小试牛刀，单链表练习题

## 1. 反转链表

[https://leetcode.cn/problems/reverse-linked-list/description/](https://leetcode.cn/problems/reverse-linked-list/description/)

```cpp

/\*

解题思路： 此题一般常用的方法有两种，三指针翻转法和头插法

1. 三指针翻转法

记录连续的三个节点，原地修改节点指向

2. 头插法

每一个节点都进行头插

\*/

// 三个指针翻转的思想完成逆置

struct ListNode\* reverseList(struct ListNode\* head) {

if(head == NULL || head->next == NULL)

return head;

struct ListNode\* n1, \*n2, \*n3;

n1 = head;

n2 = n1->next;

n3 = n2->next;

n1->next = NULL;

//中间节点不为空，继续修改指向

while(n2)

{

//中间节点指向反转

n2->next = n1;

//更新三个连续的节点

n1 = n2;

n2 = n3;

if(n3)

n3 = n3->next;

}

//返回新的头

return n1;

}

// 取节点头插的思想完成逆置

struct ListNode\* reverseList(struct ListNode\* head) {

struct ListNode\* newhead = NULL;

struct ListNode\* cur = head;

while(cur)

{

struct ListNode\* next = cur->next;

//头插新节点，更新头

cur->next = newhead;

newhead = cur;

cur = next;

}

return newhead;

}

```

---

## 2. 链表的中间结点

[https://leetcode.cn/problems/middle-of-the-linked-list/description/](https://leetcode.cn/problems/middle-of-the-linked-list/description/)

```cpp

/\*

解题思路：

通过快慢指针找到中间节点，快指针每次走两步，慢指针每次走一步，当快指针走到结尾的时候，慢指针正好走到中间位置

\*/

typedef struct ListNode Node;

struct ListNode\* middleNode(struct ListNode\* head){

Node\* slow = head;

Node\* fast = head;

while(fast!=NULL && fast->next != NULL)

{

slow = slow->next;

fast = fast->next->next;

}

return slow;

}

```

---

## 3. 合并两个有序链表

[https://leetcode.cn/problems/merge-two-sorted-lists/description/](https://leetcode.cn/problems/merge-two-sorted-lists/description/)

```cpp

/\*

解题思路：

此题可以先创建一个空链表，然后依次从两个有序链表中选取最小的进行尾插操作进行合并。

\*/

typedef struct ListNode Node;

struct ListNode\* mergeTwoLists(struct ListNode\* l1, struct ListNode\* l2){

if(l1 == NULL)

return l2;

else if(l2 == NULL)

return l1;

Node\* head = NULL, \*tail = NULL;

//创建空链表

head = tail = (Node\*)malloc(sizeof(Node));

tail->next = NULL;

while(l1 && l2)

{

// 取小的进行尾插

if(l1->val < l2->val)

{

tail->next = l1;

tail = tail->next;

l1 = l1->next;

}

else

{

tail->next = l2;

tail = tail->next;

l2 = l2->next;

}

}

//剩余元素直接拼接

if(l1)

tail->next = l1;

else

tail->next = l2;

Node\* list = head->next;

free(head);

return list;

}

```

---

## 4. 链表分割

[https://www.nowcoder.com/practice/0e27e0b064de4eacac178676ef9c9d70](https://www.nowcoder.com/practice/0e27e0b064de4eacac178676ef9c9d70)

```cpp

/\*

解题思路

创建两个链表，分别存放小于x的节点和大于等于x的节点，分别进行尾插

\*/

class Partition {

public:

ListNode\* partition(ListNode\* pHead, int x) {

if(pHead == NULL)

return NULL;

struct ListNode\* lessHead, \*lessTail,\*greaterHead, \*greaterTail;

//创建链表表头

lessHead = lessTail = (struct ListNode\*)malloc(sizeof(struct ListNode));

greaterHead = greaterTail = (struct ListNode\*)malloc(sizeof(struct ListNode));

struct ListNode\* cur = pHead;

while(cur)

{

//小于x的尾插到lessTail

if(cur->val < x)

{

lessTail->next = cur;

lessTail = lessTail->next;

}

//大于等于x的尾插到greaterTail

else

{

greaterTail->next = cur;

greaterTail = greaterTail->next;

}

cur = cur->next;

}

//链接两个链表

lessTail->next = greaterHead->next;

greaterTail->next = NULL;

//获取表头

pHead = lessHead->next;

free(lessHead);

free(greaterHead);

return pHead;

}

};

```

---

## 5. 链表的回⽂结构

[https://www.nowcoder.com/practice/d281619e4b3e4a60a2cc66ea32855bfa](https://www.nowcoder.com/practice/d281619e4b3e4a60a2cc66ea32855bfa)

```cpp

/\*

解题思路：

此题可以先找到中间节点，然后把后半部分逆置，最近前后两部分一一比对，如果节点的值全部相同，则即为回文。

\*/

class PalindromeList {

public:

bool chkPalindrome(ListNode\* A) {

if (A == NULL || A->next == NULL)

return true;

ListNode\* slow, \*fast, \*prev, \*cur, \*nxt;

slow = fast = A;

//找到中间节点

while (fast && fast->next)

{

slow = slow->next;

fast = fast->next->next;

}

prev = NULL;

//后半部分逆置

cur = slow;

while (cur)

{

nxt = cur->next;

cur->next = prev;

prev = cur;

cur = nxt;

}

//逐点比对

while (A && prev)

{

if (A->val != prev->val)

return false;

A = A->next;

prev = prev->next;

}

return true;

}

};

/\*

此题也可以先把链表中的元素值全部保存到数组中，然后再判断数组是否为回文。不建议使用这种解法，因为如果没有告诉链表最大长度，则不能同此解法

\*/

```

---

## 6. 相交链表

[https://leetcode.cn/problems/intersection-of-two-linked-lists/description/](https://leetcode.cn/problems/intersection-of-two-linked-lists/description/)

```cpp

/\*

解题思路：

此题可以先计算出两个链表的长度，让长的链表先走相差的长度，然后两个链表同时走，直到遇到相同的节点，即为第一个公共节点

\*/

struct ListNode \*getIntersectionNode(struct ListNode \*headA, struct ListNode \*headB) {

int lenA = 0, lenB = 0;

struct ListNode\* curA = headA, \*curB = headB;

//计算链表长度

while(curA) {

++lenA;

curA = curA->next;

}

while(curB) {

++lenB;

curB = curB->next;

}

int gap = abs(lenA-lenB);

struct ListNode\* longList = headA, \*shortList = headB;

if(lenA < lenB) {

longList = headB;

shortList = headA;

}

//让长链表先走几步

while(gap--){

longList = longList->next;

}

//两个链表同时走，直到遇到相同的节点

while(longList && shortList)

{

if(longList == shortList) {

return longList;

}

else {

longList = longList->next;

shortList = shortList->next;

}

}

return NULL;

}

```

---

## 7. 环形链表I

[https://leetcode.cn/problems/linked-list-cycle/description/](https://leetcode.cn/problems/linked-list-cycle/description/)

```cpp

/\*

解题思路：

定义快慢指针fast,slow, 如果链表确实有环，fast指针一定会在环内追上slow指针。

\*/

typedef struct ListNode Node;

bool hasCycle(struct ListNode \*head) {

Node\* slow = head;

Node\* fast = head;

while(fast && fast->next)

{

slow = slow->next;

fast = fast->next->next;

if(slow == fast)

return true;

}

return false;

}

```

---

## 8. 环形链表II

[https://leetcode.cn/problems/linked-list-cycle-ii/description/](https://leetcode.cn/problems/linked-list-cycle-ii/description/)

```cpp

/\*

解题思路：

如果链表存在环，则fast和slow会在环内相遇，定义相遇点到入口点的距离为X,定义环的长度为C,定义头到入口的距离为L,fast在slow进入环之后一圈内追上slow,则会得知：

slow所走的步数为:L + X

fast所走的步数为：L + X + N \* C

并且fast所走的步数为slow的两倍，故：

2\*(L + X) = L + X + N \* C

即： L = N \* C - X

所以从相遇点开始slow继续走，让一个指针从头开始走，相遇点即为入口节点

\*/

typedef struct ListNode Node;

struct ListNode \*detectCycle(struct ListNode \*head) {

Node\* slow = head;

Node\* fast = head;

while(fast && fast->next)

{

slow = slow->next;

fast = fast->next->next;

//走到相遇点

if(slow == fast)

{

// 求环的入口点

Node\* meet = slow;

Node\* start = head;

while(meet != start)

{

meet = meet->next;

start = start->next;

}

return meet;

}

}

return NULL;

}

```

---

## 9. 随机链表的复制

[https://leetcode.cn/problems/copy-list-with-random-pointer/description/](https://leetcode.cn/problems/copy-list-with-random-pointer/description/)

```cpp

/\*

解题思路：

此题可以分三步进行：

1.拷贝链表的每一个节点，拷贝的节点先链接到被拷贝节点的后面

2.复制随机指针的链接：拷贝节点的随机指针指向被拷贝节点随机指针的下一个位置

3.拆解链表，把拷贝的链表从原链表中拆解出来

\*/

class Solution {

public:

Node\* copyRandomList(Node\* head) {

// 1.拷贝链表，并插入到原节点的后面

Node\* cur = head;

while(cur)

{

Node\* next = cur->next;

Node\* copy = (Node\*)malloc(sizeof(Node));

copy->val = cur->val;

// 插入

cur->next = copy;

copy->next = next;

// 迭代往下走

cur = next;

}

// 2.置拷贝节点的random

cur = head;

while(cur)

{

Node\* copy = cur->next;

if(cur->random != NULL)

copy->random = cur->random->next;

else

copy->random = NULL;

cur = copy->next;

}

// 3.解拷贝节点，链接拷贝节点

Node\* copyHead = NULL, \*copyTail = NULL;

cur = head;

while(cur)

{

Node\* copy = cur->next;

Node\* next = copy->next;

// copy解下来尾插

if(copyTail == NULL)

{

copyHead = copyTail = copy;

}

else

{

copyTail->next = copy;

copyTail = copy;

}

cur->next = next;

cur = next;

}

return copyHead;

}

};

```

---