1. The problem I am trying to solve is the consumer-producer problem. The producer and consumer share a common buffer so that they may run concurrently. However, I need to make sure that the consumer does not attempt to consume an item that has not yet been produced and that the producer does not attempt to add an item to a full queue. To accomplish this, I use pthreads. By surrounding the critical sections of the consumer and producer with pthread mutex locks and unlocks, mutual exclusion is accomplished.

2. The busy-wait solution may produce the desired output, but it is still a wasteful solution. This is because when the conditions for a thread are not met, the thread continues *spinning*, wasting CPU cycles.

3. I am confident that my solution is correct because no producer ever attempts to add to a queue after it realizes it is full. Additionally, the producers to not exit until there is nothing left to produce, and the consumers do not exit until there is nothing left to consume. The producers and consumers are all operating on unique items, because my implementation adheres to the principles of mutual exclusion. Each item is both produced and consumed before the termination of the program. Futhermore, the producers produce the correct number of items, while the consumers consume the correct number of items. My solution passes all the tests from the makefile tests. There are exactly the same number of producer calls as there are consumer calls.