作业二参考答案

较好的作业

- 排版清楚，缩进对齐
- 有适当注释
- 书写工整

2.3

```
Status ListInsert(SqList &L, ElemType e)
{
    if (L.Length >= L.ListSize)//顺序表空间不足，添加空间
    {
        newbase = (ElempType *)malloc(L.elem, (L.ListSize + LISTINCREMENT) * sizeof(ElemType));
    }
    if (!newbase) exit(OVERFLOW);
    L.elem = newbase;
    L.ListSize += LISTINCREMENT;

    for (int j = l.length - 1; e < l.elem[j]; j--)
        l.elem[j + 1] = l.elem[j];
    l.elem[j + 1] = e;


    ++L.Length;
    return OK;

}
```

2.4.2

```
Status ListInsert(LinkList &L, ElemType e)//因为要修改指针L的指向，所以要用引用或者指针的指针。
{
    p = L;
    if (L == NULL || e<p->data)//情况一：链表为空  情况二：在链表头插入
    {
        s = (LinkList)malloc(sizeof(LNode));
        s->data = e;
        s->next = L;
        L = s;  return OK;
    }
    while (p->next != NULL)//情况三：在表中插入,先定位插入位置，然后插入。
    {
```

```
        if (e < p->next->data)
        {
            break;
        }
        p = p->next;
    }
    s = (LinkList)malloc(sizeof(LNode));
    s->data = e;
    s->next = p->next;
    p->next = s;
    return OK;
}
```

2.7.1

```
Status ListDelete(SqList &L)
{
    if (L.Length <= 1)
        return OK;
    //将表分成两部分，前面下标为0到OutLength-1的元素满足无重复条件，用k来遍历；后面为待检查元素，用j来遍历
    int i = 0, j = 1, OutLength = 1;
    while (j<L.Length)//遍历整个线性表
    {
        int k;
        for (k = 0; k < OutLength; k++)//遍历未重复线性表
        {
            if (L.elem[k] == L.elem[j])
            {
                j++;
                break;
            }
        }
        if (k == OutLength)//当前元素未于前面的元素重复
        {
            L.elem[OutLength] = L.elem[j];
            OutLength++;
        }
    }
    L.Length = OutLength;//修改表长

}
```

2.7.2

```
Status ListDelete(SqList &L)
```

```
{
    if (L.Length <= 1)
        return OK;
    //将表分成两部分，前面下标为0到OutLength-1的元素满足无重复条件，后面为待检查元素
    //检查时，只需与无重复元素表的最后一个元素比较即可
    int i = 0, j = 1;
    while (j<L.Length)
    {
        if (L.elem[i] == L.elem[j]) j++;
        else
        {
            i++;
            L.elem[i] = L.elem[j];
            j++;
        }
    }
    L.Length = i + 1;
    return OK;
}
```

2.8.1

```
// 2.8.1
Status DelDup(Linklist L)
{
    // 删除无序链表中重复的元素，带头结点
    for (p = L->next; p; p = p->next)
    {
        // 遍历链表
        q = p->next;
        r = p;
        while (q)
        {
            if (p->data == q->data)
            {
                r->next = q->next;
                free(q);
                q = r;
            }
            r = q;
            q = q->next;

        }
    }
    return OK;
```

```
}// DelDup
```

2.8.2

```
// 2.8.2
Status DelDup_Order(LinkList L)
// 删除非递减链表的中的重复元素，带头结点
{
    for (p = L->next; p->next; p = p->next) {
        // 遍历链表，比较相邻结点是否相等
        while (p->data == p->next->data) {
            q = p->next; p->next = q->next;
            free(q);
        }
    }
    return OK;
}// DelDup_Order
```

2.10

```
// 2.10
Status DelPrior(LNode *s)
// 删除循环链表s结点的前驱
{
    p = s;
    // 找到s结点的前驱的前驱p
    while (p->next->next != s) p = p->next;
    free(p->next);
    p->next = s;
    return OK;
}// DelPrior
```

2.12

```
// 2.12
Status PartOddEven(SqList L)
// 将顺序表中的奇数放在左侧，偶数放在右侧
{
    temp = L.elem[0];
    i = 0; j = L.length - 1; dir = 0;
    while (i<j) {
        if (dir == 0) { // 从后向前找到第一个奇数
            if (L.elem[j] % 2 == 1) { // 奇数
                L.elem[i] = L.elem[j];
```

```
                i++; dir = 1;
            }
            else j--;
        }
        else { // 从前向后找第一个偶数
            if (L.elem[i] % 2 == 0) { // 偶数
                L.elem[j] = L.elem[i];
                j--; dir = 0;
            }
            else i++;
        }
    }// while
    L.elem[i] = temp;
}// PartOddEven
```