Chapter 2 线性表

书P84. 2.6

```
template <typename T>
void Reverse ( T A[ ], int n, int arraySize ) {
 //将一维数组A[arraySize]中存储的n个数组元素原地逆置
    if (n>ayyaySize) {
          cerr<<“参数不合理!”<<endl;  exit(1);
    }
    T temp;
    for ( int i=0; i<n/2; i++) {
        temp=A[i];
        A[i]=A[n-i-1];
        A[n-i-1]=temp;
    }
}
```

书P85. 2.9

```
template <typename T>
void Exchange( T A[ ], int m, int n, int arraySize ) {
//将数组A[m+n]中依次存放的两个顺序表(a0, a1, …, am-1)
//与(b0, b1, …, bn-1)的位置互换，即将b表放到a表前面
   Reverse(A, m, arraySize);  //将前面a表的m个元素逆置
   Reverse(A, m+n, arraySize);  //将全部元素逆置
   Reverse(A, n, arraySize); //将换到前面的b表的n个元素逆置
}
```

补充：（2010考研统考题）

设将n(n>1)个整数存放到一维数组R中。试设计一个在 时间和空间两方面尽可能高效的算法，将R中的序列循环左移 p（0<p<n）个位置，即将R中的数据由（$x_0, x_1, \ldots, x_{n-1}$）变换为（$x_p, x_{p+1}, \ldots, x_{n-1}, x_0, x_1, \ldots, x_{p-1}$）。要求：

（1）给出算法的基本设计思想。

（2）根据设计思想，采用C或C++或JAVA语言描述算法，关键之处给出注释。

（3）说明你所设计算法的时间复杂度和空间复杂度。

解：（1）算法的基本设计思想

先将n个数（$x_0, x_1, \ldots, x_{n-1}$） 原地逆置，得到（$x_{n-1}, \ldots, x_p, x_{p-1}, \ldots, x_0$），然后再将前n-p个和后p个元素分别原地逆置，得到最终结果：($x_p, x_{p+1}, \ldots, x_{n-1}, x_0, x_1, \ldots, x_{p-1}$)。

算法用两个函数实现：一个是逆置函数reverse()，它将给定的数据逆置。另一个是循环左移函数leftShift()，它调用reverse()函数三次，实现相应功能。

（2）算法实现

```
void  reverse(int R[ ], int left, int right)
 //将数组R中left到right之间的元素原地逆置
 {  int i=left, j=right, temp;
    while (i<j)
       {   //交换R[i]与R[j]
          temp=R[i];
          R[i]=R[j];
          R[j]=temp;
          i++;       //i右移一个位置
          j--;       //j 左移一个位置
       }
 }
```

```
void leftShift( int R[ ], int n, int p)
 //将数组R中元素循环左移p个位置
{  if ( p>0 && p<n)
      {
        reverse(R, 0, n-1);       //将全部元素逆置
        reverse(R, 0, n-p-1);     //将前n-p个元素逆置
        reverse(R, n-p, n-1);     //将后p个元素逆置
      }
}
```

（3）算法分析

算法的时间复杂度为O(n)，由于是原地逆置，所以空间复杂度为O(1)。

书P85. 2.12：将两个整数类型的递增有序的顺序表A和B归并为一个递增有序的顺序表C（C由函数返回）

```
SeqList<int> Merge ( SeqList<int> & A,  SeqList<int> & B {
    SeqList<int> temp;
    if (A.Length()+B.Length() > temp.Size())
       { cerr<<“空间不够！”<<endl; exit(1); }
    int m = A.Length( ), n=B.Length(), valA, valB;
    int i=j=1, k=0;
    while ( i<=m && j<=n ) {
          A.getData(i,valA);  B.getData(j,valB);
          if (valA<=valB)
                { temp.Insert(k, valA); i++; }
          else { temp.Insert(k, valB); j++; }
          k++;
     }
    while ( i<=m )
          { temp.Insert(k, valA); i++; k++; }
    while ( j<=n )
          { temp.Insert(k, valB); j++; k++; }
    return temp;
}
```

书P85. 2.12：将两个整数类型的递增有序的顺序表A和B归并为一个递增有序的顺序表C

```
void Merge ( SeqList<int> & A,  SeqList<int> & B, SeqList<int> & C ) {
    if (A.Length()+B.Length() > C.Size())
       { cerr<<“空间不够！”<<endl; exit(1); }
    int m = A.Length( ), n=B.Length(), valA, valB;
    int i=j=1, k=0;
    while ( i<=m && j<=n ) {
        A.getData(i,valA);  B.getData(j,valB);
        if (valA<=valB)
            { C.Insert(k, valA); i++; }
        else { C.Insert(k, valB); j++; }
        k++;
     }
    while ( i<=m )
        { C.Insert(k, valA); i++; k++; }
    while ( j<=n )
        { C.Insert(k, valB); j++; k++; }
}
```

书P86. 2.15：法一：将非递减有序的单链表ha和hb合并成非递增有序的单链表，结果放在ha，要求利用原表结点。

```
template <typename T>
void Merge ( List<T> & ha, List<T> & hb ) {
    LinkNode<T> *pa, *pb, *last, *q;
    pa=ha.getHead()->link; pb=hb.getHead()->link;
    last=ha.getHead();  last->link=NULL;
    while ( pa!=NULL && pb!=NULL ) {
        if (pa->data<=pb->data) { q=pa; pa=pa->link; }
        else { q=pb; pb=pb->link; }
        q->link=last->link;  last->link=q;    //按前插法插入
    }
    if (pb!=NULL)  pa=pb;
    while (pa!=NULL)
        { q=pa; pa=pa->link; q->link=last->link;  last->link=q; }
    q=hb.getHead();   delete q;     //删除hb的表头结点
}
```

书P86. 2.15：法二：将hb合并到ha，作为单链表成员函数

```
template <typename T>
void List<T>::Merge (List<T> & hb ) {
    LinkNode<T> *pa, *pb, *q;
    pa=first->link; pb=hb.getHead()->link; first->link=NULL;
    while ( pa!=NULL && pb!=NULL ) {
        if (pa->data<=pb->data) { q=pa; pa=pa->link; }
        else { q=pb; pb=pb->link; }
        q->link=first->link;  first->link=q;    //按前插法插入
    }
    if (pb!=NULL)  pa=pb;
    while (pa!=NULL)
        { q=pa; pa=pa->link; q->link=first->link;  first->link=q; }
    q=hb.getHead();   delete q;     //删除hb的表头结点
}
```

书P86. 2.19：改造双向链表，rLink不变，利用lLink域按值从小到大链接起来

```
template <typename T>
void OrderedLink ( DblList<T> & DL) {
 //将带表头结点的循环双向链表DL利用lLink域按值从小到大
 //链接起来，采用直接插入排序思想
    DblNode<T> *pr, *p, *s, *h;
    h=DL.getHead();
    s=h->rLink->rLink;
    h->rLink->lLink=h;
    h->lLink=h->rLink;
    while ( s!=h ) {
          pr=h;  p=h->lLink;
          while ( p!=h && p->data < s->data)
                { pr=p; p=p->lLink; }
          pr->lLink=s;   s->lLink=p;   //插入在pr与p间左链上
          s=s->rLink;
    }
}
```

书P86. 2.22：删除一个带头结点的递增有序单链表中所有元素值大于min且小于max的结点。

```
template <typename T>
void rangDelete ( List<T> & L, T min, T max) {
    LinkNode<T> *pr=L.getHead(), *p=pr->link;
    while ( p!=NULL && p->data <= min )
          { pr=p;  p=p->link;  }
    while ( p!=NULL && p->data < max )
          { pr->link=p->link;  }
            delete p;
            p=pr->link; }
}
```

补充：（2009考研统考题）已知一个带头结点的单链表，假设该链表只给出了头指针，在不改变链表的前提下，请设计一个尽可能高效的算法，查找链表中倒数第k个结点（k为正整数），若查找成功，算法输出该结点的data域的值，并返回1，否则，只返回0。

```
template <typename T>
int Locatek ( List<T> & L, int k ) {
        LinkNode<T> *p,*q;   int count=0;
        p=q=L.getHead()->link;
        while (p!=NULL)
            {  //先移动k次p指针，然后再同时移动p、q指针，
               //直至p指针为空
              p=p->link;
              if  (count<k)   count++;
              else  q=q->link; }
       if (count<k)   return 0;          //k值超过表长，查找失败
       else   {  cout<<q->data<<endl;
                 return 1;   }
}
```