# Exercise: Regular Expressions

Problems for exercises and homework for the ["Programming Fundamentals" course @ SoftUni](https://softuni.bg/trainings/3366/java-fundamentals-may-2021).

You can check your solutions in [Judge.](https://judge.softuni.bg/Contests/1673/Regular-Expressions-Exercises)

## 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 a floating-point number or a whole number. Store the names of the furniture and the total price. At the end print each bought furniture on a 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 |
| >>TV<<312.23!3  >>Monitor<<300!5  >>Invalid<<<<!5  >>Sink<<220!10  >>>>>>Invalid<<!5  Purchase | Bought furniture:  TV  Monitor  Sink  Total money spend: 4636.69 |  |

## Race

Write a program that processes information about a race. On the **first line** you will be given a **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$o553m&6  end of race | 1st place: George  2nd place: Peter  3rd place: Tom | On the 3rd input line we have Ray. He is not in the list, so we do not count his result. The other ones are valid. George has total of 55 km, Peter has 25 and Tom has 19. We do not print Bill because he is on 4th place. |
| Joro, George, Georgi, Stamat  ^&^%^232St#$ama&&^^t  ^&^%^232St#$ama&&^^t  G7667eorge&^^&^  &^&&&J99999oro&^^57  end of race | 1st place: Joro  2nd place: George  3rd place: Stamat |  |

## \*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  **ST**CDoghudd4=63333$D$0**A**53333  EHf**s**y**ts**nhf?8555&I&2C9555**SR** | Attacked planets: 1  -> Alderaa  Destroyed planets: 1  -> Cantonica | We receive two messages, to decrypt them we calculate the key:  First message has decryption key 3. So we substract from each characters code 3.  **PQ@Alderaa1:30000!A!->20000**  The second message has key 5.  **@Cantonica:3000!D!->4000NM**  **Both messages are valid** and they contain planet, population, attack type and soldiers count.  After decrypting all messages we print each planet according the format given. |
| **Input** | **Output** | **Comments** |
| 3  **tt**(''DG**s**vywge**r**x>6444444444%H%1B9444  GQh**rr**|**A**977777(H(**TTTT**  EHf**s**y**ts**nhf?8555&I&2C9555**SR** | Attacked planets: 0  Destroyed planets: 2  -> Cantonica  -> Coruscant | We receive three messages.  Message one is decrypted with key 4:  **pp$##@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

## \*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 |