

# Exercise: Strings and Text Processing

Problems for exercise and homework for the ["C# Fundamentals" course @ SoftUni](#)

You can check your solutions in [Judge](#)

## 1. Valid Usernames

Create a program that **reads user** names on a **single** line (joined by " , ") and **prints** all **valid usernames**.

A valid username

- has **length** between 3 and 16 characters
- **contains** only **letters**, **numbers**, **hyphens** and **underscores**

### Examples

| Input  | Output                         |
|--|--------------------------------|
| sh, too_long_username, !lleg@l ch@rs, jeffbutt | jeffbutt                       |
| Jeff, john45, ab, cd, peter-ivanov, @smith     | Jeff<br>John45<br>peter-ivanov |

## 2. Character Multiplier

Create a **method** that takes **two strings** as arguments and returns the **sum** of their **character codes multiplied**. Multiply **str1[0]** with **str2[0]** and add to the total sum. Then continue with the next two characters. If one of the strings is **longer** than the other, **add** the **remaining** character codes to the **total sum** without multiplication.

### Examples

| Input        | Output |
|--------------|--------|
| Peter George | 52114  |
| 123 522      | 7647   |
| a aaaa       | 9700   |

## 3. Extract File

Create a program that reads the path to a file and **subtracts** the **file name** and its **extension**.

### Examples

| Input                                       | Output                                      |
|---|---|
| C:\Internal\training-internal\Template.pptx | File name: Template<br>File extension: pptx |
| C:\Projects\Data-Structures\LinkedList.cs   | File name: LinkedList<br>File extension: cs |

## 4. Caesar Cipher

Create a program that returns an **encrypted version** of the same text. Encrypt the text by **shifting each character** with **three positions forward**. For example, **A** would be replaced by **D**, **B** would become **E** and so on. Print the **encrypted text**.

### Examples

| Input                  | Output                 |
|------------------------|------------------------|
| Programming is cool!   | Surjudpplqj#lv#frro\$  |
| One year has 365 days. | Rqh# hdu#kdv#698#gd v1 |

## 5. Multiply Big Number

You are given **two lines** – the **first** one can be a really **big** number (**0 to  $10^{50}$** ). The **second** one will be a **single-digit** number (**0 to 9**). Your task is to display the product of these numbers.

Note: do not use the **BigInteger** class.

### Examples

| Input                            | Output                       |
|----------------------------------|------------------------------|
| 23<br>2                          | 46                           |
| 9999<br>9                        | 89991                        |
| 923847238931983192462832102<br>4 | 3695388955727932769851328408 |

## 6. Replace Repeating Chars

Create a program that reads a string from the console and **replaces** any **sequence of the same letter** with a **single corresponding letter**.

### Examples

| Input                    | Output     |
|--------------------------|------------|
| aaaaabbbbbccdddeeeedssaa | abcdedsa   |
| qqqwerqweccwd            | qwerqwecwd |

## 7. String Explosion

Explosions are marked with '**>**'. Immediately after the mark, there will be an **integer**, which signifies the **strength** of the explosion.

You should **remove x characters** (where **x** is the **strength** of the explosion), **starting after** the punched character ('>').

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

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

You should **not** delete the **explosion** character – '>', but you should **delete** the **integers**, which represent the **strength**.

## Input

You will receive a **single line** with the string.

## Output

Print what is left from the string after the explosions.

## Constraints

- You will **always** receive **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              | <p><b>1<sup>st</sup></b> explosion is at index <b>3</b> and it is with a <b>strength</b> of <b>1</b>. We delete <b>only</b> the <b>digit</b> after the explosion character. The string will look like this: <b>abv&gt;&gt;1&gt;2&gt;2asdasd</b></p> <p><b>2<sup>nd</sup></b> explosion is with strength <b>one</b> and the string transforms to this: <b>abv&gt;&gt;&gt;2&gt;2asdasd</b></p> <p><b>3<sup>rd</sup></b> explosion is now with a strength of <b>2</b>. We delete the digit and we find <b>another</b> explosion. At this point, the string looks like this: <b>abv&gt;&gt;&gt;&gt;2asdasd</b>.</p> <p><b>4<sup>th</sup></b> explosion is with strength <b>2</b>. We have <b>1</b> strength <b>left</b> from the previous explosion, we <b>add</b> the strength of the <b>current</b> explosion to what is <b>left</b>, and that adds up to a <b>total</b> strength of <b>3</b>. We <b>delete</b> the next <b>three characters</b> and we <b>receive</b> the string <b>abv&gt;&gt;&gt;&gt;&gt;dasd</b></p> <p>We do <b>not</b> have <b>any more explosions</b> and we print the result: <b>abv&gt;&gt;&gt;&gt;&gt;dasd</b></p> |
| peter>2sis>1a>2akarate>4hexmaster | peter>is>a>karate>master |  |

## 8. \*Letters Change Numbers

Nakov likes Math. But he also likes the English alphabet. He invented a game with numbers and letters from the **English** alphabet. The game was simple. You get a string consisting of a **number between two letters**. Depending on whether the letter was in front of the number or after it, you would perform different mathematical operations on the number to achieve the result.

**First**, you start with the letter **before** the number.

- If it's **uppercase** you **divide** the number by the letter's **position** in the alphabet.
- If it's **lowercase** you **multiply** the number with the letter's **position** in the alphabet.

**Then** you move to the **letter after** the number.

- If it's **uppercase** you **subtract** its position from the resulted number.
- If it's **lowercase** you **add** its position to the resulted number.

But the game became too easy for Nakov. He decided to complicate it a bit by doing the same but with **multiple** strings keeping track of only the **total sum** of all results. Once he started to solve this with more strings and bigger numbers, it became quite hard to do it only in his mind. So he kindly asks you to write a program that **calculates the sum of all numbers after the operations on each number have been done**.

**For example**, you are given the sequence "**A12b s17G**":

We have two strings – "**A12b**" and "**s17G**". We do the operations on each and sum them. We start with the letter before the number on the first string. **A is Uppercase** and its position in the alphabet is **1**. So we divide the number 12 with position 1 (**12/1 = 12**). Then we move to the letter after the number. **b is lowercase** and its position is 2. So we add 2 to the resulting number (**12+2=14**). Similarly for the second string **s is lowercase** and its position is 19 so we multiply it with the number (**17\*19 = 323**). Then we have Uppercase G with position 7, so we subtract it from the resulted number (**323 – 7 = 316**). Finally, we sum the 2 results and we get **14 + 316=330**.

### Input

The input comes from the console as a **single line, holding the sequence of strings**. Strings are separated by **one or more white spaces**.

The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

Print at the console a single number: the **total sum of all processed numbers** rounded up to **two digits** after the decimal separator.

### Constraints

- The **count** of the strings will be in the range [1...10].
- The numbers between the letters will be integers in the range [1...2147483647].
- Time limit: 0.3 sec. Memory limit: 16 MB.

### Examples

| Input     | Output | Comments   |
|-----------|--------|--|
| A12b s17G | 330.00 | 12/1=12, 12+2=14, 17*19=323, 323-7=316, 14+316=330 |

|                      |          |  |
|----------------------|----------|--|
| P34562Z q2576f H456z | 46015.13 |  |
| a1A                  | 0.00     |  |