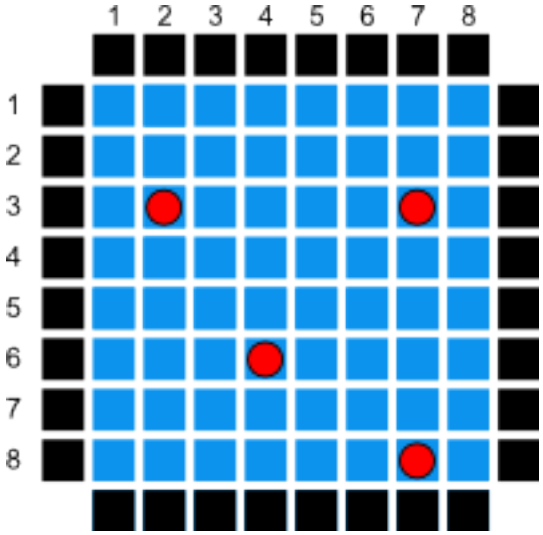
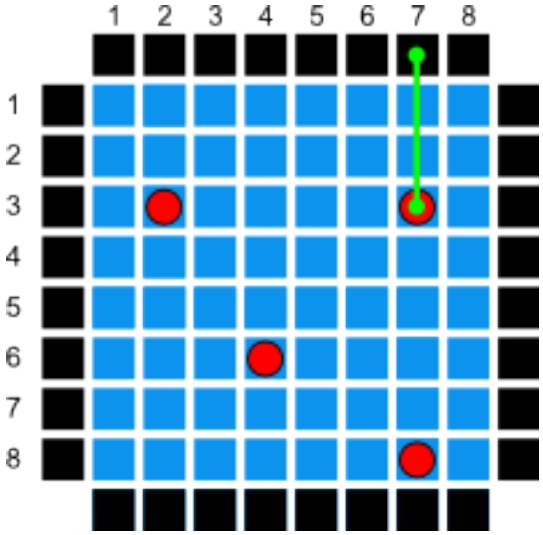
The product design department has invented a new game called the Reybox, but they need some help working out the final details and have asked us to write them a simulator for the game.

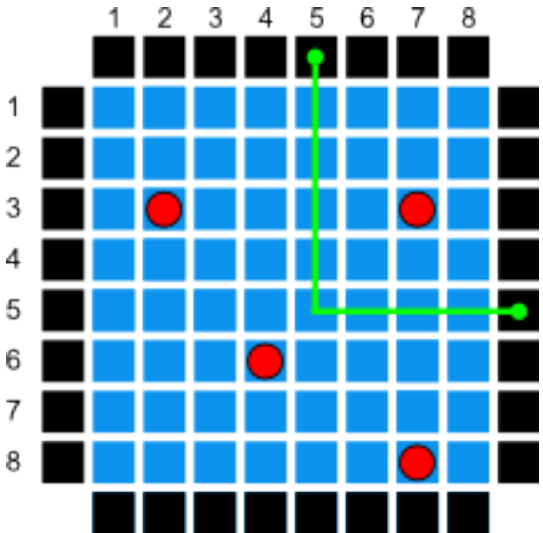
The Reybox is square and flat, sealed so the players can't see inside and with a number of holes along each of the edges, the same number on each edge, equally spaced. Product design has specified there should be at least 8 holes on each side. The players shoot a ray of light into one of the holes and see where it comes out. If the box was empty, the ray of light would come straight out of the opposite hole, but inside are a few magic mirrors which can absorb or deflect the light so it may come out of a different hole, or may not come out at all. To make the game more random, some of the magic mirrors have a limited life and evaporate after they have absorbed, or deflected, a number ofrays



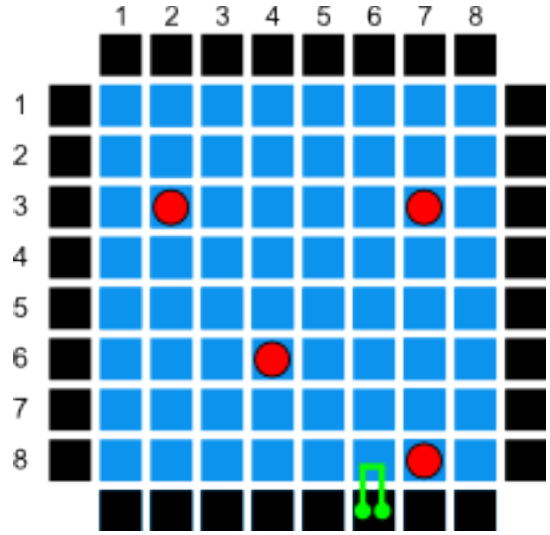
One possible grid might look like this. It has 8 holes per side and mirrors at {3,2}, {3,7}, {6,4} and {8,7}.



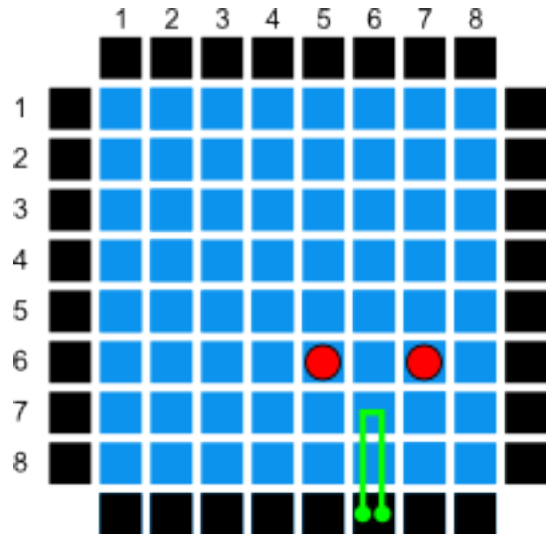
The rules are quite simple, if a ray of light hits a magic mirror directly, it's absorbed. Here a ray starting at {1,7} is absorbed by the mirror at {3,7} and doesn't come out at all.



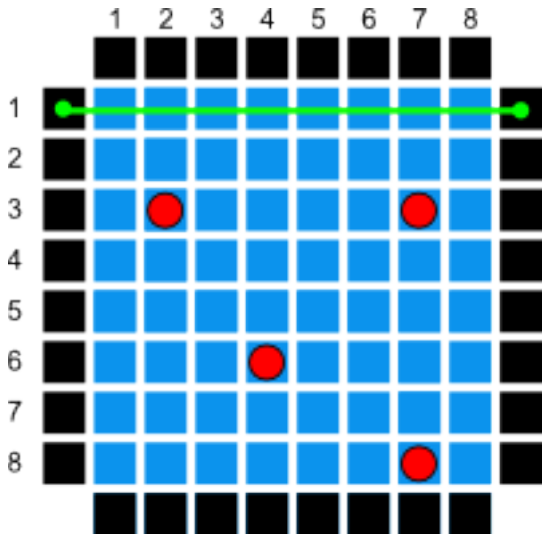
If a ray passes diagonally adjacent to a mirror, it's deflected 90 degrees away from that mirror. Here the mirror at {6,4} deflects a ray starting at {1,5}, causing it to emerge at {5,8}



If a ray enters next to a mirror which is at the edge of the box, it's instantly reflected back out. The mirror at {8,7} reflects the ray coming in at {8,6}, causing it to exit at the same place it came in.

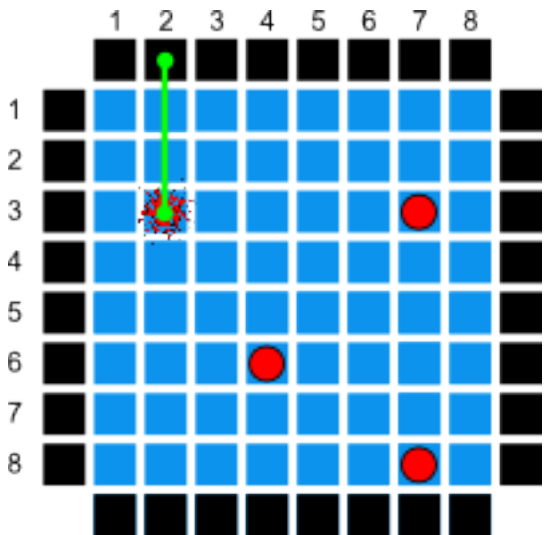


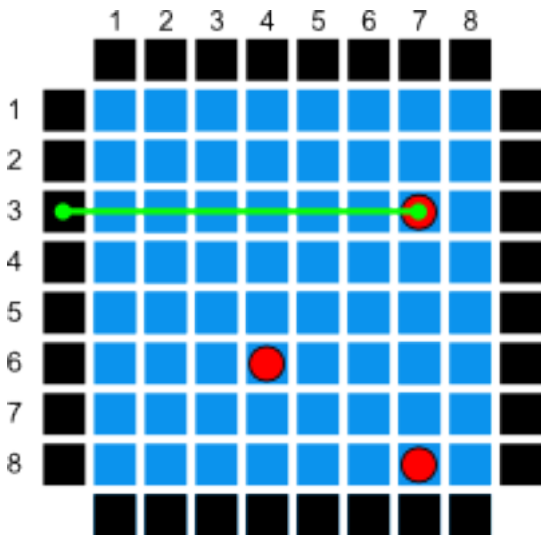
A ray may be deflected more than once, in this example it's deflected once by the mirror at {6,5} and once by the one at {6,7}, reflecting it back to where it came from.



Rays can pass right through the box if no mirrors affect them.

And, below, a mirror has reached its maximum count of strikes and evaporates, absorbing one last ray. The next time a ray passes through, the mirror has gone and has no effect.





In a case where the positions of mirrors mean a ray could be absorbed or deflected in the next step, the ray will be absorbed.

Input to the simulator comes from two files. One is a design file which specifies the size of the grid and the positions, and types, of the mirrors. The other is a test file with a list of positions around the edge of the box where rays are fired in, the simulator must work out where each of those test rays comes out, if at all and print it out.

For both input files, blank lines and lines starting with a '#' are ignored. The first valid line in the design file has a single number defining how many holes per side this model has, the rest of the valid lines have a pair of numbers which are the row and column reference of one mirror; if the mirror has a limited life, that life follows the grid reference. If there is no life number, the mirror lasts forever.

Here is the input file for the first example above, assuming the mirror at {8,7} has a limited life of 10 hits but the rest of the mirrors last forever.

```
# first line has the number of holes
8
# the rest of the lines specify the mirror positions
# the final one at {8,7} has a life of 10 hits or deflections
3 2
3 7
6 4
8 7 10
```

The test file lists the places in order a ray is fired in. There's a letter, either R for row or C for column, a number representing the row or column and a `+` or `-` sign which indicates the direction of travel. For rows a `+` means left to right, a `-` right to left and for columns `+` is top to bottom and `-` bottom to top. For the example above which has 8 rows and columns here's one possible

```
# one test per line
# this is a ray travelling down column 7 starting at {1,7}
C7+
# a ray travelling down column 5, starting at {1,5}
C5+
# or travelling right to left along row 5 starting at {5,8}
R5-
```

The required output repeats the places the ray entered and shows the cell it exits, separated by an arrow. If the ray is absorbed, there is nothing after the arrow. So the test above should produce

```
C7+ ->
C5+ -> {5,8}
R5- -> {1,5}
```

The output should have one line for each actual input. The line should start with the input as it was in the file, then a space, an ascii arrow '->' and then either nothing if the ray does not exit the box, or a space followed by the coordinates at which it exits, separated by a comma, enclosed in curly braces, if it does exit. Please see the examples above and follow the format.

Your solution should be written in C++ and consist of one or more source files along with a Makefile and instructions on how to compile and run your code. You are free to develop the answer on any platform you like, but we should be able to compile it with make and run it on Linux. If you choose to use external libraries please package them with the answer. And if you have any questions, please do not hesitate to ask.