1. **List all of the synchronization primitives and shared variables that you have used to synchronize the cats and mice and identify the purpose of each one.**

   The shared variables used include an array (*bowlsArray*) tracking the occupancy of the bowls, an integer (*eatingAnimal*) representing the type of animal dining right now, and two more integers (*cat_queue_count*, *mice_queue_count*) each representing the capacity of the queue containing cats, and the capacity of the queue containing mice.

   The synchronization primitives used are locks, semaphores, and condition variables. Locks are used for general mutual exclusion inside the "dining room" and also they are there to support the use of condition variables, which are the ones doing the heavy work for synchronization. Lastly, the semaphore used is a counting semaphore, which controls the number of animals to enter and eat at a time.

2. **Briefly explain how the listed variables and synchronization primitives are used to synchronize the cats and mice.**

   The abstraction used in the solution resembles that of a dining room with a table holding *NumBowls* of food. Animals that are not of the type *eatingAnimal* or have to simply wait because the dining room is already full will be put in two queues (cat-only and mice-only) "outside the room". These queues are made using condition variables with *cv_wait*. The first animal that arrives at an empty dining room will decide the value for *eatingAnimal*.

   Dining: we let in a batch of animals of *eatingAnimal* type to dine. Entry is controlled using a counting semaphore to ensure everyone who comes in has a bowl of food to eat. The eaters will then go through the *bowlsArray* and pick a vacant bowl to eat. When finished eating, each animal will "wash" the bowl and put it back to place before exiting the dining room. The last animal of the batch to leave the dining room will signal the next batch to come in. No batch will be called if the queues are empty (i.e. *cat_queue_count = 0* and *mice_queue_count = 0*); in this case the value of *eatingAnimal* will be reset.

3. **Briefly explain why it is not possible for two creatures to eat from the same bowl at the same time under your synchronization technique**

   First of all, the bowl selection process is a critical section that is mutually exclusive (inside a lock acquire). This means that only one animal will select from *bowlsArray* at a time. A vacant bowl has value 0. When an animal picks a vacant bowl, the bowl is then checked with 1 to indicate occupancy. When the animal finishes with the bowl it will be checked back to 0.

4. **Briefly explain why it is not possible for mice to be eaten by cats under your synchronization technique.**

   The dining sequence lets in either a batch of mice or a batch of cats at a time. When waiting outside, cats are put in a cat-only queue and mice in a mice-only queue. Cats and Mice are not mixed in any queue (so they will never see each other

face to face). A new batch is only called after all animals in the current batch have finished and exited the dining room.

5. **Briefly explain why it is not possible for cats or mice to starve under your synchronization technique.**

   Recall there are 2 queues waiting outside the dining room, one containing only mice and one containing only cats. The default scheme is to alternate between a batch of cats dining and a batch of mice dining, which is fair. However, say if the mice queue is empty when a batch of cats have finished, then we will just send in another batch of cats (assuming cats queue is not empty). There is no reason to let the cats wait for a mouse to show up in the other queue. Therefore, the combination of first-come-first-serve and queue-alternation ensures that no animal has to wait for too long and starve.

6. **Briefly discuss the fairness and efficiency of your synchronization technique, and describe any unfairness (bias against cats or mice) that you can identify in your design. Did you have to balance fairness and efficiency in some aspect of your design? If so, explain how you chose to do so.**

   The goal of the design is to achieve a balance between fairness and efficiency. By letting in a batch of cats or mice to eat at once (instead of one cat or one mouse at a time), it promotes efficiency because both queue waiting time and idle bowls are minimized. Fairness is governed through alternating between batches of mice and cats while keeping in mind that, among the 2 waiting queues, a non-empty queue should not waste time waiting on an empty queue (first come first serve). Therefore there is no bias against cats or mice in this design because opportunities are distributed equally - queues take turns to go. A queue does not lose its turn if it is non-empty.