## 第3章：排序基础

### 1、以顺序表L的L.rcd[L.length+1]作为监视哨,改写升序直接插入排序算法

/\*\*\*\*\*\*\*\*\*\*  
【题目】试以顺序表L的L.rcd[L.length+1]作为监视哨，  
改写教材3.2节中给出的升序直接插入排序算法。  
顺序表的类型RcdSqList定义如下：  
typedef struct {  
 KeyType key;   
 ...   
} RcdType;  
typedef struct {  
 RcdType rcd[MAXSIZE+1]; // rcd[0]闲置  
 int length;  
} RcdSqList;  
\*\*\*\*\*\*\*\*\*\*/  
void InsertSort(RcdSqList &L)  
{  
 int i = 0,j = 0;  
 for(i = 1;i < L.length;i ++){  
 if(L.rcd[i+1].key < L.rcd[i].key){  
 L.rcd[L.length+1] = L.rcd[i+1];  
 j = i + 1;  
 do{  
 j --;  
 L.rcd[j+1] = L.rcd[j];  
 }while(j>1 && L.rcd[j-1].key > L.rcd[L.length+1].key);  
 /\*  
 1、之所以用do...while是因为无论如何都  
 要交换一次位置。  
  
 2、之所以用j-1跟L.length+1比，是因为  
 L.rcd[j+1] = L.rcd[j]这行代码移动  
 后，j位置和j+1位置的值是一样的，  
 所以应改用j-1位置的比较，若依然大，  
 通过j--后则此时j确实指向大的位置了。  
  
 3、把j>1作为第一个条件是因为当不满足该条件，  
 循环结束，不执行后面的比较。  
 \*/  
 L.rcd[j] = L.rcd[L.length+1];  
 }   
 }  
}

### 2、改写冒泡排序，用change变量每一趟排序中进行交换的最后一个记录的位置，并以它作为下一趟起泡排序循环终止的控制值。

/\*\*\*\*\*\*\*\*\*\*  
【题目】如下所述，改写教材1.3.2节例1-10的冒泡排序算法：  
将算法中用以起控制作用的布尔变量change改为一个整型  
变量，指示每一趟排序中进行交换的最后一个记录的位置，  
并以它作为下一趟起泡排序循环终止的控制值。  
顺序表的类型RcdSqList定义如下：  
typedef struct {  
 KeyType key;   
 ...   
} RcdType;  
typedef struct {  
 RcdType rcd[MAXSIZE+1]; // rcd[0]闲置  
 int length;  
} RcdSqList;  
\*\*\*\*\*\*\*\*\*\*/  
  
/\*  
 下面这个是常规的冒泡排序，但是不符合题目要求  
 题目是要求通过change变量存储上一次循环最后一个  
 交换的位置下标，以此作为下一次循环的终止条件  
\*/  
void BubbleSort(RcdSqList &L)  
/\* 元素比较和交换必须调用如下定义的比较函数和交换函数： \*/  
/\* Status LT(RcdType a, RcdType b); 比较："<" \*/  
/\* Status GT(RcdType a, RcdType b); 比较：">" \*/  
/\* void Swap(RcdType &a, RcdType &b); 交换 \*/  
{  
 for(int i = 0;i < L.length - 1;i ++)//n-1趟  
 {  
 for(int j = 1;j < L.length - i;j ++)  
 {  
 if(LT(L.rcd[j+1],L.rcd[j]))//j+1处的key小于j  
 {  
 Swap(L.rcd[j+1],L.rcd[j]);  
 }  
 }  
 }  
}  
  
/\*  
 改进如下，把change作为第二个循环的终止条件，  
 但是还是跟测试数据的compare不匹配，似乎是因为  
 第一轮循环多了  
\*/  
void BubbleSort(RcdSqList &L)  
/\* 元素比较和交换必须调用如下定义的比较函数和交换函数： \*/  
/\* Status LT(RcdType a, RcdType b); 比较："<" \*/  
/\* Status GT(RcdType a, RcdType b); 比较：">" \*/  
/\* void Swap(RcdType &a, RcdType &b); 交换 \*/  
{  
 int change = L.length;//刚开始进行n次比较  
 int temp = 0;//记录最后一次交换的位置  
 for(int i = 1;i < L.length;i ++)//n-1趟  
 {  
 /\*进行change-1次比较，因为之后已经比较过了\*/  
 for(int j = 1;j < change;j ++)  
 {  
 if(GT(L.rcd[j],L.rcd[j+1]))  
 {  
 Swap(L.rcd[j+1],L.rcd[j]);  
 temp = j;  
 }  
 }   
 change = temp;  
 if(change == 1) break;  
 }  
}  
  
/\*  
 控制第一层循环测试通过，但是if(change == 2) break;  
 这个代码不能解决所有问题，如IWYEPSPY这个测试数据，当进行到  
 EIPPSWYY时，此时change为5，此后便是死循环，因为i永远大于0  
\*/  
void BubbleSort(RcdSqList &L)  
/\* 元素比较和交换必须调用如下定义的比较函数和交换函数： \*/  
/\* Status LT(RcdType a, RcdType b); 比较："<" \*/  
/\* Status GT(RcdType a, RcdType b); 比较：">" \*/  
/\* void Swap(RcdType &a, RcdType &b); 交换 \*/  
{  
 int change = 0;  
 for(int i = L.length;i > 0;i --)  
 {   
 for(int j = 1;j < i;j ++)  
 {  
 if(GT(L.rcd[j],L.rcd[j+1]))  
 {  
 Swap(L.rcd[j+1],L.rcd[j]);  
 change = j + 1;//记录最后一次交换的位置  
 }  
 }   
 if(change == 2) break;//这行代码不能省略，要不然i=1后一直循环   
 i = change;  
 }  
}  
  
  
/\*  
 解决方法：设置一个flag标志，初始化为0，  
 当进行GT比较时设为1，然后每次内层循环结束又重置为0。  
 在此前判断flag是否为0，若是，则退出，因为说明已排序好，  
 不必调用交换函数，也就不能使flag为1  
\*/  
void BubbleSort(RcdSqList &L)  
/\* 元素比较和交换必须调用如下定义的比较函数和交换函数：\*/  
/\* Status LT(RcdType a, RcdType b); 比较："<" \*/  
/\* Status GT(RcdType a, RcdType b); 比较：">" \*/  
/\* void Swap(RcdType &a, RcdType &b); 交换 \*/  
{  
 int change = 0;  
 int flag = 0;  
 for(int i = L.length;i > 0;i --)  
 {   
 for(int j = 1;j < i;j ++)  
 {  
 if(GT(L.rcd[j],L.rcd[j+1]))  
 {  
 flag = 1;  
 Swap(L.rcd[j+1],L.rcd[j]);  
 change = j + 1;//记录最后一次交换的位置  
 }  
 }   
 //if(change == 2) break  
 //应该设置一个flag来控制，  
 //当第二层循环不比较时，退出  
 if(flag == 0) break;  
 i = change;  
 flag = 0;  
 }  
}

### 3、统计序列中关键字比它小的记录个数存于c[i]，依c[i]值的大小对序列中记录进行重新排列

/\*\*\*\*\*\*\*\*\*\*  
【题目】已知记录序列L.rcd[1..L.length]中的关键  
字各不相同，可按如下所述实现计数排序：另设数组  
c[1..n]，对每个记录a[i]， 统计序列中关键字比它  
小的记录个数存于c[i]，则c[i]=0的记录必为关键字  
最小的记录，然后依c[i]值的大小对序列中记录进行  
重新排列。试编写算法实现上述排序方法。  
顺序表的类型RcdSqList定义如下：  
typedef struct {  
 KeyType key;   
 ...   
} RcdType;  
typedef struct {  
 RcdType rcd[MAXSIZE+1]; // rcd[0]闲置  
 int length;  
} RcdSqList;  
\*\*\*\*\*\*\*\*\*\*/  
void CountSort(RcdSqList &L)  
/\* 采用顺序表存储结构，在函数内自行定义计数数组c \*/  
{  
 int \*c;   
 c = (int \*)malloc(sizeof(int)\*(L.length+1));   
 int i;//遍历  
 for(i = 0;i <= L.length;i ++){  
 c[i] = 0;//初始化为0  
 }  
 KeyType temp;   
 if(c == NULL)return ;  
 /\*对每个L.rcd[i],统计序列中关键字比它小的记录个数存于c[i]\*/  
 for(i = 1;i <= L.length;i ++){  
 temp = L.rcd[i].key;  
 for(int j = 1;j <= L.length;j ++){  
 if(temp > L.rcd[j].key){  
 c[i] ++;  
 }  
 }  
 }  
 RcdType \*rcd1;  
 rcd1 = (RcdType \*)malloc(sizeof(RcdType)\*(L.length+1));  
 if(rcd1 == NULL)return ;  
 for(i = 1;i <= L.length;i ++){  
 /\*  
 c[i]记录了对应rcd记录比多少个记录大，  
 那么该记录应该在rcd1的c[i]+1处，因为  
 记录从1开始，所有c[i]为0的处于rcd1的第一个下标处  
 \*/  
 rcd1[c[i] + 1] = L.rcd[i];  
 }  
 /\*把排序后的记录赋值给L.rcd\*/  
 for(i = 1;i <= L.length;i ++){  
 L.rcd[i] = rcd1[i];  
 }  
 free(c);  
 free(rcd1);  
}