C++ Fundamentals: Exam 2

The following tasks should be submitted to the SoftUni Judge system, which will be open starting **Sunday, 29 July 2018, 09:00** (in the morning) and will close on **Sunday, 29 July 2018, 15:00**. Submit your solutions here: https://judge.softuni.bg/Contests/Compete/Index/1117.

For this exam, the code for each task should be a single C++ file, the contents of which you copy-paste into the Judge system.

Please be mindful of the strict input and output requirements for each task, as well as any additional requirements on running time, used memory, etc., as the tasks are evaluated automatically and not following the requirements strictly may result in your program's output being evaluated as incorrect, even if the program's logic is mostly correct.

You can use C++03 and C++11 features in your code.

Unless explicitly stated, any integer input fits into **`int`** and any floating-point input can be stored in **`double`**. On the Judge system, a C++ **`int`** is a **`32-bit`** signed integer and a C++ **`double`** is a **`64-bit`** IEEE754 floating point number.

NOTE: the tasks here are NOT ordered by difficulty level.

Task 2 – Glitches (Exam-2-Task-2-Glitches)

The agents in the Matrix have noticed a lot more glitches have been happening. In addition to that, each **glitch seems to propagate to the adjacent cells** (up, down, left, right), and then **those cells propagate the glitch to their adjacent cells** and so on.

So, the **glitches** in the Matrix are **symmetrical both horizontally and vertically** and their height and width (as numbers of cells affected) are odd numbers. Another way to describe the glitches is to consider that each glitch has a center, and each glitch has a radius (in number of cells) which is an even number, and each cell that is at a Manhattan distance equal or less than the radius is part of the glitch. Manhattan distance in this case is calculated as the sum of the difference by row and by column – e.g. the Manhattan distance between cell `[2][7]` and cell `[4][1]` is `(4 - 2) + (7 - 1) = 8`.

Additionally, **each glitch has a different signature** than the others – i.e. the cells in one glitch are differently affected than those in another glitch.

The agents could of course write code to clear the glitches, but they suspect that Neo and his friends are causing the glitches and they think they can catch them by looking for the center of each glitch. So, they want code that clears the glitches but leaves their centers, so that they can look for Neo in those centers.

Write a program that, when given a **square matrix** with glitches, **clears everything in it except the centers of the glitches**, and prints the resulting matrix.

Input

The first line of the standard input will contain a single positive integer number **S** – the **size**, i.e. the **number** of **rows** and **columns** in the square matrix.

On each of the next S lines there will be exactly S characters, representing the cells in each row of the matrix. **Normal** (unaffected by glitches) **cells** will be described by the `'.'` (dot) symbol, and **glitched cells** will be described by one of the following symbols: `!?@#$%^&*()_+-=[]{}|:`

Output

Exactly **S** lines, each containing exactly **S** characters representing the matrix, where **each normal cell remains normal** as in the input, and **every glitched cell** – **except the centers** of the glitches – is **changed to a normal cell**.

Restrictions

`1 <= S <= 100`.

No two glitches will have the same symbol.

There will be **at least 1 normal cell** horizontally and vertically **between two glitches**.

There will be at least **1** glitch in the matrix.

In **50%** of the tests there will be exactly **1** glitch.

The total running time of your program should be no more than **0.1s**

The total memory allowed for use by your program is **16MB**

Hints

Notice that the topmost affected cell of each glitch is exactly above the center, and the bottom is exactly below the center, and that the center is exactly between (the same distance from) the top and the bottom. Analogously for the leftmost and rightmost cell.

Another way to look for the center is to consider that the center of each glitch is at the average of all of the coordinates of the glitch.

Example I/O

Example Input	Expected Output
7	....
....	....
..?....	..?....
.???...	....
..?....	....
....	....
....	....
....	
14	....
....	....
.......?......	....
......???.....	....
.....?????....	....
....???????...	.......?......
...?????????..	....
....???????...	....
.....?????....	....
...*..???..#..	...........#..
..***..?..###.	...*..........
.*****.....#..	....
..***.........	....
...*..........	....
....	
14	....
....	....
.......?......	....
......???.....	....
.....?????....	....
....???????...	.......?......

```
...?????????..
....???????...
.....?????....
....*.???..#..
...***.?..###.
..*****....#..
...***........
....*.........
..............
```

```
..............
..............
..............
...........#..
....*.........
..............
..............
..............
```