```
class Allocator {
    Mutex lock
    Node* head
    Function myMalloc(int ID, int size){
        lock.Acquire()
        // Allocation logic
        lock.Release()
    }
    Function Deallocate(int ID, int index) {
        lock.Acquire()
        // Deallocation logic
        lock.Release()
    }
}
```

My allocator class has the private field std::mutex mtx;. In myMalloc and myFree functions I used the locking mechanism std::lock_guard<std::mutex> lock(mtx); because I think atomicity is needed for the whole body of these functions. Assume a thread executes Node* curr = head; (first line of both myMalloc and myFree) which is not in critical section and then some other thread updated the linked lists which is in critical section. In this scenario the first thread would have worked on the not updated linked lists so the whole body of the function should be executed atomically in my opinion. std::lock_guard mechanism acquires the lock whenever initialized and releases the lock when the execution of the thread on the function is done. I thought this mechanism is simple and meet my requirements so I chose it.