Consider the following **whoever-pick-the-last-one-lose** game. The game is played on a 5×5 board. Initially every array cell has a piece in it. Two players remove pieces alternatively from the board. The player can remove any number of consecutive pieces in a row or column. For example, in the configuration depicted below where one indicates a piece, the player can either remove one piece (**A1**, **A2**, or **B1**), or remove two pieces (**A1** and **A2**, or **A1** and **B1**) simultaneously. The game ends when one player is forced to take the last piece, and the other player wins the game.

	1	**2**	**3**	**4**	**5**
A	1	1	0	0	0
B	1	0	0	0	0
C	0	0	0	0	0
D	0	0	0	0	0
E	0	0	0	0	0

Write a program that evaluates board configurations from this game. The program must output '`winning`' when there exists a winning move that no matter how the opponent responds, it will force the opponent to take the last piece. Otherwise, the program must output '`losing`'.

Note that during the game tree evaluation, if the current configuration has a winning move, then it is not necessary to search any further because the configuration is guaranteed to be winning. This can greatly reduce the game tree search time.

Input

The input file contains $5c + 1$ lines.

Line 1	c	the number of configurations
Lines 2-6	...	configuration #1
...	...	
Lines $5c - 3$ to $5c + 1$	...	configuration #c

Output

The output contains c lines.

Line 1	evaluation result of configuration #1
...	
Line c	evaluation result of configuration #c

Sample Input

```
3
1 1 0 0 0
1 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
1 1 0 0 0
0 0 0 0 0
1 1 0 0 0
0 0 0 0 0
0 0 0 0 0
1 1 1 0 0
1 0 0 0 0
0 0 0 0 0
0 0 0 0 0
0 0 0 0 0
```

Sample Output

```
winning
losing
winning
```