09020328_王亮_第4章作业

作业事项：

- 测试环境：VS2019_C++14 ->使用了C++11 auto nullptr explicit关键字；
- P184 T5 对于同等数值节点，不予合并，仍加入合并后链表；
- P184 T6中对于指针r设置为0的描述，采取NULL宏定义(#define 0)而未使用nullptr
- P184 T6默认left初始在0位置，right在1位置，主要是将首元素的next指针设为nullptr初始化
- 其余详见各处代码注释，后附完整代码CV可检验成效,或直接跳转最下方看效果
- 索引：P184_T5 P184_T6 P194_T4 完整代码 效果图

P184 T5 索引：P184_T5 P184_T6 P194_T4 完整代码 效果图复杂度O(n+m)（n，m分别为两链表长度）

```
    static Node<T>* MergeTwoLists(Node<T>* x, Node<T>* y) {
        Node<T>* z,*tempx,*tempy;
        if (!x || !y)
            return nullptr;
        if (x->data < y->data)
            swap(x, y); //首先选取第一个元素较小的作为主链，默认为y，否则swap
        z = y,tempy = y,y = y->next,tempx = x,x = x->next;//四指针，tempy在y前，
tempx在x前
        //对于同等数值的节点，不予合并
        while (tempx && y)
        {

            while (y && (y->data < tempx->data)) //找不到更小节点，y向后走
                tempy = y,y = y->next;
            while (y && (y->data >= tempx->data)) //当y大于tempx时，插入节点与y前
            {
                tempy->next = tempx;
                tempy = tempx;
                tempx->next = y;
                if (tempx == x) //下方避免x越界，此处在下一次tempx会与x重叠
                {
                    tempx = nullptr; //设置退出循环(break外界while，也可以用goto)
                    break;
                }
                tempx = x;
                if (x->next == nullptr) break; //避免x越界
                x = x->next;
            }
        }
        if (tempx) tempy->next = tempx;

        return z;
    }
```

P184 T6 索引：P184_T5 P184_T6 P194_T4 完整代码 效果图

```
static void traverseForward(Node<T>** left,Node<T>** right,int pos)
    {
        if (!(*left) || !(*right) || pos == 0) return;
        if (pos < 0) return traverseBack(left, right, -pos);

        while (pos-- && *right)
        {
            Node<T>* now = *right;
            *right = (*right)->next;
            now->next = *left;
            *left = now;
        }
        if (!(*right)) *right = NULL; //题目描述设置为0，考虑到NULL宏定义为0，选择了
NULL，而非nullptr
    }
    static void traverseBack(Node<T>** left, Node<T>** right, int pos)
    {
        if (!(*left) || !(*right) || pos == 0) return;
        if (pos < 0) return traverseForward(left, right, -pos);
        return traverseForward(right, left, pos);
    }
```

P194 T4 索引：P184_T5 P184_T6 P194_T4 完整代码 效果图

```
static int specialSum(Node<int>* lst)
    {
        Node<int>* lst2 = lst;
        int sum = 0;
        for (int i = 0; i < 5; i++)
            lst2 = lst2->next;
        while (lst2)
        {
            sum += lst2->data* lst->data;
            lst2 = lst2->next;
            lst = lst->next;
        }
        return sum;
    }
```

完整代码(CV可运行) 索引：P184_T5 P184_T6 P194_T4 完整代码 效果图

```
#include <iostream>
using namespace std;
template<class T>
class Node {
public:
    T data;
    Node* next;
    explicit Node(T val = 0, Node* nex = nullptr) : data(val), next(nex) {}
};
template<class T>
```

```
class Merge {
public:
    Merge() {}
    static Node<T>* MergeTwoLists(Node<T>* x, Node<T>* y) {
        Node<T>* z,*tempx,*tempy;
        if (!x || !y)
            return nullptr;
        if (x->data < y->data)
            swap(x, y); //首先选取第一个元素较小的作为主链，默认为y，否则swap
        z = y,tempy = y,y = y->next,tempx = x,x = x->next;//四指针，tempy在y前，
tempx在x前
        //对于同等数值的节点，不予合并
        while (tempx && y)
        {

            while (y && (y->data < tempx->data)) //找不到更小节点，y向后走
                tempy = y,y = y->next;
            while (y && (y->data >= tempx->data)) //当y大于tempx时，插入节点与y前
            {
                tempy->next = tempx;
                tempy = tempx;
                tempx->next = y;
                if (tempx == x) //下方避免x越界，此处在下一次tempx会与x重叠
                {
                    tempx = nullptr; //设置退出循环(break外界while，也可以用goto)
                    break;
                }
                tempx = x;
                if (x->next == nullptr) break; //避免x越界
                x = x->next;
            }
        }
        if (tempx) tempy->next = tempx;

        return z;
    }

    void static createTest(Node<T>** x, Node<T>** y) {
        auto x9 = new Node<T>(125);
        auto x8 = new Node<T>(100, x9);
        auto x7 = new Node<T>(89, x8);
        auto x6 = new Node<T>(77, x7);
        auto x5 = new Node<T>(45, x6);
        auto x4 = new Node<T>(23, x5);
        auto x3 = new Node<T>(9, x4);
        auto x2 = new Node<T>(7, x3);
        auto x1 = new Node<T>(7, x2);
        auto x0 = new Node<T>(5, x1);


        auto y5 = new Node<T>(201);
        auto y4 = new Node<T>(147, y5);
        auto y3 = new Node<T>(135, y4);
        auto y2 = new Node<T>(102, y3);
        auto y1 = new Node<T>(74, y2);
        auto y0 = new Node<T>(4, y1);

        *x = x0, *y = y0;
```

```
    }
    void static createTest(Node<T>** x) {
        auto x9 = new Node<T>(125);
        auto x8 = new Node<T>(100, x9);
        auto x7 = new Node<T>(89, x8);
        auto x6 = new Node<T>(77, x7);
        auto x5 = new Node<T>(45, x6);
        auto x4 = new Node<T>(23, x5);
        auto x3 = new Node<T>(9, x4);
        auto x2 = new Node<T>(7, x3);
        auto x1 = new Node<T>(7, x2);
        auto x0 = new Node<T>(5, x1);




        *x = x0;
    }

    void static check(Node<T>* lst) {
        if (!lst)
        {
            cout << "Error:List is empty!\n";
            return;
        }
        int cnt = 0;
        while (lst) {
            cout << "[" << cnt++ << "]" << "->" << lst->data << " ";
            lst = lst->next;
        }
        cout << endl;
    }
    static void traverseForward(Node<T>** left,Node<T>** right,int pos)
    {
        if (!(*left) || !(*right) || pos == 0) return;
        if (pos < 0) return traverseBack(left, right, -pos);

        while (pos-- && *right)
        {
            Node<T>* now = *right;
            *right = (*right)->next;
            now->next = *left;
            *left = now;
        }
        if (!(*right)) *right = NULL; //题目描述设置为0，考虑到NULL宏定义为0，选择了
NULL，而非nullptr
    }
    static void traverseBack(Node<T>** left, Node<T>** right, int pos)
    {
        if (!(*left) || !(*right) || pos == 0) return;
        if (pos < 0) return traverseForward(left, right, -pos);
        return traverseForward(right, left, pos);
    }
    static void specialCheck(Node<T>* left, Node<T>* right)
    {
        if (!left)
        {
            cout << "Error:left is empty!\n";
```

```
        }
        else if (!right)
        {
            cout << "Error:right is empty!\n";


        }
        int cnt = 0,cnt2 = 0,cnt3=0;
        Node<T>* stack[100];
        while (left) {
            stack[cnt++] = left;
            //if (left->next == stack[cnt - 2]) break; //栈&下方队列均为辅助输出用，同时因为首部和尾部的next指针难以操作和确定，所以也可用于判断成环
            left = left->next;
        }
        while (cnt > 1)
        {
            cout << stack[--cnt]->data << "<-" << "[" << cnt2++ << "]" << " ";
        }
        if(cnt)cout << stack[--cnt]->data << "<-" << "[*left*]" << "[" << cnt2++ << "]" << " ";

        Node<T>* queue[100];
        while (right){
            queue[cnt++] = right;
            //if (right->next == stack[cnt - 2]) break;
            right = right->next;
        }
        if (cnt)cout << "[*right*]" << "[" << cnt2++ << "]" << "->" << queue[cnt3++]->data << " ";
        while (cnt3 < cnt)
        {
            cout << "[" << cnt2++ << "]" << "->" << queue[cnt3++]->data << " ";
        }
        cout << endl;
    }

    static int specialSum(Node<int>* lst)
    {
        Node<int>* lst2 = lst;
        int sum = 0;
        for (int i = 0; i < 5; i++)
            lst2 = lst2->next;
        while (lst2)
        {
            sum += lst2->data* lst->data;
            lst2 = lst2->next;
            lst = lst->next;
        }
        return sum;
    }

};


int main() {
    std::cout << "------Pro5-----" << std::endl;
```

```
    Node<int>* x, * y;
    Merge<int>::createTest(&x, &y);
    cout << "ChainX: " << endl;
    Merge<int>::check(x);
    cout << "ChainY: " << endl;
    Merge<int>::check(y);
    cout << "ChainZ(After Merging): " << endl;
    Merge<int>::check(Merge<int>::MergeTwoLists(x,y));

    cout << "Next Group: " << endl;
    Merge<int>::createTest(&y, &x);

    cout << "ChainX: " << endl;
    Merge<int>::check(x);
    cout << "ChainY: " << endl;
    Merge<int>::check(y);
    cout << "ChainZ(After Merging): " << endl;
    Merge<int>::check(Merge<int>::MergeTwoLists(x, y)); //test with two groups
of data


    std::cout << "------Pro6-----" << std::endl;
    Node<int>* left, * right;
    //By default, left is set to be[0], and right be [1]
    Merge<int>::createTest(&left);
    right = left->next;
    cout << "Original: " << endl;
    Merge<int>::check(left); //Print original list for comparison
    left->next = nullptr; //adjust 'next' of the first element to avoid a circle
occurring

    //考虑到提供的链表并没有设置头结点，只能预设首元素next指向nullptr
    cout << "Cross the right bound:" << endl;
    Merge<int>::traverseForward(&left,&right,40); //[右越界测试]题中所述越界设为0，这
里设置为NULL宏定义，故不可再次操作原链表
    Merge<int>::specialCheck(left, right);


    Merge<int>::createTest(&left); //Reset left_list
    right = left->next;
    cout << "Original: " << endl;
    Merge<int>::check(left);
    left->next = nullptr;

    cout << "3 steps to the right: " << endl;
    Merge<int>::traverseForward(&left, &right, 3); //three steps to right
    Merge<int>::specialCheck(left, right);
    cout << "2 steps to the right: " << endl;
    Merge<int>::traverseBack(&left, &right, 2); //two steps to left
    Merge<int>::specialCheck(left, right);
    cout << "1 step to the right: " << endl;
    Merge<int>::traverseBack(&left, &right, 1); //one step to left
    Merge<int>::specialCheck(left, right);
    cout << "Cross the left bound:" << endl;
    Merge<int>::traverseBack(&left, &right, 1); //[左越界测试]
    Merge<int>::specialCheck(left, right);
```

```
    std::cout << "------Pro4-----" << std::endl;
    Merge<int>::createTest(&left);
    Merge<int>::check(left);
    right = left;
    int testArray[100]{0};
    int cnt = 1;
    while (right)
    {
        testArray[cnt++] = right->data;
        right = right->next;
    }
    int sum = 0;
    for (int i = 1; i <= cnt - 5; i++)
        sum += testArray[i] * testArray[i + 5];
    cout << "Answer: " << sum << endl;
    cout << "Result: " << Merge<int>::specialSum(left) << endl;
    sum == Merge<int>::specialSum(left) ? cout << "Correct!\n" : cout <<
"False!\n";
    return 0;
}
```

效果图 索引：P184_T5 P184_T6 P194_T4 完整代码 效果图