Design a least-recently-used (LRU) cache (storing integers) that supports:

- (1) Inserting an element into a cache
    - (2) If the cache is full: evict the least recently used member
- (3) Getting the most-recently-used element in the cache
- (4) Accessing an element in the cache
    - Cache.access(1)

Map[integer value] => Node*

```
Linked List: {
        // Most recently accessed
        Node* head;

        // Least recently accessed
        Node* last;
}

struct Node {
        Int val;
        Node* next;
        Node* prev;
}

Node insert (int x) {
        // Check not in map already
        If x is not in map:

        // Create a new Node

        // If (map.size > n) { evictOldest };

        // Set the head of linked list equal t new Node

        // Set the next of node to the old head

        // map[x] = &newNode;

        // return newNode;
}

evictOldest () {

        // copy last.prev
```

```
        // delete map[last]

        // free the memory associated with Node*last

        // set linked.last = last.prev; last.next = null;
}


[ 1 ] -> [ 2 ] -> [ 3 ]
          |

Pointer             Node
0x23423423      { .val = 234, .next = 0x234234 }

Access(2)

Node access(int x) {
        // Check element exists in the map
        If (Map[x] exists) {

                linkedlist.delete(map[x]);

                map[x].delete;

                Node newNode = insert(x);

                Return newNode;
        }
        Else {
                Throw error
        }
}

// Delete node
Boolean delete(int x) {
        // Check exists
        If map[x]:
                If (map[x].prev == null) {
                        Linkedlist.head = map[x].next;
                        Map[x].next.prev = null;
                        free(map[x]);
                }
                If (map[x].next == null) {
                        Linkedlist.tail = map[x].prev;
```

```
                    Map[x].prev.next = null;
                    free(map[x]));
                }
                Else {
                    Map[x].prev.next = map[x].next;
                    Map[x].next.prev = map[x].prev;
                    Free(map[x]);
                }
                Return true;
        }
        Return false;
}
```