Names:1- Mostafa Ahmed Abdelrhman / Class:3 ID:12300674

2-Mohsen Ibrahim Hasan / Class:6 ID:12200037

Old Bridge Problem

Under Supervision of TA: **Nada Mohamed Abd El-Maboud**

I. **Description of the problem:** The Old Bridge Problem is also known as the Crossing the Bridge Problem classic synchronization challenge in concurrent programming. It involves managing access to a single-lane bridge where only one direction of traffic can cross at a time, and often there's a limit to the number of vehicles that can be on the bridge simultaneously.

II. The problems we encountered during the solving:

I. Utilizing the **Semaphore** Library is only available in C20 or Later so if we depend on this library most of our solving won't be workable on many systems.

II. We struggled in our solution because we made a mistake by placing unique_lock inside loop itself

III. The issue arises in synchronizing vehicle movement without causing data races or deadlocks. Without adequate synchronization, the following concerns may arise:

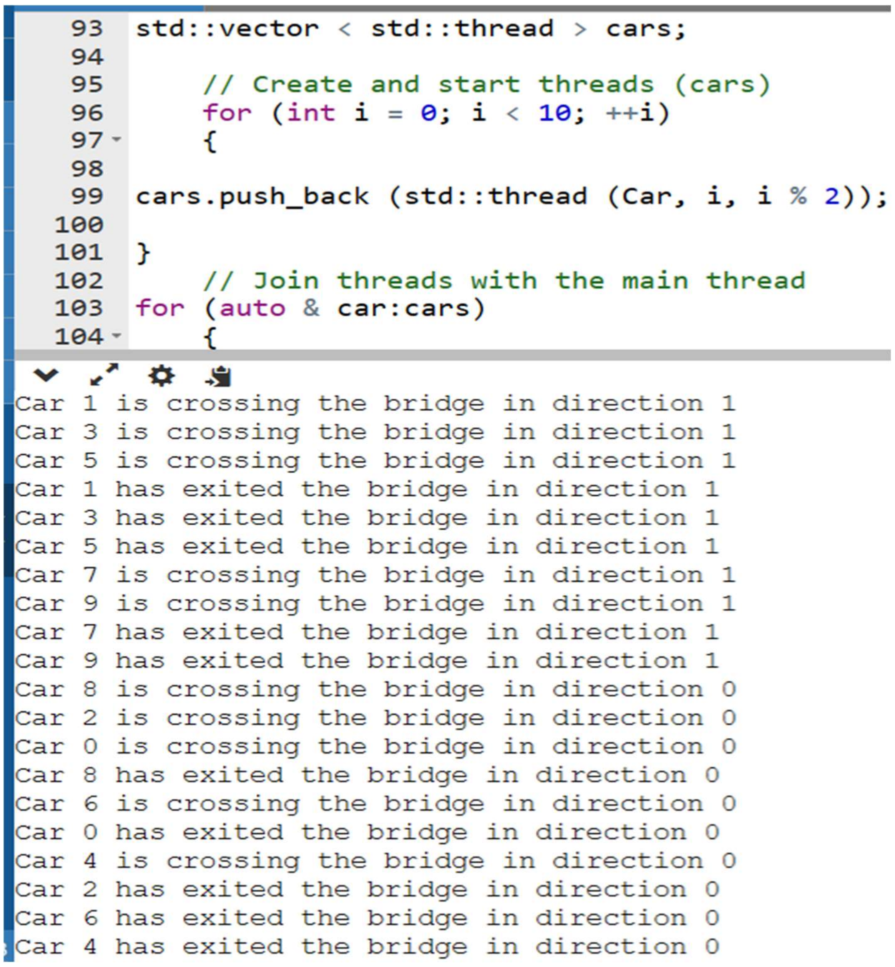
Data Race: If many threads representing vehicles attempt to access the bridge at the same time, they may overwrite the current state (for example, the number of vehicles on the bridge), causing inconsistencies.
Deadlock: If two cars (one light and one heavy) collide on a bridge, they may become stuck, waiting for the other to leave while neither can move.

III. The Solution of the Problem:

Several synchronization techniques can be used to solve the Old Bridge Problem:

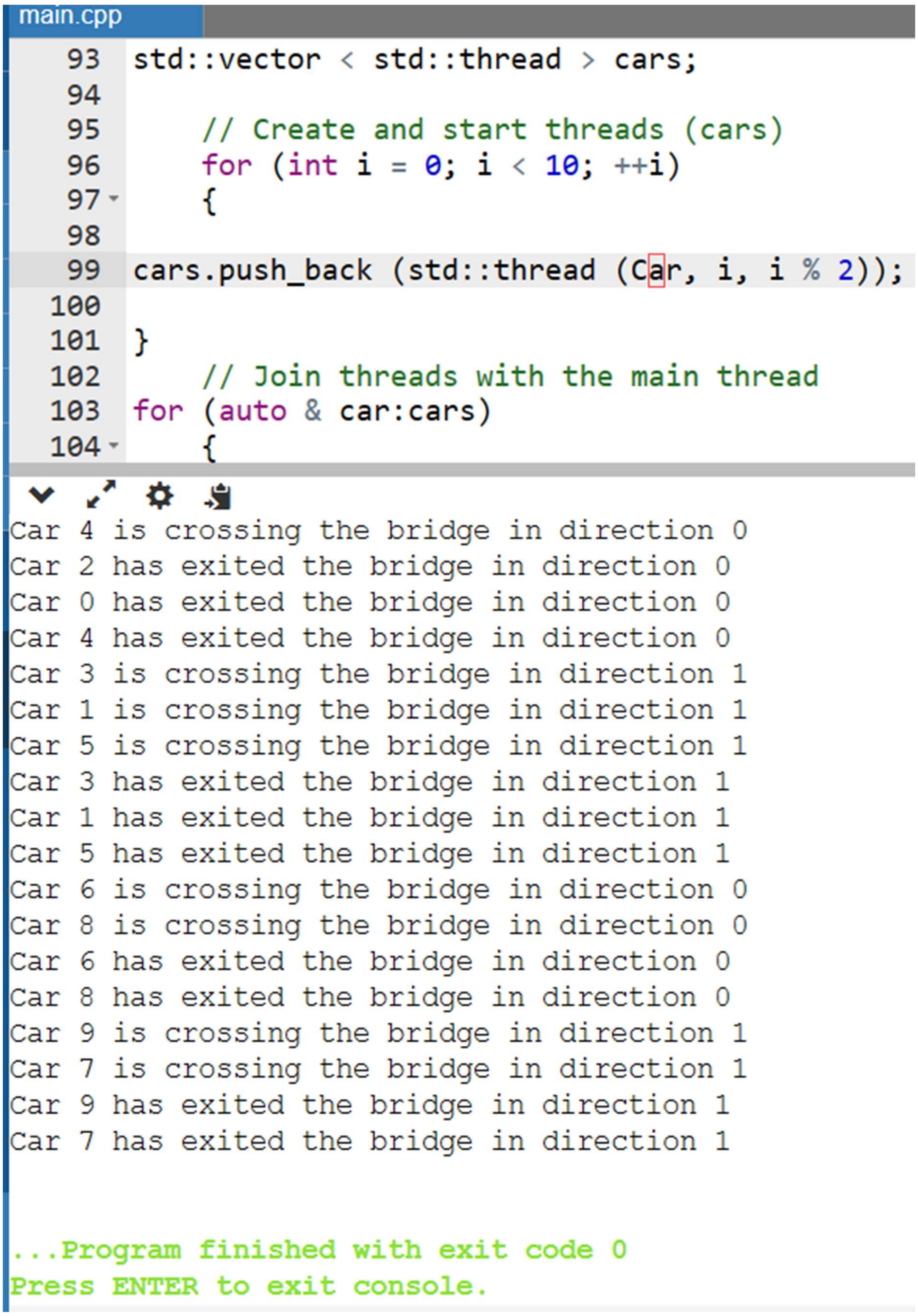
- **Mutex:** A single mutex can be used to control access to the bridge. Only one thread (vehicle) can acquire the mutex at a time, ensuring mutual exclusion and preventing overloading.
- **Semaphores:** Two semaphores can be used, one for light vehicles and one for heavy vehicles. Each semaphore controls the number of vehicles of its type allowed on the bridge, ensuring both mutual exclusion and fairness.
- **Atomic Operations:** Languages that support atomic operations can be used to update the bridge state (e.g., the number of vehicles on it) atomically, preventing data races.

Screenshot of the Runtime:



```
std::vector < std::thread > cars;

    // Create and start threads (cars)
    for (int i = 0; i < 10; ++i)
    {

cars.push_back (std::thread (Car, i, i % 2));

}
    // Join threads with the main thread
for (auto & car:cars)
    {
```

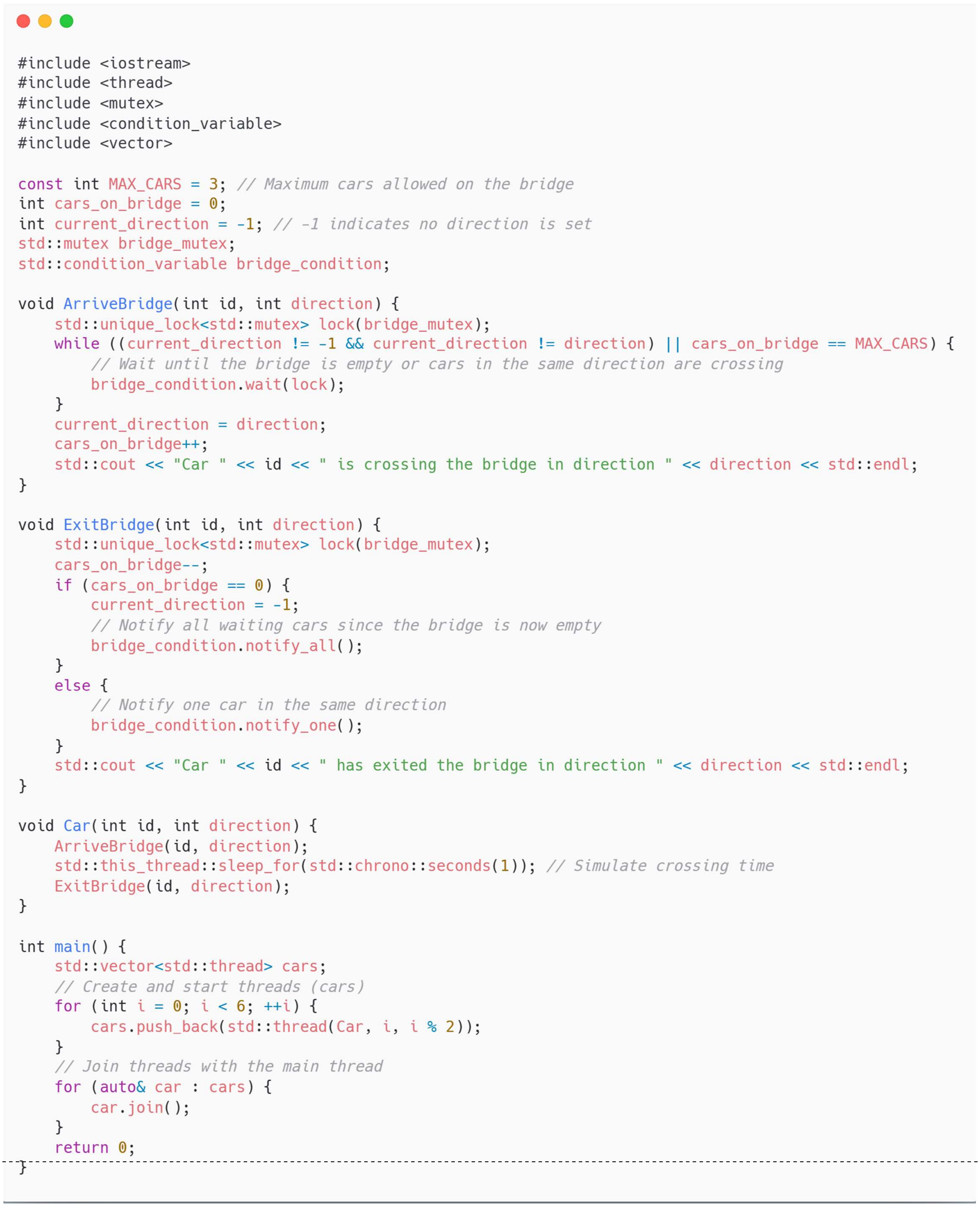
```
Car 1 is crossing the bridge in direction 1
Car 3 is crossing the bridge in direction 1
Car 5 is crossing the bridge in direction 1
Car 1 has exited the bridge in direction 1
Car 3 has exited the bridge in direction 1
Car 5 has exited the bridge in direction 1
Car 7 is crossing the bridge in direction 1
Car 9 is crossing the bridge in direction 1
Car 7 has exited the bridge in direction 1
Car 9 has exited the bridge in direction 1
Car 8 is crossing the bridge in direction 0
Car 2 is crossing the bridge in direction 0
Car 0 is crossing the bridge in direction 0
Car 8 has exited the bridge in direction 0
Car 6 is crossing the bridge in direction 0
Car 0 has exited the bridge in direction 0
Car 4 is crossing the bridge in direction 0
Car 2 has exited the bridge in direction 0
Car 6 has exited the bridge in direction 0
Car 4 has exited the bridge in direction 0
```



```cpp
std::vector < std::thread > cars;

    // Create and start threads (cars)
    for (int i = 0; i < 10; ++i)
    {

cars.push_back (std::thread (Car, i, i % 2));

}
    // Join threads with the main thread
for (auto & car:cars)
    {
```

```
Car 4 is crossing the bridge in direction 0
Car 2 has exited the bridge in direction 0
Car 0 has exited the bridge in direction 0
Car 4 has exited the bridge in direction 0
Car 3 is crossing the bridge in direction 1
Car 1 is crossing the bridge in direction 1
Car 5 is crossing the bridge in direction 1
Car 3 has exited the bridge in direction 1
Car 1 has exited the bridge in direction 1
Car 5 has exited the bridge in direction 1
Car 6 is crossing the bridge in direction 0
Car 8 is crossing the bridge in direction 0
Car 6 has exited the bridge in direction 0
Car 8 has exited the bridge in direction 0
Car 9 is crossing the bridge in direction 1
Car 7 is crossing the bridge in direction 1
Car 9 has exited the bridge in direction 1
Car 7 has exited the bridge in direction 1


...Program finished with exit code 0
Press ENTER to exit console.
```

The Code of the Problem :



```cpp
#include <iostream>
#include <thread>
#include <mutex>
#include <condition_variable>
#include <vector>

const int MAX_CARS = 3; // Maximum cars allowed on the bridge
int cars_on_bridge = 0;
int current_direction = -1; // -1 indicates no direction is set
std::mutex bridge_mutex;
std::condition_variable bridge_condition;

void ArriveBridge(int id, int direction) {
    std::unique_lock<std::mutex> lock(bridge_mutex);
    while ((current_direction != -1 && current_direction != direction) || cars_on_bridge == MAX_CARS) {
        // Wait until the bridge is empty or cars in the same direction are crossing
        bridge_condition.wait(lock);
    }
    current_direction = direction;
    cars_on_bridge++;
    std::cout << "Car " << id << " is crossing the bridge in direction " << direction << std::endl;
}

void ExitBridge(int id, int direction) {
    std::unique_lock<std::mutex> lock(bridge_mutex);
    cars_on_bridge--;
    if (cars_on_bridge == 0) {
        current_direction = -1;
        // Notify all waiting cars since the bridge is now empty
        bridge_condition.notify_all();
    }
    else {
        // Notify one car in the same direction
        bridge_condition.notify_one();
    }
    std::cout << "Car " << id << " has exited the bridge in direction " << direction << std::endl;
}

void Car(int id, int direction) {
    ArriveBridge(id, direction);
    std::this_thread::sleep_for(std::chrono::seconds(1)); // Simulate crossing time
    ExitBridge(id, direction);
}

int main() {
    std::vector<std::thread> cars;
    // Create and start threads (cars)
    for (int i = 0; i < 6; ++i) {
        cars.push_back(std::thread(Car, i, i % 2));
    }
    // Join threads with the main thread
    for (auto& car : cars) {
        car.join();
    }
    return 0;
}
```