COP 2535: Data Structures

Lab 04, LRU wth Circular List

1. Read pages 91 – 96 in Mastering Algorithms with C
2. Implement the following program.
3. Upload the output of your execution as text.

Save as `LRUCache.cpp`

```
//https://bhrigu.me/post/lru-cache-c-plus-plus-implementation/
#include <iostream>
#include <map>
using namespace std;
class Node {
public:
    int key, value;
    Node* prev, * next;
    Node(int k, int v) : key(k), value(v), prev(NULL), next(NULL) {}
};

class DoublyLinkedList {
    Node* front, * rear;

    bool isEmpty() {
        return rear == NULL;
    }

public:
    DoublyLinkedList() : front(NULL), rear(NULL) {}

    Node* add_page_to_head(int key, int value) {
        Node* page = new Node(key, value);
        if (!front && !rear) {
            front = rear = page;
        }
        else {
            page->next = front;
            front->prev = page;
            front = page;
        }
        return page;
    }

    void move_page_to_head(Node* page) {
        if (page == front) {
            return;
        }
        if (page == rear) {
            rear = rear->prev;
            rear->next = NULL;
        }
        else {
            page->prev->next = page->next;
            page->next->prev = page->prev;
        }

        page->next = front;
        page->prev = NULL;
        front->prev = page;
        front = page;
    }

    void remove_rear_page() {
        if (isEmpty()) {
            return;
        }
        if (front == rear) {
            delete rear;
            front = rear = NULL;
        }
        else {
            Node* temp = rear;
            rear = rear->prev;
            rear->next = NULL;
            delete temp;
        }
    }
    Node* get_rear_page() {
        return rear;
    }

};

class LRUCache {
    int capacity, size;
    DoublyLinkedList* pageList;
    map<int, Node*> pageMap;

public:
    LRUCache(int capacity) {
        this->capacity = capacity;
        size = 0;
        pageList = new DoublyLinkedList();
        pageMap = map<int, Node*>();
    }

    int get(int key) {
        if (pageMap.find(key) == pageMap.end()) {
            return -1;
        }
        int val = pageMap[key]->value;
        // move the page to front
        pageList->move_page_to_head(pageMap[key]);
        return val;
    }

    void put(int key, int value) {
        if (pageMap.find(key) != pageMap.end()) {
            // if key already present, update value and move page to head
            pageMap[key]->value = value;
            pageList->move_page_to_head(pageMap[key]);
            return;
        }

        if (size == capacity) {
            // remove rear page
            int k = pageList->get_rear_page()->key;
            pageMap.erase(k);
            pageList->remove_rear_page();
            size--;
        }

        // add new page to head to Queue
        Node* page = pageList->add_page_to_head(key, value);
        size++;
        pageMap[key] = page;
    }

    ~LRUCache() {
        map<int, Node*>::iterator i1;
        for (i1 = pageMap.begin(); i1 != pageMap.end(); i1++) {
            delete i1->second;
        }
        delete pageList;
    }
};
```

Save as `Source.cpp`

```
//https://bhrigu.me/post/lru-cache-c-plus-plus-implementation/
#include <iostream>
#include "LRUCache.cpp"
using namespace std;

int main() {
        LRUCache cache(2);        // cache capacity 2
        cache.put(2, 2);
        cout << cache.get(2) << endl;
        cout << cache.get(1) << endl;
        cache.put(1, 1);
        cache.put(1, 5);
        cout << cache.get(1) << endl;
        cout << cache.get(2) << endl;
        cache.put(8, 8);
        cout << cache.get(1) << endl;
        cout << cache.get(8) << endl;

}
```