Exercises: Regular Expressions

Problems for exercise for the "PHP Fundamentals" course @ SoftUni.

You can check your solutions in Judge.

1. Extract Emails

Write a program to **extract all email addresses from a given text**. The text comes at the only input line. Print the emails on the console, each at a separate line. Emails are considered to be in format **<user>@<host>**, where:

- **<user>** is a sequence of **letters** and **digits**, where '**.**', '**-**' and '**_**' can appear between them.
  - Examples of valid users: "**stephan**", "**mike03**", "**s.johnson**", "**st_steward**", "**softuni-bulgaria**", "**12345**".
  - Examples of invalid users: "**--123**", "**.....**", "**nakov_-**", "**_steve**", "**.info**".
- **<host>** is a sequence of at least two words, separated by dots '**.**'. Each word is sequence of letters and can have hyphens '**-**' between the letters.
  - Examples of hosts: "**softuni.bg**", "**software-university.com**", "**intoprogramming.info**", "**mail.softuni.org**".
  - Examples of invalid hosts: "**helloworld**", "**.unknown.soft.**", "**invalid-host-**", "**invalid-**".
- Examples of **valid emails**: **info@softuni-bulgaria.org**, **kiki@hotmail.co.uk**, **no-reply@github.com**, **s.peterson@mail.uu.net**, **info-bg@software-university.software.academy**.
- Examples of **invalid emails**: **--123@gmail.com**, **…@mail.bg**, **.info@info.info**, **_steve@yahoo.cn**, **mike@helloworld**, **mike@.unknown.soft.**, **s.johnson@invalid-**.

Examples

Input	Output
`Please contact us at: support@github.com.`	`support@github.com`
`Just send email to s.miller@mit.edu and j.hopking@york.ac.uk for more information.`	`s.miller@mit.edu` `j.hopking@york.ac.uk`
`Many users @ SoftUni confuse email addresses. We @ Softuni.BG provide high-quality training @ home or @ class. -- steve.parker@softuni.de.`	`steve.parker@softuni.de`

2. Furniture

Write a program 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

3. Race

Write a program 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.

4. SoftUni Bar Income

Let`s take a break and visit the game bar at SoftUni. 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

5. Santa's Secret Helper

After the successful second Christmas, Santa needs to gather information about the behavior of children to plan the presents for next Christmas. He has a secret helper, who is sending him **encrypted** information. Your task is to **decrypt it** and create a list of the children who have been good.

You will receive an **integer**, which represents **a key** and afterwards some **messages**, which you **must decode** by **subtracting the key** from the **value** of **each character**. After the decryption, to be considered a valid match, a message should:

- Have a name, which **starts after '@'** and contains **only letters from the Latin alphabet**
- Have a behaviour type - **"G"(good) or "N"(naughty)** and must be **surrounded by "!"** (exclamation mark).

The order in the message should be: **child's name -> child's behavior.** They can be separated from the others by **any character except: '@', '-', '!', ':' and '>'.**

You will be receiving message until you are given the "**end**" command. Afterwards, print the names of the children, who will receive a present, each on a new line.

Input / Constraints

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

Output

Print the **names of the children**, each on a new line

Examples

Input	Output	Comments
3 CNdwhamigyenumje$J$ CEreelh-nmguuejn$J$ CVwdq&gnmjkvng$Q$ end	Kate Bobbie	We receive three messages and to decrypt them we use the key: First message has decryption key 3. So we substract from each characters code 3 and we receive: @Kate^jfdvbkrjgb!G! @Bobbie*kjdrrbgk!G! @Stan#dkjghskd!N! **They are all valid** and they contain a child's name and behavior – G for good and N for naughty.

Input	Output	Comments
3 N}eideidmk$'(mnyenmCNlpamnin$J$ ddddkkkkmvkvmCFrqqru-nvevek$J$nmgievnge ppqmkkkmnolmnnCEhq/vkievk$Q$ yyegiivoguCYdohqwlqh/kguimhk$J$ end	Kim Connor Valentine	We receive four messages. They are with key 3: Kzbfabfajh!$%jkvbkj@Kim^jkfk!G! aaaahhhhjshsj@Connor*ksbsbh!G!kjdfbskdb mmnjhhhjklijkk@Ben,shfbsh!N! vvbdffsldr@Valentine,hdrfjeh!G!

6. *Winning Ticket

Lottery is exciting. What is not, is checking a million tickets for winnings only by hand. So, you are given the task to create a program which automatically checks if a ticket is a winner.

You are given a **collection of tickets separated by commas and spaces**. You need to check every one of them if it has a winning combination of symbols.

A valid ticket should have exactly 20 characters. The winning symbols are '**@**', '**#**', '**$**' and '**^**'. But in order for a ticket to be a winner the symbol should uninterruptedly repeat for at least **6 times** in both the **tickets left half** and the **tickets right half**.

For example, a valid winning ticket should be something like this:

"**Cash$$$$$$Ca$$$$$$sh**"

The left half "**Cash$$$$$$**" contains "**$$$$$$**", which is also contained in the tickets right half "**Ca$$$$$$sh**". A winning ticket should contain symbols repeating up to 10 times in both halves, which is considered a Jackpot (for example: "**$$$$$$$$$$$$$$$$$$$$**").

Input

The input will be read from the console. The input consists of a **single line** containing all tickets **separated by commas and one or more white spaces** in the format:

- **"{ticket}, {ticket}, … {ticket}"**

Output

Print the result for every ticket in the order of their appearance, each on a separate line in the format:

- **Invalid ticket - "invalid ticket"**
- **No match - "ticket "{ticket}" - no match"**
- **Match with length 6 to 9 - "ticket "{ticket}" - {match length}{match symbol}"**
- **Match with length 10 - "ticket "{ticket}" - {match length}{match symbol} Jackpot!"**

Constrains

- Number of tickets will be in range [0 … 100]

Examples

Input	Output
`Cash$$$$$Ca$$$$$sh`	`ticket "Cash$$$$$Ca$$$$$sh" - 6$`

`$$$$$$$$$$$$$$$$$$, aabb , th@@@@@@eemo@@@@@@ey`	`ticket "$$$$$$$$$$$$$$$$$$" - 10$ Jackpot!`
	`invalid ticket`
	`ticket "th@@@@@@eemo@@@@@@ey" - 6@`

`validticketnomatch:(`	`ticket "validticketnomatch:(" - no match`