PSEUDO CODE IS AT THE END!

Some Global Variables which are important for understanding the main flow:

CounterA: Team A thread counter

CounterB: Team B thread counter

GameTurn: To differentiate whether we are forming a group or finding a car and deciding a captain. If it is zero, threads are waiting for enough people to share a ride. If it is 1, they are forming the proper group to share a ride. This will guarantee that we have ordered termination.

Main Thread: Firstly, the given parameters are checked whether they are valid or not. If they are not valid, main thread terminates itself. Secondly, semaphore and conditional thread which will be used in the thread function are created. Then, threads are created according to given parameters. Each thread has their own argument as character which indicates their team. Since we want main thread to wait for the other threads, we use "join" method to proper order of execution.

Thread Function: Each thread goes to same function. Function's first line of code is a semaphore wait operation. Our initialized semaphore has a value as 1. It means that after one thread execute semaphore wait, it will be decremented by 1 and become 0. Therefore, other threads will wait until a thread execute semaphore post method which will increment the semaphore's value by 1. Then, the tread who executed the semaphore wait, will say it is looking for a car and increment the global variable of one of the counters (CounterA or CounterB) according to own argument. Then, it will look for if there are enough people to form a group to share a ride. If there are not enough people, it will post the semaphore to allow one more thread. This process will repeat itself until enough people to share ride. Threads who passed this part by posting the semaphore, wait in a loop to "GameTurn"

becomes 1 to ensure ordered execution. The last thread of the group will not post the semaphore, so the others will wait the execution of “I have found a car” and “I am the captain” statements.

Threads which passed to first part, will acquire a lock one by one to ensure there is no race condition. In the lock, there are several locks to decide what is the type of ride share. In other words, is there 4 Team A supporters or 4 Team B supporters, or 3 Team A, 2 Team B and so on. If there are extra people while we are forming to group, these extra threads will sleep with conditional thread wait method. In other words, if there are 3 A and 2 B threads, one of the A threads will conditionally wait in this lock. For waking up, the signal will come from the first part where the last thread is not posting the semaphore but give the signal of waking up for these sleeping threads.

At the last part, after 4 threads come to this part with the help of while loop and barrier, one of them announced itself as a captain of the car and they will update CounterA and CounterB global variables. Also, the last person of the car will post the semaphore which is located at the first line of the function. Therefore, this process will repeat itself until all threads form a group.

After all thread’s termination, main thread terminates itself by printing out “The main terminates”.

Correctness: Since I divided my function into sub parts and also used semaphores, conditional threads, barrier and turn based while loops, my program provides correctness criteria.

PSEUDO CODE

For easier understanding I will divide my function 3 sub-parts: forming a group, finding a car and deciding the captain respectively part 1, part 2 and part3.

Thread Function:

Part 1 (Forming a group):

"Semaphore **wait**()"

Print out "I am looking for a car."

Update global counters (team A counter and team B counter)

Check if there are enough people to form a group to share a ride:

If there are **not enough** people:

"Semaphore **post**()"

If there are **enough** people:

"**Game turn = 1**" and "**Condition Signal**" for sleeping thread in part 2

Between Part 1 and Part 2:

While (game turn != 1) // this is for ensuring that threads which completed the first part will wait until a proper group formation to proceed to Part 2. Therefore, announcement of finding a car will never execute before announcements of looking a car. **(Correctness Criteria)**

Part 2 (Finding a car):

"**Acquire** Mutex Lock"

Check the counters:

If counterA == 4: // Team A threads will announce they **found** a car

If there is a thread B: "**Conditionally** wait"

If CounterB == 4: // Team B threads will announce they **found** a car

If there is a thread A:

"**Conditionally** wait"

If CounterA == 2 & CounterB == 2:

Both team A and team B threads will announce they **found** a car

If CounterA == 3 & CounterB ==2:

One of the team A threads will sleep, others will announce they **found** a car

If CounterA == 2 & CounterB == 3:

One of the team A threads will sleep, others will announce they **found** a car

P.S. After each announcement threads will "unlock the mutex".

Between Part 2 and Part 3:

There is a **barrier** and **while** loop to ensure all threads ended the part 2 to ready for going in Part 3. Therefore, announcement of captain will never execute before announcements of finding a car. **(Correctness Criteria)**

Part 3 (Deciding the captain):

"Acquire Mutex Lock"

First thread which acquired to lock, announced itself as the **captain** of the car

Then, we will wait until 4th thread to lock the mutex with the help of a global thread counter variable: **(Correctness Criteria)**

The 4th thread will update global variables such as CounterA and CounterB, also it will **"post the semaphore"** which is in the Part 1. To repeat all these parts.

"Unlock the Lock"

Main Thread:

Check validity of parameters

Initialize semaphore, **barrier**, and **conditional thread**.

Create Threads

Join Threads

Terminate itself, **print** out (The main terminates)