线性表的顺序表示

顺序表

存储块

逻辑上相邻的两个元素在物理位置上也相邻

顺序表的定义：

```
#define MaxSize 50                  //定义线性表的长度
typedef struct{
   ElemType data[MaxSize];          //顺序表的元素
   int len;                         //顺序表的当前长度
}SqList;                            //顺序表的类型定义
```

优缺点

优点

- 可以随机存取（根据表头元素地址和元素序号）表中任意一个元素。
- 存储密度高，每个结点只存储数据元素。

缺点

- 插入和删除操作需要移动大量元素。
- 线性表变化较大时，难以确定存储空间的容量。
- 存储分配需要一整段连续的存储空间，不够灵活。

插入操作

- 最好情况：在表尾插入元素，不需要移动元素，时间复杂度为O(1)。
- 最坏情况：在表头插入元素，所有元素依次后移，时间复杂度为O(n)。
- 平均情况：在插入位置概率均等的情况下，平均移动元素的次数为n/2，时间复杂度为O(n)。

代码片段：

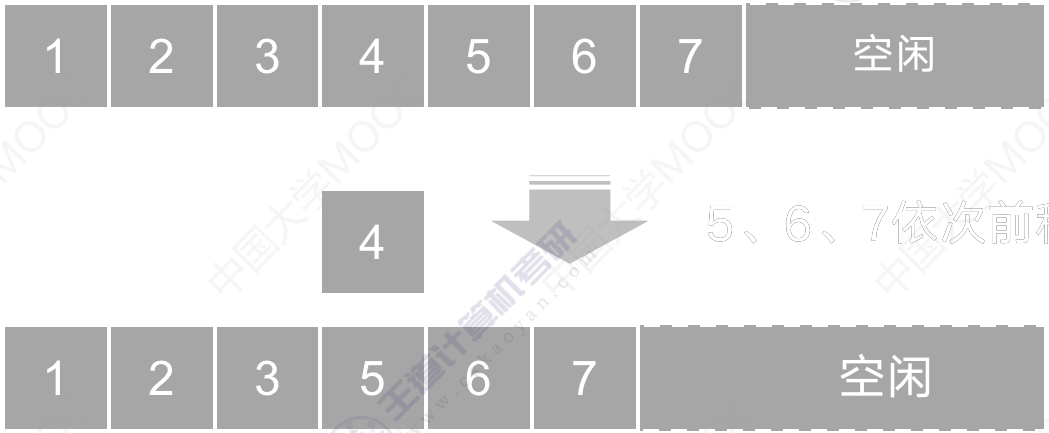
```
//判断插入位置i是否合法（满足1≤i≤len+1）
//判断存储空间是否已满（即插入x后是否会超出数组长度）
for(int j=L.len;j>=i;j--)   //将最后一个元素到第i个元素依次后移一位
    L.data[j]=L.data[j-1];
L.data[i-1]=x;              //空出的位置i处放入x
L.len++;                    //线性表长度加1
```

注意：线性表第一个元素的数组下标是0。

有能力的同学，请用C语言完整实现上述伪代码。

要求删除第4个元素

删除操作

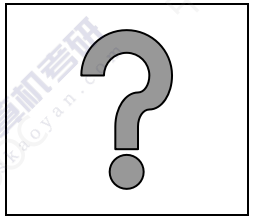


- 最好情况：删除表尾元素，不需要移动元素，时间复杂度为O(1)。
- 最坏情况：删除表头元素，之后的所有元素依次前移，时间复杂度为O(n)。
- 平均情况：在删除位置概率均等的情况下，平均移动元素的次数为(n-1)/2，时间复杂度为O(n)。

代码片段：

```
//判断删除位置i是否合法（满足1≤i≤len）
 e=L.data[i-1];                          //将被删除的元素赋值给e
for(int j=i;j<L.len;j++)                 //将删除位置后的元素依次前移
     L.data[j-1]=L.data[j];
L.len--;                                 //线性表长度减1
```

注意：插入和删除时，i的合法范围是不一样的。
有能力的同学，请用C语言完整实现上述伪代码。



请思考：

动态分配的数组还属于顺序存储结构吗？

动态分配

```
#define InitSize 100                    //表长度的初始定义
typedef struct{
    ElemType *data;                     //指示动态分配数组的指针
    int MaxSize,length;                 //数组的最大容量和当前个数
}SeqList;                               //动态分配数组顺序表的类型定义
```

C的初始动态分配语句为：

```
L.data=(ElemType*)malloc(sizeof(ElemType)*InitSize);
```

C++的初始动态分配语句为：

```
L.data=new ElemType[InitSize];
```