1a

The `lk.lock()` ensures mutual exclusion, meaning that if one thread holds the lock, the other thread must wait until it is released. Therefore, P and Q cannot print simultaneously since one thread must hold the lock and complete its sequence before the other can start.

P first:

- p1,p2,p3
- q1,q2,q3
- **Output:** "P" "Q"

Q first:

- q1,q2,q3
- p1,p2,p3
- **Output:** "Q" "P"

1b

Once acquired, Thread P will no longer release the lock. If Thread P acquires the lock first, Thread Q must wait indefinitely.

P first:

- p1,p2
- Q tries to execute `q1: lk.lock()`, but Thread P has not released the lock, so Thread Q is blocked indefinitely.
- Thread Q will never be able to proceed past `q1`.
- Thread Q never prints "Q"
- **Output:** "P"

Q first:

- q1,q2,q3
- p1,p2

Both threads have printed their respective outputs, but P still holds the lock, which can cause further consequences depending on the overall system behavior. If P terminates without releasing the lock, this could lead to deadlock situations later if other threads or processes need the same lock.

- **Output:** "Q" "P"

2a

- Thread P acquires the lock before Thread Q can acquire it.
- Thread P executes `p1: lk.lock()`, sets `B = True,` and then releases the lock.
- After Thread P releases the lock, Thread Q can acquire it.
- When Thread Q reaches `q2: while not B`, it finds that `B = True,` so it skips the loop entirely and releases the lock.
- Thread Q does not print any '*' because it never enters the `while not B` loop.

2b

- Thread Q acquires the lock before Thread P can acquire it.
- Thread Q begins executing its loop in `q2: while not B` and prints '*' as long as `B` is `False.`
- While Thread Q is inside the loop, Thread P cannot set `B = True` because it cannot acquire the lock.
- Thread Q repeatedly prints '*'

3

Setup

- Integer n = 0 and Lock lk
- lk.lock()

Thread P	Thread Q
p1: for i from 1 to 20 do	q1: lk.lock()
p2: temp = n	q2: print(n)
p3: n = temp + 1	
p4: lk.lock()	