# Exercise: STRINGS, REGULAR EXPRESSIONS AND TEXT PROCESSING

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

## Trainegram

It’s like telegram, but … between trains. You have been tasked to write a software that read encoded trainegram messages. Trainegram messages are used to send meta information about trains.

A **locomotive** consists of **2** **surrounding square brackets** and a “**less than” symbol** and a **dot** at the **end** (“<[].”). **Between** the **brackets**, the **locomotive** may contain **ANY symbol** except **letters** and **digits**.

**Valid** **locomotives**: “<[\*/\*].”, “<[---].”, “<[.;[].”... **Invalid locomotives**: “[//].”, “<[\*-\*]”, “<[asd1].”

A **locomotive** might be **accompanied** by **several wagons**.A **wagon** consists of a **dot**, **2 surrounding square brackets** and **another** **dot** (“.[].”). **Between** the **brackets**, the **wagon** may contain **ONLY letters** and **digits**.

**Valid wagons**: “.[asd].”, “.[131].”, “.[as2].”, “.[].” ... **Invalid wagons**: “[asd]”, “.[3-D].”.

On **each input line** you will **receive** a **message**. You should **check** if that message **IS** a **train**.

A **train ALWAYS** **has** a **locomotive** and **may** have **several wagons**.

A **train** **ALWAYS** **starts** at the **start** of the **message** and **ends** at its **end**.

A **train** **ALWAYS** **starts** with a **locomotive**. Example: “<[/\*\*]..[asd]..[3dx].”

The **input sequence** **ends** when you receive the command “Traincode!”. When that happens, you must print all **valid trains** you’ve **found**, **each** on a **new line**, by **order** of **input**.

### Input

* The input will come in the form of several lines containing messages.
* The input ends when you receive the command “Traincode!”.

### Output

* As output you must print **all valid trains**, **each** on a **new line**, by **order** of **input**.

### Constrains

* There will be no more than **1000 input lines**.
* The strings in the input lines may contain **any ASCII character**.
* The strings in the input lines may be **INVALID**.
* Allowed time / memory: 100ms / 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| <[/\*\*]..[asd]..[3dx]..[]..[].  <>  Traincode! | <[/\*\*]..[asd]..[3dx]..[]..[]. |
| <[{]..[7]..[]..[]..[C2I43].  <[(\_#/}$)$]..[GO5A]..[G5]..[3W4].  <[^]..[54]..[S].  <[@].  <[)$-{,]..[PB1N]..[R757G].  <[]..[]..[10]..[223F]..[GBM4].  <[!]..[]  <[)\_]..[3N]..[TS]..[0NS58].  Traincode! | <[{]..[7]..[]..[]..[C2I43].  <[(\_#/}$)$]..[GO5A]..[G5]..[3W4].  <[^]..[54]..[S].  <[@].  <[)$-{,]..[PB1N]..[R757G].  <[]..[]..[10]..[223F]..[GBM4].  <[)\_]..[3N]..[TS]..[0NS58]. |

## Anonymous Vox

The Anonymous’s main communication channel is based on encoded messages. The CIA has targetted that channel, assuming that it holds sensitive information. You have been hired to decode and break their internal com. system.

You will receive an input line containing a **single string** – the **encoded text**. Then, on the **next line** you will receive several values in the following format: “{value1}{value2}{value3}...”.

You must find the **encoded placeholders** in the **text** and **REPLACE** each one of them with the **value** that corresponds to its **index**.   
**Example**: **placeholder1 – value1**, **placeholder2 – value2** etc. There may be **more values** than **placeholders** or **more placeholders** than **values**.

The **placeholders** consist of 3 blocks {start}{placeholder}{end}. The **start** should consist only of **English alphabet letters**. The **placeholder** may contain **ANY ASCII** character. The **end** should be **EXACTLY EQUAL** to the **start**. The idea is that you have to find the **placeholders**, and **REPLACE** their placeholder block with the **value** at that **index**.

Example Placeholders: “a.....a”, “b!d!b”, “asdxxxxxasd”, “peshogoshopesho”...

You **must** **ALWAYS** match the placeholder with the **LONGEST** start and the **RIGHTMOST** end. For example if you have “asddvdasd” you should **NOT** match “dvd” as a placeholder, you should match “asddvdasd”.

At the end you must **print** the **result** **text**, after you’ve **replaced** the **values**.

### Input

* On the **first input line** you will receive the **encoded text**.
* On the **second input line** you will receive the **placeholders**.

### Output

* As output you must print a **single line** containing the **resulting text**, after the replacing of values.

### Constrains

* The **given** **text** may contain **ANY ASCII** character.
* The **given values** may contain **ANY ASCII** character except ‘{’ and ‘}’.
* The **given values** will **AWLAYS** follow the format specified above.
* Allowed working **time/memory**: **100ms / 16MB**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Hello\_mister,\_Hello  { Jack } | Hello Jack Hello |
| ASD\_\_\_asdfffasd  {this}{exam}{problem}{is}{boring} | ASD\_\_\_asdthisasd |
| Whatsup\_ddd\_sup  {Dude} | WhatsupDudesup |
| HeypalHey\_\_\_\_\_\_how\_ya\_how\_doin\_how  {first}{second} | HeyfirstHey\_\_\_\_\_\_howsecondhow |

## Hornet Comm.

The Hornet Ex-King – Horny, has established an innovative technology providing communication for his fellow hornets, called Hornet Comm. Inc. Hornet Comm. provides functionality from private messages to wide broadcasts.

You will be given data of several tracked comm. channels, which you must decrypt. The input data will come in the following format:

{firstQuery} <-> {secondQuery}

If the **first query** consists **only of digits** and the **second one** consists of **digits and / or letters**, it is a **private message**.

If the **first query** consists of **anything but digits**, and the **second one** consists of **letters and / or digits,** it is a **broadcast**.

Any input that **does** **NOT** **follow** the format, specified above, should be **IGNORED**.

If the **given data** is a **private message**, the first query is the **recipient’s code**, and the second query is the **message**. You must **reverse** the **recipient’s code** and **store** it along with the message.

If the **given data** is a **broadcast**, the first query is the message, and the second query is the **frequency**. You must take the **frequency** and make **all capital letters** – **small** and **all small letters** – **capital**. Then you must **store** it, along with the **message**.

You must **keep** all input broadcasts and messages after you **decrypt** them.

When you receive the command “**Hornet is Green**”, the input sequences **ends**, and you must print the stored broadcasts and messages.

### Input

* The input comes in the form of several input lines in the format specified above.
* The input ends when you receive the command “**Hornet is Green**”.

### Output

* As output you must print all broadcasts and messages, printing first the broadcasts, in the following format:
  + Broadcasts:
  + {frequency} -> {message}
  + . . .
  + Messages:
  + {recipient} -> {message}
  + . . .
* If there are **no messages**, or **no broadcasts**, print “**None**” in their place.

### Constrains

* The input lines may consist of **any ASCII** character.
* There will be **NO** more than 1000 lines of input.
* **All data** must be printed in **order of input**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 213094 <-> BeeQueenDown  213094 <-> Repeat  Foxtrot <-> 123321  213094 <-> BeeQueenDown  Unicorn <-> 871203  Charlie <-> 56210  Kilo <-> 423211  Hornet is Green | Broadcasts:  123321 -> Foxtrot  871203 -> Unicorn  56210 -> Charlie  423211 -> Kilo  Messages:  490312 ->BeeQueenDown  490312 ->Repeat  490312 ->BeeQueenDown |
| <+>.<+> <-> az13b6  <->.<-> <-> P2Z4x789  12345 <-> Pr1v@teM3ssage  Hornet is Green | Broadcasts:  AZ13B6 -> <+>.<+>  p2z4X789 -> <->.<->  Messages:  None |