**READ ME Q3**

The code uses Semaphores for synchronization. In general, the code ensures all the conditions that are mentioned in the problem are followed.

We initialized four semaphores: mutex, bridge, l\_mutex, and r\_mutex. These control the access to various critical sections as provided in the code.

The left and right functions represent cars from the left and right sides of the bridge, respectively. Each function checks if there's space available on the bridge for a car to cross. If available, it proceeds; otherwise, it employs a timeout mechanism similar to the previous code to avoid deadlock situations.

The main function initializes semaphores, prompts the user for the number of cars on each side, creates threads for left and right cars, waits for their completion, and then destroys the semaphores.

Mutex controls access to critical sections modifying shared variables.

Bridge limits the number of cars allowed on the bridge concurrently.

l\_mutex and r\_mutex control access to the left and right sides of the bridge, respectively.

They ensure that possible race conditions and concurrency bugs are avoided. Error handling has also been done for any kind of discrepancies.