

## 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

```
1  /*****
2  【题目】试以顺序表L的L.rcd[L.length+1]作为监视哨，
3  改写教材3.2节中给出的升序直接插入排序算法。
4  顺序表的类型RcdSList定义如下：
5  typedef struct {
6      KeyType key;
7      ...
8  } RcdType;
9  typedef struct {
10     RcdType rcd[MAXSIZE+1]; // rcd[0]闲置
11     int length;
12 } RcdSList;
13 *****/
14 void InsertSort(RcdSList &L)
15 {
16     int i = 0, j = 0;
17     for(i = 1; i < L.length; i++){
18         if(L.rcd[i+1].key < L.rcd[i].key){
19             L.rcd[L.length+1] = L.rcd[i+1];
20             j = i + 1;
21             do{
22                 j--;
23                 L.rcd[j+1] = L.rcd[j];
24             }while(j>1 && L.rcd[j-1].key > L.rcd[L.length+1].key);
25             /*
26              1、之所以用do...while是因为无论如何都
27              要交换一次位置。
28
29              2、之所以用j-1跟L.length+1比，是因为
30              L.rcd[j+1] = L.rcd[j]这行代码移动
31              后，j位置和j+1位置的值是一样的，
32              所以应改用j-1位置的比较，若依然大，
33              通过j--后则此时j确实指向大的位置了。
34
35              3、把j>1作为第一个条件是因为当不满足该条件，
36              循环结束，不执行后面的比较。
37             */
38             L.rcd[j] = L.rcd[L.length+1];
39         }
40     }
41 }
```

### 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。

```
1  /*****
2  【题目】如下所述，改写教材1.3.2节例1-10的冒泡排序算法：
```

```

3  将算法中用以起控制作用的布尔变量change改为一个整型
4  变量，指示每一趟排序中进行交换的最后一个记录的位置，
5  并以它作为下一趟起泡排序循环终止的控制值。
6  顺序表的类型RcdSqlList定义如下：
7  typedef struct {
8      KeyType key;
9      ...
10 } RcdType;
11 typedef struct {
12     RcdType rcd[MAXSIZE+1]; // rcd[0]闲置
13     int length;
14 } RcdSqlList;
15 *****/
16
17 /*
18     下面这个是常规的冒泡排序，但是不符合题目要求
19     题目是要求通过change变量存储上一次循环最后一个
20     交换的位置下标，以此作为下一次循环的终止条件
21 */
22 void BubbleSort(RcdSqlList &L)
23 /* 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： */
24 /* Status LT(RcdType a, RcdType b); 比较: "<" */
25 /* Status GT(RcdType a, RcdType b); 比较: ">" */
26 /* void Swap(RcdType &a, RcdType &b); 交换 */
27 {
28     for(int i = 0; i < L.length - 1; i++) // n-1趟
29     {
30         for(int j = 1; j < L.length - i; j++)
31         {
32             if(LT(L.rcd[j+1], L.rcd[j])) // j+1处的key小于j
33             {
34                 Swap(L.rcd[j+1], L.rcd[j]);
35             }
36         }
37     }
38 }
39
40 /*
41     改进如下，把change作为第二个循环的终止条件，
42     但是还是跟测试数据的compare不匹配，似乎是因为
43     第一轮循环多了
44 */
45 void BubbleSort(RcdSqlList &L)
46 /* 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： */
47 /* Status LT(RcdType a, RcdType b); 比较: "<" */
48 /* Status GT(RcdType a, RcdType b); 比较: ">" */
49 /* void Swap(RcdType &a, RcdType &b); 交换 */
50 {
51     int change = L.length; // 刚开始进行n次比较
52     int temp = 0; // 记录最后一次交换的位置
53     for(int i = 1; i < L.length; i++) // n-1趟
54     {
55         /* 进行change-1次比较，因为之后已经比较过了 */
56         for(int j = 1; j < change; j++)
57         {
58             if(GT(L.rcd[j], L.rcd[j+1]))
59             {
60                 Swap(L.rcd[j+1], L.rcd[j]);

```

```

61         temp = j;
62     }
63 }
64 change = temp;
65 if(change == 1) break;
66 }
67 }
68
69 /*
70 控制第一层循环测试通过，但是if(change == 2) break;
71 这个代码不能解决所有问题，如IWYEPSPY这个测试数据，当进行到
72 EIPPSWYY时，此时change为5，此后便是死循环，因为i永远大于0
73 */
74 void BubbleSort(RcdSqlList &L)
75 /* 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： */
76 /* Status LT(RcdType a, RcdType b); 比较: "<" */
77 /* Status GT(RcdType a, RcdType b); 比较: ">" */
78 /* void Swap(RcdType &a, RcdType &b); 交换 */
79 {
80     int change = 0;
81     for(int i = L.length; i > 0; i --)
82     {
83         for(int j = 1; j < i; j ++ )
84         {
85             if(GT(L.rcd[j], L.rcd[j+1]))
86             {
87                 Swap(L.rcd[j+1], L.rcd[j]);
88                 change = j + 1; //记录最后一次交换的位置
89             }
90         }
91         if(change == 2) break; //这行代码不能省略，要不然i=1后一直循环
92
93         i = change;
94     }
95 }
96
97 /*
98 解决方法：设置一个flag标志，初始化为0，
99 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。
100 在此前判断flag是否为0，若是，则退出，因为说明已排序好，
101 不必调用交换函数，也就不能使flag为1
102 */
103 void BubbleSort(RcdSqlList &L)
104 /* 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： */
105 /* Status LT(RcdType a, RcdType b); 比较: "<" */
106 /* Status GT(RcdType a, RcdType b); 比较: ">" */
107 /* void Swap(RcdType &a, RcdType &b); 交换 */
108 {
109     int change = 0;
110     int flag = 0;
111     for(int i = L.length; i > 0; i --)
112     {
113         for(int j = 1; j < i; j ++ )
114         {
115             if(GT(L.rcd[j], L.rcd[j+1]))
116             {
117                 flag = 1;

```

```

118         Swap(L.rcd[j+1],L.rcd[j]);
119         change = j + 1; //记录最后一次交换的位置
120     }
121 }
122 //if(change == 2) break
123 //应该设置一个flag来控制,
124 //当第二层循环不比较时,退出
125 if(flag == 0) break;
126 i = change;
127 flag = 0;
128 }
129 }

```

### 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

```

1  /*****
2  【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
3  c[1..n],对每个记录a[i],统计序列中关键字比它
4  小的记录个数存于c[i],则c[i]=0的记录必为关键字
5  最小的记录,然后依c[i]值的大小对序列中记录进行
6  重新排列。试编写算法实现上述排序方法。
7  顺序表的类型RcdSList定义如下:
8  typedef struct {
9      KeyType key;
10     ...
11 } RcdType;
12 typedef struct {
13     RcdType rcd[MAXSIZE+1]; // rcd[0]闲置
14     int length;
15 } RcdSList;
16 *****/
17 void CountSort(RcdSList &L)
18 /* 采用顺序表存储结构,在函数内自行定义计数数组c */
19 {
20     int *c;
21     c = (int *)malloc(sizeof(int)*(L.length+1));
22     int i; //遍历
23     for(i = 0; i <= L.length; i++){
24         c[i] = 0; //初始化为0
25     }
26     KeyType temp;
27     if(c == NULL) return ;
28     /*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]*/
29     for(i = 1; i <= L.length; i++){
30         temp = L.rcd[i].key;
31         for(int j = 1; j <= L.length; j++){
32             if(temp > L.rcd[j].key){
33                 c[i] ++;
34             }
35         }
36     }
37     RcdType *rcd1;
38     rcd1 = (RcdType *)malloc(sizeof(RcdType)*(L.length+1));
39     if(rcd1 == NULL) return ;
40

```

```
41     for(i = 1;i <= L.length;i ++){
42         /*
43             c[i]记录了对应rkd记录比多少个记录大，
44             那么该记录应该在rkd1的c[i]+1处，因为
45             记录从1开始，所有c[i]为0的处于rkd1的第一个下标处
46         */
47         rkd1[c[i] + 1] = L.rkd[i];
48     }
49     /*把排序后的记录赋值给L.rkd*/
50     for(i = 1;i <= L.length;i ++){
51         L.rkd[i] = rkd1[i];
52     }
53     free(c);
54     free(rkd1);
55 }
```