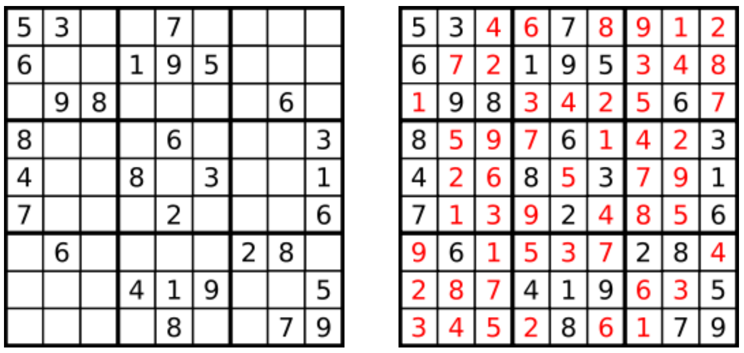
Just su-DO-ku It!

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 Sudoku puzzle. It is a puzzle in which you have to complete a puzzle of size N^2 x N^2 with the numbers [1 … N^2] such that a certain set of requirements is met. Their goal is to find someone who can solve this puzzle first, and award that person with a huge sum of money. However, one problem arises: they want to verify whether the submitted solution is correct or not. A manual check is possible, albeit slow (not to mention tedious). They don't want participants to complain about the slowness of the committee in checking the solutions, do they?

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



1. Each row must contain all numbers between 1 to N^2 (both inclusive) with each number appearing exactly once.
2. Each column must contain all numbers between 1 to N^2 (both inclusive) with each number appearing exactly once.
3. There are N^2 non-overlapping N x N submatrices (defined by the bolded black lines in the diagram). Each of them must contain all numbers between 1 to N^2 (both inclusive) with each number appearing exactly once.
4. The left diagram represents an empty Sudoku puzzle, while a valid solution to the given puzzle is given on the right diagram, given N = 3.

You have one job. Help verify whether the submitted solutions are correct. More precisely, write an automated checker to check whether the given input is a valid solution to the Sudoku puzzle. Good luck!

Input

The input consists of a single number **N** (2 <= N <= 14). **N^2** rows follow. Each row represents the ith row of the Sudoku puzzle. In each row, there are N^2 distinct numbers from 1 to N^2 (inclusive), each occupying a cell in the Sudoku puzzle. There is a single space separating each number.

Output

If the given input is a valid solution to the Sudoku puzzle, print "VALID". Otherwise, print "INVALID".

Sample Input 1

```
3
5 3 4 6 7 8 9 1 2
6 7 2 1 9 5 3 4 8
1 9 8 3 4 2 5 6 7
8 5 9 7 6 1 4 2 3
4 2 6 8 5 3 7 9 1
7 1 3 9 2 4 8 5 6
9 6 1 5 3 7 2 8 4
2 8 7 4 1 9 6 3 5
3 4 5 2 8 6 1 7 9
```

Sample Output 1

```
VALID
```

Sample Input 2

```
3
7 9 2 1 3 5 4 6 8
5 4 6 8 7 9 2 1 3
3 8 1 6 2 4 9 5 7
1 3 5 4 6 8 7 9 2
8 7 9 2 1 3 5 4 6
6 2 4 9 5 7 3 8 1
4 6 8 7 9 2 1 3 5
2 1 3 5 4 6 8 7 9
9 5 7 3 8 1 6 2 5
```

Sample Output 2

```
INVALID
```

Explanation

Sample input 1 is illustrated in the Sudoku puzzle example. It is clear that sample input 1 provides a valid solution as illustrated in the example, whereas sample input 2 is not a valid solution. If you look at the last column of sample input 2, there are two 5s in that column (row 7 and 9). It already violates one of the requirements of a valid Sudoku solution. Therefore, it is not a valid solution.

Skeleton

You are given the skeleton file Sudoku.java

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

import java.util.*;

public class Sudoku {

    public static void main(String[] args) {

    }

}
```

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%