More Exercises: Objects

1. System Components

You will be given a register of systems with components and subcomponents. You need to build an ordered database of all the elements that have been given to you.

The elements are registered in a very simple way. When you have processed all of the input data, you must print them in a specific order. For every System, you must print its components in a specified order, and for every Component, you must print its Subcomponents in a specified order.

The Systems you've stored must be ordered by the **number of components**, in **descending order**, as **first criteria**, and by **alphabetical order** as **second criteria**. The **Components** must be ordered by the **number of Subcomponents**, **in descending order**.

Input

The **input** comes as an array of strings. Each element holds **data** about a **system**, a **component** in that **system**, and a **subcomponent** in that **component**. If the given **system already exists**, you should just **add the new component** to it. If even the **component exists**, you should just **add** the **new subcomponent** to it. The **subcomponents** will **always be unique**. The input format is:

"`{systemName} | {componentName} | {subcomponentName}`"

All of the elements are strings, and can contain **any ASCII character**. The **string comparison** for the alphabetical order is **case-insensitive**.

Output

As **output**, you need to print all of the elements and order them exactly in the way specified above. The format is:

```
"{systemName}
|||{componentName}
|||{component2Name}
||||||{subcomponentName}
||||||{subcomponent2Name}
{system2Name}
..."
```

Examples

Input	Output
['KLLS | Main Site | Home Page', 'KLLS | Main Site | Login Page', 'KLLS | Main Site | Register Page', 'KLLS | Populi Site | Login Page', 'KLLS | Populi Site | Submission Page', 'Lambda | CoreA | A23', 'KLLS | Digital Site | Login Page', 'Lambda | CoreB | B24',	KLLS |||Main Site ||||||Home Page ||||||Login Page ||||||Register Page |||Populi Site ||||||Login Page ||||||Submission Page

'Lambda | CoreA | A24', 'Lambda | CoreA | A25', 'Lambda | CoreC | C4', 'Indice | Session | Default Storage', 'Indice | Session | Default Security']	|||Digital Site ||||||Login Page Lambda |||CoreA ||||||A23 ||||||A24 ||||||A25 |||CoreB ||||||B24 |||CoreC ||||||C4 Indice |||Session ||||||Default Storage ||||||Default Security

Hints

- Creating a sorting function with two criteria might seem a bit daunting at first, but it can be simplified to the following:
    - If elements **a** and **b** are different based on the **first criteria**, then that result is the result of the sorting function, checking the second criteria is not required.
    - If elements **a** and **b** are **equal** based on the **first criteria**, then the result of comparing **a** and **b** on the **second criteria** are the result of the sorting.

What to submit?

Function Signature: function main(components)

2. Usernames

You are tasked to create a catalog of usernames. The usernames will be strings that **may contain any ASCII** character. You **need to order** them **by their length** in **ascending order**, as **first criteria**, and by **alphabetical order** as **second criteria**.

Input

The **input** comes as an array of strings. Each element represents a **username**. Sometimes the input may contain **duplicate usernames**. Make it so that there are **NO duplicates** in the output.

Output

The **output** is all of the usernames, **ordered** exactly as **specified above** – each printed on a new line.

Examples

Input	Output
['Ashton', 'Kutcher', 'Ariel', 'Lilly', 'Keyden',	Aizen Ariel Billy Lilly Ashton

Input	Output
['Denise', 'Ignatius', 'Iris', 'Isacc', 'Indie',	Rot Dean Iris Biser Indie

'Aizen', 'Billy', 'Braston']	Keyden Braston Kutcher

'Dean', 'Donatello', 'Enfuego', 'Benjamin', 'Biser', 'Bounty', 'Renard', 'Rot']	Isacc Bounty Denise Renard Enfuego Benjamin Ignatius Donatello

Hints

- Try to find a **structure** which **does NOT allow duplicates.** It will be best for the current problem.

What to submit?

Function Signature: function main(usernames)

3. Unique Sequences

You will be given an array that contains an **array of numbers** that is **formatted as JSON**. Create a function that stores **unique** arrays and discards duplicate arrays. An array is considered the **same** (**NOT unique**) if it contains the **same numbers** as another array, **regardless of their order**.

Print the arrays in **ascending** order based on their **length**. If two arrays have the same length, they should be printed in the **order of appearance**. The values of each array should also be sorted in **descending order**. Check the examples below.

Input

The **input** comes as an array of strings where each entry is a JSON representing an array of numbers.

Output

The **output** should be printed on the console - each array printed on a new line in the format **"[a_1, a_2, a_3,... a_n]"** , following the above-mentioned ordering.

Examples

Input	Output
["[-3, -2, -1, 0, 1, 2, 3, 4]", "[10, 1, -17, 0, 2, 13]", "[4, -3, 3, -2, 2, -1, 1, 0]"]	[13, 10, 2, 1, 0, -17] [4, 3, 2, 1, 0, -1, -2, -3]

Input	Output
["[7.14, 7.180, 7.339, 80.099]", "[7.339, 80.0990, 7.140000, 7.18]", "[7.339, 7.180, 7.14, 80.099]"]	[80.099, 7.339, 7.18, 7.14]

Hints

- Think of an easy way to compare arrays.
- Sometimes the most obvious collection choice is not the best one.

What to submit?

Function Signature: function main(input)

4. Arena Tier

Pesho is a pro gladiator and he is struggling to become master of the Arena.

You will receive an **array of strings** with each element is formatted by:

`"{gladiator} -> {technique} -> {skill}"`

or

`"{gladiator} vs {gladiator}"`

The **'gladiator'** and **'technique'** are strings and **'skill'** is a number. You need to keep track of **each of the gladiators.**

When you receive a **gladiator with its technique and skill**, you should add it to the gladiator pool. If the gladiator exists in the pool, add its technique and skill or update its technique's skill if the new technique skill is higher than the current one by replacing the value.

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

- Look for a technique that both exists between them. The gladiator with the highest technique skill wins while the losing gladiator should get demoted from the tier and remove it from the pool.
- If no common technique exists, the battle is canceled and you should proceed to the next array element.

You should end your program when you receive the command **"Ave Cesar"**. At that point, you should print the gladiators ordered by **total skill in descending order**, then ordered by **name in ascending order**. For each gladiator print, their technique and skill ordered **skill in descending order**, then ordered by technique **name in ascending order**.

Input / Constraints

You will receive an **array of strings** as a parameter to your solution.

- The input comes in the form of commands in one of the formats specified above.
- Gladiator and technique **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 programm ends when you receive the command **`"Ave Cesar"`**.

Output

- The output format for each gladiator is:

  `"{gladiator}: {totalSkill} skill"`

  `"- {technique} <!> {skill}"`

Scroll down to see examples.

Examples

Input	Output	Comments

Pesho -> BattleCry -> 400 Gosho -> PowerPunch -> 300 Stamat -> Duck -> 200 Stamat -> Tiger -> 250 Ave Cesar	Stamat: 450 skill - Tiger <!> 250 - Duck <!> 200 Pesho: 400 skill - BattleCry <!> 400 Gosho: 300 skill - PowerPunch <!> 300	We order the gladiators by total skill points descending, then by name. We print every technique along its skill ordered descending by skill, then by technique name.
Input	**Output**	
Pesho -> Duck -> 400 Julius -> Shield -> 150 Gladius -> Heal -> 200 Gladius -> Support -> 250 Gladius -> Shield -> 250 Pesho vs Gladius Gladius vs Julius Gladius vs Gosho Ave Cesar	Gladius: 700 skill - Support <!> 250 - Shield <!> 250 - Heal <!> 200 Pesho: 400 skill - Duck <!> 400	Gladius and Pesho don`t have common technique, so the duel isn`t valid. Gladius wins vs Julius /common technique: "Shield". Julius is demoted. Gosho doesn`t exist so the duel isn`t valid. We print every gladiator left in the tier.

What to submit?

Function Signature: function main(commands)

5. Game of Epicness

Write a JavaScript program that **determines** the **winner** from **all battles**. You will receive **two** arguments:

The **first** argument is an **array of kingdoms** with **generals** and their **army** in the form of an **object** with the format:

`{ kingdom: String, general: String, army: Number }`

Every **general** has their **army** that fights for a certain **kingdom**. Note that every **kingdom's name** is **unique**, and every **general's name** is also **unique** in their **kingdom**. If the **general** already **exists in their kingdom**, add the army to their current one. After storing all the kingdoms with their generals and armies, the battles can be simulated.

The **second** argument is a **matrix of strings** showing which **kingdom's generals** will be **fighting** in this format:

[

["{AttackingKingdom} ", "{AttackingGeneral}", "{DefendingKingdom} ", "{DefendingGeneral}"],

...

]

The **first two elements** are the **attacking kingdom and it's general**, respectively while the **other two elements** are the **defending kingdom and it's general**, respectively. Compare the two general armies and whoever has the **larger army wins**. The **winner's army** will **increase** by **10%** and the **loser's army** will **decrease** by **10%**. Keep in mind that armies should be **round down** if there is any **excess army after the battle**. If the battle is a **draw**, **do not do anything**. Keep track of the **win** and **lose count** of each general.

Note that, **generals** from the **same kingdom cannot attack each other**.

After you finished all battles, you need to **determine** which **kingdom wins** the game. To decide that, **order them** by all their **general's wins (descending)** then by their **losses (ascending),** and finally by the **kingdom's name** in **ascending alphabetical** order. In short, the kingdom with the most wins and few losses is the game-winner.

Input

You will receive **two arguments –** an **array of objects** with properties and a **matrix of strings** as shown above.

Output

Print the winning kingdom and **sort** the generals by their **armies in descending** order, **formatted** as seen in the examples.

Constraints

- The **number** of **elements** in the **first input argument** will be in range **[1..100] inclusive.**
- The **number** of **elements** in the **second input argument** will be in range **[0..100] inclusive.**
- General's **army** will be always an **integer** in range **[0..1,000,000] inclusive.**
- There **will** be **no invalid input.**
- There **will** be **no matching number** of **armies** in the **output.**

Examples

Input

`[ { kingdom: "Maiden Way", general: "Merek", army: 5000 },`
`  { kingdom: "Stonegate", general: "Ulric", army: 4900 },`
`  { kingdom: "Stonegate", general: "Doran", army: 70000 },`
`  { kingdom: "YorkenShire", general: "Quinn", army: 0 },`

```
  { kingdom: "YorkenShire", general: "Quinn", army: 2000 },
  { kingdom: "Maiden Way", general: "Berinon", army: 100000 } ],
[ ["YorkenShire", "Quinn", "Stonegate", "Ulric"],
  ["Stonegate", "Ulric", "Stonegate", "Doran"],
  ["Stonegate", "Doran", "Maiden Way", "Merek"],
  ["Stonegate", "Ulric", "Maiden Way", "Merek"],
  ["Maiden Way", "Berinon", "Stonegate", "Ulric"] ]
```

Output

```
Winner: Stonegate
/\general: Doran
---army: 77000
---wins: 1
---losses: 0
/\general: Ulric
---army: 5336
---wins: 2
---losses: 1
```

Explanation

After you successfully store the kingdoms information, the first battle's result is victory for the defender Ulric and a loss for the attacker Quinn. Second battle is ignored because the generals are from the same kingdom. Third battle is a victory for Doran and a loss for Merek. Fourth battle is a win for Ulric and a loss for Merek. Fifth battle is a victory for Berinon and a defeat for Ulric. All winners increase their armies with 10% for each win and all losers decrease their armies with 10% for each loss.

The result from the battles are – Stonegate: 3 wins and 1 loss; Maiden Way: 1 win and 2 losses; YorkenShire: 0 wins and 1 loss. Making Stonegate the winner of the games because they have the most wins from kingdoms.

Input

```
[ { kingdom: "Stonegate", general: "Ulric", army: 5000 },
  { kingdom: "YorkenShire", general: "Quinn", army: 5000 },
  { kingdom: "Maiden Way", general: "Berinon", army: 1000 } ],
```

```
[ ["YorkenShire", "Quinn", "Stonegate", "Ulric"],
  ["Maiden Way", "Berinon", "YorkenShire", "Quinn"] ]
```

Output

```
Winner: YorkenShire
/\general: Quinn
---army: 5500
---wins: 1
---losses: 0
```

Explanation

The first battle between Quinn and Ulric is a draw because they have even armies because of that it is not recorded and their armies size does not change. The second battle is a win for Quinn and a loss for Berinon making YorkenShire the winner of the game with 1 win and 0 losses.

Input

```
[ { kingdom: "Maiden Way", general: "Merek", army: 5000 },
  { kingdom: "Stonegate", general: "Ulric", army: 4900 },
  { kingdom: "Stonegate", general: "Doran", army: 70000 },
  { kingdom: "YorkenShire", general: "Quinn", army: 0 },
  { kingdom: "YorkenShire", general: "Quinn", army: 2000 } ],
[ ["YorkenShire", "Quinn", "Stonegate", "Doran"],
  ["Stonegate", "Ulric", "Maiden Way", "Merek"] ]
```

Output

```
Winner: Maiden Way
/\general: Merek
---army: 5500
---wins: 1
---losses: 0
```

What to submit?

Function Signature: function main(kingdoms, battles)