Exercise: Text Processing

Problems for exercise and homework for the Python Fundamentals Course @SoftUni.
Submit your solutions in the SoftUni judge system at
https://judge.softuni.bg/Contests/1740

1. Valid Usernames

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

A valid username is:

- Has **length** between 3 and 16 characters
- **Contains** only letters, numbers, hyphens and underscores
- Has **no redundant symbols** before, after or in between

Examples

Input	Output
`sh, too_long_username, !lleg@l ch@rs, jeffbutt`	`jeffbutt`

`Jeff, john45, ab, cd, peter-ivanov, @smith`	`Jeff`
	`John45`
	`peter-ivanov`

2. Character Multiplier

Create a program that receives **two strings** on a **single lines** separated by **space** and prints 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
`George Peter`	`52114`
`123 522`	`7647`
`a aaaa`	`9700`

3. Extract File

Write 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 File extension: pptx
C:\Projects\Data-Structures\LinkedList.cs	File name: LinkedList File extension: cs

4. Caesar Cipher

Write 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. Emoticon Finder

Find all emoticons in the text. An emoticon **always starts with ":"** and is followed by a **symbol.**
The input will be provided as a **single string.**

Example

Input	Output
There are so many emoticons nowadays :P. I have many ideas :0 what input to place here :)	:P :0 :)

6. Replace Repeating Chars

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

Examples

Input	Output
aaaaabbbbbcdddeeeedssaa	abcdedsa
qqqwerqwecccwd	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 punch **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 **single line** with the string.

Output

Print what is left from the string after explosions.

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 explosion is at index **3** and it is with **strength** of **1**. We delete **only** the **digit after** the explosion character. The string will look like this: **abv>>1>2>2asdasd** 2nd explosion is with strength **one** and the string transforms to this: **abv>>>2>2asdasd** 3rd explosion is now with strength of 2. We delete the digit and we find **another** explosion. At this point the string looks like this: **abv>>>>2asdasd**. 4th explosion is with strength **2**. We have **1** strength **left** from the previous explosion, we **add** the strength of the **current** explosion 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 explosions** and we print the result: **abv>>>>dasd**
pesho>2sis>1a>2akarate>4hexmaster	pesho>is>a>karate>master	

8. *Letters Change Numbers

Nakov likes Math. But he also likes the English alphabet a lot. 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 really quick. 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 the 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 resulted 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 range `[1 … 2 147 483 647]`.
- 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`	

9. *Rage Quit

Every gamer knows what rage-quitting means. It's basically when you're just not good enough and you blame everybody else for losing a game. You press the CAPS LOCK key on the keyboard and flood the chat with gibberish to show your frustration.

Chochko is a gamer, and a bad one at that. He asks for your help; he wants to be the most annoying kid in his team, so when he rage-quits he wants something truly spectacular. He'll give you **a series of strings followed by non-negative numbers**, e.g. **"a3"**; you need to print on the console **each string repeated N times**; **convert the letters to uppercase beforehand**. In the example, you need to write back **"AAA"**.

On the output, print first a statistic of the **number of unique symbols** used (the casing of letters is irrelevant, meaning that **'a'** and **'A'** are the same); the format shoud be **"Unique symbols used {0}"**. Then, **print the rage message** itself.

The **strings and numbers will not be separated by anything**. The input will always start with a string and for each string there will be a corresponding number. The entire input will be given on a **single line**; Chochko is too lazy to make your job easier.

Input

- The input data should be read from the console.
- It consists of a single line holding a series of **string-number sequences**.
- The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

- The output should be printed on the console. It should consist of **exactly two lines**.
- On the first line, print the **number of unique symbols used** in the message.
- On the second line, print the **resulting rage message** itself.

Constraints

- The count of **string-number pairs** will be in the range [1 … 20 000].
- Each string will contain any character **except digits**. The **length** of each string will be in the range [1 … 20].

- The **repeat count** for each string will be an integer in the range [0 … 20].
- Allowed working time for your program: 0.3 seconds. Allowed memory: 64 MB.

Examples

Input	Output	Comments
a3	Unique symbols used: 1 AAA	We have just one string-number pair. The symbol is 'a', convert it to uppercase and repeat 3 times: AAA. Only one symbol is used ('A').
aSd2&5s@1	Unique symbols used: 5 ASDASD&&&&&S@	"aSd" is converted to "ASD" and repeated twice; "&" is repeated 5 times; "s@" is converted to "S@" and repeated once. 5 symbols are used: 'A', 'S', 'D', '&' and '@'.

10. *Winning Ticket

Lottery is exciting. What is not, is checking a million tickets for winnings only by hand. So, you are given the task to create a program which automatically checks if a ticket is a winner.

You are given a **collection of tickets separated by commas and spaces**. You need to check every one of them if it has a winning combination of symbols.

A valid ticket should have exactly 20 characters. The winning symbols are '@', '#', '$' and '^'. But in order for a ticket to be a winner the symbol should uninterruptedly repeat for at least **6 times** in both the **tickets left half** and the **tickets right half**.

For example, a valid winning ticket should be something like this:

"Cash$$$$$$Ca$$$$$$sh"

The left half **"Cash$$$$$$"** contains **"$$$$$$"**, which is also contained in the tickets right half **"Ca$$$$$$sh"**. A winning ticket should contain symbols repeating up to 10 times in both halves, which is considered a Jackpot (for example: **"$$$$$$$$$$$$$$$$$$$$"**).

Input

The input will be read from the console. The input consists of a **single line** containing all tickets **separated by commas and one or more white spaces** in the format:

- **"{ticket}, {ticket}, … {ticket}"**

Output

Print the result for every ticket in the order of their appearance, each on a separate line in the format:

- **Invalid ticket - "invalid ticket"**
- **No match - "ticket "{ticket}" - no match"**
- **Match with length 6 to 9 - "ticket "{ticket}" - {match length}{match symbol}"**
- **Match with length 10 - "ticket "{ticket}" - {match length}{match symbol} Jackpot!"**

Constrains

- Number of tickets will be in range [0 … 100]

Examples

Input	Output
`Cash$$$$$$Ca$$$$$$sh`	`ticket "Cash$$$$$$Ca$$$$$$sh" - 6$`

`$$$$$$$$$$$$$$$$$$$$, aabb , th@@@@@@eemo@@@@@@ey`	`ticket "$$$$$$$$$$$$$$$$$$$$" - 10$`
	`Jackpot!`
	`invalid ticket`
	`ticket "th@@@@@@eemo@@@@@@ey" - 6@`

`validticketnomatch:(`	`ticket "validticketnomatch:(" - no match`