# 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).

## Wardrobe

You just bought a new wardrobe. The weird thing about it is that it came prepackaged with a big box of clothes (just like those refrigerators, which ship with free beer inside them)! So, you’ll need to find something to wear.

### Input

On the first line of the input, you will receive n – the **number of lines** of clothes, which came prepackaged for the wardrobe.

On the next n lines, you will receive the clothes for each color in the format:

* “{color} -> {item1},{item2},{item3}…”

If a color is added a **second** time, **add** **all items** from it and **count** the **duplicates**.

**Finally**, you will receive the **color** and **item** of the clothing, that you need to look for.

### Output

Go through all the **colors** of the clothes and print them in the following format:

|  |
| --- |
| **{color}** clothes:  \* **{item1}** - **{count}**  \* **{item2}** - **{count}**  \* **{item3}** - **{count}**  …  \* **{itemN}** - **{count}** |

If the **color** lines up with the **clothing item**, print “(found!)” alongside the item. See the examples to better understand the output.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4  Blue -> dress,jeans,hat  Gold -> dress,t-shirt,boxers  White -> briefs,tanktop  Blue -> gloves  Blue dress | Blue clothes:  \* dress - 1 (found!)  \* jeans - 1  \* hat - 1  \* gloves - 1  Gold clothes:  \* dress - 1  \* t-shirt - 1  \* boxers - 1  White clothes:  \* briefs - 1  \* tanktop - 1 |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4  Red -> hat  Red -> dress,t-shirt,boxers  White -> briefs,tanktop  Blue -> gloves  White tanktop | Red clothes:  \* hat - 1  \* dress - 1  \* t-shirt - 1  \* boxers - 1  White clothes:  \* briefs - 1  \* tanktop - 1 (found!)  Blue clothes:  \* gloves - 1 |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  Blue -> shoes  Blue -> shoes,shoes,shoes  Blue -> shoes,shoes  Blue -> shoes  Blue -> shoes,shoes  Red tanktop | Blue clothes:  \* shoes - 9 |

## Key-Key Value-Value

Write a program, which searches for a **key** and **value** inside of several **key-value** pairs.

### Input

* On the **first** line, you will receive a **key**.
* On the **second** line, you will receive a **value**.
* On the **third** line, you will receive **N**.
* On the next **N** lines, you will receive strings in the following format:

“key => {value 1};{value 2};…{value X}”

After you receive **N** **key -> values** pairs, your task is to go through them and print **only** the **keys**, which contain the **key** and the **values**, which contain the **value**. Print them in the following format:

|  |
| --- |
| **{key}**:  -**{value1}**  -**{value2}**  …  -**{valueN}** |

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| bug  X  3  invalidkey => testval;x;y  debug => XUL;ccx;XC  buggy => testX;testY;XtestZ | debug:  -XUL  -XC  buggy:  -testX  -XtestZ |
| key  valu  2  xkeyc => value;value;valio  keyhole => valuable;x;values | xkeyc:  -value  -value  keyhole:  -valuable  -values |

## Travel Company

Write a program, which categorizes information about a travel company.

Companies have various vehicles planned for different cities. Every city has prepared several **types of vehicles**. Each vehicle type has a different **capacity**.

Until you receive the command “ready”, you will receive all the **cities** the company offers holiday packages for, and their respective **vehicle types** + **capacities** in the format:

* “{city}:{type}-{capacity},{type}-{capacity}…”

An example city with its transportation options would look like this:

* “Athens:bus-40,airplane-300,train-150”

If a city is entered a **second time**, add all transport which **hasn’t already been added** and **replace** **existing** transports’ capacities with the **new** ones.

After the “ready” command, you will start receiving **groups** for various cities in the format: “{city} {peopleCount}” until you receive “travel time!”.

After that, calculate whether the **group** will have **enough seats** to accommodate the passengers and print the status per these conditions:

If the group’s size is **smaller than or equal to** the **combines seats** in all the vehicles, print:

* “{city} -> all {peopleCount} accommodated”

If the group’s size is **larger than** the **combines seats** in all the vehicles, print:

* “{city} -> all except {peopleCount - transportCapacities} accommodated”

### Constraints

* There will be **no redundant whitespaces** anywhere in the input.
* The input will **always** be in the **format specified**.
* The **group cities** will **always** be **existing** cities.
* The **group sizes** will **always** be **positive**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Athens:bus-40,airplane-300,train-150  Athens:minibus-8,airplane-350  Warsaw:bus-30,train-150,airplane-120  ready  Athens 400  Warsaw 500  travel time! | Athens -> all 400 accommodated  Warsaw -> all except 200 accommodated |

|  |  |
| --- | --- |
| **Input** | **Output** |
| Sofia:bus-30,airplane-150,train-25  Istanbul:minibus-6,train-80  Sofia:bus-45  Sofia:bus-50  Berlin:airplane-120  ready  Berlin 115  Istanbul 200  Sofia 200  travel time! | Berlin -> all 115 accommodated  Istanbul -> all except 114 accommodated  Sofia -> all 200 accommodated |

## Shellbound

Vladi is a crab. Crabs are scared of 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 out. 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 to have 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} -> {shell 1, shell 2…, shell n…} ({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)).

### Constraints

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

### 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) |

## 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} -> {value 1, value 2, …, value n}
* {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 existing 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, …, #tagN}

**NOTE: The tags have a number sign (‘**#**’) 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} {commenterName} {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**, **dislikes** 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**.

### 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.

### 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... |