A: Raindrops Keep Falling on our Sensors

Input file: a.in
Output: Standard output

The University of Chicago Weather Service has purchased a *new* rainfall sensor. This sensor takes measurements every few seconds, which can be handed off to a computer for further analysis. Each measurement is a non-negative integer representing how many millimeters of rainfall have been recorded since the previous measurement.

Given this information, we would like to compute the average rainfall, which is simply the average of the non-negative integers produced by the sensor. However, there's one catch: this new sensor occasionally produces *negative* integers representing faulty measurements. These have to be discarded and shouldn't be taken into account.

Input
The input contains several datasets with measurements from the sensor. Each dataset is represented by two lines. The first line contains a single positive integer $n < 100$ that specifies the number of measurements in this data set. The second line contains the measurements, each separated by a single space.

The end of the input is indicated by a line with a single integer: 0.

Output
For each dataset in the input, you must print a line with a single integer: the average rainfall for that dataset (*rounded down*; i.e., only the integer part of the average), taking into account that negative measurements should be discarded.

If a dataset contains only negative measurements, then you must print the word `NONE`

Sample Input	Sample Output

3
5 10 15
5
14 -5 39 -5 7
0
	10
	20

This page is intentionally left blank.

B: The Apaxians Strike Back

Input file: b.in
Output: Standard output

Researchers at the Oriental Institute have spent all summer analyzing a collection of Apaxian scrolls unearthed, of all places, on the shores of Lake Michigan. Once again, the Oriental Institute has turned to the Department of Computer Science to analyze the ancients texts of this venerable civilization that we totally did not make up.

More specifically, the linguists and archaeologists of the Oriental Institute have discovered a twist in the Apaxian naming system. It seems that, in Apaxian society, the first letter in your name, and how many times that letter appears in your name, was crucial in determining social status. Thus, an Apaxian named "axeziolotenimos" would have belonged to the lower castes of society (the letter "a" only appears once in his name), while "bobabeeboobabibobuboo" was renowned and admired throughout the entire Apaxian Empire (the first letter of his name, "b", appears several times in his name).

The Oriental Institute has sent us a list of common Apaxian names, and has asked us to determine how important the name is, based on how many times the first letter of the name appears in the name *(including the first letter).* For example, for "alexitas", the letter "a" appears 2 times. For "bobabeeboobabibobuboo", the letter "b" appears 9 times.

Input
The input contains a list of names, one per line. Each name contains only lowercase letters, no whitespace, and has a maximum length of 50 characters. The end of the input is denoted by a line with a single string: "END"

Output
For each name in the input, the output contains a line with a single integer: the number of times the first letter of the name appears in the name *(including the first letter)*.

Sample Input	**Sample Output**
`axeziolotenimos`	1
`alexitas`	2
`bobabeeboobabibobuboo`	9
`END`	

This page is intentionally left blank.

C: Dijkstra's *Other* Algorithm

Input file: c.in
Output: Standard output

You've probably heard of Dijkstra's algorithm for finding the shortest path in graphs (if you haven't, don't worry, you'll learn about it in the Masters Program in Computer Science). This exercise isn't about *that* algorithm, but about a different algorithm that Dijkstra came up with to compute the Greatest Common Divisor (GCD) of two numbers (i.e., the largest positive integer that divides two numbers with the remainder being zero).

Dijkstra's Other Algorithm is a simple and elegant variation on Euclid's classic algorithm for finding the GCD, but it only works if both numbers, *a* and *b*, are larger than zero (Euclid's Algorithm doesn't have that requirement). The GCD is defined recursively like this:

$$gcd(a,b)=\begin{cases} a & \text{if } a=b \\ gcd(a-b,b) & \text{if } a>b \\ gcd(a,b-a) & \text{if } a<b \end{cases}$$

Input
Each line contains a pair of non-zero positive integers separated by a single space. The end of the input is indicated by a line with two zeros separated by a space.

Output
For each pair of numbers, you must print a single integer: the GCD, computed using the method described above. *You* must *use a recursive solution*.

Sample Input	Sample Output
12 15	3
5 15	5
84 268	4
0 0	

This page is intentionally left blank.

D: Reverse Minesweeper

Input file: d.in
Output: Standard output

Minesweeper is a popular game requiring skill and a little bit of luck. If you're not familiar with it, it involves a *minefield* with *x* rows and *y* columns. Each position of the field can have a mine or not. For example, this minefield has two mines:

	●		
			●

However, the player cannot see where the mines are. Instead, she must select a position of the minefield; if the position has a mine, the game is over. If not, a number is revealed. This number indicates how many mines are present in the positions immediately adjacent to that position (including the diagonal ones). For example, the above board would contain the following numbers:

1	1	1	0
1	●	1	0
1	1	2	1
0	0	1	●

In a game of minesweeper, you would use those numbers to figure out where the mines are. However, you will be writing a program to perform a different task that requires no guesswork. You will be given the specification of a minefield, and the location of all the mines. Based on this information, you will need to compute the *solved minefield* with all the correct numbers in the positions that do not contain mines.

Input
The input contains the specification of several minefields. Each minefield starts with a line with two non-zero positive integers, *x* and *y*, that specify the dimensions of the minefield (*x* is the number of rows, and *y* is the number of columns; both are less than or equal to 50). This is followed by *x* lines, each corresponding to a line of the minefield. Each of these lines has *y* integers, each separated by a single space, corresponding to each column of that row. A one ("1") denotes that there is a mine in that position, while a zero ("0") denotes that there is no mine in that position.

Output
For each minefield, print the solved minefield. The solved minefield must be composed of *x* lines, and each line must contain *y* characters. If the solved position is a zero, print a hyphen ("-"). If the solved position is a non-zero number, print the number. If the solved position is a mine, print an uppercase x ("X").

Each solved minefield is separated by an empty line.

Note that, for the sample data presented below, the first minefield corresponds to the minefield shown in the figures above.

Input	Output
4 4	111-
0 0 0 0	1X1-
0 1 0 0	1121
0 0 0 0	--1X
0 0 0 1	
4 6	----1X
0 0 0 0 0 1	111-11
0 0 0 0 0 0	2X1---
0 1 0 0 0 0	X21---
1 0 0 0 0 0	
0 0	