



Data Structures

# 线性表 Linear Lists

2023年9月5日

学而不厌 诲人不倦

- ➡ 2.1 引言
- ➡ 2.2 线性表的逻辑结构
- ➡ 2.3 线性表的顺序存储结构及实现
- ➡ 2.4 线性表的链接存储结构及实现
- ➡ 2.5 顺序表和链表的比较
- ➡ 2.6 约瑟夫环与一元多项式求和

## 2.1 引言



随处可见的线性结构





## 2.1 引言



### 学籍管理问题

学号	姓名	性别	出生日期	籍贯
15041001	王 军	男	19970102	吉林省图们市
15041002	李 明	男	19980328	吉林省吉林市
15041003	汤晓影	女	19971116	吉林省长春市
...	...	...	...	...

二维表



线性结构

## 2.1 引言



### 工资管理问题

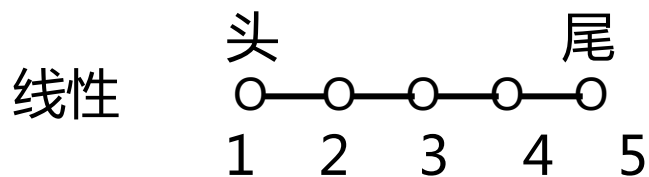
职工号	姓名	性别	基本工资	岗位津贴	业绩津贴
000826	王一梅	女	3480	1900	1380
000235	李明	男	3860	2400	1600
000973	郑浩	男	2850	1600	1050
...	...	...	...	...	...

二维表



线性结构

### 线性结构特点:

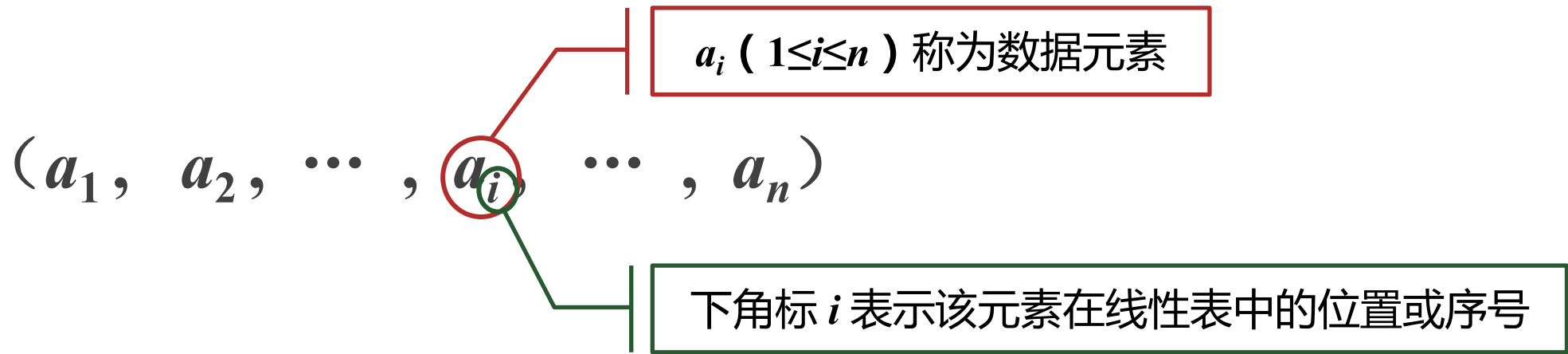


- 唯一头元素
- 唯一尾元素
- 除头元素外，均有一个直接前驱
- 除尾元素外，均有一个直接后继

## 2.2 线性表的逻辑结构

### 线性表的定义

✦ 线性表（表）： $n$  ( $n \geq 0$ ) 个具有**相同类型**的数据元素的**有限序列**



✦ 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

## 2.2 线性表的逻辑结构

### 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**  $\Rightarrow$  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





## 2.2 线性表的逻辑结构

### 线性表的抽象数据类型定义

ADT List

DataModel

线性表中的数据元素具有相同类型，相邻元素具有前驱和后继关系

Operation

**InitList**：表的初始化，建一个空表

**DestroyList**：销毁表，释放表所占用的存储空间

**Length**：求表的长度

**Get**：在表中取序号为  $i$  的数据元素

**Locate**：在线性表中查找值等于  $x$  的元素

**Insert**：在表的第  $i$  个位置处插入一个新元素  $x$

**Delete**：删除表中的第  $i$  个元素

**Empty**：判断表是否为空

endADT

## 2.2 线性表的逻辑结构

### 线性表的抽象数据类型定义

#### **InitList**

输入：无

功能：表的初始化，建一个空表

输出：无

#### **DestroyList**

输入：无

功能：销毁表，释放表所占用的存储空间

输出：无

#### **Length**

输入：无

功能：求表的长度

输出：表中数据元素的个数

## 2.2 线性表的逻辑结构

### 线性表的抽象数据类型定义

#### Get

输入：元素的序号  $i$

功能：在表中取序号为  $i$  的数据元素

输出：若  $i$  合法，返回序号为  $i$  的元素值

#### Locate

输入：数据元素  $x$

功能：在线性表中查找值等于  $x$  的元素

输出：若查找成功，返回  $x$  在表中的序号，否则返回0

#### Insert

输入：插入位置  $i$ ，待插  $x$

功能：在表的第  $i$  个位置处插入一个新元素  $x$

输出：若插入成功，表中增加一个新元素；否则给出失败信息

## 2.2 线性表的逻辑结构

### 线性表的抽象数据类型定义

#### Delete

输入：删除位置  $i$

功能：删除表中的第  $i$  个元素

输出：若删除成功，表中减少一个元素；否则给出失败信息

#### Empty

输入：无

功能：判断表是否为空

输出：若是空表，返回 1，否则返回 0

- (1) 线性表的基本操作根据实际应用而定
- (2) 复杂的操作可以通过基本操作的组合来实现
- (3) 对不同的应用，操作的接口可能不同

## 2.3 顺序表的存储结构及实现

### 顺序表的存储方法

✦ 顺序表（向量）：线性表的顺序存储结构 例：（34, 23, 67, 43）

34	23	67	43		4
----	----	----	----	--	---

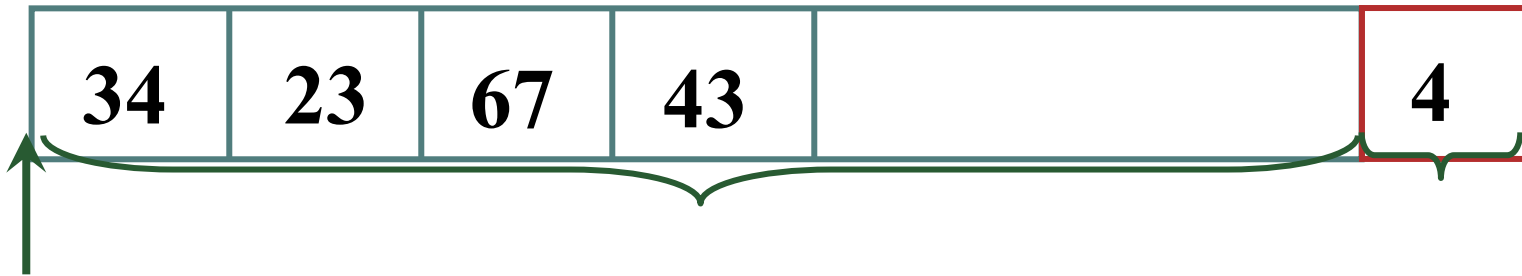
💡 存储要点 { 用一段地址**连续**的存储单元  
**依次**存储线性表中的数据元素

🕒 某些内存单元可能是空吗？

## 2.3 顺序表的存储结构及实现

### 顺序表的存储方法

✦ 顺序表（向量）：线性表的顺序存储结构 例：（34, 23, 67, 43）



🕒 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）

📎 顺序表的当前长度



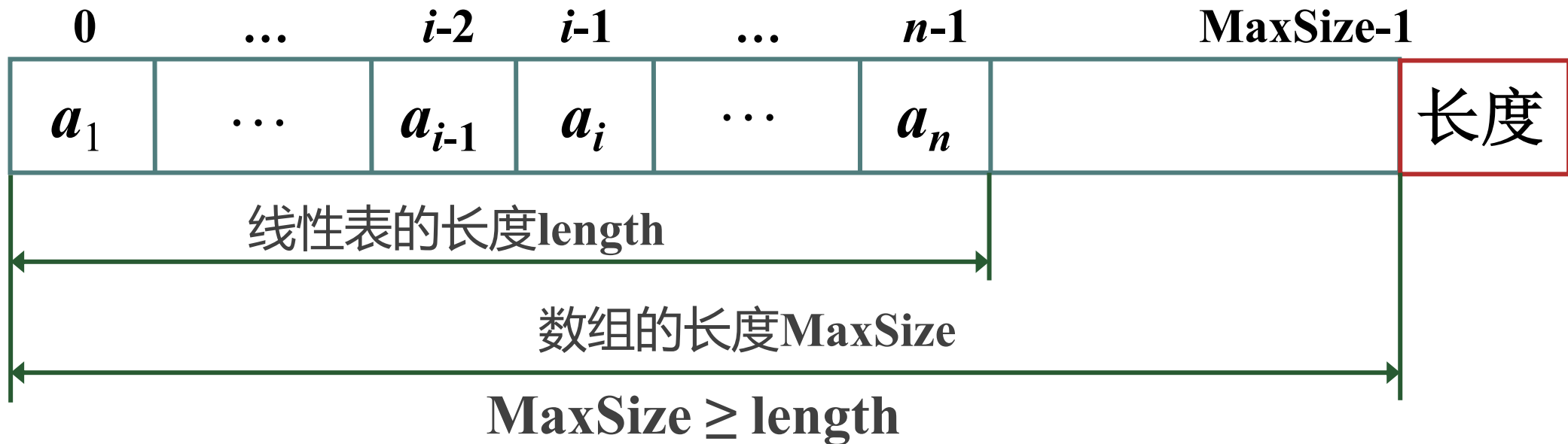


## 2.3 顺序表的存储结构及实现

### 顺序表的存储方法

📌 顺序表：线性表的顺序存储结构

$(a_1, \dots, a_{i-1}, a_i, \dots, a_n)$



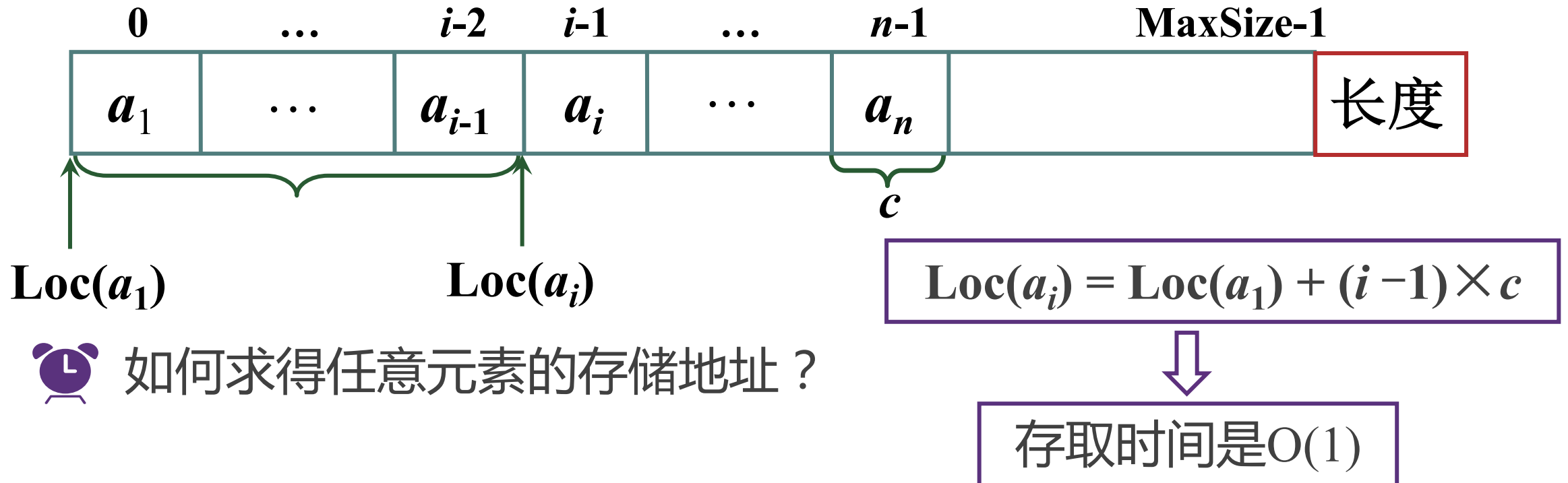


## 2.3 顺序表的存储结构及实现

存取访问

✦ 顺序表：线性表的顺序存储结构

$(a_1, \dots, a_{i-1}, a_i, \dots, a_n)$



🕒 如何求得任意元素的存储地址？



## 2.3 顺序表的存储结构及实现

### 存储结构与存取结构


✦ 随机存取：在  $O(1)$  时间内存取数据元素

- 存储结构——数据及其逻辑结构在计算机中的表示
- 存取结构——在一个数据结构上对（按位置）查找操作的时间性能

顺序表是一种随机存取的存储结构，其含义为：在顺序表这种存储结构上进行的（按位置）查找操作，其时间性能为  $O(1)$

## 2.3 顺序表的存储结构及实现

### 顺序表的实现——类定义

 线性表的抽象数据类型定义？

**InitList**：表的初始化，建一个空表

**DestroyList**：销毁表，释放表所占用的存储空间

**Length**：求表的长度

**Get**：在表中取序号为  $i$  的数据元素

**Locate**：在线性表中查找值等于  $x$  的元素

**Insert**：在表的第  $i$  个位置处插入一个新元素  $x$

**Delete**：删除表中的第  $i$  个元素

**Empty**：判断表是否为空



```
const int MaxSize = 100;
template <typename DataType>
class SeqList
{
public:
    SeqList( );
    SeqList(DataType a[ ], int n);
    ~SeqList( );
    int Length( );
    DataType Get(int i);
    int Locate(DataType x );
    void Insert(int i, DataType x);
    DataType Delete(int i); int Empty( );
    int Empty( );
    void PrintList( );
private:
    DataType data[MaxSize];
    int length;
};
```

## 2.3 顺序表的存储结构及实现

### 顺序表的实现——初始化

 初始化顺序表的函数原型是什么？

#### InitList

输入：无

功能：表的初始化，建一个空表

输出：无

```
SeqList<DataType> :: SeqList()  
{  
    Length = 0;  
}
```



## 2.3 顺序表的存储结构及实现

### 顺序表的实现——建立

```
SeqList<DataType> :: SeqList(DataType a[ ], int n)
{
    if (n > MaxSize) throw “参数非法”;
    for (int i = 0; i < n; i++)
        data[i] = a[i];
    length = n;
}
```

数组  $a$

35	12	24	33	42
----	----	----	----	----



顺序表  $L$

35	12	24	33	42		5
----	----	----	----	----	--	---

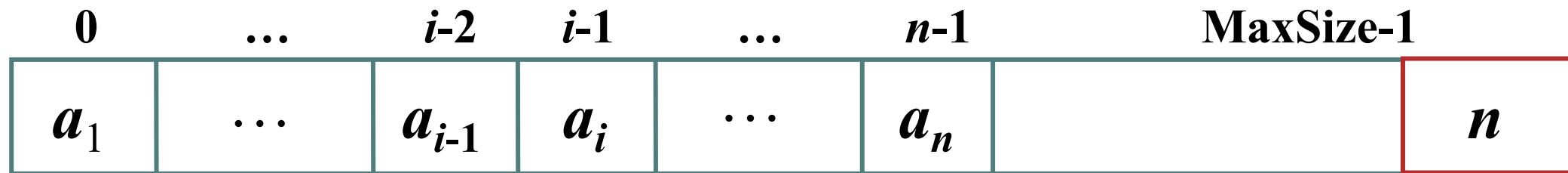




## 2.3 顺序表的存储结构及实现

### 顺序表的实现——判空

```
int SeqList<DataType> :: Empty( )  
{  
    if (length == 0) return 1;  
    else return 0;  
}
```





## 2.3 顺序表的存储结构及实现

### 顺序表的实现——长度

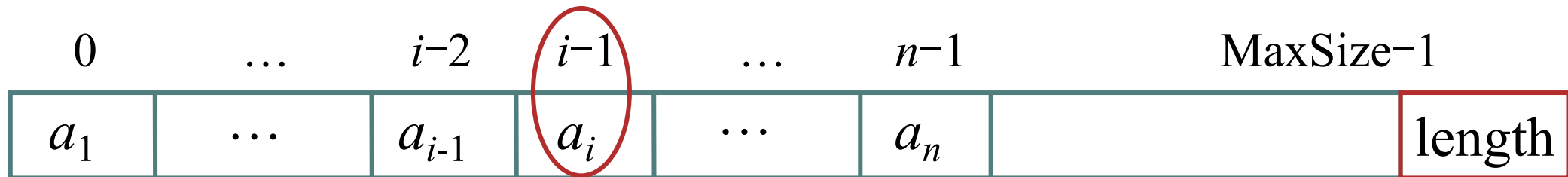
```
int SeqList<DataType> :: Length( )  
{  
    return length ;  
}
```

0	...	$i-2$	$i-1$	...	$n-1$	MaxSize-1
$a_1$	...	$a_{i-1}$	$a_i$	...	$a_n$	$n$

## 2.3 顺序表的存储结构及实现

### 顺序表的实现——按位查找

```
template <typename DataType>
DataType SeqList<DataType> :: Get(int i)
{
    if (i < 1 && i > length) throw "查找位置非法";
    else return data[i - 1];
}
```



按位查找的时间复杂度？



$O(1)$



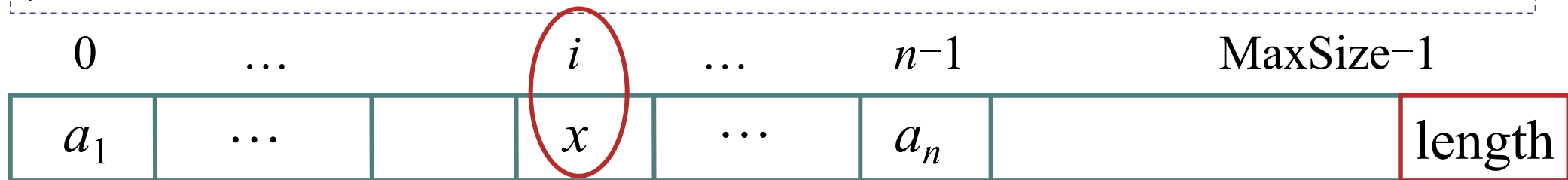
随机存取

## 2.3 顺序表的存储结构及实现

### 顺序表的实现——按值查找

```
template <typename DataType>
int SeqList<DataType> :: Locate(DataType x)
{
    for (int i = 0; i < length; i++)
        if (data[i] == x) return i+1;
    return 0;
}
```

//返回其序号i+1  
//退出循环，说明查找失败



 按值查找的时间复杂度？  $\Rightarrow O(n)$

## 2.3 顺序表的存储结构及实现

### 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

0	i	2	3	4	
$a_1$	$a_2$	$a_3$	$a_4$		
35	12	24	42		

↑  
33



什么情况下插入无法进行？



注意边界条件



表满： $\text{length} \geq \text{MaxSize}$



合理的插入位置： $1 \leq i \leq \text{length} + 1$ （注意： $i$  指的是元素的序号）

## 2.3 顺序表的存储结构及实现

### 顺序表的实现——插入

```
template <typename DataType>
void SeqList<DataType> :: Insert(int i, DataType x)
{
    if (length == MaxSize) throw "上溢";
    if (i < 1 || i > length + 1) throw "插入位置错误";
    for (int j = length; j >= i; j--)
        data[j] = data[j - 1];           //第j个元素存在数组下标为j-1处
    data[i - 1] = x;
    length++;
}
```



基本语句？执行多少次？



## 2.3 顺序表的存储结构及实现

### 顺序表的实现——插入

- ✈️ 最好情况 (  $i = n+1$  ) : 执行0次, 时间复杂度为  $O(1)$
- ✈️ 最坏情况 (  $i = 1$  ) : 执行  $n+1$  次, 时间复杂度为  $O(n)$
- ✈️ 平均情况 (  $1 \leq i \leq n+1$  ) : 时间复杂度为  $O(n)$

$$\sum_{i=1}^{n+1} p_i (n-i+1) = \frac{1}{n+1} \sum_{i=1}^{n+1} (n-i+1) = \frac{n}{2} = O(n)$$

```
for (int j = length; j >= i; j--)  
    data[j] = data[j - 1];
```

🕒 基本语句？执行多少次？

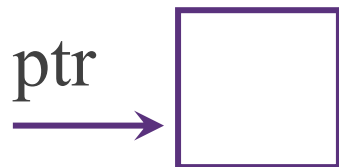


## 2.3 顺序表的存储结构及实现

### 顺序表的实现——删除

例 2 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		<del>5</del> 4



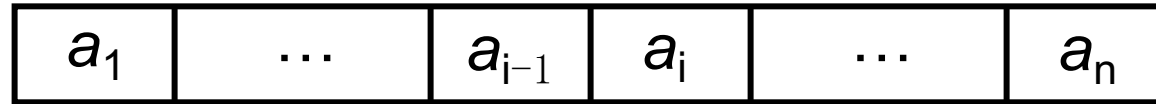
请仿照顺序表的插入算法，完成删除算法：

1. 分析边界条件
2. 分别给出伪代码和C++语言实现
3. 分析时间复杂度

## 小结



### 顺序存储线性表特点



以元素在计算机内的“**物理位置相邻**”来表示线性表中数据元素之间的“**逻辑关系相邻**”。

只要确定线性表的起始地址，线性表中的任意数据元素都可随意存取，故线性表的**顺序存储结构**又称为**随机存取存储结构**。

#### 优点:

- 可随机存取表中任意数据元素
- 直接可获取线性表的长度

#### 缺点:

- 数据元素的**插入**、**删除**相对麻烦

## 实验一、顺序存储结构线性表的建立及操作

### 一、实验目的

1. 掌握线性表的存储结构的特点，理解顺序存储结构表示线性表的方法。
2. 掌握顺序存储结构线性表数据元素类型定义的格式与方法。
3. 掌握对线性表的元素进行删除和插入的原理与方法。
4. 用C++语言实现**顺序结构存储线性表的建立，元素的删除与插入算法**，并上机调试。

### 二、实验内容

1. 设计C++类及相关方法，用于维护学生成绩表：

**基础信息：**学号姓名分数：`long num; char name[10]; float score;`

2. 写出建立线性表，并向线性表中输入数据的函数。
3. 写出删除指定学号的学生信息，及按学生的成绩顺序插入新的学生信息的函数。（假定学生的成绩已有序排列）
4. 写出输入及输出的内容。
5. 合并两张有序表（**扩展内容**）

**实验时间：** 第2周周四晚

**22网安：**18:30-20:10 **22物联网：**20:10-21:50

**实验地点：** 软件学院502

## 算法设计1：两个线性表La，Lb的合并。

要求: 扩展线性表La，将存在于Lb中而不存在于La中的数据元素插入到La中。

例， La = (3,13,7,9)

Lb = (5,7,10)

La	Lb	
3	5	← 查找La, 未找到
13	7	← 查找La, 找到
7	10	← 查找La, 未找到
9		
5		
10		

思想:

1. 依次取出 Lb 中的数据元素进行处理
2. 判断该数据元素是否存在于 La 中
3. 在则取下一个数据元素，否则插入到 La 中

## 算法设计2：两个有序线性表La, Lb的合并。

**要求:** 线性表La、Lb中的数据元素按值**非递减**有序排列，合并La、Lb构造Lc，使Lc中的数据元素仍按值**非递减**有序排列。

思想:

例,  $La = (3, 5, 8, 11)$   
 $Lb = (2, 6, 8, 9, 11, 15, 20)$

构造  $Lc = (2, 3, 5, 6, 8, 8, 9, 11, 11, 15, 20)$

Diagram illustrating the merge process:

- Indices  $i$  and  $j$  point to the current elements in  $La$  and  $Lb$  respectively.
- Arrows indicate the selection of elements from  $La$  and  $Lb$  to be merged into  $Lc$ .





*Thank You !*

*Q & A*