

# 实验二实验报告

2112514 辛浩然

## 实验内容

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。

## 实验代码

```
#include <iostream>
using namespace std;
class ListNode
{
public:
    int data;
    ListNode* link;
    ListNode() : data(0), link(0) {}
    ListNode(int x) : data(x), link(0) {}
};
class LinkedList
{
private:
    ListNode* first;
    int num;

public:
    LinkedList()
    {
        num = 0;
        first = new ListNode(0);
    }
    ~LinkedList() //从第一个节点遍历析构
    {
        ListNode* next;
        while (first)
        {
            next = first->link;
            delete first;
            first = next;
        }
        delete first;
    }
    void Create() //尾插
```

```

{
    ListNode* p = first;
    int x;
    while (cin >> x)
    {
        ListNode* newNode = new ListNode(x);
        p->link = newNode;
        p = newNode;
        num++;
        if (cin.get() == '\n')
        {
            break;
        }
    }
}

void Output()
{
    ListNode* p = first;
    while (p)
    {
        p = p->link;
        if (p)
            cout << p->data << ' ';
    }
    cout << endl;
}

void Reverse() //将头节点指向nullptr, 后面的节点依次插入头节点之后
{
    ListNode* p, * q, * r;
    p = first;
    q = first->link;
    p->link = nullptr;
    while (q)
    {
        r = q->link;
        q->link = p->link;
        p->link = q;
        q = r;
    }
}

void DeleteDuplicates(ListNode* yFirst)
{
    ListNode* y = yFirst->link;
    while (y && y != yFirst)
    { //遍历B的节点, 比较
        ListNode* p = first, * q = first->link;
        while (q)
        {
            if (q->data == y->data) //数据相同, 删除节点, 继续遍历A
            {
                p->link = q->link;
                delete q;
                num--;
                q = p->link;
            }
        }
    }
}

```

```

        }
        else //不相同继续遍历A
        {
            p = q;
            q = q->link;
        }
    }
    y = y->link;
}

void SearchNthFromtheEnd(int n)
{
    if (n < 1 || n > num)
    {
        cout << "ERROR" << endl;
        return;
    }
    ListNode* p = first, * q = first;
    while (n--)
    { //快指针先走n步
        p = p->link;
    }
    while (p)
    { //快指针到最后一个节点时，慢指针到达目标节点
        p = p->link;
        q = q->link;
    }
    cout << q->data;
    return;
}

};

class CircleList // 循环链表类
{
private:
    ListNode* first;
    int num;

public:
    CircleList()
    {
        num = 0;
        first = new ListNode(0);
        first->link = first;
    }
    ~CircleList()
    {
        ListNode* p, * q;
        p = first->link;
        while (p != first)
        {
            q = p->link;
            delete p;
            p = q;
        }
    }
}

```

```

        if (p == first)
            p->link = first;
        //成为只有头节点的空表
        delete first;
        //删除头节点
    }
    ListNode* getFirst()
    {
        return first;
    }
    void Create()
    {
        ListNode* p = first;
        int x;
        while (cin >> x)
        {
            ListNode* newNode = new ListNode(x);
            p->link = newNode;
            p = newNode;
            num++;
            if (cin.get() == '\n')
            {
                break;
            }
        }
        p->link = first; //尾指针指向头节点
    }
    void Output()
    {
        ListNode* p = first;
        while (p->link != first)
        {
            p = p->link;
            cout << p->data << ' ';
        }
        cout << endl;
    }
    void Reverse()
    {
        ListNode* p, * q, * r;
        p = first;
        q = first->link;
        p->link = p;
        while (q != first)
        {
            r = q->link;
            q->link = p->link;
            p->link = q;
            q = r;
        }
    }
};
int main()
{

```

```

LinkedList x;
CircleList y;
cout << "Enter list A:" << endl;
x.Create();
cout << "Enter list B:" << endl;
y.Create();
cout << "Reverse the lists:" << endl;
x.Reverse();
y.Reverse();
x.Output();
y.Output();
cout << "Delete the node with the same data as list B in list A:" << endl;
x.DeleteDuplicates(y.getFirst());
x.Output();
cout << "Enter the Nth node from the end to be searched:";
int n;
cin >> n;
cout << "The data of the node is:";
x.SearchNthFromtheEnd(n);
}

```

## 实验测试

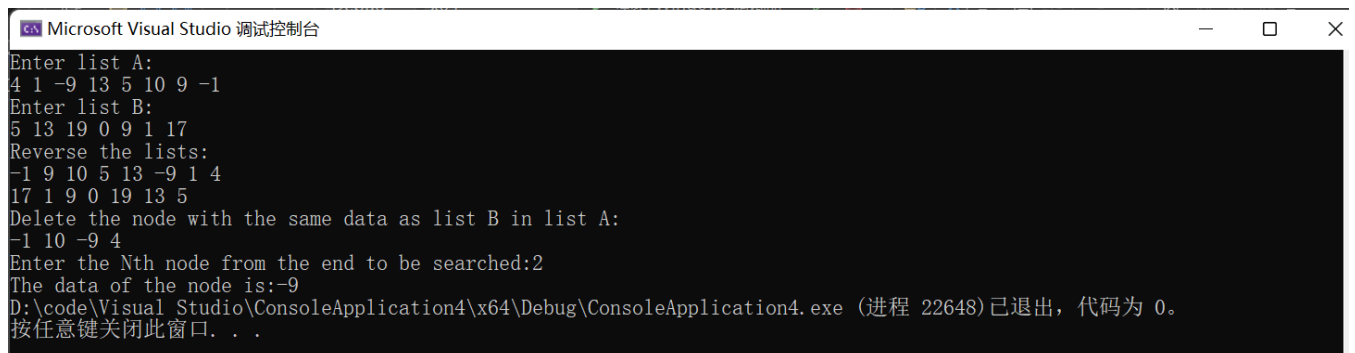
### 测试用例一

输入:

4 1 -9 13 10 9 1

5 13 19 0 9 1 17

输出:



```

Microsoft Visual Studio 调试控制台
Enter list A:
4 1 -9 13 10 9 -1
Enter list B:
5 13 19 0 9 1 17
Reverse the lists:
-1 9 10 5 13 -9 1 4
17 1 9 0 19 13 5
Delete the node with the same data as list B in list A:
-1 10 -9 4
Enter the Nth node from the end to be searched:2
The data of the node is:-9
D:\code\Visual Studio\ConsoleApplication4\x64\Debug\ConsoleApplication4.exe (进程 22648) 已退出, 代码为 0。
按任意键关闭此窗口。 . .

```

### 测试用例二

输入:

1 2 3 4 5 6 7 8 9 10 11 12

12 11 10 9 8 7 6 5 4 3 2

1

输出:

```
Microsoft Visual Studio 调试控制台
Enter list A:
1 2 3 4 5 6 7 8 9 10 11 12
Enter list B:
12 11 10 9 8 7 6 5 4 3 2
Reverse the lists:
12 11 10 9 8 7 6 5 4 3 2 1
2 3 4 5 6 7 8 9 10 11 12
Delete the node with the same data as list B in list A:
1
Enter the Nth node from the end to be searched:1
The data of the node is:1
D:\code\Visual Studio\ConsoleApplication4\x64\Debug\ConsoleApplication4.exe (进程 4920)已退出，代码为 0。
按任意键关闭此窗口。 . . .
```

## 测试用例三

输入：

1 3 4 6 7 8 9 19 -3 -4 0 22 2 2 2 2 1

1 2 3 4 5 6 7 8 9 10

输出：

```
Microsoft Visual Studio 调试控制台
Enter list A:
1 3 4 6 7 8 9 19 -3 -4 0 22 2 2 2 2 1
Enter list B:
1 2 3 4 5 6 7 8 9 10
Reverse the lists:
1 2 2 2 2 22 0 -4 -3 19 9 8 7 6 4 3 1
10 9 8 7 6 5 4 3 2 1
Delete the node with the same data as list B in list A:
22 0 -4 -3 19
Enter the Nth node from the end to be searched:5
The data of the node is:22
D:\code\Visual Studio\ConsoleApplication4\x64\Debug\ConsoleApplication4.exe (进程 16876)已退出，代码为 0。
按任意键关闭此窗口。 . . .
```

## 算法分析

### 创建链表

尾插创建链表，生成结点接在头结点之后。

对于循环链表，在链表创建完毕后，将最后一个结点指向头结点。

### 链表逆置

逆置单链表，首先更改头结点的指向，使其指向nullptr，随后将原链表的各个结点依次插在头结点之后。

插入的具体实现：插入结点指向头结点指向的结点，头结点指向插入结点。

而对于循环链表的逆置，与单链表不同的地方在于将最初的头结点指向自身，随后的头插操作相同。

### 删除与B链表相同的结点

遍历B链表，对于B的每个结点，遍历A链表进行数据比较，若数据相同，则删除该结点，随后继续遍历；若不同，则继续遍历。

删除结点的具体实现：双指针一先一后一起移动，前指针移动到所要删除的结点时，后指针指向要删除结点前的结点，将前一结点的指向修改为要删除结点之后的结点。

## 找到倒数第k个结点

双指针都在头结点，快指针先走k步，随后两个指针同步向后移动，当快指针到最后一个结点时，慢指针到达目标节点，这样就找到了倒数第k个结点。

## 复杂度分析

### 链表逆置

时间复杂度：while循环执行n次，时间复杂度为 $O(n)$

空间复杂度： $O(1)$

### 删除与B链表相同的结点

时间复杂度：双层循环，时间复杂度为 $O(n^2)$

空间复杂度： $O(1)$

### 找到倒数第k个结点

时间复杂度：两个指针遍历链表，快指针遍历链表一次，慢指针遍历不完整的一次，时间复杂度为 $O(n)$

空间复杂度： $O(1)$

## 心得总结

- 1.对链表结构的认识更为深刻，熟练链表构造、删除、插入等基本操作。
- 2.在链表逆置操作时，一开始思路比较混乱，结点指向的处理出现错误，导致出现了bug；后来通过查找相关资料，理清了思绪，修改了出现的问题。
- 3.通过找倒数第k个结点，掌握了快慢指针的思路，可以迁移运用到一些其他问题，如判断链表是否存在环。