typedef struct Node{

    int key;

    int val;

    struct Node\* pre ;

    struct Node\* next ;

    UT\_hash\_handle hh; /\* makes this structure hashable \*/

} Node;

typedef struct LRUCache{//資料結構中含有左側跟右側Node的節點

    int cap;

    struct Node\* LEFT ;

    struct Node\* RIGHT ;

} LRUCache;

struct Node \*hashtable = NULL;  /\* makes this structure hashable \*/

void addNode( LRUCache\* obj, Node\* node ){//增加節點

    Node \*pre\_node = obj->RIGHT->pre;//將右側node節點的前一個節點紀錄

    Node \*next\_node = obj->RIGHT;//將右側node節點紀錄

    pre\_node->next=node;//前一個節點的next指項新node

    next\_node->pre=node;//將右側節點pre指向前一個新的node

    node->next=next\_node;//將新的節點的next指向右側節點

    node->pre=pre\_node;//將新的節點的pre指向前一個節點

}

void rmNode( Node\* tag ){//刪除節點

        tag->next->pre=tag->pre;//將該節點的下一個的pre指向該解點的pre

        tag->pre->next=tag->next;//將該節點的上一個的next指向該解點的next

}

LRUCache\* lRUCacheCreate(int capacity) {//製作出開頭資料結構

    //製作出一個LRUCache的資料結構包含兩個node的資料結構

    LRUCache \*LRU = malloc(sizeof(LRUCache));

    LRU->LEFT = malloc(sizeof(Node));

    LRU->RIGHT = malloc(sizeof(Node));

    LRU->cap=capacity;//輸入可容納的量

    //LRU的LEFT與相接一個nodes

    LRU->LEFT->pre=NULL;

    LRU->LEFT->next=LRU->RIGHT;

    //LRU的RIGHT與相接一個nodes

    LRU->RIGHT->next=NULL;

    LRU->RIGHT->pre=LRU->LEFT;

    return LRU;

}

int lRUCacheGet(LRUCache\* obj, int key) {

    //printf("\n");

    //printf("lRUCacheGet\n");

    struct Node \*s;

    HASH\_FIND\_INT( hashtable, &key, s );  /\* s: output pointer \*/

    int ret\_val=-1;

    int ret\_key=-1;

    if( s == NULL){

        //printf("find %s","NULL");

    }else{//如果找到值

        rmNode( s );//就先刪除該節點

        ret\_val=s->val;//更新新的值

        addNode(  obj,s);//再增加該節點//這樣才能將動過的值放到右側

    }

  return ret\_val;

}

void lRUCachePut(LRUCache\* obj, int key, int value) {

   // printf("\n");

    //printf("lRUCachePut\n");

    Node \*s;

    HASH\_FIND\_INT( hashtable, &key, s );  /\* s: output pointer \*/

    if(s){//如果該節點值已經存在

        rmNode( s );//就先刪除該節點

        s->val=value;//更新新的值

    }else{//如果沒有該節點

        s = malloc(sizeof(Node));//製作出新的節點

        s->key=key;

        s->val=value;

        HASH\_ADD\_INT( hashtable,key,s );

    }

    addNode( obj, s);//新增該節點

    if(HASH\_COUNT(hashtable) > obj->cap){//檢查hashtable內的值是否大於Cap(題目可容納量)

        Node\* LRUNode = obj->LEFT->next;//如果太大就要將左側節點刪掉

        rmNode(LRUNode);//如果太大就要將左側節點刪掉

        HASH\_DEL( hashtable,LRUNode);//並且將hashtable中的紀錄刪除

        free(LRUNode);//free掉節點記憶體位置

    }

    // HASH\_FIND\_INT( hashtable, &key, s );

    // printf("find %d %d",s->val,s->key);

}

void lRUCacheFree(LRUCache\* obj) {

    // Deallocate the nodes first

    free(obj->LEFT);

    free(obj->RIGHT);

    // Deallocate the LRUCache

    free(obj);

    // Delete and deallocate all the nodes from the hash(如果不做這個，hashtable的值可能會帶到下一題中)

    Node \*currentNode, \*tmp;

    HASH\_ITER(hh, hashtable, currentNode, tmp) {

        HASH\_DEL(hashtable, currentNode);  /\* delete; nodes advances to next \*/

        free(currentNode);

   }

}

/\*\*

 \* Your LRUCache struct will be instantiated and called as such:

 \* LRUCache\* obj = lRUCacheCreate(capacity);

 \* int param\_1 = lRUCacheGet(obj, key);

 \* lRUCachePut(obj, key, value);

 \* lRUCacheFree(obj);

\*/