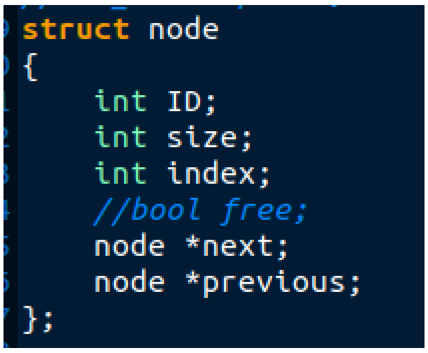
REPORT

In my class implementation, I used doubly linked list. In other words, all nodes have both next and previous pointers. Typical node structure can be seen below:



```
struct node
{
    int ID;
    int size;
    int index;
    //bool free;
    node *next;
    node *previous;
};
```

I used a global Mutex to have synchronization. In other words, before both functions myFree() and myMalloc(), I locked the mutex and before returning I unlocked it. Therefore, there will be no race condition between threads because my lock ensures the function's statements are atomic.

myMalloc() function as pseudo:

-Lock the Mutex

-Find a node with ID == -1 and its size should be larger than requested size.

-Make necessary updates (create a new node in the left and update size value of free memory)

-Print out the operation result. (Allocation succeed or not)

-Unlock the Mutex

-Return

myFree() function as pseudo:

-Lock the Mutex

-Find the node with given index and ID

-Check neighbours whether they are free memory or not

-Freed the allocated memory and make necessary combinations according to upper step's results

-If there are any combinations of nodes, make sure that there is no memory leak (delete properly old memory nodes)

-Print out the operation result

-Unlock the Mutex

-Return