle <::> 300	We order the players by total skill points descending, then by name. We print every position along its skill ordered descending by skill, then by position name.
Peter -> Adc -> 400 Bush -> Tank -> 150 Frank -> Mid -> 200 Frank -> Support -> 250 Frank -> Tank -> 250 Peter vs Frank Frank vs Bush Frank vs Hide Season end	Frank: 700 skill - Support <::> 250 - Tank <::> 250 - Mid <::> 200 Peter: 400 skill - Adc <::> 400	Frank and Peter don`t have a common position, so the duel isn`t valid. Frank wins vs Bush /common position: "Tank". Bush is demoted. Hide doesn`t exist so the duel isn`t valid. We print every player left in the tier.

4. Snow White

Snow White loves her dwarfs, but there are so many, and she doesn't know how to order them. Does she order them by name? Or by the color of their hat? Or by physics? She can't decide, so it's up to you to write a program that does it for her.

You will be receiving **several input lines** which contain **data** about each **dwarf** in the following format:

`{dwarf_name} <:> {dwarf_hat_color} <:> {dwarf_physics}`

The **`"dwarf_name"`** and the **`"dwarf_hat_color"`** are **strings**. The **`"dwarf_physics"`** is an **integer**.

You must **store** the data about the **dwarfs** in your program. There are several rules though:

- If **2 dwarfs** have the **same name** but **different colors**, they should be **considered different dwarfs**, and you should store them **both**.
- If **2 dwarfs** have the **same name** and the **same color**, **store** the **one** with the **higher physics**.

When you receive the command **`"Once upon a time"`**, the input ends. You must **order** the **dwarfs** by **physics** in **descending order** and then by the **total count** of **dwarfs** with the **same hat color** in **descending order**.
Then you must print them all.

Input

- The input will consist of **several input lines**, containing **dwarf data** in the format, specified above.
- The input **ends** when you receive the command **"Once upon a time"**.

Output

- As output, you must print the **dwarfs**, **ordered** in the way, specified above.
- The output format is: **"({hat_color}) {name} <-> {physics}"**

Constraints

- The **"dwarf_name"** will be a **string** that may contain **any ASCII** character except ' ' (**space**), '**<**', '**:**', '**>**'.
- The **"dwarf_hat_color"** will be a **string** that may contain **any ASCII** character except ' ' (**space**), '**<**', '**:**', '**>**'.
- The **"dwarf_physics"** will be an **integer** in the **range [0, $2^{31} - 1$]**.
- There will be **no invalid** input lines.
- If **all sorting criteria fail**, the order should be by **order** of **input**.
- Allowed working **time** / **memory**: **100ms** / **16MB**.

Examples

Input	Output
Peter <:> Red <:> 2000 Teodor <:> Blue <:> 1000 George <:> Green <:> 1000 Simon <:> Yellow <:> 4500 Dopey <:> Simon <:> 1000 Once upon a time	(Yellow) Simon <-> 4500 (Red) Peter <-> 2000 (Blue) Teodor <-> 1000 (Green) George <-> 1000 (Simon) Dopey <-> 1000
Grumpy <:> Red <:> 5000 Grumpy <:> Blue <:> 10000 Grumpy <:> Red <:> 10000 Happy <:> Blue <:> 10000 Once upon a time	(Blue) Grumpy <-> 10000 (Blue) Happy <-> 10000 (Red) Grumpy <-> 10000

5. Dragon Army

*Heroes III is the best game ever. Everyone loves it and everyone plays it all the time. Simon is no exclusion to this rule. His favorite units in the game are all **types** of dragons – black, red, gold, azure, etc. He likes them so much that he gives them **names** and keeps logs of their **stats**: **damage, health,** and **armor**. The process of aggregating all the data is quite tedious, so he would like to have a program doing it. Since he is no programmer, it's your task to help him.*

You need to categorize dragons by their **type**. For each dragon, identified by **name,** keep information about his **stats (damage, health,** and **armor)**. Type is **preserved** as in the order of input, but dragons are **sorted** alphabetically by their name. For each type, you should also print the average **damage**, **health,** and **armor** of the dragons. For each dragon, print **his own stats**.

There **may** be **missing** stats in the input, though. If a stat is missing you should assign it default values. Default values are as follows: health **250**, damage **45**, and armor **10**. Missing stat will be marked as **"null"**.

The input is in the following format **"{type} {name} {damage} {health} {armor}"**.

The **"type"** and the **"name"** are **strings**. The **"damage"**, the **"health"**, and the **"armor"** are **integers**. Any of the integers may be assigned a **"null"** value. See the examples below for a better understanding of your task.

If the same dragon is added a second time, the new stats should **overwrite** the previous ones. Two dragons are considered **equal** if they match by **both name** and **type**.

Input

- On the **first** line, you are given the **number N** -> the number of dragons to follow.
- On the **next N** lines, you are given input in the above-described format. There will be a **single space** separating each element.

Output

- Print the aggregated data on the console.
- For each **type**, print the **average stats** of its **dragons** in the format **"{type}::({damage}/{health}/{armor})"**.
- **Damage**, **health**, and **armor** should be rounded to **two digits** after the decimal separator.
- For each **dragon**, print **its stats** in the format **"-{Name} -> damage: {damage}, health: {health}, armor: {armor}"**.

Constraints

- N is in the range [1…100].
- The dragon type and name are one word only, starting with a capital letter.
- Damage health and armor are integers in the range [0 … 100000] or **null.**

Examples

Input	Output
5 Red Bazgargal 100 2500 25 Black Dargonax 200 3500 18 Red Obsidion 220 2200 35 Blue Kerizsa 60 2100 20 Blue Algordox 65 1800 50	Red::(160.00/2350.00/30.00) -Bazgargal -> damage: 100, health: 2500, armor: 25 -Obsidion -> damage: 220, health: 2200, armor: 35 Black::(200.00/3500.00/18.00) -Dargonax -> damage: 200, health: 3500, armor: 18 Blue::(62.50/1950.00/35.00) -Algordox -> damage: 65, health: 1800, armor: 50 -Kerizsa -> damage: 60, health: 2100, armor: 20
4 Gold Zzazx null 1000 10 Gold Traxx 500 null 0 Gold Xaarxx 250 1000 null Gold Ardrax 100 1055 50	Gold::(223.75/826.25/17.50) -Ardrax -> damage: 100, health: 1055, armor: 50 -Traxx -> damage: 500, health: 250, armor: 0 -Xaarxx -> damage: 250, health: 1000, armor: 10 -Zzazx -> damage: 45, health: 1000, armor: 10