puzzle 100 points

Source code: **puzzle.c, puzzle.cpp, puzzle.pas**
Input files: **puzzle.in**
Output files: **puzzle.out**
Time limit: **0.5 s**
Memory limit: **64 MB**

Little P is very fond of the chips produced by Company X because they are crisp, very tasty and with each package he now increases his chances of winning the big prize at the end-of-year lottery. To take part in this lottery, all he has to do is collect the pieces of puzzle found in the packages. Whenever he finds four pieces that can be assembled to form a square (the square representing the assembled puzzle), he is to stick them to a sheet of paper, label them "Chips forever" and send them to Company X.

The puzzle pieces are made from cardboard, both sides having the same colour, and can be used in 8 distinct ways (in their current position or rotated by `90°,180°,270°` on both sides).

The initial pieces were `n` by `n` squares. From each piece we chose `2` adjacent sides and we cut out some small squares (`1` by `1`). We cut at least one small square from each chosen side, but also leave at least one small square on each chosen side.

When assembled together they form a `2n–1` by `2n–1` square.

Each piece is given a number from `1` to `4`. All the elements for a given piece have the same number, except for the cut out squares that are `0`.

This is the last puzzle Little P was able to assemble:

1	1	1	1
1	1	1	0
1	1	1	1
1	1	0	1

2	2	2	2
2	2	2	0
2	2	2	0
0	2	2	0

0	3	0	0
3	3	3	0
3	3	3	0
3	3	3	3

4	4	4	0
4	4	4	4
4	4	4	4
0	0	4	0

He put these pieces together to form a `7` by `7` square as illustrated in the figure below.

1	1	1	1	3	3	3
1	1	1	3	3	3	3
1	1	1	1	3	3	3
1	1	4	1	2	2	3
4	4	4	4	2	2	2
4	4	4	4	2	2	2
4	4	4	2	2	2	2

Little P needs a program to help him assemble the puzzle.

Task

You are given the four `n` by `n` puzzle pieces in order. They can be rotated and mirrored.

Your task is to assemble these four pieces to form a `2n-1` by `2n-1` square so that the pieces fit perfectly without overlapping and without any spaces between the pieces.

Description of input

The `puzzle.in` file contains the integer `n` on the first line, representing the size of the puzzle pieces. The next lines describe the `4` puzzle pieces in order.

A puzzle piece is represented by `n` lines. Each line contains `n` digits separated by one space.

A blank line separates the puzzle pieces.

Description of output

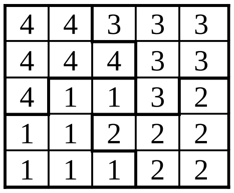
The `puzzle.out` file will contain `2n-1` lines with `2n-1` digits separated by one space that will represent an assembled puzzle.

Restrictions and remarks

- `3 ≤ n ≤ 20`
- In all the test cases the puzzle can be solved.
- Any correct solution will be accepted.
- For `30%` of the tests, the pieces do not need any operations (rotation, mirroring)
- For another `40%` of the tests, at least one piece will fit only after being rotated by `90`°, `180`°, `270`°.
- For another `30%` of the tests, the pieces will be assembled using rotations on both sides.

Examples

puzzle.in	puzzle.out	Explanation

3
0 1 1
1 1 0
1 1 1
 	
0 0 2
2 2 2
0 2 2
 	
3 3 3
0 3 3
0 3 0
 	
4 4 0
4 4 4
4 0 0
	4 4 3 3 3
	4 4 4 3 3
	4 1 1 3 2
	1 1 2 2 2
	1 1 1 2 2
		The puzzle assembled in `puzzle.out` contains all the pieces without any rotations or mirroring.
		
		4 4 3 3 3
		4 4 4 3 3
		4 1 1 3 2
		1 1 2 2 2
		1 1 1 2 2

<table>
<tr><th>puzzle.in</th><th>puzzle.out</th><th>Explanation</th></tr>
<tr><td rowspan="2">4
1 1 1 1
1 1 1 0
1 1 1 1
1 1 0 1

2 2 2 2
2 2 2 0
2 2 2 0
0 2 2 0

0 3 0 0
3 3 3 0
3 3 3 0
3 3 3 3

4 4 4 0
4 4 4 4
4 4 4 4
0 0 4 0</td><td>1 1 1 1 3 3 3
1 1 1 3 3 3 3
1 1 1 1 3 3 3
1 1 4 1 2 2 3
4 4 4 4 2 2 2
4 4 4 4 2 2 2
4 4 4 2 2 2 2</td><td>The puzzle assembled in puzzle.out, given in left, contains
• piece 1 – used without rotation or mirroring;
• piece 2 – rotated by 180°;
• piece 3 – mirrored and rotated;
• piece 4 – mirrored.</td></tr>
<tr><td colspan="2">Another correct solution is:
3 3 3 1 1 1 1
3 3 3 3 1 1 1
3 3 3 1 1 1 1
3 2 2 1 4 1 1
2 2 2 4 4 4 4
2 2 2 4 4 4 4
2 2 2 2 4 4 4</td></tr>
</table>