线性表及顺序存储结构

学习目标

- 熟知线性表的定义与结构
- 掌握线性表的顺序存储和相关操作
- 熟练分析相关存储算法的时间复杂度

线性表

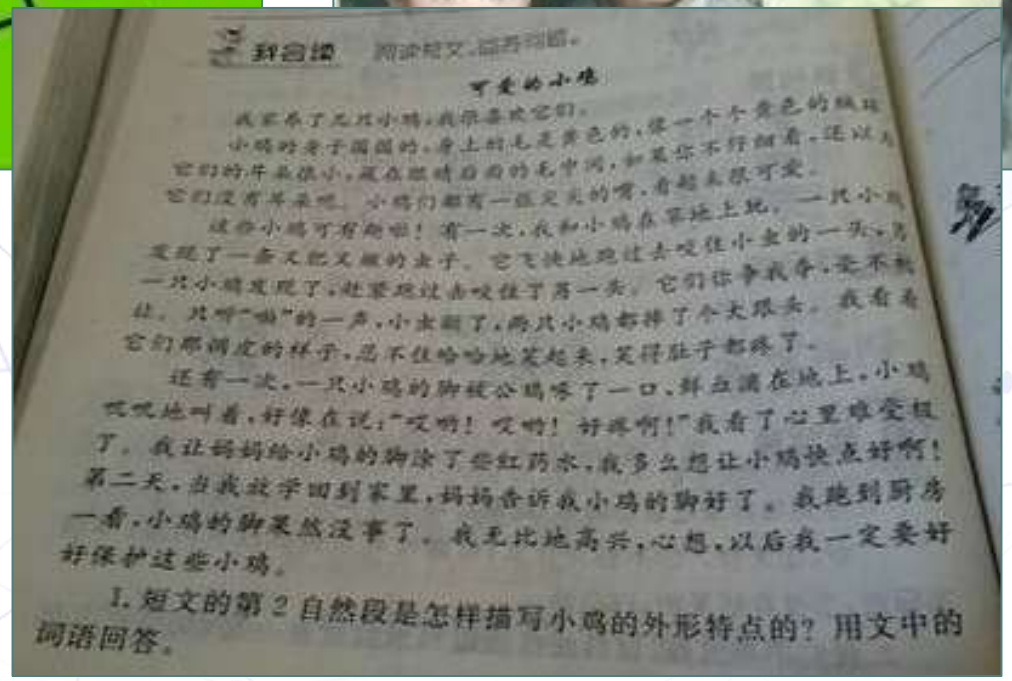
线性表是一种最基本、最简单的数据结构，用来描述数据元素之间单一的前驱和后继关系。

现实生活中，许多问题抽象出的数据模型是线性表。

随处可见的线性结构


一星期有7天。


亥
子
丑
寅
卯
辰
巳
午
未
申
酉
戌


可爱的小鸡
我家养了几只小鸡，我很喜欢它们。
小鸡的身子圆圆的，身上的毛是黄色的，像一个个黄色的绒球。它们的爪子很小，藏在脚背后面的毛中间，如果你不仔细看，还以为它们没有爪呢。小鸡们都有一张尖尖的嘴，看起来很可爱。
这些小鸡可有趣啦！有一次，我和小鸡在草地上玩，一只小鸡发现了一条又肥又嫩的虫子，它飞快地跑过去咬住小虫的一头，另一只小鸡发现了，赶紧跑过去咬住了另一头，它们你争我夺，谁不肯让。只听"啪"的一声，小虫断了，两只小鸡都掉了个大跟头，我看着它们那滑稽的样子，忍不住哈哈地笑起来，笑得肚子都疼了。
还有一次，一只小鸡的脚被公鸡啄了一口，鲜血滴在地上，小鸡吱吱地叫着，好像在说："哎哟！哎哟！好疼啊！"我看了心里难受极了。我让妈妈给小鸡的脚涂了些红药水，我多么想让小鸡快点好啊！第二天，当我放学回到家里，妈妈告诉我小鸡的脚好了。我跑到厨房一看，小鸡的脚果然没事了。我无比地高兴，心想，以后我一定要好好保护这些小鸡。
1. 短文的第2自然段是怎样描写小鸡的外形特点的？用文中的词语回答。

线性表

线性表不仅有着广泛的应用，而且也是其它数据结构的基础。

线性表的存储结构具有代表性，特别链式存储在其它结构（树、图等）也会涉及。

问题引入：一元多项式

例2.1 一元多项式及其运算

- 一元多项式：$f(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1} + a_n x^n$
- 主要运算：多项式相加、相减、相乘等

- 问题：如何在计算机中表示一元多项式并实现相关的运算？
- 多项式的关键数据：多项式项数、每一项系数（以及对应指数）

方法1：采用顺序存储结构直接表示一元多项式

用数组a存储多项式的相关数据：数组分量a[i]表示项x^i的系数

例如：$4x^5-3x^2+1$

$a[i]$	1	0	-3	0	0	4	……
下标i	0	1	2	3	4	5	……

- 这种表示的优点和缺点？

优点：多项式相加容易

缺点：如何表示 $1+2x^{30000}$

方法2：采用顺序存储结构表示多项式的非零项

- 每个非零项 $a_i x^i$ 涉及两个信息：指数i和系数a_i
- 可以将一个多项式看成是一个(a_i，i)二元组的集合。

$$\{(a_n,\ n),\ (a_{n-1},\ n-1),\ \ldots\ldots,\ (a_0,\ 0)\}$$

$$P_1(x) = 9x^{12} + 15x^8 + 3x^2$$

系数	9	15	3	-
指数	12	8	2	-
数组下标i	0	1	2	……

$$P_2(x) = 26x^{19} - 4x^8 - 13x^6 + 82$$

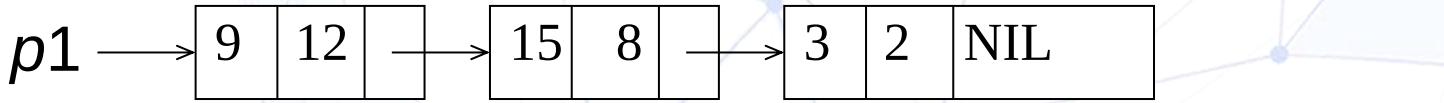
系数	26	-4	-13	82	-
指数	19	8	6	0	-
数组下标i	0	1	2	3	……

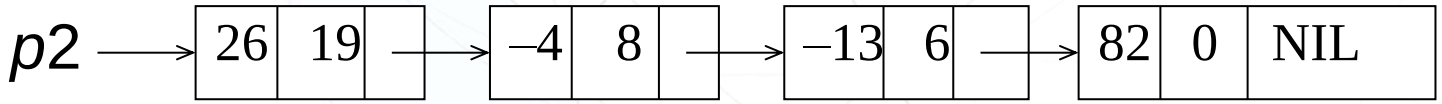
- 这种表示的优点和缺点？

方法3：采用链表结构来存储多项式的非零项

链表表示多项式：链表结点对应一个非零项，包括：系数、指数、指针域

$P_1(x) = 9x^{12} + 15x^8 + 3x^2$　　$P_2(x) = 26x^{19} - 4x^8 - 13x^6 + 82$





- 这种表示的优点和缺点？

启示：数据结构的设计

- 往往需要在算法简洁、可理解性与时间、空间效率之间权衡
- 针对具体问题选择合适的数据结构及设计相应的算法

关于线性结构

- 什么是线性结构？在逻辑上有什么特点？
- 如何存储线性结构？
- 在不同的存储结构上，如何实现插入、删除、查找等基本操作？
- 在不同的存储结构上，基本操作的时空性能如何？

线性表的定义

线性表（表）：n（$n \geq 0$）个具有**相同类型**的数据元素构成的有限且有序的序列。

$(a_1, a_2, \ldots, a_i, \ldots, a_n)$

a_i（$1 \leq i \leq n$）称为数据元素

下角标 i 表示该元素在线性表中的位置或序号

- 线性表的长度：线性表中数据元素的个数
- 空表：长度等于零的线性表

线性表的抽象数据类型定义

代码2-1：线性表的抽象数据类型定义

ADT List {

数据对象：

{ a_i| a_i∈ElemSet，i=1,2,…n, n > 0 } 或Φ，即空表；ElemSet为元素集合。 注意：线性表元素是从1开始计数

数据关系：

{<a_i, a_{i+1}>|a_i, a_{i+1} ∈ElemSet，i=1,2,…n-1 }，a_1为表头元素，a_n为表尾元素。

基本操作：

InitList(*list*)：初始化一个空的线性表*list*。

DestroyList(*list*)：释放线性表*list*占用的所有空间。

Clear (*list*)：清空线性表*list*。

IsEmpty (*list*)：当线性表*list*为空时返回真值，否则返回假值。

Length (*list*)：返回线性表*list*中的元素个数，即表的长度。

Get (*list, i*)：返回线性表*list*中第*i*个元素的值。

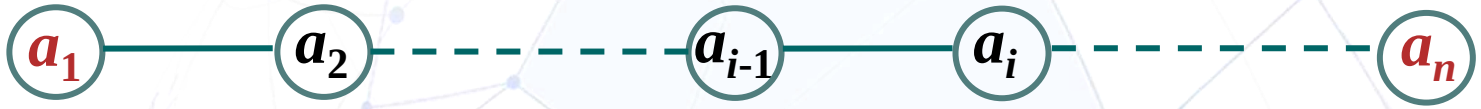
Search(*list, x*)：在线性表*list*中查找元素*x*，查找成功，返回*x*的位置，否则返回NIL。

Insert (*list, i, x*)：在线性表*list*的第*i*个位置上插入元素*x*。

Remove (*list, i*)：从线性表*list*中删除第*i*个元素。

}

线性表的逻辑特征



1. 数据元素**个数的有限性**　$L_1 = (1, 3, 5, 7, 9)$　$L_2 = ('a', 'e', 'i', 'o', 'u')$
2. 数据元素**类型的相同性**　$L_3 = ((Li, 8, 3), (Wang, 7, 4), (Zhang, 5, 5))$
3. 数据元素**类型的抽象性** ⇨ 不确定、任意
4. 相邻数据元素的**序偶关系**，且 a_1 无前驱，a_n 无后继

📌 序偶：两个具有固定次序的元素组成的序列，记作<a，b>，且称 a 是 b 的前驱，b 是 a 的后继

线性表的结构

线性表的逻辑结构：

数据元素之间线性的序列关系，即数据元素之间的前驱和后继关系。

线性表的物理结构：

线性表在计算机中的存储方式，又称为存储结构，即从程序实现的角度将逻辑结构映射到计算机存储单元中。

线性表的存储结构

存储结构主要有两种形式：顺序存储结构和链式存储结构。

- **顺序存储结构**：数据元素被**顺序地**存储在**连续的**内存空间中，前驱和后继元素在物理空间上是相邻的，对应C++中的数组。
- **链式存储结构**：可以动态地申请存储数据的结点空间，并使用类似指针这样的手段将结点按顺序前后链接起来

顺序存储方法

线性表的顺序存储结构(也称作顺序表)

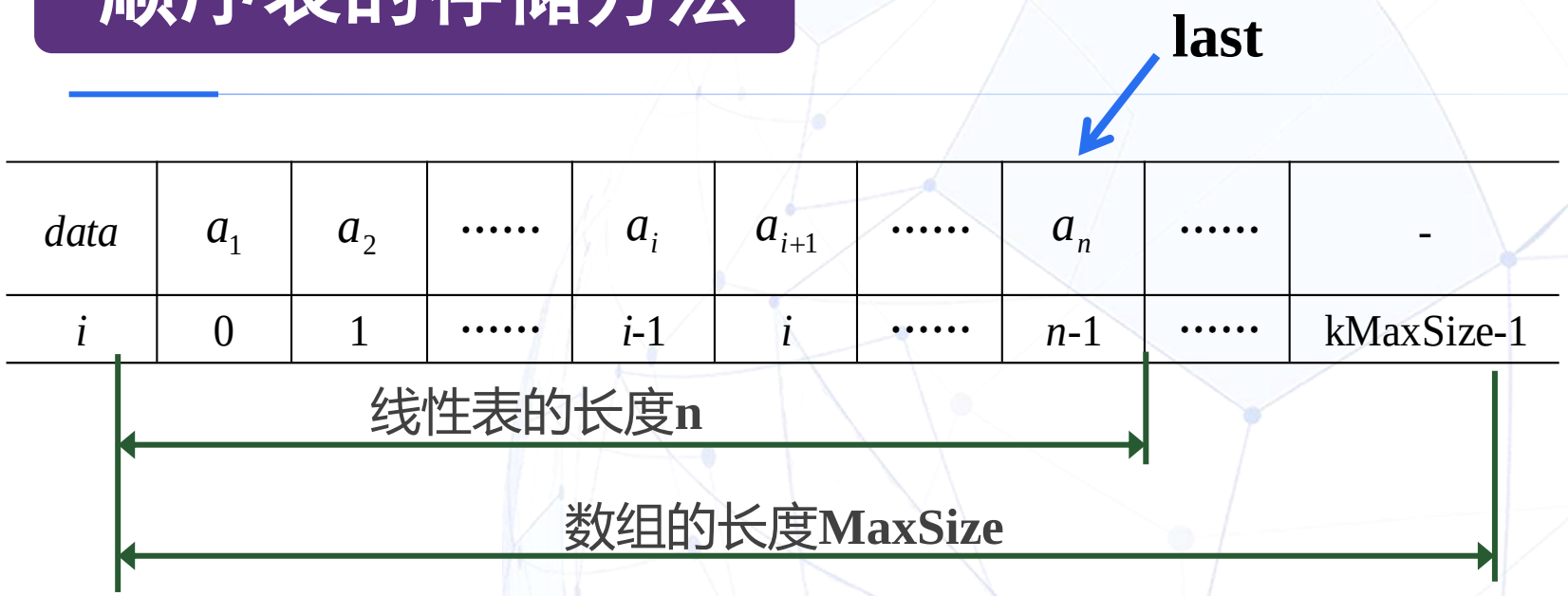
例：（34, 23, 67, 43）

存储要点

- 用一段地址**连续**的存储单元
- **依次**存储线性表中的数据元素

某些内存单元可能是空吗?

顺序表的存储方法



用什么属性来描述顺序表?

存储空间的起始位置

顺序表的容量（最大长度）

顺序表的最后一个元素在数组中的位置，-1表示空表

```
const int MaxSize = 100;
template <typename DataType>
class SeqList
{
    public:
        [methods]
    private:
        DataType* data;
        DataType data[MaxSize];
        int last;
};
```

顺序表的基本操作

1.初始化

2.查找

3.插入

4.删除

顺序表的基本操作-初始化

1.初始化：顺序表的初始化即构造一个空表

(1)动态分配表结构所需要的存储空间

(2)将表中 list.last 指针置为-1，表示表中没有数据元素。

代码 2-2　产生一个初始空顺序表 InitList(*list*)

输入： 顺序表 *list*

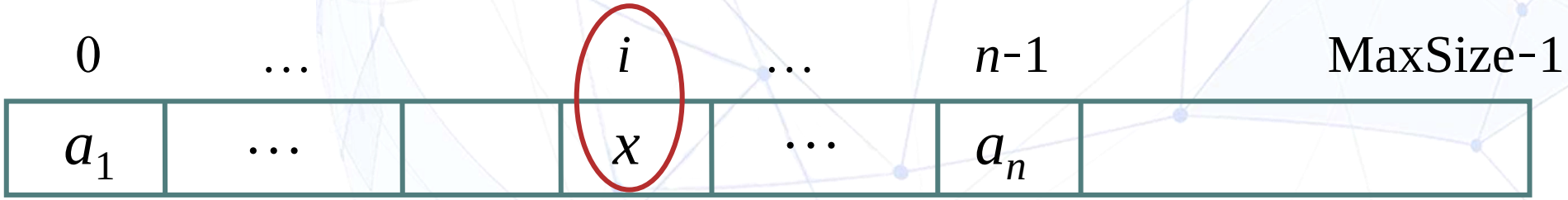
输出： 完成了初始化的空顺序表 *list*

```
1    list.data ← new ElemSet[kMaxSize]    //申请顺序表空间
2    list.last ← -1;          //空表中 list.last 值为-1
```

顺序表的基本操作-查找

查找：在线性表中查找与给定值x相等的数据元素

方法：从第一个元素起依次和x比较，直到找到一个与x相等的数据元素，返回它在顺序表中的存储下标；或者查遍整个表都没有找到与x相等的元素，则返回NIL。



按值查找的时间复杂度？ ⇨ $O(n)$

顺序表的基本操作-查找

算法 2-1 在顺序表 *list* 中查找元素 *x*　Search(*list*, *x*)

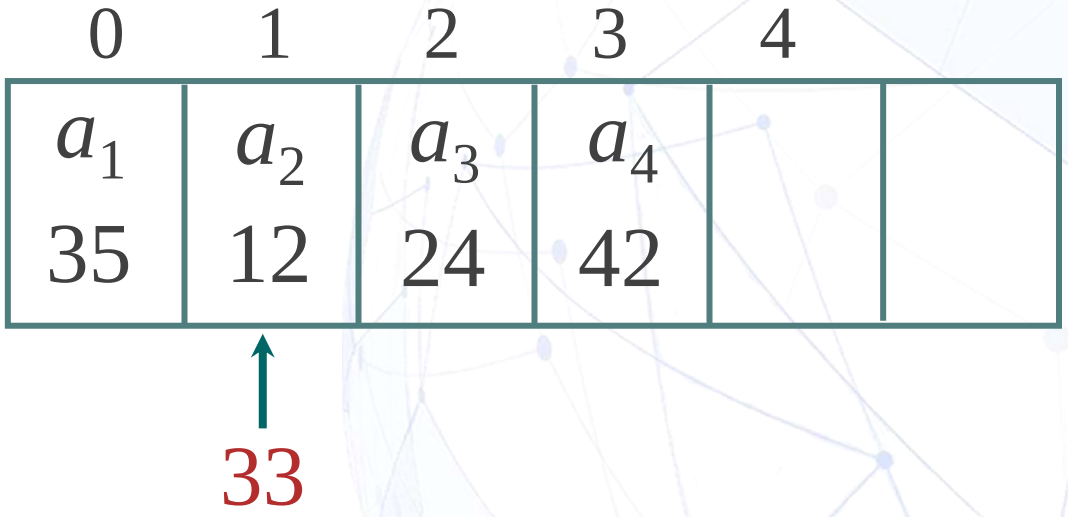
输入： 顺序表 *list*，$x \in$ ElemSet

输出： 元素 *x* 在顺序表 *list* 中的位置 *i*（由于顺序表中的位置从 0 开始，*x* 的实际位序是 *i*+1）；
如果 *x* 不在顺序表中返回 NIL

```
1   i ← 0;
2   while i ≤ list.last 并且 list.data[i] ≠ x do
3   |   i←i+1
4   end
5   if  i > list.last then      // 如果没找到，返回 NIL
6       i ← NIL
7   end
8   return i
```

顺序表的实现——插入

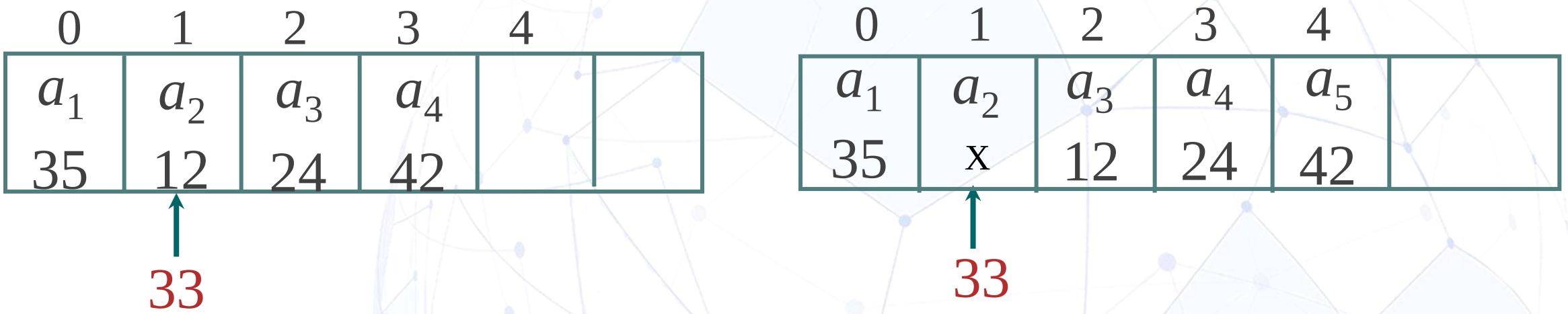
例 1　对于线性表（35，12，24，42），在 $i = 2$的位置上插入元素33



什么情况下插入无法进行？ ⇨ 注意边界条件

表满：**last >= MaxSize-1**

顺序表的基本操作-插入



在表的第i个位序上插入一个值为x的新元素

$(a_1,a_2,\ldots,a_{i-1},a_i,a_{i+1},\ldots,a_n)$ --> $(a_1,a_2,\ldots,a_{i-1},x,a_i,a_{i+1},\ldots,a_n)$

执行步骤：

（1）将$a_i \sim a_n$顺序向后移动（移动次序从后到前），为新元素让出位置；

（2）将x置入空出的第i个位序；

（3）修改 list.last 指针(相当于修改表长)，使之仍指向最后一个元素。

顺序表的基本操作-插入

算法 2-2 在顺序表 *list* 的第 *i* 个位置上插入元素 *x*　Insert (*list*, *i*, *x*)

输入： 顺序表 *list*，*i* 是插入位置的序号（从 1 开始），*x*∈ ElemSet

输出： 完成插入后的顺序表 *list*

```
1   if  list.last = kMaxSize−1 then    // 表空间已满，不能插入
2   |   表满不能插入，退出
3   end
4   if  i<1 或者 i>list.last+2 then    //检查 i 的合法性。注意 i 代表位序，不是数组下标
5   |   插入位置不合法，退出
6   end
7   for  j ← list.last downto i-1 do
8   |   list.data[j+1] ← list.data[j]    //将 a_i~a_n 顺序向后移动
9   end
10  list.data[i-1] ← x    // 新元素插入
11  list.last ← list.last + 1 // list.last 仍指向最后元素
```

注意边界条件

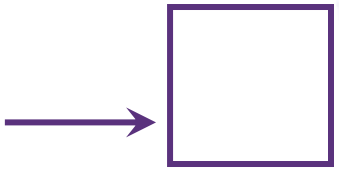
注意：last是数组下标，是从0开始的
存储必须连续

最好/坏情况
时间复杂度为O(n）

顺序表的实现——删除

对于线性表（35, 33, 12, 24, 42），删除位置 $i = 2$的数据元素

0	1	2	3	4	
a_1	a_2	a_3	a_4	a_5	
35	33	12	24	42	



顺序表的基本操作-删除

将线性表中的第i个位序上的元素删除

$(a_1,a_2,\ldots,a_{i-1},\mathbf{a_i},a_{i+1},\ldots,a_n)$ --> $(a_1,a_2,\ldots,a_{i-1},a_{i+1},\ldots,a_n)$

■ 执行步骤：

(1)将～顺序向前移动，元素被覆盖；

(2)修改list.last指针(相当于修改表长)使之仍指向最后一个元素。

顺序表的基本操作-删除

算法 2-3 从顺序表 *list* 中删除第 *i* 个元素　Remove (*list*, *i*)

输入： 顺序表 *list*，*i* 是删除元素的位置序号（从 1 开始）

输出： 完成删除后的顺序表 *list*

```
1  if  i<1 或 i > list.last + 1 then  // 检查空表及删除位置的合法性
2  |   不存在这个元素，退出
3  end
4  for  j←i to list.last do
5  |   list.data[j-1]← list.data[j]  // 将 a_{i+1} ~ a_n 顺序向前移动
6  end
7  list.last ←list.last – 1  // list.last 仍指向最后元素
```

注意边界条件

时间复杂度为O(n）

顺序表的优缺点

优点：

- 存储密度大
- 可以随机存取表中的任一元素

$$\mathbf{Loc}(a_i) = \mathbf{Loc}(a_1) + (i-1) \times c \quad //c是一个元素需要的字节数$$

缺点：

- 在插入、删除某一元素时，需要移动大量元素
- 浪费存储空间
- 属于静态存储形式，数据元素的个数不能自由扩充

课堂小结

线性表的定义和结构介绍

线性表的顺序存储（顺序表）的结构和操作实现