Exercises: Strings and Text Processing

This document defines the **exercise assignments** for the "Programming Fundamentals" course @ Software University. Please submit your solutions (source code) of all below described problems in Judge.

Problem 1. Convert from Base-10 to Base-N

Write a program that takes a base-10 number (0 to 10^{50}) and converts it to a base-N number, where 2 <= N <= 10. The input consists of 1 line containing two numbers separated by a single space. The first number is the base N to which you should convert. The second one is the base 10 number to be converted. **Do not use any built in converting functionality, try to write your own algorithm.**

Hints

About the algorithm (from base-10 to base-2) you can read this article.

The algorithm for converting from base-10 to base-N is similar: instead of "**% 2**", use "**% N**".

Examples

Base-10	Base-N
7 10	13
3 154	12201
5 123	443
4 1000	33220
9 3487	4704

Problem 2. Convert from Base-N to Base-10

Write a program that takes a base-N number and converts it to a base-10 number (0 to 10^{50}), where 2 <= N <= 10. The input consists of 1 line containing two numbers separated by a single space. The first number is the base N to which you have to convert. The second one is the base N number to be converted. **Do not use any built in converting functionality, try to write your own algorithm.**

Hints

See this picture for more clarity about base-2 to base-10. Again, the algorithm for N-base is similar.

Examples

Base-N	Base-10
7 13	10
3 12201	154
5 443	123
4 33220	1000
9 4704	3487

Problem 3. Unicode Characters

Write a program that converts a string to a sequence of Unicode character literals.

Examples

Input	Output
`Hi!`	`\u0048\u0069\u0021`
`What?!?`	`\0057\0068\0061\0074\003f\0021\003f`
`SoftUni`	`\0053\006f\0066\0074\0055\006e\0069`

Problem 4. Character Multiplier

Create a **method** that takes two strings as arguments and returns the sum of their character codes multiplied (multiply str1.charAt (0) with str2.charAt (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
`Gosho Pesho`	`53253`
`123 522`	`7647`
`a aaaa`	`9700`

Problem 5. Magic Exchangeable Words

Write a **method** that takes as input two strings, and returns Boolean if they are exchangeable or not. Exchangeable are words where the characters in the first string can be replaced to get the second string. Example: "**egg**" and "**add**" are exchangeable, but "**aabbccbb**" and "**nnooppzz**" are not. (First '**b**' corresponds to '**o**', but then it also corresponds to '**z**'). The two words may not have the same length, if such is the case they are exchangeable only if the longer one doesn't have more types of characters then the shorter one ("**`Clint`**" and "**`Eastwaat`**" are exchangeable because '**a**' and '**t**' are already mapped as '**l**' and '**n**', but "**`Clint`**" and **"`Eastwood`**" aren't exchangeable because '**o**' and '**d**' are not contained in "**Clint**").

Examples

Input	Output
`gosho hapka`	`true`
`aabbaa ddeedd`	`true`
`foo bar`	`false`
`Clint Eastwood`	`false`

Problem 6. Sum Big Numbers

You are given two lines - each can be a really big number (0 to 10^{50}). You must display the sum of these numbers.

Note: do not use the **`BigInteger`** or **`BigDecimal`** classes for solving this problem.

Examples

Input	Output
23 23	46

Input	Output
9999 1	10000

Input	Output
923847238931983192462832102 934572893617836459843471846187346	934573817465075391826664309019448

Problem 7. 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). You must display the product of these numbers.

Note: do not use the **`BigInteger`** or **`BigDecimal`** classes for solving this problem.

Examples

Input	Output
23 2	46

Input	Output
9999 9	89991

Input	Output
923847238931983192462832102 4	93457381746507539182666430 9019448

Problem 8. *Letters Change Numbers

This problem is from the Java Basics exam (8 February 2015). You may check your solution here.

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	Comment
`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`	

Problem 9. ** Melrah Shake

You are given a **string** of random characters, and a **pattern** of random characters. You need to “shake off” (**remove**) all of the **border** occurrences of that pattern, in other words, the **very first match** and the **very last match** of the pattern you find in the string.

When you successfully shake off a match, you **remove** from the pattern the character which corresponds to the **index** equal to **the pattern’s length / 2**. Then you continue to shake off the border occurrences of the new pattern until the pattern becomes **empty** or until there is **less** than the - needed for shake, matches in the remaining string.

In case you have found at least **two** matches, and you have successfully shaken them off, you print “**`Shaked it.`**” on the console. Otherwise you print “No shake.”, the remains of the main string, and you end the program. See the examples for more info.

Input

- The input will consist only of two lines.
- On the first line, you will get a string of random characters.
- On the second line, you will receive the pattern and that ends the input sequence.

Output

- You must print “**`Shaked it.`**” for every time you successfully do the melrah shake.

- If the melrah shake fails, you print "**`No shake.`**", and on the next line you print what has remained of the main string.

Constraints

- The two strings may contain **ANY** ASCII character.
- Allowed time/memory: 250ms/16MB.

Examples

Input	Output

`astalavista baby`
`sta`
	`Shaked it.`
	`No shake.`
	`alavi baby`

Input	Output

`##mtm!!mm.mm*mtm.#`
`mtm`
	`Shaked it.`
	`Shaked it.`
	`No shake.`
	`##!!.*.#`