**21.A number of cats and mice inhabit a house. The cats and mice have worked out a deal where the mice can steal pieces of the cats’ food, so long as the cats never see the mice actually doing so. If the cats see the mice, then the cats must eat the mice (or else lose face with all of their cat friends). There are NumBowls cat food dishes, NumCats cats, and NumMice mice. Your job is to synchronize the cats and mice so that the following requirements are satisfied: No mouse should ever get eaten. You should assume that if a cat is eating at a food dish, any mouse attempting to eat from that dish or any other food dish will be seen and eaten. When cats aren’t eating, they will not see mice eating. In other words, this requirement states that if a cat is eating from any bowl, then no mouse should be eating from any bowl. Only one mouse or one cat may eat from a given dish at any one time. Neither cats nor mice should starve. A cat or mouse that wants to eat should eventually be able to eat. For example, a synchronization solution that permanently prevents all mice from eating would be unacceptable. When we actually test your solution, each simulated cat and mouse will only eat a finite number of times; however, even if the simulation were allowed to run forever, neither cats nor mice should starve.**

**Solution:**

In a rectangular field of size n by m squares there is a mouse and two cats. The mouse is the first to make a move, then each of the cats makes a move, then again, it’s the mouse's turn, and so on. In each move both the mouse and the cats can move exactly one square vertically or horizontally. If the mouse is standing at the edge of the field then in its next move it can jump off the field and is saved from the cats. If in the next move one of the cats moves to the field with the mouse then there is no escape for the mouse

**Function Description**

Complete the *catAndMouse* function in the editor below. It should return one of the three strings as described.

**catAndMouse** has the following parameter(s):

• x: an integer, Cat A’s position

• y: an integer, Cat B's position

• z: an integer, Mouse C's position

**Input Format**

The first line contains a single integer, q, denoting the number of queries. Each of the q subsequent lines contains three space-separated integers describing the respective values of x (cat A's location), y (cat B's location), and z (mouse C's location).

**Constraints**

• 1<=q<=100

• 1<=x,y,z<=100

**Output Format**

For each query, return Cat A if cat A catches the mouse first, Cat B if cat B catches the mouse first, or Mouse C if the mouse escapes.

**Sample Input 0:**

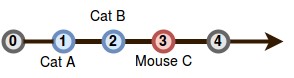
|  |
| --- |
| **2**  **1 2 3**  **1 3 2** |

**Sample Output 0:**

|  |
| --- |
| **CAT B**  **MOUSE C** |

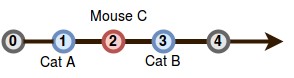
**Explanation 0**

**Query 0: The positions of the cats and mouse are shown below:**



**Cat B will catch the mouse first, so we print Cat B on a new line.**

**Query 1: In this query, cats A and B reach mouse C at the exact same time:**



**Because the mouse escapes, we print Mouse C on a new line.**