Exercises: Regular Expressions

1. Furniture

Write a function to calculate the total cost of different types of furniture. You will be given some lines of input until you receive the line "Purchase". For the line to be valid it should be in the following format:

">>{furniture name}<<{price}!{quantity}"

The price can be floating point number or whole number. Store the names of the furniture and the total price. At the end print the each bought furniture on separate line in the format:

"Bought furniture:

{1st name}

{2nd name}

…"

And on the last line print the following: **"Total money spend: {spend money}"** formatted to the second decimal point.

Examples

Input	Output	Comment
>>Sofa<<312.23!3 >>TV<<300!5 >Invalid<<!5 Purchase	Bought furniture: Sofa TV Total money spend: 2436.69	Only the Sofa and the TV are valid, for each of them we multiply the price by the quantity and print the result

2. Race

Write a function that processes information about a race. On the **first line** you will be given **list of participants separated by ", "**. On the next few lines until you receive a line **"end of race"** you will be given some info which will be some **alphanumeric characters**. In between them you could have some **extra characters which you should ignore**. For example: **"G!32e%o7r#32g$235@!2e"**. The **letters are the name** of the person and the **sum of the digits is the distance** he ran. So here we have **George** who ran **29 km**. Store the information about the person only **if the list of racers contains the name of the person**. If you receive the **same person more than once just add the distance to his old distance**. At the end **print the top 3 racers** ordered by **distance in descending** in the format:

"1st place: {first racer}

2nd place: {second racer}

3rd place: {third racer}"

Examples

Input	Output	Comment
[`George, Peter, Bill, Tom`, `G4e@55or%6g6!68e!!@`, `R1@!3a$y4456@`, `B5@i@#123ll`, `G@e54o$r6ge#`, `7P%et^#e5346r`, `T$o553m&6`, `end of race`]	1st place: George 2nd place: Peter 3rd place: Tom	On the 3rd input line we have Ray. He is not in the list, so we do not count his result. The other ones are valid. George has total of 55 km, Peter has 25 and Tom has 19. We do not print Bill because he is on 4th place.

3. Kingsland Bar Income

Let`s take a break and visit the game bar at Kingsland. It is about time for the people behind the bar to go home and you are the person who has to draw the line and calculate the money from the products that were sold throughout the day. Until you receive a line with text **"end of shift"** you will be given lines of input. But before processing that line you have to do some validations first.

Each valid order should have a **customer, product, count and a price:**

- Valid customer's name should be **surrounded by '%'** and must **start with a capital letter**, followed by **lower-case letters**
- Valid product **contains any word character** and must be **surrounded by '<' and '>'**
- Valid count is an **integer**, **surrounded by '|'**
- Valid price is any **real number followed by '$'**

The parts of a valid order should appear in the order given: **customer, product, count and a price**.

Between each part there can be other symbols, except (**'|', '$', '%' and '.'**)

For each valid line print on the console: **"{customerName}: {product} - {totalPrice}"**

When you receive **"end of shift"** print the total amount of money for the day **rounded to 2 decimal places** in the following format: **"Total income: {income}".**

Input / Constraints

- Strings that you have to process until you receive text **"end of shift".**

Output

- Print all of the valid lines in the format **"{customerName}: {product} - {totalPrice}"**
- After receiving **"end of shift"** print the total amount of money for the day rounded to 2 decimal places in the following format: **"Total income: {income}"**
- Allowed working **time** / **memory**: **100ms** / **16MB**.

Examples

Input	Output	Comment
%George%<Croissant>|2|10.3$ %Peter%<Gum>|1|1.3$ %Maria%<Cola>|1|2.4$ end of shift	George: Croissant - 20.60 Peter: Gum - 1.30 Maria: Cola - 2.40 Total income: 24.30	Each line is valid, so we print each order, calculating the total price of the product bought. At the end we print the total income for the day
%InvalidName%<Croissant>|2|10.3$ %Peter%<Gum>1.3$ %Maria%<Cola>|1|2.4 %Valid%<Valid>valid|10|valid20$ end of shift	Valid: Valid - 200.00 Total income: 200.00	On the first line, the customer name isn`t valid, so we skip that line. The second line is missing product count. The third line don`t have a valid price. And only the forth line is valid

4. *Star Enigma

The war is in its peak, but you, young Padawan, can turn the tides with your programming skills. You are tasked to create a program to **decrypt** the messages of The Order and prevent the death of hundreds of lives.

You will receive several messages, which are **encrypted** using the legendary star enigma. You should **decrypt the messages**, following these rules:

To properly decrypt a message, you should **count all the letters [s, t, a, r]** – **case insensitive** and **remove** the count from the **current ASCII value of each symbol** of the encrypted message.

After decryption:

Each message should have a **planet name, population, attack type ('A', as attack or 'D', as destruction) and soldier count.**

The planet name **starts after '@'** and contains **only letters from the Latin alphabet**.

The planet population **starts after ':'** and is an **Integer**;

The attack type may be **"A"(attack) or "D"(destruction)** and must be **surrounded by "!"** (exclamation mark).

The **soldier count** starts after **"->"** and should be an Integer.

The order in the message should be: **planet name -> planet population -> attack type -> soldier count.** Each part can be separated from the others by **any character except: '@', '-', '!', ':' and '>'.**

Input / Constraints

- The **first line holds n** – the number of **messages– integer in range [1…100];**
- On the next **n** lines, you will be receiving encrypted messages.

Output

After decrypting all messages, you should print the decrypted information in the following format:

First print the attacked planets, then the destroyed planets.

```
"Attacked planets: {attackedPlanetsCount}"
"-> {planetName}"
"Destroyed planets: {destroyedPlanetsCount}"
"-> {planetName}"
```

The planets should be **ordered by name alphabetically.**

Examples

Input	Output	Comments
2 STCDoghudd4=63333D0A53333 EHfsytsnhf?8555&I&2C9555SR	Attacked planets: 1 -> Alderaa Destroyed planets: 1 -> Cantonica	We receive two messages, to decrypt them we calculate the key: First message has decryption key 3. So we substract from each characters code 3. **PQ@Alderaa1:30000!A!->20000** The second message has key 5. **@Cantonica:3000!D!->4000NM** **Both messages are valid** and they contain planet, population, attack type and soldiers count. After decrypting all messages we print each planet according the format given.
Input	**Output**	**Comments**
3 tt(''DGsvywgerx>6444444444%H% 1B9444 GQhrr|A977777(H(TTTT EHfsytsnhf?8555&I&2C9555SR	Attacked planets: 0 Destroyed planets: 2 -> Cantonica -> Coruscant	We receive three messages. Message one is decrypted with key 4: **pp$##@Coruscant:2000000000!D!->5000** Message two is decrypted with key 7: **@Jakku:200000!A!MMMM** This is **invalid message**, missing soldier count, so we continue. The third message has key 5. **@Cantonica:3000!D!->4000NM**

"It's a trap!" – Admiral Ackbar