# Exercises: Advanced Collections

Problems for exercises and homework for the [“Programming Fundamentals” course @ SoftUni](https://softuni.bg/courses/programming-fundamentals).

Check your solutions here: [https://judge.softuni.bg/Contests/433/Advanced-Collections-Exercises](https://judge.softuni.bg/Contests/433).

## Shellbound

Vladi is a crab. Crabs are scared from almost anything, which is why they usually hide in their shells. Vladi is a rich crab though. He has acquired many outer shells, in different regions, in which he can hide – each with a different size.

However, it is really annoying to look after all your shells when they are so spread. That is why, Vladi decided to merge all shells in each region, so that he has one big shell for every region. He needs your help to do that.

You will be given several input lines containing a **string** and an **integer**, **separated** by a **space**. The **string** will represent the **region’s name**, and the **integer** – the shell, in the **given region**, **size**.

You must store all shells in their corresponding regions.  
If the region **already exists**, **add** the **new shell** to it. Make sure you have **NO** duplicate shells (shells with **same size**). Vladi doesn’t like shells with the same size.

When you receive the command “**Aggregate**”, you must stop reading input lines, and you must print **every region**, **all of the shells** in that region, and the **size** of the **giant shell** after you’ve merged them in the following format:

{regionName} -> {shell1, shell2, shell3…} ({giantShell})

The giant shell size is calculated when you subtract the average of the shells from the sum of the shells.  
Or in other words: (sum of shells) – (sum of shells) / (count of shells).

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Sofia 50  Sofia 20  Sofia 30  Varna 10  Varna 20  Aggregate | Sofia -> 50, 20, 30 (67)  Varna -> 10, 20 (15) |
| Sofia 2  Sofia 1  Plovdiv 100  Plovdiv 50  Aggregate | Sofia -> 2, 1 (2)  Plovdiv -> 100, 50 (75) |

### Constraints

* All numeric data should be represented with **integer** variables in **range [0, 1.000.000.000]**.

## Dict-Ref-Advanced

Remember the DIct-Ref Problem from the previous exercise? Well this one is an Advanced Version.

You will begin receiving input lines containing information in one of the following formats:

* {key} -> {value1, value2, value3. . .}
* {key} -> {otherKey}

The **keys** will always be **strings**, and the **values** will always be **integers**, **separated** by a **comma** and a **space**.

If you are **given a key** and **values**, you must **store** the **values** to the **given key**. If the **key** already **exists**, you must **add** the **given values** to the old ones.

If you are **given a key** and **another key**, you must **copy** the **values** of the **other key** to the **first one**. If the **other key** **does not exist**, this input line must be **IGNORED**.

When you receive the command “**end**”, you must stop reading input lines, and you must print all keys with their values, in the following format:

{key} === {value1, value2, value3. . .}

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Isacc -> 5, 4, 3  Peter -> 6, 3, 3  Derek -> 2, 2, 2  end | Isacc === 5, 4, 3  Peter === 6, 3, 3  Derek === 2, 2, 2 |
| Donald -> 2, 2, 2  Isacc -> 1  George -> John  John -> Isacc  end | Donald === 2, 2, 2  Isacc === 1  John === 1 |

## Forum Topics

You have been tasked to store a few forum topics, and filter them by several given tags.  
You will be given several input lines, containing data about topics in the following format:

{topic} -> {tag1, tag2, tag3. . .}

**The topic and tags will be strings. They will NOT contain spaces or ‘-’, ‘>’ symbols.**

**If you receive an existent topic, you must add the new tags to it. There should be NO duplicate tags.**

**When you receive the command “filter”, you must end the input sequence. On the next line (after “filter”), you will receive a sequence of tags, separated by a comma and a space. You must print ONLY those topics, which CONTAIN ALL tags in the given sequence.**

**The topics must be printed in the following format:**{topic} | {#tag1, #tag2, #tag3. . .}

**NOTE: The tags have a hashtag (‘#’) as a prefix.**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| HelloWorld -> hello, forum, topic  HelpWithHomework -> homework, forum, topic  filter  forum, topic | HelloWorld | #hello, #forum, #topic  HelpWithHomework | #homework, #forum, #topic |
| First -> this  First -> that  First -> who  Second -> this, what, why  First -> this  Third -> this, third  Third -> that  filter  that, this | First | #this, #that, #who  Third | #this, #third, #that |

## ****Social Media Posts****

You have been tasked to create a Console Social Media Database.

You will receive several input lines in one of the following formats:

* post {postName}
* like {postName}
* dislike {postName}
* comment {postName} {commentatorName} {content}

If you receive the **post** command, you must **create** a **post** with the **given name**.

If you receive the **like** command you must **add** a **like** to the **given post**.

If you receive the **dislike** command you must add a **dislike** to the **given post**.

If you receive the **comment** command, you must **add** a **comment** to the **given post**. The **comment’s** **writer** must be the **given commentator name**, and the **content** of the **comment** must be the **given content**.

By default the posts have **0 likes**, **0 dislikes** and **0 comments**, when created.

When you receive the command “**drop the media**” you must end the input sequence, and you must print every post with its likes, disliked and comments in the following format:

Post: {postName} | Likes: {likes} | Dislikes {dislikes}  
Comments:  
\* {commentator1}: {comment1}  
\* {commentator2}: {comment2}  
. . .

If a certain **post** does **NOT** have **ANY** comments, you should just print “**None**”.

The comments have a **prefix** of a **single** **asterisk** (‘**\***’) and **2 spaces**.

There is **NO space** between the **commentator’s name** and the **colon**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| post FirstPost  like FirstPost  like FirstPost  dislike FirstPost  post SecondPost  comment FirstPost Isacc Cool  comment SecondPost Isacc Lol  like SecondPost  drop the media | Post: FirstPost | Likes: 2 | Dislikes: 1  Comments:  \* Isacc: Cool  Post: SecondPost | Likes: 1 | Dislikes: 0  Comments:  \* Isacc: Lol |
| post SomePost  like SomePost  like SomePost  dislike SomePost  post OtherPost  comment OtherPost Isacc Naaais  comment OtherPost Peter GoodPost  comment OtherPost John Meh...  drop the media | Post: SomePost | Likes: 2 | Dislikes: 1  Comments:  None  Post: OtherPost | Likes: 0 | Dislikes: 0  Comments:  \* Isacc: Naaais  \* Peter: GoodPost  \* John: Meh... |

### Constraints

* The **post name** will be a **string** of **letters** and **digits**.
* The **commentator’s name** will be a **string** of **letters**.
* The comment’s **CONTENT**, may contain **ANY ASCII** character.
* There will be **NO invalid** input data.