# Problem 1: Minesweeper

Minesweeper is a popular computer game in which "mines" are hidden in a grid of squares. The player clicks on squares one at a time. If the square contains a mine, the player "blows up" and the game ends. A square that is not adjacent to a mine we'll call *lonely*. (Here, adjacent means any of the up to eight surrounding squares.) A square that does not contain a mine but is adjacent to a mine we'll call a *boundary square*. If the clicked-on square is a boundary square, the number of adjacent mines is printed in the square. If the clicked-on square is lonely, then that square is uncovered and the process repeats for all adjacent squares (i.e., as if each of those squares had been clicked-on). (So, if any of these adjacent squares are themselves lonely, then the process repeats for all of their adjacent squares and so on.)

For example, figure 1 below shows a grid with the location of some mines. This would be hidden from the player. Suppose the player clicks on the square in the third row, fourth column. The revealed board is shown in figure 2. (Here, * represent squares still covered.)

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| M | | | M | | | | |
| | | | | | | | |
| | | | | | | M | M |
| M | | | | | | | |
| M | | | | | | | |
| | | M | | | | | |
| | | | | | | M | |
| | | | M | | M | | |

figure 1

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| * | * | * | * | * | * | * | * |
| * | 1 | 1 | 1 | 1 | 1 | * | * |
| * | 1 | | | | 1 | * | * |
| * | 2 | | | | 1 | 2 | 2 |
| * | 2 | 1 | 1 | | | | |
| * | * | * | 1 | | 1 | 1 | 1 |
| * | * | * | 2 | 2 | 2 | * | * |
| * | * | * | * | * | * | * | * |

figure 2

In this problem there will be multiple problem instances. The first line of each problem instance will consist of two integers, $n$ and $m$. (A value of $n = 0$ indicates end of input.) Neither $n$ nor $m$ will exceed 20. Next will be $n$ lines, each containing $m$ characters with no spaces separating them. Each character will be either `M` or `E`, indicating either a mined (`M`) or an empty (`E`) square. We'll say the first character is in row 1, column 1; the next character in row 1, column 2 and so on. Following these $n$ lines will be a line containing the positive integer $k$. Next follow $k$ rows each containing a row number and a column number. These indicate the row and column that the player clicks on.

The effect of clicking on the squares is cumulative. If a mined square is clicked on, the remaining row/column input for this problem instance should be ignored. Once the $k$ squares have been clicked on, or a mined square has been clicked on, the board should be displayed in the format shown below. An asterisk (`*`) will indicate a square not yet revealed, a space indicates a revealed lonely square, and an `M` indicates a clicked on mined square. If a mined square has been clicked on, *all* mines should be displayed in addition to the squares revealed.

A blank line should follow each board displayed.

**Example Input:**

```
8 8
MEEMEEEE
EEEEEEEE
EEEEEEMM
MEEEEEEE
MEEEEEEE
EEMEEEEE
EEEEEEME
EEEMEMEE
3
3 4
8 1
1 2
2 3
MEE
EME
4
1 1
1 2
2 1
2 2
0 0
```

**Corresponding Output:**

```
*1******
*11111**
*1   1**
*2   122
*311
12*1 111
 12222**
  1*****

M**
*M*
```

# Problem 2: Telephone Rings

The telephone number 462-4211 has the interesting property that $462 = 42 * 11$. Likewise, 350-5710 can be written $350 = 5*7*10$ and 639-971 can be written $639 = 1*9*71$. We'll say a telephone number is *multiplicative* if one or more multiplication signs (*) can be inserted in the digits of the four rightmost digits so that the resulting product is equal to the exchange number. (The exchange number is the three leftmost digits.)

There will be multiple problem instances. The first line of input will contain an integer $n$ indicating the number of problem instances. Each of the next $n$ lines will contain a telephone number in the form `DDD-DDDD`, where each `D` is a digit. You may assume that neither the first digit of the exchange nor the first digit of the four rightmost digits is zero.

For each input instance, output one line of output, either

*DDD-DDDD* `is multiplicative.`

or

*DDD-DDDD* `is not multiplicative.`

where *DDD-DDDD* is the telephone number supplied.

**Example Input:**

```
6
462-4211
999-2211
639-1971
555-1051
196-2283
123-1123
```

**Corresponding Output:**

```
462-4211 is multiplicative.
999-2211 is not multiplicative.
639-1971 is multiplicative.
555-1051 is not multiplicative.
196-2283 is not multiplicative.
123-1123 is multiplicative.
```

# Problem 3: Lucky Numbers

Lucky numbers are generated in the following manner. First list all positive integers.

Step 1: Remove all even integers.

Step 2: Now 3 is the 2nd number in the list, so remove every 3rd number in the list.

Step 3: Now 7 is the 3rd number in the list, so remove every 7th number in the list.

Repeat this pattern. So, on step $k$, remove every $m$th number in the list where $m$ is the value of the $k$th number in the list after step $k - 1$.

The numbers that remain are called Lucky Numbers. So, the first seven Lucky Numbers are 1,3,7,9,13,15 and 21.

Each line of input will contain a non-negative integer $n$. (A value of $n = 0$ indicates end of input.) For each input value, you should print the $n$th Lucky Number on its own line. You may assume no output will exceed 10,000.

**Example Input:**

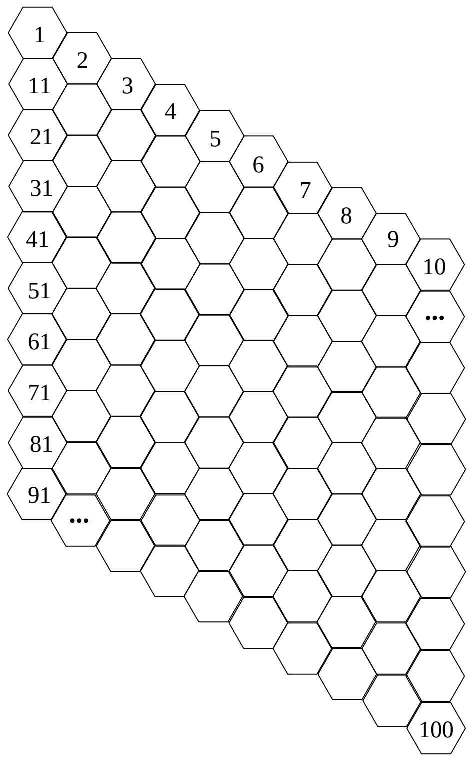
```
1
5
10
100
1000
0
```

**Corresponding Output:**

```
1
13
33
613
8809
```

# Problem 4: The Game of Hex

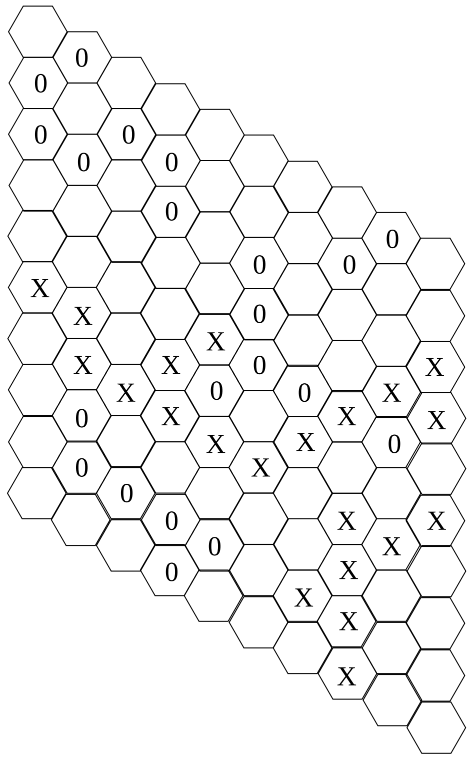
The Game of Hex is played on a 10 by 10 honeycomb of hexagons. For reference, let's number the hexagons on this honeycomb as below.




Hex is played by two players, alternating marking hexagons with X's and O's. (A hexagon can be marked only once.) The objective of player X is to get a connecting path of X's between the left and right borders. (That is, between any hexagon numbered 1, 11, 21, ..., 91 and any hexagon numbered 10, 20, ... 100.) The objective of player O is to get a connecting path of O's between the top row (those hexagonals numbered 1 through 10) and the bottom row (those numbered 91 through 100). Once either player has met their objective, that player wins the game. Note that both players can't have a winning configuration.

Given which hexagons are marked with X's and which are marked with O's, you are to determine which player, if any, has won the game.

For example, if player X has marked squares 30, 39, 40, 45, 48, 51, 25, 54, 57, 60, 62, 63, 64, 65, 66, 68, 69, 78, 87, 88, and 98 and player O has marked hexagons 2, 9, 11, 13, 14, 18, 21, 22, 24, 26, 36, 46, 47, 49, 55, 72, 82, 83, 84, 85 and 94, then the game board looks like the following and you can see that player X has won the game.



There will be multiple problem instances. The first line of a problem instance will consist of an integer n, indicating the number of hexagons marked with X's, which will be equal to the number of hexagons marked with O's. (A value of $n = 0$ indicates end of input.) The second line will have n positive integers, in increasing order and separated by single spaces, giving the number of the the hexagons marked with X's. The third line will give the hexagons marked with O's, similarly formatted. You are guaranteed that no number will appear in both listings and all numbers listed will be between 1 and 100, inclusive.

Output for each problem instance should be one of the following lines, as appropriate:

```
Player X wins.
Player O wins.
Neither player wins.
```

**Example Input:**

```
21
30 39 40 45 48 51 52 54 57 60 62 63 64 65 66 68 69 78 87 88 98
2 9 11 13 14 18 21 22 24 26 36 46 47 49 55 72 82 83 84 85 94
5
1 4 56 77 78
3 5 16 17 21
0
```

**Corresponding Output:**

```
Player X wins.
Neither player wins.
```

# Problem 5: Numbered Square

We fill in a 3 by 3 grid with 9 given integers. The rule is that each integer across the top must occupy a square in the column below it and each integer listed to the left must occupy a square in the row to it's right. Each square must be occupied and an integer can occupy only one square. Such an assignment of integers to squares we call an occupation.

The diagram on the left shows a starting configuration and the next two grids show two legal occupations.

| | | 3 | 7 | 1 |
|---|---|---|---|---|
| 2 | 10 | | | |
| 1 | 5 | | | |
| 7 | 1 | | | |

| | | |
|---|---|---|
| 3 | 10 | 2 |
| 5 | 7 | 1 |
| 1 | 7 | 1 |

| | | |
|---|---|---|
| 2 | 7 | 10 |
| 3 | 5 | 1 |
| 7 | 1 | 1 |

The value of a certain occupation is determined by calculating the product of the integers in each row and the product of the integers in each column then summing these six values. For example, the value of the first occupation above is 609 and the value of the second one is 249. You are to find the maximum value of a legal occupation given a certain setup.

There will be multiple problem instances. The first line of input will be a positive integer $n$ indicating the integer of problem instances. Each of the next $n$ lines contains data of a configuration and will be of the form

*c1 c2 c3 r11 r12 r21 r22 r31 r32*

where *c1 c2* and *c3* are the integers at the top of columns 1, 2 and 3, respectively, and *ri1* and *ri2* are the two integers to the left of row $i$.

Output for each problem instance should be one integer on a line.

**Example Input:**

```
2
3 7 1 2 10 1 5 7 1
1 1 1 1 2 1 2 1 2
```

**Corresponding Output:**

```
609
13
```