实践8小结

- 编译错误——warning
- 运行错误
  - *逻辑：差一误差*
    - 头指针——链表的必备元素。
    - 头结点——有/无在处理上的差别。
    - 尾结点。
    - 栈区越界。

实践8小结

- 运行错误，C/C++的罪恶之源——内存管理
  - *最显著：如：debug assertion failed，access violation*
    - 本质都是对非法地址的访问——悬空指针(野指针)——例：局部指针变量没有被初始化，堆区越界，对已销毁指针的操作[读，写，释放——作用域有效指针变量未清零，析构时重复释放]， 错误的强制类型转换。
    - 善用NULL，可避免一部分。
  - *非显性：内存泄漏*——未释放的内存，暂时非显性的！
    - operator= 重载时，有少数同学忘记清除左值对象占用的空间！

套路版“=”

```
template<typename T>
List<T>& List<T>::operator=(const List<T> &t)
{
   Node<T>*temp=head,*p;
   while(temp!=NULL)    {
      p=temp->link;
      delete []temp;
      temp=p;    }
   //以下同复制构造，略
}
```

机智版“=”

```
MakeEmpty();  或   this->MakeEmpty();
```

实践8小结

- 定义类成员函数的一致性需求
  - 静态函数，只可访问静态成员变量。
  - const类引用， const类指针，类的只读成员函数，只可以访问类的只读成员函数。

  如本次作业中

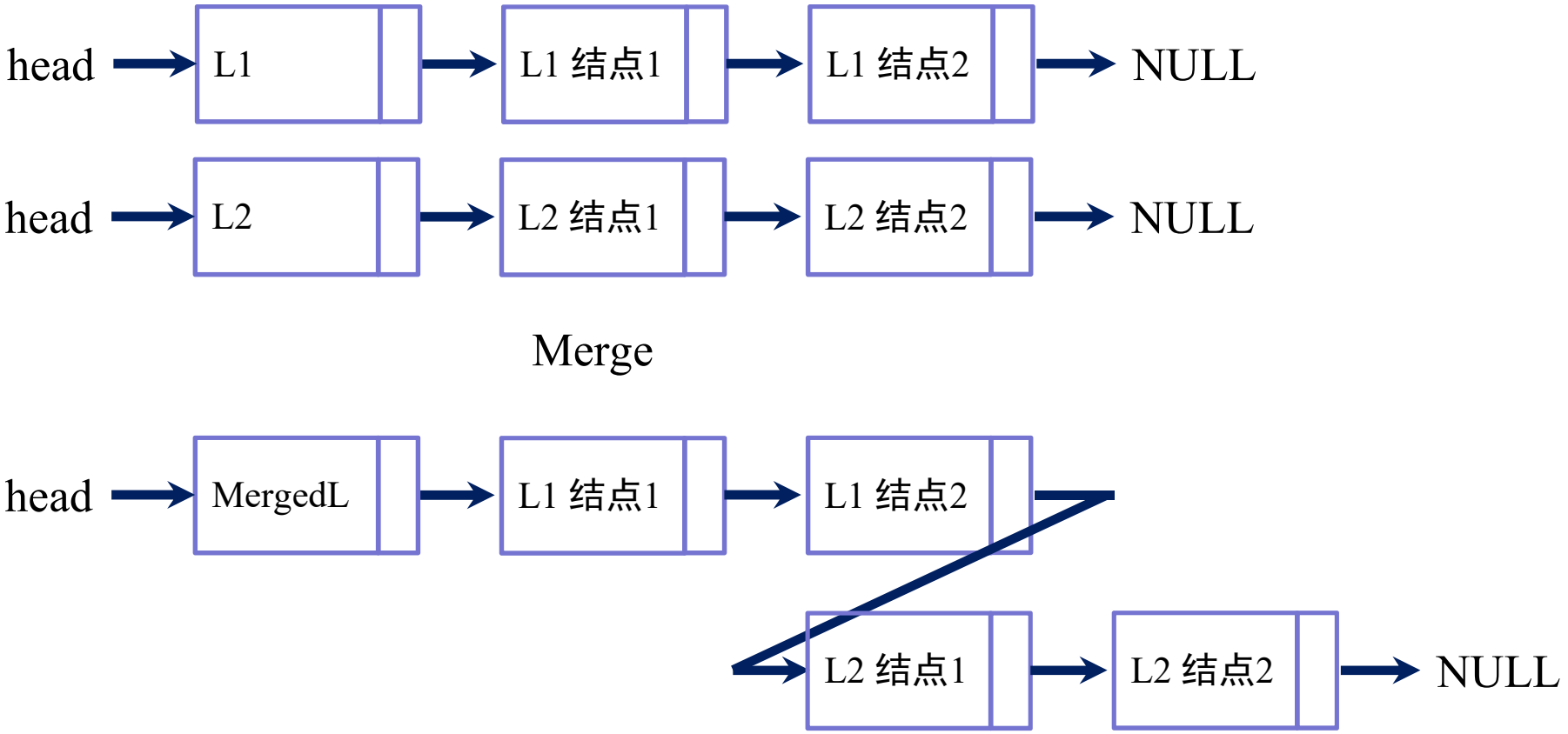
  List(const List<T> &l);

  Node<T>* FindK(int K) const;

  List<T> operator+(const List<T> &L2) const;

  在函数定义过程中若调用了length函数，则需要将length函数也定义为只读。

实践8 小结



注意：重载operator+函数，返回的链表的每一个结点，都应该重新在堆内申请空间；不应该是取L1或L2中的结点地址赋值给objL结点。原理同深浅复制/赋值。

套路版“+”

```
template<typename T>List<T>
List<T>::operator+(const List<T> &L2){
    List<T> now(*this); //正确深复制构造为前提
    Node<T> *temp=L2.head->link,*p;
    while(temp!=NULL) {
        p=new Node<T>(*temp);
        p->link=now.tail->link;
        now.tail->link=p;
        now.tail=p;
        temp=temp->link;   }
    return now; }
```

机智版 “+”

```
template<typename T> List<T>
List<T>::operator+(const List<T>& L2) {
    List<T> L1(*this);  //正确深复制构造为前提
    Node<T>* tem = L2.head->link;
    while (tem != NULL) {
        L1.InsertRear(CreatNode(tem->info));
        tem = tem->link;
    }
    return L1;                  //正确深复制构造为前提
}
```

劳动人民曲线救国的智慧是无穷的!

```
void List<T>::Sort() {
        int len=Length();        T *arr=new T [len];
        Node<T> *temp1=head;
        for(int i=0;i<len;i++)  {
                arr[i]=temp1->link->info;
                temp1=temp1->link;   }
        MakeEmpty();    SelectSort(arr,len);
        for(int i=0;i<len;i++)      {
                Node<T> *P1=CreatNode(arr[i]);
                InsertRear(P1);   }
        return ;
}
```

算法思路by@D212015,25,26,@D212017

实诚孩子链表排序，套路版算法一例！

```
template<typename T>void List<T>::Sort(){
    Node<T> *pre,*cur,*next,*end;
    while(head->link!=end){
        pre=head,cur=pre->link,next=cur->link;
        for( ;next!=NULL; )
        {
            if(cur->info  >  next->info)    {
                pre->link=next;          cur->link=next->link;
                next->link=cur;          Node<T> *temp=cur;
                cur=next;                next=temp;    }
            pre=pre->link, cur=cur->link, next=next->link;
        }
        end=cur;  }
}
```

算法思路by@D212013
@D212020 4,05,12,15,22,23,25,26

机智版，利用已有函数的排序算法一例！

```
template<typename T>void List<T>::Sort()
{
    Node<T>*now=head->link,*temp;  //缓存原链表
    head->link=NULL,tail=head;   //从空链表开始
    while(now!=NULL)
    {
        temp=now;        //从缓存的链表依次摘下结点
        now=now->link;
        InsertOrder(temp);
    }
}
```

by@D2120123

实践8 小结

- 面向对象的思想方便了程序的设计和维护。
- 但在每一个功能块的实现，面向过程的开发过程仍然适用：分解任务，分而治之，大问题分解为若干小的易处理的问题，逐一化解！
  - Top-down，从上到下依次实现每个函数，函数未实现时，函数体可以是空语句，函数可被调用，不影响整体结构。
  - Bottom-up，每实现一个细化的功能/函数，即测试其正确性，逐步完善，逐步求精。

范例：Expo8_ooo_01_单链表类模板.cpp

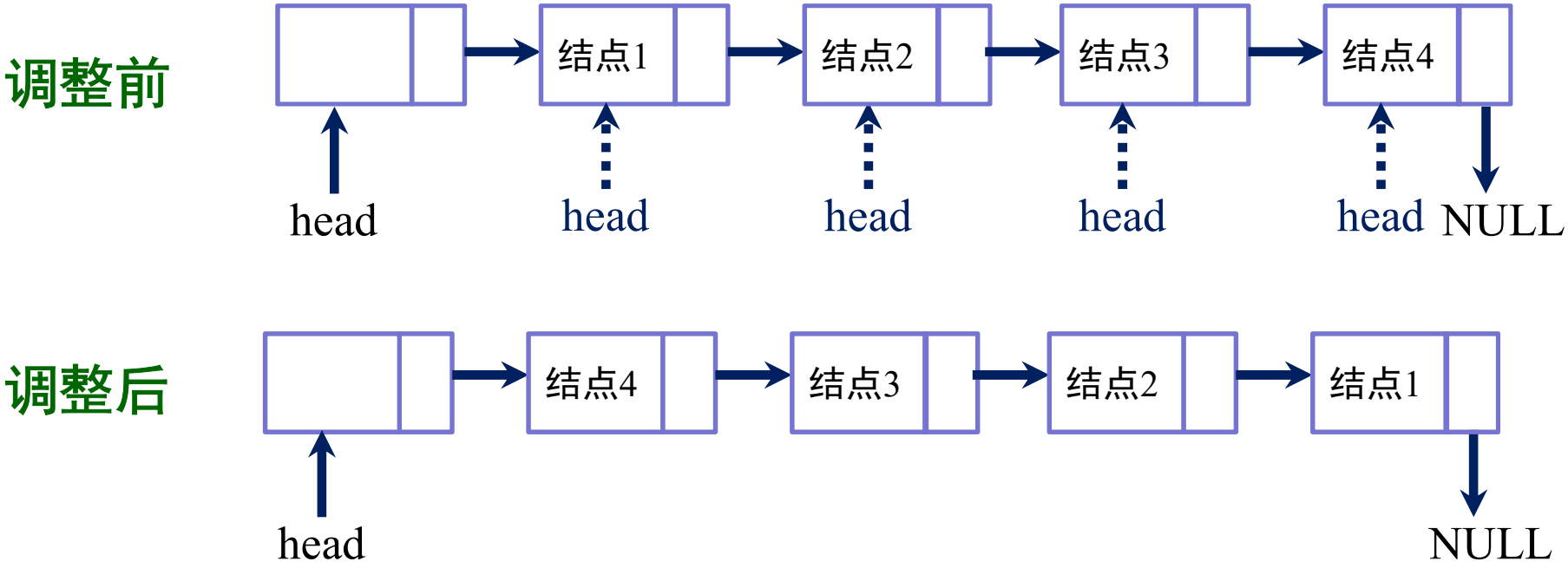
Reverse()——渐进地完成：第一个结点的处理→添加循环执行重复的结点的处理→最后，头/尾结点的处理→测试及“差一”误差的处理。

operator+()——先完成合并，测试无误后，再完成去重。

实践8 小结

- 在class List中添加成员函数;
  - 习题7.5，难点在不改变所有结点的存储，仅通过指针域连接的改变。

```
void Reverse();    //通过指针域连接的改变，将链表逆转。
```



Reverse的一种实现方案

```
template<typename T>void List<T>::Reverse() {
    if(head==tail || head->link==tail) return; //空链表或单结点链表
    Node<T> *prev=NULL, *next=NULL;
    Node<T> *pHead=head;  //缓存头结点
    head=tail=head->link;      //从第一个结点开始逆转
    while(head != NULL)    {
        next = head->link; //从头向尾依次摘下结点
        head->link = prev; //首轮置0，后续将指针域指向原前一个结点
        prev = head;
        head = next; }       // head指针向后一结点移动
    //取来缓存的(空的)头结点作为头结点，将逆转后的链表接到头结点后
    head=pHead;
    head->link=prev;    return;
}
```

实践9：栈与队列的操作

- 要求
  - 栈、队列结构的特点及操作方法，能用面向对象的方法定义并检验栈、队列结构。
- 编程：
  - 1.链表元素类型不同及链表的应用的实验：用链表结构定义一个矩阵。
  - 2.实验20，二、1，习题7.8，改造顺序栈类模板。
  - 3.实验20，二、2，习题7.9，用单链表类模板表示一个双端队列。
  - 4.模仿实验20，二、1，改造课本例7.10循环队列类模板，具体要求见cpp文件。