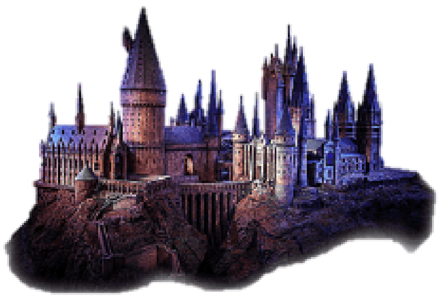
Problem 1 - Hogwarts



Welcome, wizard. We have been waiting for you here, in Hogwarts. Get ready to learn a lot of new potions and spells to fight The one who shall not be named. Your first class is Spellcasting.

First, you will receive **a spell that needs to be deciphered.** Next, you will be receiving **commands** split by a single space until you get the "**Abracadabra**" command. There are **5 possible commands:**

- **"Abjuration"**
  - **Replace** all letters with **upper case** and **print** the result.
- **"Necromancy"**
  - **Replace** all letters with **lower case** and **print** the result.
- **"Illusion {index} {letter}"**
  - **Replace** the letter at the index with the given one and print **"Done!"**
  - If the index is **invalid**, print: **"The spell was too weak."**
- **"Divination {first substring} {second substring}"**
  - **Replace** the **first substring (all matches)** with the **second** and **print the result.**
  - If the substring **does not exist**, **skip** the command.
- **"Alteration {substring}"**
  - **Remove** the **substring from the string** and **print the result.**
  - If the substring **does not exist**, **skip** the command.

If the input command is **not** in the list, print **"The spell did not work!".**

Input / Constraints

- On the **1st line,** you are going to receive the **string**.
- On the next **lines**, until you receive **"Abracadabra"**, you will be receiving commands.
- All commands are **case-sensitive.**

Output

- **Print** the **output** of the **commands** in the **format described above**.

Input

- The **possible** commands are:
    - `"Abracadabra"`
    - `"Abjuration"`
    - `"Necromancy"`
    - `"Illusion {index} {letter}"`
    - `"Divination {first substring} {second substring}"`
    - `"Alteration {substring}"`

Output

- The **possible** outputs are:
    - `"The spell did not work!"`
    - `"The spell was too weak."`
    - `"Done!"`

Examples

JS Examples

Input	Output
(["A7ci0", "Illusion 1 c", "Illusion 4 o", "Abjuration", "Abracadabra"])	Done! Done! ACCIO
(["TR1GG3R", "Necromancy", "Illusion 8 m", "Illusion 9 n", "Abracadabra"])	tr1gg3r The spell was too weak. The spell was too weak.
(["SwordMaster", "Target Target Target", "Abjuration", "Necromancy", "Alteration master", "Abracadabra"])	The spell did not work! SWORDMASTER swordmaster sword

Problem 2 - Message Decrypter

Create a program that **checks** if **inputs** have a **valid message** and **decrypt** it. On the **first** line, you will **receive** a **number** that **indicates** how **many inputs** you will **receive** on the **following** lines.

A message is **valid** when:

- There is **nothing** else **before** and **after it**
- It **starts** with a **tag**, which is **surrounded** by either **"$" or "%"** (but **not both** at the same time). The tag itself has to be **minimum 3** characters long, **start** with an **uppercase** letter, **followed only** by **lowercase** letters
- There is a **colon** and a single **white space** after the tag
- There are **3 groups** consisting of **numbers** between **"["** and **"]"**, followed by a **pipe** (**"|"**)

Example for a valid message:

"$Request$: [73]|[115]|[32]|"

You must **check** if the **message** is **valid,** and if it **is** - **decrypts** it. If it **isn't** - **print** the following **message:**

"Valid message not found!"

Decrypting a **message** means taking **all numbers** and **turn** them **into ASCII symbols.** After successful decrypt, print it in the following format:

"{tag}: {decryptedMessage}"

Input

- On the **first** line - **n** - the count of inputs.
- On the **next** n lines - **input** that you have to **check** if it has a **valid message.**

Output

- Print all **results from each input, each on a new line.**

Input

Example for a valid message:

"$Request$: [73]|[115]|[32]|"

Output

- The **possible** outputs are:
    - **`"{tag}: {decryptedMessage}"`**
    - **`"Valid message not found!"`**

Examples

JS Examples

The input will be provided as an array of strings**.**

Input	Output	Comment

`(["4",`
`"$Request$: [73]|[115]|[105]|",`
`"%Taggy$: [73]|[73]|[73]|",`
`"%Taggy%: [118]|[97]|[108]|",`
`"$Request$: [73]|[115]|[105]|[32]|[75]|"])`
	`Request: Isi`
	`Valid message not found!`
	`Taggy: val`
	`Valid message not found!`
		We have **4 input** lines to **check**. The **first** one **follows** the **rules** and **is valid**. The **second** one **doesn't** because the tag is **surrounded** by **both** '**%**' and '**$**'. **The third one** also is **a valid message**. The **last** one is **invalid** because it has **more than 3 groups of numbers**.

`(["3",`
`"This shouldnt be valid%Taggy%: [118]|[97]|[108]|",`
`"$tAGged$: [97][97][97]|",`
`"$Request$: [73]|[115]|[105]|true"])`
	`Valid message not found!`
	`Valid message not found!`
	`Valid message not found!`	

Problem 3 - Hero Recruitment

Create a program that keeps track of enrolled heroes and their collection of spells (spellbook). You will be receiving the following commands until you receive the command "**End**":

- **"Enroll {HeroName}"**:
  - Adds the hero to your collection of heroes.
  - If the hero is already present in your collection, print: **"{HeroName} is already enrolled."**
- **"Learn {HeroName} {SpellName}"**:
  - Adds the spell to the hero's spellbook.
  - If the hero does not exist in the collection, print: **"{HeroName} doesn't exist."**
  - If the hero already has the spell in his spellbook, print: **"{HeroName} has already learnt {SpellName}."**
- **"Unlearn {HeroName} {SpellName}"**:
  - Removes the spell from the hero's spellbook.
  - If the hero doesn't exist in the collection, print: **"{HeroName} doesn't exist."**
  - If the spell doesn't exist in the hero's spellbook, print: **"{HeroName} doesn't know {SpellName}."**
  -

After receiving the **"End"** command, print all the heroes:

```
"Heroes:
== {name1}: {spell1}, {spell2}, {spelln}
== {name2}: {spell1}, {spell2}, {spelln}
…
== {nameN}: {spell1}, {spell2}, {spelln}"
```

Input / Constraints

- You will be receiving **lines** until you receive the **"End"** command.

Output

- Print the **heroes** in the **format** described above.

Input

- The **possible** commands are:
  - **"End"**
  - **"Enroll {HeroName}"**
  - **"Learn {HeroName} {SpellName}"**
  - **"Unlearn {HeroName} {SpellName}"**

Output

- The **possible** outputs are:
  - **`"{HeroName} is already enrolled."`**
  - **`"{HeroName} doesn't exist."`**
  - **`"{HeroName} has already learnt {SpellName}."`**
  - **`"{HeroName} doesn't know {SpellName}."`**
  -
    ```
    "Heroes:
    == {name1}: {spell1}, {spell2}, {spelln}
    == {name2}: {spell1}, {spell2}, {spelln}
    …
    == {nameN}: {spell1}, {spell2}, {spelln}"
    ```

Examples

JS Examples

The input will be provided as an array of strings.

Input	Output

`(["Enroll Stefan",`
`"Enroll Peter",`
`"Enroll Stefan",`
`"Learn Stefan ItShouldWork",`
`"Learn John ItShouldNotWork",`
`"Unlearn George Dispel",`
`"Unlearn Stefan ItShouldWork",`
`"End"])`
	`Stefan is already enrolled.`
	`John doesn't exist.`
	`George doesn't exist.`
	`Heroes:`
	`== Stefan:`
	`== Peter:`

`(["Enroll Stefan",`
`"Learn Stefan ItShouldWork",`
`"Learn Stefan ItShouldWork",`
`"Unlearn Stefan NotFound",`
`"End"])`
	`Stefan has already learnt ItShouldWork.`
	`Stefan doesn't know NotFound.`
	`Heroes:`
	`== Stefan: ItShouldWork`

`(["Enroll Stefan",`
`"Enroll Peter",`
`"Enroll John",`
`"Learn Stefan Spell",`
`"Learn Peter Dispel",`
`"End"])`
	`Heroes:`
	`== Stefan: Spell`
	`== Peter: Dispel`
	`== John:`