习题课：线性表

假设用不带头结点的的单链表作为线性表List的存储结构,List类的数据成员只含有一个head指针。为List类添加一个成员函数定位中间位置（倒数第k个）的结点。

如果表为(1,2,3,4,5),则中间位置元素为3，如果表为(1,2,3,4)，则中间位置元素为2;

基本思想：设置一快、一慢两个动态指针

```
template <class T>
LinkNode *  List<T>::middle_point(){
Node <T> * p=head;
Node <T>* q=head;
while (q->next!=NULL){
    q=q->next;
    if (q->next!=NULL){
            q=q->next;
            p=p->next;
     }}
return p;
}
```

假设用不带头结点的的单链表作为线性表List的存储结构,为List类添加一个递归成员函数，统计表中值为item的结点的个数。

例如：线性表为：（7，2，1，7，2，3，6，5），统计链表中值为7的结点数，返回结果应为2。

基本思想：一分为二。
空表，或者首号元素+剩余线性表

```
template <class T>
int  List<T> :: rec_count(LinkNode<T>*head,T item){
   if (head==NULL)
            return 0;
   else
            if (head->data==item)
                     return 1+rec_count(head->next,item);
            else
                     return rec_count(head->next,item);
}
```

假设用带头结点的的单链表作为线性表List的存储结构,为List添加一个成员函数，删除线性表中所有值为item的结点。
例如：原线性表为：（7，2，1，7，2，3，6，5）
删除7之后的表为：（2，1，2，3，6，5）
请按函数原型int List<T> :: removealloccurance(T item)；进行设计，函数返回删除元素的个数。

基本思想：一前一后两个指针，前一个指针负责比较，后一个指针负责删除动作。

```
template <class T>
int  List<T> :: removealloccurance(T item){
    count=0;
     q=head;p=head->next; //以下要求保证p,q保持前后相邻
     while (p!=NULL)
     if (p->entry==item){
                    q->next=p->next;
                    delete p;
                    count++;//删除p结点
                    p=q->next;//维护p的值
    }else{
                    q=p;p=p->next;//q,p分别向后移动
    }
     return  count;
}
```

a,b分别为两个不带头结点的单链表，
试设计一个从la为头指针表a中删除自第i个元素起共len个元素，
并将这len个元素插入到表b中的第i个元素之前的算法。

基本思想：

- 动态指针的定位
- 修改任何指针都必须确保有其他指针指向对应内存
- 画图

```
template <class List_entry>
Error_code   move(Node <List_entry >* &la , Node <List_entry>*&lb,int i,int len){
   if (i<0 )
      return fail;//参数不合法的情况处理
   p=la;q=lb;
   for (int j=0;j<i-1;j++){
      if (p==NULL ||q==NULL)   return fail;
      p=p->next;q=q->next;
   }
   s=q->next;r=p;
   for (int j=0;j<len;j++){
      p=p->next;
      if (p==NULL)  return fail;
   }
   q->next=r->next;
   r->next=p->next;
   p->next=s;
   return success;
}
```

join two ordered list in a List, ***Error_code Join(List a_list,List b_list,List &c_list)***
example : a_list=(1,3,5,7,9) b_list=(2,4,7,10,13) After the joint, we get the result list, c_list=(1,2,3,4,5,7,7,9,10,13).

```
template <class List_entry>
Error_code Join(List<List_entry> a_list, List<List_entry> b_list, List<List_entry> &c_list){
    ...
            if (item_a<=item_b){
              if (c_list.insert(k++,item_a)==overflow) return overflow ;
              i++;
            }else{
              if (c_list.insert(k++,item_b)==overflow) return overflow ;
              j++;}
        return success;
    }
```

基本

-

- **类似思考：**
  - ✓ **设有两个有序的单链表，一为升序，一为降序。试设计一算法将这两个链表合并成一个有序链表。**
  - ✓ 设带头结点且头指针为ha和hb的两线性表A和B分别表示两个集合。两表中的元素皆为递增有序。请写一算法求A和B的并集（交集）。要求该并集中的元素仍保持递增有序,且要利用A和B的原有结点空间。

- 已知线性表(a1,a2,……,an)按顺序存于内存，每个元素都是整数，试设计用最少时间把所有值为负数的元素移动到全部正数值元素前面的算法。

基本思想：

- 并不要求正、负元素的次序与原次序保持相同
- 设定两个索引指示量p、q，其中p从小到大找正数，q从大到小找负数，并交换之，直到p、q相遇或交叉而过

- 编写算法将单链表L1拆分成两个链表，其中以L1为头的链表保持原来向后的链接，另一个链表的头为L2，其链接方向与原L1相反。L1包含原链表中奇数序号的结点，L2包含原链表中偶数序号的结点。

基本思想：

- L1以队列方式工作
- L2以栈的方式工作
- 将L2初始化成空栈，遍历L1同时计数，遇到偶数结点实施从L1中“卸载”，再将该结点入栈L2

- 设有一头指针为L的带头结点的非循环双向链表，其每个结点中除有pre，data和next域外，还有一个访问频度域freq。在链表启用前freq均为0。每当链表中进行一次Locate（L,x）运算时，所有data域为x的结点的freq增加1，并使此链表中结点保持按访问频度非增的顺序排列，同时最近访问的结点排在频度相同结点的最后，以便使频繁访问的结点总是靠近表头。编写符合上述要求的Locate(L,x)。

基本思想：

- 若查找x不成功，在链表尾部进行插入
- 若查找x成功，freq++后，按向前（pre）方向找第一个freq<=当前freq的结点（或者头结点），然后进行后插

- 编写递归函数完成求n个数的全排列。

基本思想：

- 利用数组A[n]保存n个自然数
- n=1时，一个元素的全排列只有一种，即为本身
- n>1时，求n-1个元素的全排列+n^{th}个元素

求n-1个元素的全排列 + n^{th}个元素			
1^{st}子问题	$a_1, a_2, \ldots, a_{n-1}$	a_n	//
2^{nd}子问题	$a_1, \ldots, a_{n-2}, a_n$	a_{n-1}	//A[n-2]↔A[n-1]
3^{rd}子问题	$a_1, \ldots, a_n, a_{n-1}$	a_{n-2}	//A[n-3]↔A[n-1]
⋮	⋮	⋮	
n^{th}子问题	$a_n, a_2 \ldots, a_{n-1}$	a_1	//A[0]↔A[n-1]

```
void Permute(int a[],int j,int n){
    //a[j]~a[n-1]的n-j个元素进行全排列，j初始为0
  int i, tmp;
  if (j==n-1){
     for (i=0;i<n;i++)  count<<a[i];
     cout<<endl;
  }
  else
     for (i=j;i<n;i++){
       tmp=a[j]; a[j]=a[i]; a[i]=tmp;
       Permute(a,j+1,n);
            tmp=a[j]; a[j]=a[i]; a[i]=tmp;
       }
}
```