Malware evolution (`evolution`)

From our last virus scan we have found that unfortunately our systems are infected with a last generation malware that expands itself in memory.

For the problem, our memory can be considered as a bidimensional grid of N rows and M columns, where each cell can be of three types:

- Empty cell (symbol `.`) that contains no data.
- Data cell (symbol `+`) that contains valid information.
- Malware cell (symbol `*`) that contains part of the malware.

Each cell on the grid, after someone run a software on the system (for simplicity called execution **round**), can update its value according to the number of non-empty cells in its neighborhood (in the 8 possible directions, or less if a border cell) and in particular:

- An empty cell that has more than 4 non-empty cells become a data cell.
- A data cell that has more than 4 non-empty cells become a malware cell.
- A data cell that has less than 4 non-empty cells become an empty cell.
- A malware cell that has more than 4 non-empty cells become a data cell.
- A malware cell that has less than 4 non-empty cells become an empty cell.

Write a program that given the initial configuration of a memory finds its final configuration after k rounds.

Input data

Among the attachments you can find templates for each available language that already correctly implements the management of input and output, use them!

Your program must read the input data from the standard input.

The first line of the input contains three space-separated integer **N**, **M** and **K**, respectively: the number of rows of the memory, the number of columns of the memory and the number of rounds to simulate.

The following **N** lines will contains a string of **M** characters representing the initial configuration of the memory.

Output data

Among the attachments you can find templates for each available language that already correctly implements the management of input and output, use them!

Your program must write the outut data into the standard output.

The output must contains **N** lines will contains a string of M characters representing the final configuration of the memory after **K** round.

Scoring

For each of the test cases the program will be tested, the following constraints are met:

- $K = 1$ and $N, M \leq 10$ for at least 25% of all the test cases.
- $K \leq 10$ and $N, M \leq 50$ for at least 50% of all the test cases.
- $K \leq 20$ and $N, M \leq 100$ for all the test cases.

Examples

input	output
5 4 2 +**. .+** *..+ +*+. **..	.... .+*. .*+. .+.. ....

Explanation

In the example, the evolution of the memory from the round 0 to the round 2 is:

[+**.] → [.**.] → [....]
[.+**] → [.*+.] → [.+*.]
[*..+] → [.++.] → [.*+.]
[+*+.] → [++..] → [.+..]
[**..] → [.*..] → [....]