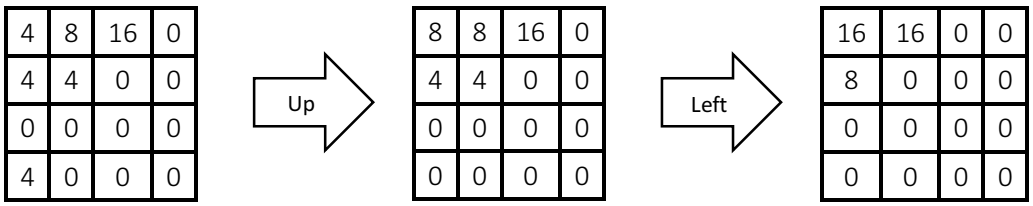
Twenty-Forty-Eight

The Association of Creative Minds (ACM) is organizing the International Creative Puzzles Competition (ICPC) for enthusiastic puzzle-solvers. This competition, usually called ACM-ICPC, is going to be held next week in Singapore, particularly in NUS School of Computing. They are searching for a talented NUS student to help them organize this competition. After interviewing many candidates, they have decided that you are the most suitable candidate out of all applicants. Congratulations!

Now they are going to brief you about your responsibilities in this competition. Out of the many puzzles being contested, one is a very famous puzzle called the 2048 puzzle[1]. It is a puzzle in which you have to slide numbered tiles to get the 2048 tile. Each numbered tile contains a number that is a power of 2, ranging from 2 to 2048. Their goal is to find someone who can get the 2048 tile first, and award that person with a huge sum of money. However, one problem arises: they want to use their own 2048 application instead of using the one available online. Short of experienced programmers, the ACM-ICPC organizer approached you to build a simple simulation of this 2048 puzzle.

Being a beginner at 2048, you don't know what a 2048 puzzle is or how it is played. Luckily, the ACM-ICPC organizer is very helpful. They give you this diagram to illustrate the nature of the puzzle:



1. You can slide the tiles in four different directions: up, left, right, and down. Each numbered tile is represented by the number of the tile while 0 represents an empty cell.
2. When you slide in a direction, chances are at least two tiles will hit each other. If two same-numbered tiles hit each other, they are "merged": their numbers are added together and become one tile. If the two tiles have different numbers, they are simply stacked together.
3. In the left diagram, we have our initial state. If we slide up, we see that, in the first column, the first two "4" tiles are merged since they hit each other and have the same number (i.e. they become a tile with the number 8). Then, the "4" tile at the bottom left corner will hit the newly-formed "8" tile and be stacked together as in the first column of the middle diagram. In the second column, we see that the numbers "4" and "8" are different, hence they are not merged and we leave them as they are since there are no empty cells to slide up to.
4. In a single move, a tile can only be merged once. If we see the middle and right diagrams, we notice that the two "8" tiles in the first row are merged together to form a new "16" tile since we slide to the left. However, in the same move, we also slide the "16" tile to the left. It is then stacked to the right of the newly-formed "16" tile since the "16" tile it hits was just formed on the very same move.
5. Normally, a new "2" or "4" tile is created at a random position after each move. We do not consider it for this problem.

You have one job. Help the ACM-ICPC build a simple 2048 simulator. More precisely, given an initial state of a 2048 puzzle and a direction of a move, print the resulting state of that move, without generating a new "2" or "4" tile at a random position. Good Luck!

[1] Based on 1024 by Veewo Studio and conceptually similar to Threes by Asher Vollmer. Available at http://gabrielecirulli.github.io/2048/.

Input

The input consists of 5 lines. Each of the first four lines shows the initial state of the puzzle, containing 4 numbers each, separated by a single space. The numbers are either powers of two (between 2 and 1024 inclusive) or the number 0. The number 0 represents an empty cell while the powers of two (between 2 and 1024 inclusive) represent a tile with that particular number. The last line of the input consists of a single number, **d**, the direction of the move. This number **d** is either 0, 1, 2, or 3, each representing a slide left, up, right, or down respectively.

Output

Based on the given initial state and slide direction, print 4 lines representing the outcome of the operation, with the i^{th} line representing the i^{th} row of the puzzle. In each of the four lines, print each number with a single space in between each number.

Sample Input 1
```
2 0 0 2
4 16 8 2
2 64 32 4
32 32 64 0
0
```

Sample Output 1
```
4 0 0 0
4 16 8 2
2 64 32 4
64 64 0 0
```

Sample Input 2
```
2 0 0 2
4 16 8 2
2 64 32 4
32 32 64 0
1
```

Sample Output 2
```
2 16 8 4
4 64 32 4
2 32 64 0
32 0 0 0
```

Sample Input 3
```
2 0 0 2
4 16 8 2
2 64 32 4
32 32 64 0
2
```

Sample Output 3
```
0 0 0 4
4 16 8 2
2 64 32 4
0 0 64 64
```

Sample Input 4
```
2 0 0 2
4 16 8 2
2 64 32 4
32 32 64 4
3
```

Sample Output 4
```
2 0 0 0
4 16 8 0
2 64 32 4
32 32 64 8
```

Explanation

Sample Input 2: The slide direction is up. In the fourth column, the two "2" tiles form a new "4" tile. The "4" tile that was in the third row of the fourth column is then located on the second row since it cannot merge with the newly-formed "4" tile. In the other columns, there are no newly-formed tile and hence all tiles only move upwards without forming any new tiles.

Sample Input 4: The slide direction is down. In the last column, we form two new tiles: "4" and "8", since there are two possible merge operations, i.e. between the two "4" tiles and the two "2" tiles.

Skeleton

You are given the skeleton file TwentyFortyEight.java

```
/*
 * Name                 :
 * Matric No.           :
 * Plab Account         :
 */

import java.util.*;

public class TwentyFortyEight {

    public static void main(String[] args) {

    }

}
```

Clarification

The matrices below show the initial and final states of each of the sample input cases above.

Initial State	Slide Direction	Final State
2 0 0 2 4 16 8 2 2 64 32 4 32 32 64 0	LEFT	4 0 0 0 4 16 8 2 2 64 32 4 64 64 0 0
2 0 0 2 4 16 8 2 2 64 32 4 32 32 64 0	UP	2 16 8 4 4 64 32 4 2 32 64 0 32 0 0 0
2 0 0 2 4 16 8 2 2 64 32 4 32 32 64 0	RIGHT	0 0 0 4 4 16 8 2 2 64 32 4 0 0 64 64
2 0 0 2 4 16 8 2 2 64 32 4 32 32 64 4	DOWN	2 0 0 0 4 16 8 0 2 64 32 4 32 32 64 8

Notes:

1. You should develop your program in the subdirectory ex1 and use the skeleton java file provided. You should not create a new file or rename the file provided.
2. If your algorithm is different from the given skeleton, you are free to write a solution according to your own algorithm.
3. You do not have to use OOP for this sit-in lab.
4. You are free to define your own methods.
5. Please be reminded that the marking scheme is:

Input	: 10%	Correctness	: 50%
Output	: 10%	Programming Style	: 30%