Task 1

We can see that the program has inconsistent state since several readers are reading while other writers are changing values, before all of the readers have had the chance to read. As well as different readers acquiring the wrong values after writers have changed the values in the program. This can occur when different processes/threads are not aware of each others execution and no monitoring is established between concurrent processes, in this case states would be useful.

I do not believe starvation is actually occuring in the program, however the notify() function selects a thread to be executed/woken up arbitrarily, so it is definitely possible for a thread to never get executed in the first place.

Implementation
I used synchronized to solve this task with wait() and notify(). I believe this was a more fun solution, and not really any other reason for not using Reentrant Lock or Semaphores. However, this was used later on in the next task.. The program behaved more structured after adding these changes.

Task 2

Implementation
The acquire functions were changed in order to get the desired behavior from the program, especially the aquireRead where an extra while (reading || writing) was added to only allow a read to be acquired if there were no processes currently reading or writing the shared data. I believe this was a good way of checking it.

Task 3

Yes, it is possible to see that more than 5 chopsticks are used by looking at the output.

The Circular Wait condition is easiest to solve, by only allowing a philosopher to pick up a chopstick if both chopsticks on either side are available. The program no longer had a deadlock after adding this change and ran better than before.