# More Exercises: Strings and Regular Expressions

This document defines the **exercise assignments** for the ["Programming Fundamentals" course @ Software University](https://softuni.bg/courses/programming-fundamentals). Please submit your solutions (source code) of all below described problems in [Judge](https://judge.softuni.bg/Contests/585).

## Censorship

Write a program, which takes as an input a single **word** and a **sentence**. Your program should **search** for the **word** in the **sentence** and replace **every** **letter** of the word with ‘\*’. You should do that for **every** **occurrence** of the word.

### Input

The input will consist of **two lines**:

* On the **first** line, will be the **word**, which you have to **censor**.
* On the **second** line, will be the **sentence**, which you need to **censor**.

### Output

**Print** only the **censored** **sentence**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Money  Show me the money | Show me the \*\*\*\*\* |
| Doom  Doom and Gloom | \*\*\*\* and Gloom |

## Email Me

Last night Pesho received the email of a girl. Unfortunately, he cannot remember whether she was worth it. He has a plan on how to decide if he should message the girl and he needs your programming skills.

He will give you her **email** and your task is to **subtract** the **sum** of the characters **after** the ‘**@**’ from the **sum** of the characters **before** the ‘**@**’.

If the result is **equal** or **greater** **than** **0** – he will **write** her email, otherwise he will **not**.

### Input

You will receive **single** **line** with the **email** of the girl.

### Output

If the result is **equal** or **greater** **than** **0** print:

* Call her!

**Otherwise** print:

* She is not the one.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| maria@abv.bg | She is not the one. |
| gergana.ivanova@yahoo.com | Call her! |

## Karate Strings

The most notorious person in SoftUni – Pesho is trying to become a karate master. Being a programmer, Pesho has no idea how to train, so he decided to train on strings.

His **punches** are marked with ‘>’. Immediately after the mark, there will be an **integer**, which signifies the **strength** of the punch.

You should **remove** x **characters** (where x is the **strength** of the punch), **starting after** the punch **character** (‘**>**’).

If you find **another** punch mark (‘>’) while you’re deleting characters, you should **add** the **strength** to your **previous** **punch**.

When all characters are processed, **print** the string **without** the **deleted** **characters**.

You should **not** delete the **punch** character – ‘**>**’, but you should **delete** the **integers**, which represent the **strength**.

### Input

You will receive **single** **line** with the string, which is used by Pesho for training.

### Output

Print what is left from the string after Pesho’s punches.

### Constraints

* You will **always** receive a **strength** for the punches
* The path will consist only of letters from the **Latin** **alphabet**, **integers** and the char ‘**>**’
* The strength of the punches will be in the interval **[0…9]**

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comments** |
| abv>1>1>2>2asdasd | abv>>>>dasd | 1st punch is at index **3** and it is with **strength** of **1**. We delete **only** the **digit** **after** the punch character. The string will look like this: abv>>1>2>2asdasd  2nd punch is with strength **one** and the string transforms to this: abv>>>2>2asdasd  3rd punch is now with strength of 2. We delete the digit and we find **another** punch. At this point the string looks like this: abv>>>>2asdasd.  4th punch is with strength **2**. We have **1** strength **left** from the previous punch, we **add** the strength of the **current** **punch** to what is **left** and that adds up to a **total** strength of **3**. We **delete** the next **three** **characters** and we **receive** the **string** abv>>>>dasd  We do **not** have **any more punches** and we print the result: abv>>>>dasd |

|  |  |
| --- | --- |
| **Input** | **Output** |
| pesho>2sis>1a>2akarate>4hexmaster | pesho>is>a>karate>master |

## \* Morse Code Upgraded

You have written new secret way to transmit coded messages. You will receive the input in the format:

{firstLetterOfTheMessage}|{secondLetterOfTheMessage}|…|{nthLetterOfTheMessage}

Each part of the message will consist only of ‘**0**’ and ‘**1**’. Each part of the message will transform into a character from the **printable range** of the **ASCII table [32…126** **(**space…~**)]**. The transformation for each part happens in the following way:

* Each **0** adds **3** to the total sum.
* Each **1** adds **5** to the total sum.
* Every time you receive a sequence of equal digits, the sum **increases** by the **count** of the **equal digits**.

The sum should give you the **ASCII** **code** of a **character**. The final message consists of all deciphered signs.

**Example**: **10101010101010101** 🡺 The message has **nine** **ones** and **eight** **zeroes**. There are **no** **consecutive** **equal** **digits**, which means the total is **8 \* 3 + 9 \* 5 = 69** 🡺 the letter ‘**E**’.

**Example 2**: **1110011111111** 🡺 The message has **eleven** **ones** and **three** **zeroes**. This sums up to **11 \* 5 + 2 \* 3 = 61** On top of that we have **three sequences** with equal digits:

* **The first three** digits are **ones**, so we add **3** to the **sum** (the current sum equals **61 + 3 = 64**)
* **The next two** digits are **zeroes**, so we add **2** to the **sum** (the current sum equals **64 + 2 = 66**)
* **The next eight** digits are **ones**, so we add **8** to the sum (the current sum equals **66 + 8 = 74**).
* We reached the **end of the string**,and the **final ASCII code is 74** ‘**J**’.

### Input

You will receive a **single** **line** with the letters from the message. They will be separated with single pipe – ‘**|**’

### Output

Print only the deciphered message.

### Constraints

* Each **coded** **letter** will consist of either ‘**1**’ or ‘**0**’.
* The **ASCII codes** will be in the interval **[32…126]**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 111000001110000|111111110111111111 | Hi |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 01010101010101011|111001111100001111110|111001111100001111110|000011000011111010110|110010011010101011100|11110000000100110011010101|110001100101110101101 | Goodbye |

## Only Letters

Write a program which takes a **string** message as input and replaces **all** **numbers** with the **letter** immediately **after** the **number**.

### Input

You will receive a **single** **line** with the **message**, which you need to correct.

### Output

Print only the **corrected** message.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| ChangeThis12andThis56k | ChangeThisaandThiskk |

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1Beware72ForThe4End88888 | BBewareFForTheEEnd88888 |

## Email Statistics

You will receive **n** emails from the console. Some of these emails will be **invalid**. In order one email to be **valid** it should pass the following conditions:

* The **username** of the user should be at least **5** characters long and consist only of **uppercase** and **lowercase** Latin letters.
* The username should be followed immediately by ‘**@**’.
* The domain part should consist of **two** parts:
  + **The mail server**, which should contain only **lowercase** **Latin letters** and should be **at least 3** **letters** long.
  + **The top-level domain**, which can be one of the following: .com, .bg or .org

At the end, print data in the **format** described in the **output** section.

### Input

* On the **first** line, you will receive **n** – the **count** of emails.
* On the next **n** lines, you will receive **emails**.

### Output

Print the **domains** in the **format**:

|  |
| --- |
| {1st domain}:  ### {1st username}  ### {2nd username}  …  ### {nth username}  …  {nth domain}  ### {1st username}  …  ### {nth username} |

Order the **domains** by the **counts** of **usernames** in the domain in **descending order**. If they are **equal,** print them in the order, in which they were **received**.

Order the **usernames** by the time of **receiving**.

If you receive **two** of the **same** **username** for one **domain** – **ignore** it.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| **5**  Pesho@abv.bg  JohnDowe@gmail.com  Maria@gmail.com  invalid123@dir.bg  nakov@yahoo.com | gmail.com:  ### JohnDowe  ### Maria  abv.bg:  ### Pesho  yahoo.com:  ### nakov |

|  |  |
| --- | --- |
| **Input** | **Output** |
| **5**  Georgi@abv.bg  Petran@gmail.com  Vladi@gmail.com  super\_man@abv.bg  superMan@abv.bg | abv.bg:  ### Georgi  ### superMan  gmail.com:  ### Petran  ### Vladi |

## Hideout

You are a detective from Scotland Yard and you need to find the hideout of a very dangerous group of criminals. You will receive a **map** in the form of a **string** and after that you will receive **clues** from the intelligence.

On the next **unknown** amount of **lines**, you will receive **arrays** containing **two** elements:

* **The first** element will be the **character**, which **marks** the **hideout**.
* **The second element** will be the **minimum** **count** of **characters**, which you need to search.

The array will be in format: “{searchedCharacter} {minimumCount}”.

If you cannot find a hideout 🡺 continue reading the next two lines.

If you find a hideout 🡺 stop the program and print the **index** where the hideout **starts** and the **length** of the hideout.

### Input

* On the **first** line, you will receive **the map**, which will contain random strings.
* On the next **unknown** amount of lines, you will receive **arrays**
  + The first element is the searched character
  + The second element is the minimum count, which should be searched

### Output

If you find the hideout, print:

“Hideout found at index {indexOfTheFirstChar} and it is with size {lengthOfTheFoundString}!”

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| **asd@@asdasd@@@@@@@asdasd asdsad**  @ 5 | Hideout found at index 11 and it is with size 7! |

|  |  |
| --- | --- |
| **Input** | **Output** |
| **asd@@asd\*\*\*asdasdsad123%4521Asdsad\*\*\*\*\*\*\*\*\*\*\*\*ASssda**  & 3  \* 20  \* 10  \* 2 | Hideout found at index 34 and it is with size 12! |

## \* Mines

You have the very prosperous and modern profession of mine examiner. The problem is that your job is quite dangerous, so you decided to write a program, which calculates the power of the mines.

The **mines** will be in format <{firstCharacter}{secondCharacter}>. When you encounter a **mine**, you should destroy **all** its **characters**. The **mine** also destroys **n** characters to the **left** and **right** of itself. **n** is determined by the **absolute** value of the **subtraction** of the **ASCII** codesof the **first** character and the **second** characters. **Replace** the **destroyed** characters with underscores – ‘\_’.

Example: we received the following string:

bewareOf<AF>TheMines

The mine is <AF>. The power of the mine will equal |A (65) – F (70)| = 5. When the mine explodes, we have the following string:

bew\_\_\_\_\_\_\_\_\_\_\_\_\_\_nes

*(Legend: red – mine, green ­– blast radius)*

### Input

You will receive single line with the string, which you need to check for mines.

### Output

Print the string after the explosions.

### Constraints

* The lengthof the text will be in the range [1...500].
* Mine explosions will not overlap with other bombs.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| bewareOf<AF>TheMines | bew\_\_\_\_\_\_\_\_\_\_\_\_\_\_nes |

|  |  |
| --- | --- |
| **Input** | **Output** |
| **TwoMin<ag>esWillBeHe<HH>reMuchDangerous** | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_BeHe\_\_\_\_reMuchDangerous |