**实验二**

**一、实验题目：**自行输入两个无序链表A和B，要求均带有头结点，A以单链表形式存储，B以循环链表形式存储。

(1)将A和B在自身基础上分别就地逆置，要求不占用额外的存储空间；

(2)删除A链表中节点值与B链表中节点值相同的结点；

(3)找到A链表经过（2）操作后的倒数第k个结点，k要求手动输入。

**二、关键代码：**

**(1)单链表删除节点函数：**

void Delete(int n) {

if (front == NULL) { return; }//空链表返回

//只有一个头节点时

if (front->data == n && front->next == NULL) {

front = NULL;

count--;

return;

}

while (front->data == n) {

if (front->data == n && front->next != NULL) {

front = front->next;

count--;

}

if (front->data == n && front->next == NULL) {

front = NULL;

count--;

return;

}

if (front == NULL)return;

}

Node\* m = new Node();

Node\* p = front;//p一直在m前面，这样好删除

m = p->next;

while (true) {

if (m == NULL)return;//到末尾了就结束

//末尾情况

if (m->data == n && m->next == NULL) {

p->next = NULL;

count--;

return;

}

//中间情况

else {

if (m->data == n) {

p->next = m->next;

count--;

}

m = m->next;

p = p->next;

if (m == NULL)return;//到末尾了就结束

}

}

}

**(2)单链表插入函数：**

void insert(int n) {

Node\* p = new Node(n);

if (front == NULL) {

front = p;

count++;

}

else {

Node\* temp = front;//用来找尾节点

while (temp->next != NULL) {

temp = temp->next;

}//找到尾节点

temp->next = p;

count++;

}

}

**(3)单链表逆置函数：**

void reverse() {

if (front == NULL || front->next == NULL) { return; }//空链表或者只有一个

//利用三个指针

Node\* x = new Node();

Node\* y = new Node();

Node\* z = new Node();

x = front->next;//防止失去联系

front->next = NULL;//将头节点孤立

y = x->next;

x->next = front;

while (1) {

z = x;

x = y;

if (x == NULL)break;

y = x->next;

x->next = z; }

front = z; }};

**(4)循环链表逆置函数：**

void reverse() {

if (front == tail) { return; }

//利用两个指针

Node\* cur1 = new Node();

Node\* cur2 = new Node();

cur1 = front->next;

front->next = NULL;//将头节点孤立

tail->next = front;

cur2 = cur1->next;

while (cur1 != tail) {

cur1->next = tail->next;

tail->next = cur1;

cur1 = cur2;

cur2 = cur1->next;

}

tail = front;

front = cur1;

tail->next = front;//因为是循环链表，所以逆置后也不要忘记这步

}

**(5)循环链表插入函数：**

void insert(int n) {

Node\* p = new Node(n);

if (front == NULL && tail == NULL) {

front = tail = p;

tail->next = front;

}

else {

tail->next = p;

tail = p;

tail->next = front; } }};

**(6)删除相同数据：**

j = B.front;

A.Delete(j->data);

//查找与B中数据相同的节点

j = B.front->next;

while (j != B.front) {

A.Delete(j->data);

j = j->next;

}

**(7)追击指针：**

if (k <= 0 || k > A.count) { cout << "k值超过范围 "; }

else {

cout << "A中倒数第k个节点为： ";

i = A.front;

j = A.front;

for (int m = 0; m < k - 1; m++) {

j = j->next;

}//先让j领先i k步

for (; j->next!=NULL; ) {

j = j->next;

i = i->next;

}//当j到尽头，i指向倒数第k个节点

1. **复杂度分析：**
2. **逆置函数**：

**空间复杂度：** 因为只多运用了几个指针，所以为O(1)

**时间复杂度:** 需要从链表头到链表尾，指针反复进行改变指向，所以两个链表的逆置函数分别为3n和2n，则为O(n)

1. **删除操作：**

**空间复杂度：**没有与n有关，所以为0

**时间复杂度：**需要双重循环进行比较，还需要调用删除函数进行删除，所以为O(n²)

1. **查找函数：**

**空间复杂度：**0

**时间复杂度：**O(n)

1. **心得体会：**
2. **在写链表自身逆置函数时，要有逻辑性，找到规律用循环实现。**
3. **写链表的插入和删除时，一定要谨慎指针是否指空，并在后面的过程要进行判断指针是否为空的操作，来防止产生bug。**
4. **使用ctrl z停止循环时，要注意已经把输入流置入错误位置，后面要想继续输出，则要使用cin.clear（）函数来恢复正常。**